Armageddon

unknown

Dec 17, 2021

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:

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

Python asteroid airburst calculator

class armageddon.**HeatMap**(*data*, *name=None*, *min_opacity=0.5*, *max_zoom=18*, *radius=25*, *blur=15*, *gradient=None*, *overlay=True*, *control=True*, *show=True*, ***kwargs*)

Create a Heatmap layer

Parameters

- **data** (*list of points of the form [lat, lng] or [lat, lng, weight]*) – The points you want to plot. You can also provide a numpy.array of shape (n,2) or (n,3).
- **name** (*string, default None*) – The name of the Layer, as it will appear in LayerControls.
- **min_opacity** (*default 1.*) – The minimum opacity the heat will start at.
- **max_zoom** (*default 18*) – Zoom level where the points reach maximum intensity (as intensity scales with zoom), equals maxZoom of the map by default
- **radius** (*int, default 25*) – Radius of each "point" of the heatmap
- **blur** (*int, default 15*) – Amount of blur
- **gradient** (*dict, default None*) – Color gradient config. e.g. {0.4: 'blue', 0.65: 'lime', 1: 'red'}
- **overlay** (*bool, default True*) – Adds the layer as an optional overlay (True) or the base layer (False).
- **control** (*bool, default True*) – Whether the Layer will be included in LayerControls.
- **show** (*bool, default True*) – Whether the layer will be shown on opening (only for overlays).

class armageddon.**MarkerCluster**(*locations=None*, *popups=None*, *icons=None*, *name=None*, *overlay=True*, *control=True*, *show=True*, *icon_create_function=None*, *options=None*, ***kwargs*)

Provides Beautiful Animated Marker Clustering functionality for maps.

Parameters

- **locations** (*list of list or array of shape (n, 2)*) – Data points of the form [[lat, lng]].
- **popups** (*list of length n, default None*) – Popup for each marker, either a Popup object or a string or None.
- **icons** (*list of length n, default None*) – Icon for each marker, either an Icon object or a string or None.

- **name** (*string, default None*) – The name of the Layer, as it will appear in LayerControls
- **overlay** (*bool, default True*) – Adds the layer as an optional overlay (True) or the base layer (False).
- **control** (*bool, default True*) – Whether the Layer will be included in LayerControls.
- **show** (*bool, default True*) – Whether the layer will be shown on opening (only for overlays).
- **icon_create_function** (*string, default None*) – Override the default behaviour, making possible to customize markers colors and sizes.
- **options** (*dict, default None*) – A dictionary with options for Leaflet.markercluster. See https://github.com/Leaflet/Leaflet.markercluster for options.

Example

```
>>> icon_create_function = '''
...     function(cluster) {
...     return L.divIcon({html: '<b>' + cluster.getChildCount() + '</b>',
...                       className: 'marker-cluster marker-cluster-small',
...                       iconSize: new L.Point(20, 20)});
...     }
... '''
```

class armageddon.**Planet**(*atmos_func='exponential'*, *atmos_filename='./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 :param atmos_func: 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'

Parameters

- **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(*f*, *u0*, *dt*, *strength*, *density*)

Solve the a coupled set of (total six) ordinary differential equations. Implement RK4 algorithm :param f: ODE function lists to solve :type f: list :param u0: a list of initial parameters

u0 = [velocity, mass, angle, altitude, distance, radius]

Parameters

- **dt** (*float*) – The timestep
- **strength** (*float*) – The strength of the asteroid (i.e., the ram pressure above which fragmentation and spreading occurs) in N/m^2 (Pa)
- **density** (*float*) – The density of the asteroid in kg/m^3

Returns Result – u = [velocity, mass, angle, altitude, distance, radius] t = [time]

Return type array

analyse_outcome(*result*)

Inspect a pre-found solution to calculate the impact and airburst stats :param result: 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 : `Airburst` or `Cratering`), as well as the following 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. :param result: A pandas dataframe with columns for the velocity, mass, angle,

altitude, horizontal distance and radius as a function of time

Parameters Returns (*DataFrame*) – Returns the dataframe with additional column `dedz` which is the kinetic energy lost per unit altitude

chelyabinsk()

Determine asteroid parameters strength and radius that fits an observed energy deposition curve and plot a fig

formula(*t*, *stepdata_asteroid*, *strength*, *density*)

calculate the ode result directly by given parameters f[0]:dv/dt f[1]:dm/dt f[2]:d(theta)/dt f[3]:dz/dt f[4]:dx/dt f[5]:dr/dt

Parameters

- **stepdata_asteroid** (*list of parameters*) – stepdata_asteroid = [velocity, mass, angle, altitude, distance, radius]
- **density** (*float*) – The density of the asteroid in kg/m^3, which is a constant
- **strength** (*float*) – The strength of the asteroid (i.e. the maximum pressure it can take before fragmenting) in N/m^2

Returns **Result** – an array consisting of 6 numbers, as shown below. f[0, 1, 2, 3, 4, 5] f[0]:dv/dt f[1]:dm/dt f[2]:d(theta)/dt f[3]:dz/dt f[4]:dx/dt f[5]:dr/dt

Return type array

`solve_atmospheric_entry`(*radius*, *velocity*, *density*, *strength*, *angle*, *init_altitude=100000.0*, *dt=0.005*, *radians=False*)

Solve the system of differential equations for a given impact scenario :param radius: The radius of the asteroid in meters :type radius: float :param velocity: The entery speed of the asteroid in meters/second :type velocity: float :param density: The density of the asteroid in kg/m^3, which is a constant :type density: float :param strength: The strength of the asteroid (i.e. the maximum pressure it can

take before fragmenting) in N/m^2

Parameters

- **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 time step, 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

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

`solver_analytic`(*v0*, *z0*, *rho0*, *theta0*, *r0*, *mass*, *dt*, *H=8000*)

Solve the system of analytical equations for a given impact scenario :param r0: The radius of the asteroid in meters :type r0: float :param mass: Mass of the asteroid in kg :type mass: float :param velocity: The entery speed of the asteroid in meters/second :type velocity: float :param rho0: The density of the asteroid in kg/m^3, which is a constant :type rho0: float :param strength: The strength of the asteroid (i.e. the maximum pressure it can

take before fragmenting) in N/m^2

Parameters

- **theta0** (*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
- **z0** (*float, optional*) – Initial altitude in m
- **dt** (*float, optional*) – The output time step, in s
- **H** (*float*) – Atmospheric scale height in m

Returns **Result** – A pandas dataframe containing the solution to the system. Includes the following column: 'velocity', 'altitude' List A list which contains analytical solution of velocity

Return type DataFrame

`class` armageddon.`PostcodeLocator`(*postcode_file='./armageddon/resources/full_postcodes.csv'*, *census_file='./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.

Parameters

- **postcodes** (*list of lists*) – list of postcode units or postcode sectors
- **sector** (*bool, optional*) – if true return populations for postcode sectors, otherwise postcode units

Returns Contains the populations of input postcode units or sectors

Return type list of lists

Examples

```
>>> locator = PostcodeLocator()
>>> locator.get_population_of_postcode([['SW7 2AZ',
                                         'SW7 2BT',
                                         'SW7 2BU',
                                         'SW7 2DD']])
>>> locator.get_population_of_postcode([['SW7  2']], True)
```

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

Return (unit or sector) postcodes within specific distances of input location.

Parameters

- **X** (*arraylike*) – Latitude-longitude pair of centre location
- **radii** (*arraylike*) – array of radial distances from X
- **sector** (*bool, optional*) – if true return postcode sectors, otherwise postcode units

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

Return type list of lists

Examples

```
>>> locator = PostcodeLocator()
>>> locator.get_postcodes_by_radius((51.4981, -0.1773), [0.13e3])
>>> locator.get_postcodes_by_radius((51.4981, -0.1773),
                                    [0.4e3, 0.2e3], True)
```

armageddon.**acos**(*x*, */*)

Return the arc cosine (measured in radians) of x.

armageddon.**asin**(*x*, */*)

Return the arc sine (measured in radians) of x.

armageddon.**atan2**(*y*, *x*, */*)
Return the arc tangent (measured in radians) of y/x.

Unlike atan(y/x), the signs of both x and y are considered.

armageddon.**cos**(*x*, */*)
Return the cosine of x (measured in radians).

armageddon.**damage_zones**(*outcome*, *lat*, *lon*, *bearing*, *pressures*)
Calculate the latitude and longitude of the surface zero location and the list of airblast damage radii (m) for a given impact scenario.

Parameters

- **outcome** (*Dict*) – the outcome dictionary from an impact scenario
- **lat** (*float*) – latitude of the meteoroid entry point (degrees)
- **lon** (*float*) – longitude of the meteoroid entry point (degrees)
- **bearing** (*float*) – Bearing (azimuth) relative to north of meteoroid trajectory (degrees)
- **pressures** (*float, arraylike*) – List of threshold pressures to define airblast damage levels

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

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])
```

armageddon.**degrees**(*x*, */*)
Convert angle x from radians to degrees.

armageddon.**fsolve**(*func*, *x0*, *args=()*, *fprime=None*, *full_output=0*, *col_deriv=0*, *xtol=1.49012e-08*, *maxfev=0*, *band=None*, *epsfcn=None*, *factor=100*, *diag=None*)
Find the roots of a function.

Return the roots of the (non-linear) equations defined by `func(x) = 0` given a starting estimate.

Parameters

- **func** (callable `f(x, *args)`) – A function that takes at least one (possibly vector) argument, and returns a value of the same length.
- **x0** (*ndarray*) – The starting estimate for the roots of `func(x) = 0`.
- **args** (*tuple, optional*) – Any extra arguments to *func*.
- **fprime** (callable `f(x, *args)`, optional) – A function to compute the Jacobian of *func* with derivatives across the rows. By default, the Jacobian will be estimated.

- **full_output** (*bool, optional*) – If True, return optional outputs.
- **col_deriv** (*bool, optional*) – Specify whether the Jacobian function computes derivatives down the columns (faster, because there is no transpose operation).
- **xtol** (*float, optional*) – The calculation will terminate if the relative error between two consecutive iterates is at most *xtol*.
- **maxfev** (*int, optional*) – The maximum number of calls to the function. If zero, then `100*(N+1)` is the maximum where N is the number of elements in *x0*.
- **band** (*tuple, optional*) – If set to a two-sequence containing the number of sub- and super-diagonals within the band of the Jacobi matrix, the Jacobi matrix is considered banded (only for `fprime=None`).
- **epsfcn** (*float, optional*) – A suitable step length for the forward-difference approximation of the Jacobian (for `fprime=None`). If *epsfcn* is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.
- **factor** (*float, optional*) – A parameter determining the initial step bound (`factor * || diag * x||`). Should be in the interval `(0.1, 100)`.
- **diag** (*sequence, optional*) – N positive entries that serve as a scale factors for the variables.

Returns

- **x** (*ndarray*) – The solution (or the result of the last iteration for an unsuccessful call).
- **infodict** (*dict*) – A dictionary of optional outputs with the keys:

  nfev number of function calls

  njev number of Jacobian calls

  fvec function evaluated at the output

  fjac the orthogonal matrix, q, produced by the QR factorization of the final approximate Jacobian matrix, stored column wise

  r upper triangular matrix produced by QR factorization of the same matrix

  qtf the vector `(transpose(q) * fvec)`

- **ier** (*int*) – An integer flag. Set to 1 if a solution was found, otherwise refer to *mesg* for more information.
- **mesg** (*str*) – If no solution is found, *mesg* details the cause of failure.

See also:

root Interface to root finding algorithms for multivariate functions. See the `method=='hybr'` in particular.

Notes

`fsolve` is a wrapper around MINPACK's hybrd and hybrj algorithms.

Examples

Find a solution to the system of equations: `x0*cos(x1) = 4, x1*x0 - x1 = 5`.

```
>>> from scipy.optimize import fsolve
>>> def func(x):
...     return [x[0] * np.cos(x[1]) - 4,
...             x[1] * x[0] - x[1] - 5]
>>> root = fsolve(func, [1, 1])
>>> root
array([6.50409711, 0.90841421])
>>> np.isclose(func(root), [0.0, 0.0])  # func(root) should be almost 0.0.
array([ True,  True])
```

armageddon.**get_lat_long_of_postcodes**(*postcodes*, *sector=False*)

Return location(latitude,longitude) of a list of postcode units or sectors.

Parameters

- **postcodes** (*list of lists*) – list of postcode units or postcode sectors
- **sector** (*bool, optional*) – if true return populations for postcode sectors, otherwise postcode units

Returns

- *list of lists*
- *Contains the latitude,longitude of input postcode units or sectors*

Examples

```
>>> get_lat_log_of_postcode([['SW7 2AZ','SW7 2BT','SW7 2BU','SW7 2DD']])
>>> get_lat_log_of_postcode([['SW7 2']], True)
```

armageddon.**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).

Parameters

- **latlon1** (*arraylike*) – latitudes and longitudes of first point (as [n, 2] array for n points)
- **latlon2** (*arraylike*) – latitudes and longitudes of second point (as [m, 2] array for m points)

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]
```

armageddon.**halfway_on_sphere**(*lat*, *lon*, *elat*, *elon*, *z*)

Calculate a point on the great circle rout of asteroid. If z= 0.5, the return shows lat & lon of harfway point.

lat: float latitude of zero point

lon: float longitude of zero point

elat: float latitude of entry point

elon: float longitude of entry point

z: float calculation point between entry and zero point.

Returns latitude and longitude of interval point.

Return type list

armageddon.**heat_map_layer**(*locations*, *weights*, *map=None*, *radius=25*)

Return heat map layer for follium map from a list of locations and a list of weights

Parameters

- **locations** (*list of lists*) – list of latitutde and longitude coordinates corresponding to postcode units or postcode sectors
- **weights** (*list of lists, array-like*) – list of weights to be plotted at locations

Returns

Return type Follium map

Examples

```
>>> locations = get_lat_long_of_postcodes(postcodes, sector=False)
>>> weights = [['10000', '20000', '30000', '40000']]
>>> heat_map_layer(locations, weights, map = None, radius = 25)
```

armageddon.**impact_risk**(*planet*, *means={'angle': 20, 'bearing': - 45.0, 'density': 3000, 'lat': 51.5, 'lon': 1.5, 'radius': 10, 'strength': 1000000.0, 'velocity': 19000.0}*, *stdevs={'angle': 1, 'bearing': 0.5, 'density': 500, 'lat': 0.025, 'lon': 0.025, 'radius': 1, 'strength': 500000.0, 'velocity': 1000.0}*, *pressure=27000.0*, *nsamples=100*, *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*) – The 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

armageddon.**latlon_to_xyz**(*lat*, *lon*)

Change lattitude and longitude into the rectangular coordinate system. The equatorial plane is the xy plane, and the axis of rotation is the z axis.

Parameters

- **lat** (*float*) – latitude(degree)
- **lon** (*float*) – longitude(degree)
- **rlat** (*float*) – latitude(rad)
- **rlon** (*float*) – longitude(rad)

Returns Points on the rectangular coordinate system.

Return type float

armageddon.**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** (*arraylike, float*) – List of distances specifying the radius of circle(s) to plot (m)
- **map** (*folium.Map*) – existing map object

Returns

Return type Folium map object

Examples

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

armageddon.**plot_line**(*lat*, *lon*, *elat*, *elon*, *map=None*, *n=100*)

Plot a black lineconnecting entry point and zero point. :param lat: latitude of circle to plot (degrees) :type lat: float :param lon: longitude of circle to plot (degrees) :type lon: float :param elat: latitude of entry point (degrees) :type elat: float :param elon: longitude of entry point(degrees) :type elon: float :param n: number of rute divisions.Default value is 100. :type n: int :param map: existing map object :type map: folium.Map

Returns

Return type Folium map object

Examples

```
>>> plot_line(52.79, -2.95, 53.48, -2.24 , map=None)
```

armageddon.**plot_marker**(*lat*, *lon*, *popup=None*, *map=None*, ***kwargs*)
Plot a point on a map (creating a new folium map instance if necessary).

Parameters

- **lat** (*float*) – latitude of point to plot (degrees)
- **lon** (*float*) – longitude of point to plot (degrees)
- **popup** (*str*) – will plot a string label at point
- **map** (*folium.Map*) – existing map object

Returns

Return type Folium map object

Examples

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

armageddon.**plot_multiple_markers**(*locations*, *popups=None*, *map=None*)
Return heat cluster of markers for follium map from a list of locations

Parameters

- **locations** (*list of lists*) – list of latitutde and longitude coordinates corresponding to postcode units or postcode sectors
- **popup** (*list of str*) – will plot a string label at points
- **map** (*folium.Map*) – existing map object

Returns

Return type Follium map

Examples

```
>>> locations = get_lat_long_of_postcodes(postcodes, sector=False)
>>> plot_multiple_markers(locations, popups= None, map = None)
```

armageddon.**radians**(*x*, */*)
Convert angle x from degrees to radians.

armageddon.**sin**(*x*, */*)
Return the sine of x (measured in radians).

armageddon.**xyz_to_latlon**(*x*, *y*, *z*)
Change coodinate from xyz coordinate system to latitude & longitude coordinates.

x: float x coordinate of Equatorial plane

y: float y coodinate of Equatorial plane

z: float z coodinate of Arctic direction

Returns Points on the earth surface(degree)

Return type float

PYTHON MODULE INDEX