TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

PROJECTILE SIMULATOR SYSTEM

Missiles are self-guided munitions that travel through the air or outer space to their targets. A ballistic missile travels along a suborbital trajectory. An intercontinental ballistic missile can travel a substantial distance around the Earth to its target.

Now, there has been a drastic increase in the computational capacity of the machines which has led to the advancement in Defense and Military aspects for very complex calculations which has led to the development of the above system.

Anti-Ballistic Missile System is a process of intercepting the warheads in mid-air accurately, or to prevent it from reaching the target. This system basically consists of five main phases-

- Detection of incoming Missile
- Deciding on whether the Missile hits a populated zone or not.
- Estimating the position of missile at regular intervals
- Initiating the intercepting warhead missile launch
- Determines the time, velocity and angle of launch

The above phases/system is implemented using Python language either as a GUI or console based project [displays the trajectory and its parameters of both Missile and the intercepting missile (Anti-missile)]. From this project we like to evaluate the complicated expressions, so as to reduce the tedious work of solving them [time consuming] and accurately [in ideal case] determine the height, range of the impact with calculated velocity and angle of launch. This project implements some modules of the entire project.

1) INTRODUCTION

Projectile Simulator system is an application which is developed to make the tedious computations easier and faster. It is useful for simulating projectile scenarios and also device a system where we can counter the incoming projectiles/missiles and visualize the trajectory in an efficient manner.

1.1 MOTIVATION

Managing the real-time systems is a hectic task but if planned with skill and intelligence and with the help of technology we can try and build efficient systems. This Projectile simulator is one of that kind, this system provides to visualize the trajectory of projectile and can simulate such projectiles for testing purposes and also this system is used to counter attack the incoming ICBMS.

2) PROBLEM STATEMENT

This is a simulation system where we can create and analyse scenarios of various aspects of the trajectory of a missile, estimate the break point or the bend of the arms based on the projectile principles. The purpose of this is to visualize our calculations with random inputs which would lead to our suitable and possible desired scenario. This also provides details or is search engine for various arms/Launchers.

3. Python

The above project has been implemented in python so, before getting into the project let me tell you why I have used python as my implementation language.

Python is frequently used for high-performance scientific applications. It is widely used in academia and scientific projects because it is easy to write and performs well.

Due to its high-performance nature, scientific computing in Python often utilizes external libraries, typically written in faster languages (like C, or FORTRAN for matrix operations). The main libraries used are NumPy, SciPy and Matplotlib. Going into detail about these libraries is beyond the scope of the Python guide. However, a comprehensive introduction to the scientific Python ecosystem can be found in the Python Scientific Lecture Notes.

Tools

IPython

IPython is an enhanced version of Python interpreter, which provides features of great interest to scientists. The *inline mode* allows graphics and plots to be displayed in the terminal (Qt based version). Moreover, the *notebook* mode supports literate programming and reproducible science generating a web-based Python notebook. This notebook allows you to store chunks of Python code along side the results and additional comments (HTML, LaTeX, Markdown). The notebook can then be shared and exported in various file formats.

Libraries

NumPy

NumPy is a low level library written in C (and FORTRAN) for high level mathematical functions. NumPy cleverly overcomes the problem of running slower algorithms on Python by using multidimensional arrays and functions that operate on arrays. Any algorithm can then be expressed as a function on arrays, allowing the algorithms to be run quickly.

NumPy is part of the SciPy project, and is released as a separate library so people who only need the basic requirements can use it without installing the rest of SciPy. NumPy is compatible with Python versions 2.4 through to 2.7.2 and 3.1+.

Numba

Numba is a NumPy aware Python compiler (just-in-time (JIT) specializing compiler) which compiles annotated Python (and NumPy) code to LLVM (Low Level Virtual Machine) through special decorators. Briefly, Numba uses a system that compiles Python code with LLVM to code which can be natively executed at runtime.

SciPy

SciPy is a library that uses NumPy for more mathematical functions. SciPy uses NumPy arrays as the basic data structure, and comes with modules for various commonly used tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving and signal processing.

Matplotlib

Matplotlib is a flexible plotting library for creating interactive 2D and 3D plots that can also be saved as manuscript-quality figures. The API in many ways reflects that of MATLAB, easing transition of MATLAB users to Python. Many examples, along with the source code to re-create them, are available in the matplotlib gallery.

Pandas

Pandas is data manipulation library based on Numpy which provides many useful functions for accessing, indexing, merging and grouping data easily. The main data structure (DataFrame) is close to what could be found in the R statistical package; that is, heterogeneous data tables with name indexing, time series operations and auto-alignment of data.

Rpy2

Rpy2 is a Python binding for the R statistical package allowing the execution of R functions from Python and passing data back and forth between the two environments. Rpy2 is the object oriented implementation of the Rpy bindings.

PsychoPy

PsychoPy is a library for cognitive scientists allowing the creation of cognitive psychology and neuroscience experiments. The library handles presentation of stimuli, scripting of experimental design and data collection.

Resources

Installation of scientific Python packages can be troublesome, as many of these packages are implemented as Python C extensions which need to be compiled. This section lists various so-called scientific Python distributions which provide precompiled and easy-to-install collections of scientific Python packages.

Unofficial Windows Binaries for Python Extension Packages

Many people who do scientific computing are on Windows, yet many of the scientific computing packages are notoriously difficult to build and install on this platform. Christoph Gohlke however, has compiled a list of Windows binaries for many useful Python packages. The list of packages has grown from a mainly scientific Python resource to a more general list.

Anaconda

Continuum Analytics offers the Anaconda Python Distribution which includes all the common scientific Python packages as well as many packages related to data analytics and big data. Anaconda itself is free, and Continuum sells a number of proprietary add-ons. Free licenses for the add-ons are available for academics and researchers.

Canopy

Canopy is another scientific Python distribution, produced by Enthought. A limited 'Canopy Express' variant is available for free, but Enthought charges for the full distribution. Free licenses are available for academics.

Since there are a wide resources and libraries provided by python, I have opted for python.But there is an another IDE called Jupyter Notebook that supports all the above libraries.

4.JUPYTER NOTEBOOK

The above ***IDE has been used for implementation*** as it is the most comfortable for implementation and the background running module is the anaconda.

The Jupyter Notebook Introduction-

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

Python module used for this project is **gmaps**.

5. Gmaps

gmaps is a plugin for Jupyter for embedding Google Maps in your notebooks. It is designed as a data visualization tool.

To demonstrate gmaps, let's plot the earthquake dataset, included in the package:

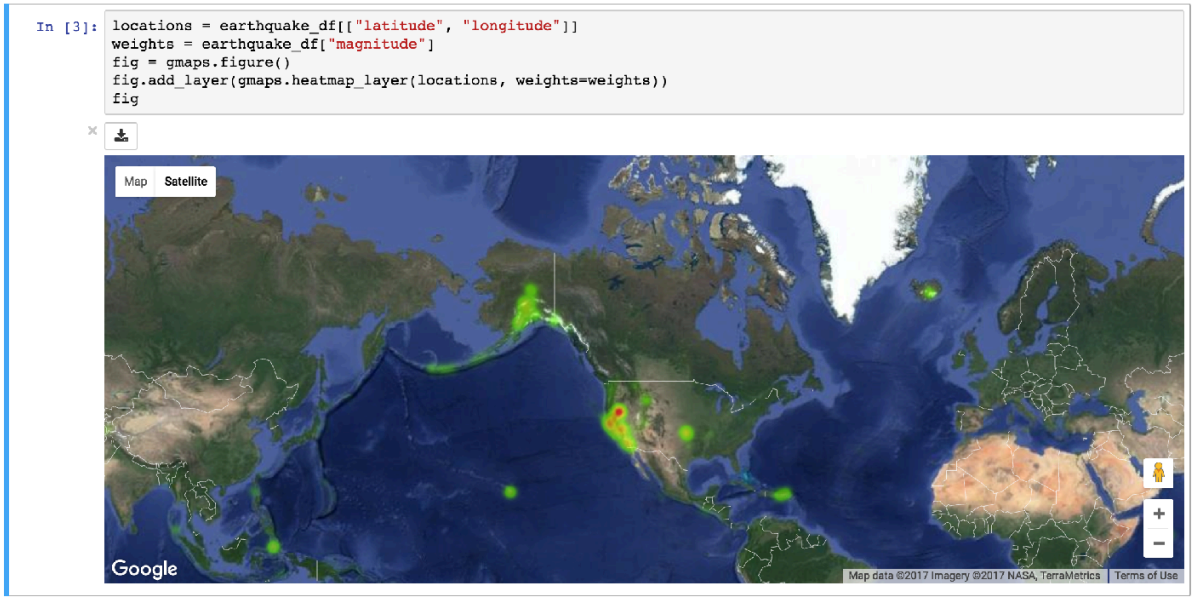
```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...") # Fill in with your API key
earthquake_df = gmaps.datasets.load_dataset_as_df("earthquakes")
earthquake_df.head()
```

The earthquake data has three columns: a latitude and longitude indicating the earthquake's epicentre and a weight

denoting the magnitude of the earthquake at that point. Let's plot the earthquakes on a

Google map:

```
locations = earthquake_df[["latitude", "longitude"]]
weights = earthquake_df["magnitude"]
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(locations, weights=weights))
fig
```



This gives you a fully-fledged Google map. You can zoom in and out, switch to satellite view and even to street view

if you really want. The heatmap adjusts as you zoom in and out.

Basic concepts

gmaps is built around the idea of adding layers to a base map. After you've authenticated with Google maps, you start

by creating a figure, which contains a base map:

```
import gmaps
gmaps.configure(api_key="AI...")
```

```
fig = gmaps.figure()
fig
```

```
In [5]: import gmaps
        gmaps.configure(api_key="AIz...")

In [6]: fig = gmaps.figure()
        fig
```

You then add layers on top of the base map. For instance, to add a heatmap layer:

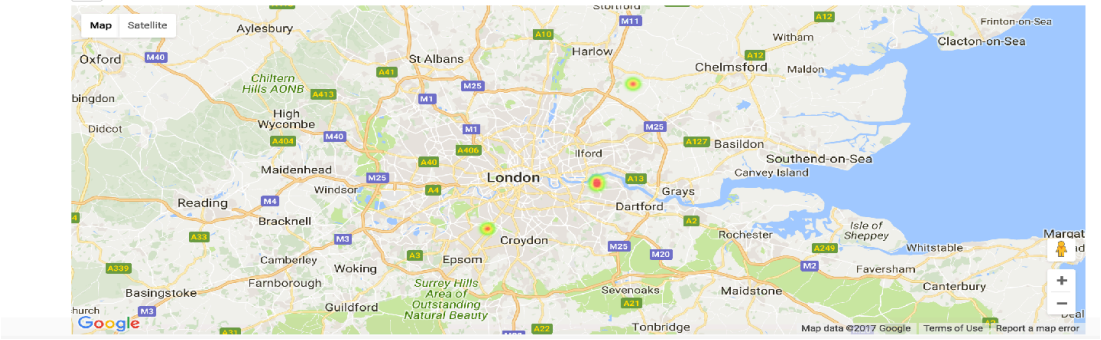
```
import gmaps
gmaps.configure(api_key="AI...")
fig = gmaps.figure()
# generate some (latitude, longitude) pairs
locations = [(51.5, 0.1), (51.7, 0.2), (51.4, -0.2), (51.49, 0.1)]
heatmap_layer = gmaps.heatmap_layer(locations)
fig.add_layer(heatmap_layer)
fig
```

```
In [5]: import gmaps
        gmaps.configure(api_key="AIz...")

In [7]: fig = gmaps.figure()

        # generate some (latitude, longitude) pairs
        locations = [(51.5, 0.1), (51.7, 0.2), (51.4, -0.2), (51.49, 0.1)]

        heatmap_layer = gmaps.heatmap_layer(locations)
        fig.add_layer(heatmap_layer)
        fig
```



The locations array can either be a list of tuples, as in the example above, a numpy array of shape $N times 2$ or a

dataframe with two columns.

Most attributes on the base map and the layers can be set through named arguments in the constructor or as instance

attributes once the instance is created. These two constructions are thus equivalent:

heatmap_layer = gmaps.heatmap_layer(locations)

heatmap_layer.point_radius = 8

and:

heatmap_layer = gmaps.heatmap_layer(locations, point_radius=8)

The former construction is useful for modifying a map once it has been built. Any change in parameters will propagate

to maps in which those layers are included.

Base maps

Your first action with gmaps will usually be to build a base map:

```
import gmaps
gmaps.configure(api_key="AI...")
gmaps.figure()
```

This builds an empty map. You can also set the zoom level and map center explicitly:

```
new_york_coordinates = (40.75, -74.00)
gmaps.figure(center=new_york_coordinates, zoom_level=12)
```

```
In [19]: new_york_coordinates = (40.75, -74.00)
         gmaps.figure(center=new_york_coordinates, zoom_level=12)
```

If you do not set the map zoom and center, the viewport will automatically focus on the data as you add it to the map.

Heatmaps

Heatmaps are a good way of getting a sense of the density and clusters of geographical events. They are a powerful

tool for making sense of larger datasets. We will use a dataset recording all instances of political violence that occurred

in Africa between 1997 and 2015. The dataset comes from the Armed Conflict Location and Event Data Project. This

dataset contains about 110,000 rows.

```
import gmaps.datasets
locations = gmaps.datasets.load_dataset_as_df("acled_africa")
locations.head()
# => dataframe with 'longitude' and 'latitude' columns
```

We already know how to build a heatmap layer:

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...")
```

```
locations = gmaps.datasets.load_dataset_as_df("acled_africa")
fig = gmaps.figure()
heatmap_layer = gmaps.heatmap_layer(locations)
fig.add_layer(heatmap_layer)
fig
```

Setting the color gradient and opacity

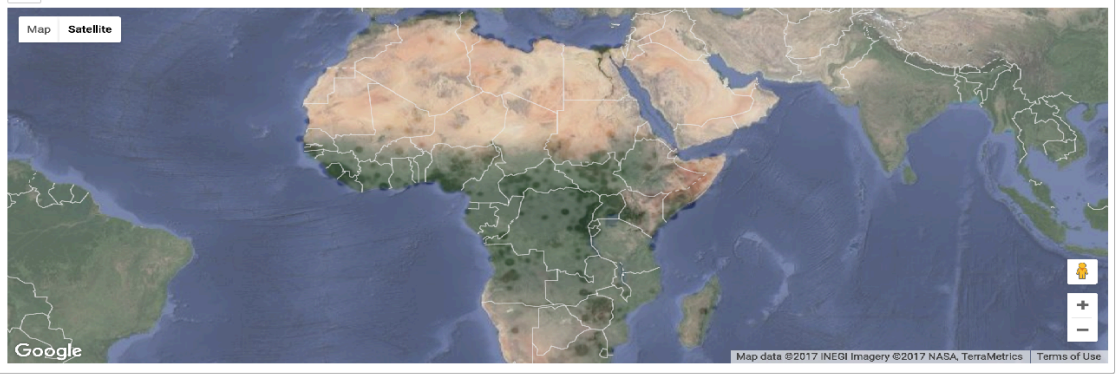
You can set the color gradient of the map by passing in a list of colors. Google maps will interpolate linearly between
those colors. You can represent a color as a string denoting the color (the colors allowed by this):

```
heatmap_layer.gradient = [
'white',
'silver',
'gray'
]
```

If you need more flexibility, you can represent colours as an RGB triple or an RGBA quadruple:

```
heatmap_layer.gradient = [
(200, 200, 200, 0.6),
(100, 100, 100, 0.3),
(50, 50, 50, 0.3)
]
```

```
In [20]: locations = gmaps.datasets.load_dataset_as_df("acled_africa")
         fig = gmaps.figure()
         heatmap_layer = gmaps.heatmap_layer(locations)
         fig.add_layer(heatmap_layer)
         fig
```




```
In [21]: heatmap_layer.max_intensity = 100
         heatmap_layer.point_radius = 5
         heatmap_layer.gradient = [
             (200, 200, 200, 0.6),
             (100, 100, 100, 0.3),
             (50, 50, 50, 0.3)
         ]
```

You can also use the opacity option to set a single opacity across the entire colour gradient:

heatmap_layer.opacity = 0.0 # make the heatmap transparent

Weighted heatmaps

By default, heatmaps assume that every row is of equal importance. You can override this by passing weights through

the weights keyword argument. The weights array is an iterable (e.g. a Python list or a Numpy array) or a single

pandas series. Weights must all be positive (this is a limitation in Google maps itself).

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...")
df = gmaps.datasets.load_dataset_as_df("earthquakes")
# dataframe with columns ('latitude', 'longitude', 'magnitude')
fig = gmaps.figure()
heatmap_layer = gmaps.heatmap_layer(
df[["latitude", "longitude"]], weights=df["magnitude"],
max_intensity=30, point_radius=3.0
)
fig.add_layer(heatmap_layer)
fig
```

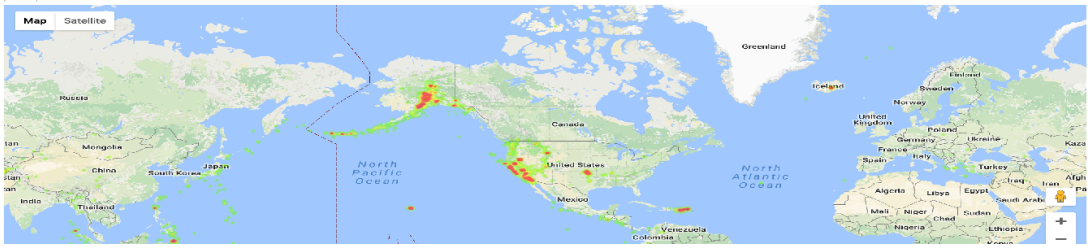
```
In [24]: import gmaps
         import gmaps.datasets
         gmaps.configure(api_key="AIz...")

In [25]: df = gmaps.datasets.load_dataset_as_df("earthquakes")
         df.head()
```

Out[25]:

	latitude	longitude	magnitude
0	65.193300	149.072500	1.70
1	38.791832	-122.780830	2.10
2	38.818001	-122.792168	0.48
3	33.601667	-116.727667	0.78
4	37.378334	-118.520836	3.64

```
In [26]: fig = gmaps.figure()
         heatmap_layer = gmaps.heatmap_layer(
             df[["latitude", "longitude"]], weights=df["magnitude"],
             max_intensity=30, point_radius=3.0
         )
         fig.add_layer(heatmap_layer)
         fig
```



Markers and symbols

We can add a layer of markers to a Google map. Each marker represents an individual data point:

```
import gmaps
gmaps.configure(api_key="AI...")
marker_locations = [
(-34.0, -59.166672),
(-32.23333, -64.433327),
(40.166672, 44.133331),
(51.216671, 5.0833302),
(51.333328, 4.25)]
fig = gmaps.figure()
markers = gmaps.marker_layer(marker_locations)
fig.add_layer(markers)
fig
```

```
In [27]: marker_locations = [
             (-34.0, -59.166672),
             (-32.23333, -64.433327),
             (40.166672, 44.133331),
             (51.216671, 5.0833302),
             (51.333328, 4.25)
         ]

         fig = gmaps.figure()
         markers = gmaps.marker_layer(marker_locations)
         fig.add_layer(markers)
         fig
```

We can also attach a pop-up box to each marker. Clicking on the marker will bring up the info box. The content of the

box can be either plain text or html:

```
import gmaps
gmaps.configure(api_key="AI...")
nuclear_power_plants = [
{"name": "Atucha", "location": (-34.0, -59.167), "active_reactors": 1},
{"name": "Embalse", "location": (-32.2333, -64.4333), "active_reactors": 1},
{"name": "Armenia", "location": (40.167, 44.133), "active_reactors": 1},
{"name": "Br", "location": (51.217, 5.083), "active_reactors": 1},
{"name": "Doel", "location": (51.333, 4.25), "active_reactors": 4},
{"name": "Tihange", "location": (50.517, 5.283), "active_reactors": 3}
]
plant_locations = [plant["location"] for plant in nuclear_power_plants]
info_box_template = """
<dl>
<dt>Name</dt><dd>{name}</dd>
<dt>Number reactors</dt><dd>{active_reactors}</dd>
</dl>
"""
plant_info = [info_box_template.format(**plant) for plant in nuclear_power_plants]
marker_layer = gmaps.marker_layer(plant_locations, info_box_content=plant_info)
fig = gmaps.figure()
fig.add_layer(marker_layer)
fig
```

```
In [29]: nuclear_power_plants = [
             {"name": "Atucha", "location": (-34.0, -59.167), "active_reactors": 1},
             {"name": "Embalse", "location": (-32.2333, -64.4333), "active_reactors": 1},
             {"name": "Armenia", "location": (40.167, 44.133), "active_reactors": 1},
             {"name": "Br", "location": (51.217, 5.083), "active_reactors": 1},
             {"name": "Doel", "location": (51.333, 4.25), "active_reactors": 4},
             {"name": "Tihange", "location": (50.517, 5.283), "active_reactors": 3}
         ]

         plant_locations = [plant["location"] for plant in nuclear_power_plants]
         info_box_template = """
         <dl>
         <dt>Name</dt><dd>{name}</dd>
         <dt>Number reactors</dt><dd>{active_reactors}</dd>
         </dl>
         """
         plant_info = [info_box_template.format(**plant) for plant in nuclear_power_plants]

         marker_layer = gmaps.marker_layer(plant_locations, info_box_content=plant_info)
         fig = gmaps.figure()
         fig.add_layer(marker_layer)
         fig
```

Markers are currently limited to the Google maps style drop icon. If you need to draw more complex shape on maps,

use the symbol_layer function. Symbols represent each latitude, longitude pair with a circle whose colour and

size you can customize. Let's, for instance, plot the location of every Starbuck's coffee shop in the UK:

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...")
df = gmaps.datasets.load_dataset_as_df("starbucks_kfc_uk")
starbucks_df = df[df["chain_name"] == "starbucks"]
starbucks_df = starbucks_df[['latitude', 'longitude']]
starbucks_layer = gmaps.symbol_layer(
starbucks_df, fill_color="green", stroke_color="green", scale=2
)
fig = gmaps.figure()
fig.add_layer(starbucks_layer)
fig
```

Directions layer

gmaps supports drawing routes based on the Google maps directions service. At the moment, this only supports

directions between points denoted by latitude and longitude:

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AIza...")
# Latitude-longitude pairs
geneva = (46.2, 6.1)
montreux = (46.4, 6.9)
zurich = (47.4, 8.5)
fig = gmaps.figure()
geneva2zurich = gmaps.directions_layer(geneva, zurich)
fig.add_layer(geneva2zurich)
fig
```

```
In [63]: geneva = (46.2, 6.1)
         montreux = (46.4, 6.9)
         zurich = (47.4, 8.5)

         fig = gmaps.figure()
         geneva2zurich = gmaps.directions_layer(geneva, zurich)
         fig.add_layer(geneva2zurich)
         fig
```

You can also pass waypoints and customise the directions request. You can pass up to 23 waypoints, and waypoints

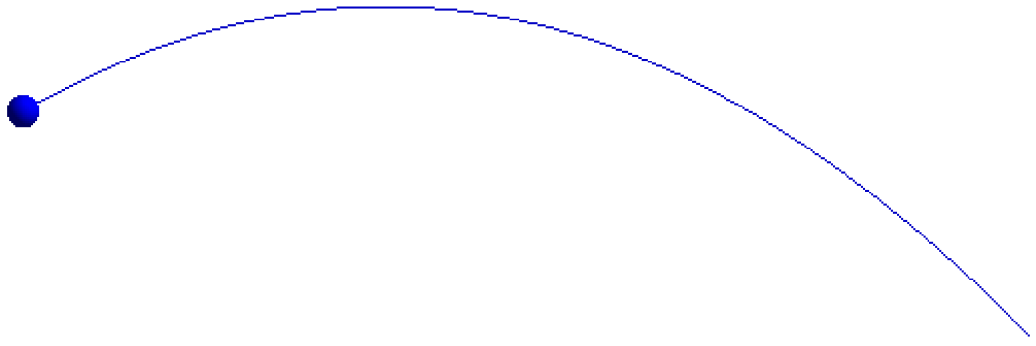
are not supported when the travel mode is 'TRANSIT' (this is a limitation of the Google Maps directions service):

```
fig = gmaps.figure()
geneva2zurich_via_montreux = gmaps.directions_layer(
geneva, zurich, waypoints=[montreux],
travel_mode='BICYCLING')
fig.add_layer(geneva2zurich_via_montreux)
fig
```

6.PROJECTILE MOTION

Now we have seen the tools that were used for implementation. But did we use them for?

Let's see now theory behind the project.



Projectile motion is a special case of two-dimensional motion. A particle moving in a vertical plane with an initial velocity and experiencing a free-fall (downward) acceleration, displays projectile motion. Some examples of projectile motion are the motion of a ball after being hit/thrown, the motion of a bullet after being fired and the motion of a person jumping off a

diving board. For now, we will assume that the air, or any other fluid through which the object is
moving, does not have any effect on the motion. In reality, depending on the object, air can play a very significant role. For example, by taking advantage of air resistance, a parachute can allow a person to land safely after jumping off an airplane. These effects are very briefly discussed at the end of this module.

Projectile Motion Analysis

Before proceeding, the following subsection provides a reminder of the three main equations of motion for constant acceleration. These equations are used to develop the equations for projectile motion.

Equations of Motion for Constant Acceleration

The following equations are three commonly used equations of motion for an object moving with a constant acceleration.

$$v = v_0 + a \cdot t$$

... Eq. (1)

$$x - x_0 = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

... Eq. (2)

$$v^2 = v_0^2 + 2 \cdot a \cdot (x - x_0)$$

... Eq. (3)

Here, a is the acceleration, v is the speed, v_0 is the initial speed, x is the position, x_0 is the initial position and t is the time.

To begin, consider an object with an initial velocity $\vec{v_0} = v_{0x}\,\hat{i} + v_{0y}\,\hat{j}$ launched at an angle of θ_0 measured from the positive x-direction. The x- and y-components of the object's initial velocity, v_{0x} and v_{0y}, can be written as

$$v_{0x} = v_0 \cdot \cos(\theta_0) \quad \text{and} \quad v_{0y} = v_0 \cdot \sin(\theta_0)$$

... Eq. (4) and Eq. (5)

Here, the x-axis corresponds to the horizontal direction and the y-axis corresponds to the vertical direction. Since there is a downward acceleration due to gravity, the y-component of the object's velocity is continuously changing. However, since there is no horizontal acceleration, the x-component of the object's velocity stays constant throughout the motion. Breaking up the motion into x- and y-components simplifies the analysis.

The Horizontal Component

Horizontal Displacement

Using Eq. (2), at a time , the horizontal displacement for a projectile can be be written As

$$x - x_0 = v_{0x} \cdot t + \frac{1}{2} \cdot a_x \cdot t^2$$

... Eq. (6)

where, x_0 is the initial position and a_x is the horizontal acceleration. Since $a_x = 0$ and $v_{0x} = v_0 \cdot \cos(\theta_0)$, the above equation reduces to

$$x - x_0 = v_0 \cdot \cos(\theta_0) \cdot t$$

... Eq. (7)

Eq. (7) is the equation for the horizontal displacement of a projectile as a function of time. This shows that, before the projectile hits the ground or encounters any other resistance, the horizontal displacement changes linearly with time.

Horizontal Velocity

Using Eq. (1), the horizontal velocity can be written as

$$v_x = v_{0x} + a_x \cdot t$$

which further reduces to

$$v_x = v_0 \cdot \cos(\theta_0)$$

... Eq. (9)

Since v_0 and θ_0 are constants, this equation shows that the horizontal velocity remains unchanged throughout the motion.

The Vertical Component

Vertical Displacement

Using Eq. (2), at a time t, the vertical displacement for the projectile can be written as

$$y - y_0 = v_{0y} \cdot t + \frac{1}{2} \cdot a_y \cdot t^2$$

... Eq. (10)

where, y_0 is the initial position and a_y is the vertical acceleration. Since $a_y = -g$ and $v_{0y} = v_0 \cdot \sin(\theta_0)$, the above equation reduces to

$$y - y_0 = v_0 \cdot \sin(\theta_0) \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

... Eq. (11)

Eq. (11) is the equation for the vertical displacement of a projectile as a function of time. Unlike the horizontal displacement, the vertical displacement does not vary linearly with time.

Vertical Velocity

Using Eq. (1), the vertical velocity can be written as

$$v_y = v_{0y} + a_y \cdot t$$

... Eq. (12)

Since $a_y = -g$ and $v_{0y} = v_0 \cdot \sin(\theta_0)$, this can be further written as

$$v_y = v_0 \cdot \sin(\theta_0) - g \cdot t$$

... Eq. (13)

This equation shows that the vertical component of the velocity continuously changes with time.

Also, using Eq. (3), the vertical velocity can be expressed as

$$v_y^2 = {v_{0y}}^2 + 2 \cdot a_y \cdot (y - y_0)$$

... Eq. (14)

which can be further written as

$$v_y^2 = (v_0 \cdot \sin(\theta_0))^2 - 2 \cdot g \cdot (y - y_0)$$

... Eq. (15)

This equation provides a relation between the vertical velocity and the vertical displacement.

Maximum Vertical Displacement

When an object is launched with a positive vertical velocity component, the vertical velocity becomes zero at the point where the maximum height is reached. Hence, substituting $v_y = 0$ in Eq. (15) gives the following equation for the maximum vertical displacement of a projectile.

$$(y - y_0) = \frac{\left(v_0 \cdot \sin\left(\theta_0\right)\right)^2}{2 \cdot g}$$

... Eq. (16)

Also, from Eq. (16) observe that, to achieve the maximum possible vertical displacement for a fixed initial velocity, $\left(\sin\left(\theta_0\right)\right)^2$ should be equal to 1. This happens when θ_0 is 90°(-90° is also valid mathematically but not physically. Mathematically, with a launch angle of -90°, the velocity is zero at a negative time). Therefore, to achieve the maximum height, an object must be launched straight up, which is also what intuition would tell us.

The Path Equation

By combining Eq. (7) and Eq. (11), an equation for the projectile's trajectory can be obtained. Rearranging Eq. (7) to isolate for t and substituting it into Eq. (11) gives

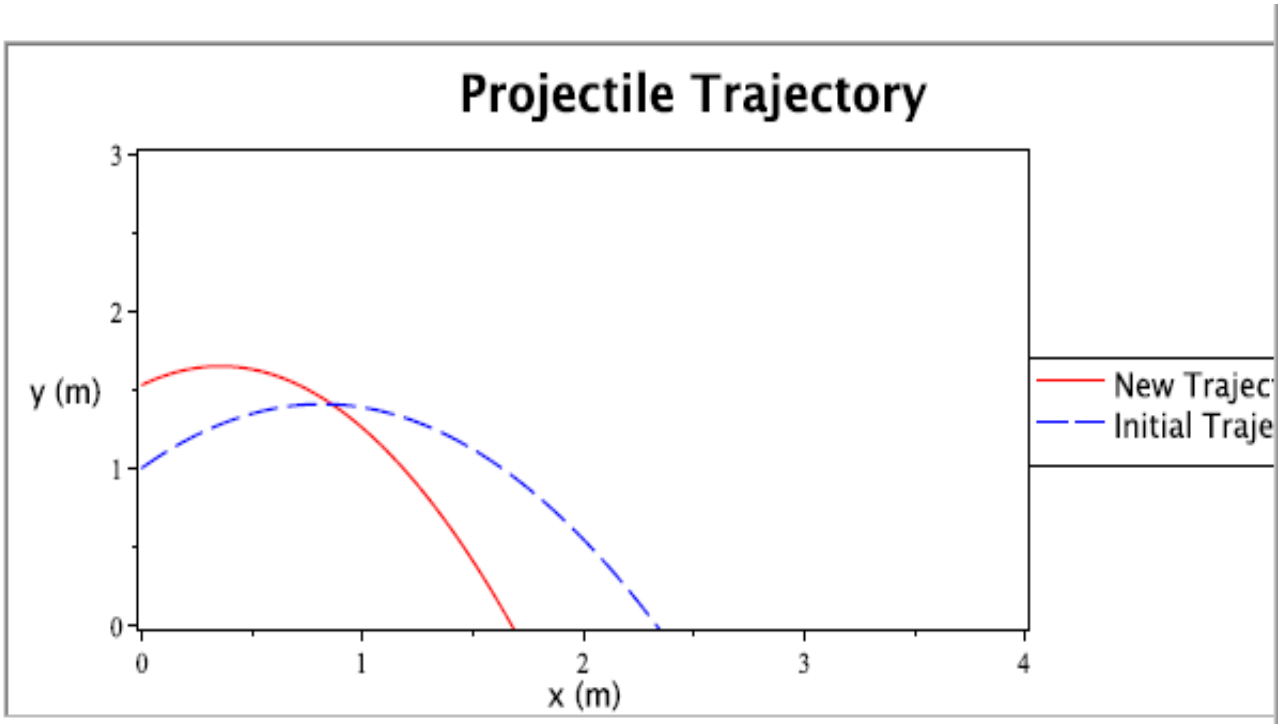
$$y = y_0 + \left(x - x_0\right) \cdot \tan\theta_0 - \frac{g \cdot \left(x - x_0\right)^2}{2\left(v_0 \cdot \cos\left(\theta_0\right)\right)^2}$$

... Eq. (17)

This is the equation for the path of a projectile. If the initial conditions are known, then the path of the projectile can be determined. Due to the quadratic form of the equation, each y-position has two corresponding x-positions. These different x-positions correspond to different time values. For example, when a ball is thrown, it can cross a certain height

twice, once on the way up and once on the way down. It should also be noted that, due to the tan and cos terms, this path equation is not defined at 90° and -90°. This can also be understood by the fact that there will be many y-positions for the same initial x-position.

In the following plot, the dashed line shows the trajectory of a projectile launched at an initial height of 1m, with an initial velocity of 4m/s and at an angle of 45° from the horizontal. The sliders can be used to adjust the initial conditions and observe the new trajectory (solid line).



Horizontal Range

The horizontal range can be defined as the horizontal distance a projectile travels before returning to its launch height. When the final height is equal to the launch height (i.e. $y - y_0 = 0$), Eq. (11) reduces to the following,

$$0 = v_0 \cdot \sin(\theta_0) \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

... Eq. (18)

When t is non-zero, this equation can be written as

$$t = \frac{2 \cdot v_0 \cdot \sin(\theta_0)}{g}$$

... Eq. (19)

By substituting Eq. (18) into Eq. (7), the following equation is obtained

$$x - x_0 = \frac{2 \cdot v_0^2}{g} \cdot \sin(\theta_0) \cdot \cos(\theta_0)$$

... Eq. (20)

Using the identity $\sin(2 \cdot \theta_0) = 2 \cdot \sin(\theta_0) \cdot \cos(\theta_0)$ gives

$$x - x_0 = \frac{2 \cdot v_0^2}{g} \cdot \sin(2 \cdot \theta_0)$$

.... Eq. (21)

This shows that the horizontal displacement, when the projectile returns to the launch height, is greatest when $\sin(2 \cdot \theta_0) = 1$. This means that the range is maximum when the launch angle is 45°. The following plot shows the trajectory of a projectile launched with an initial velocity of 10 m/s, at an angle of 45° and with no initial height (dashed line). The launch angle can be varied to observe the effect on the range.

6.1. HAVERSINES FORMULA

The Earth is round but big, so we can consider it flat for short distances. But, even though the circumference of the Earth is about 40,000 kilometers, flat-Earth formulas for calculating the distance between two points start showing noticeable errors when the distance is more than about 20 kilometers. Therefore, calculating distances on a sphere needs to consider spherical geometry, the study of shapes on the surface of a sphere.

Spherical geometry considers spherical trigonometry which deals with relationships between trigonometric functions to calculate the sides and angles of spherical polygons. These spherical polygons are defined by a number of intersecting great circles on a sphere. Some rules found in spherical geometry include:

- There are no parallel lines.
- Straight lines are great circles, so any two lines meet in two points.
- The angle between two lines is the angle between the planes of the corresponding great circles.

The Haversine

Did you know that there are more than the 3 trigonometric functions we are all familiar with sine, cosine and, tangent? These additional trigonometric functions are now obsolete, however, in the past, they were worth naming.

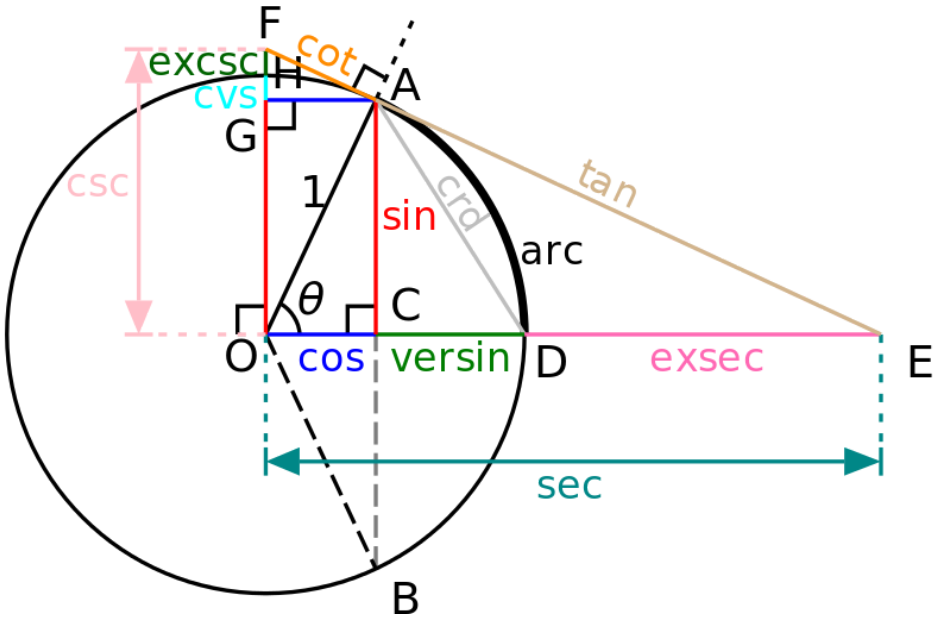
Additional trigonometric function are *versine*, *haversine*, *coversine*, *hacoversine*, *exsecant*, and *excosecant*. All of these can be expressed simply in terms of the more familiar trigonometric functions. For example, haversine(θ) = $\sin^2(\theta/2)$.

The haversine formula is a very accurate way of computing distances between two points on the surface of a sphere using the latitude and longitude of the two points. The haversine formula is a re-formulation of the spherical law of cosines, but the formulation in terms of haversines is more useful for small angles and distances.

One of the primary applications of trigonometry was navigation, and certain commonly used navigational formulas are stated most simply in terms of these archaic function names. But

you might ask, why not just simplify everything down to sines and cosines? The functions listed above were from a time without calculators, or efficient computer processors, when the user calculated angles and direction by hand using log tables, every named function took appreciable effort to evaluate. The point of these functions is if a table simply combines two common operations into one function, it probably made navigational calculations on a rocking ship more efficient.

These function names have a simple naming pattern and in this example, the "Ha" in "Haversine" stands for "half versed sine" where haversin(θ) = versin(θ)/2.



Haversine Formula

The Haversine formula is perhaps the first equation to consider when understanding how to calculate distances on a sphere. The word "Haversine" comes from the function:

haversine(θ) = sin²(θ/2)

The following equation where φ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km) is how we translate the above formula to include latitude and longitude coordinates. Note that angles need to be in radians to pass to trig functions:

a = sin²(φB - φA/2) + cos φA * cos φB * sin²(λB - λA/2)

c = 2 * atan2(√a, √(1−a))

d = R · c

```
Calculate distance using the Haversine Formula
'''

def haversine(coord1: object, coord2: object):
    import math

    # Coordinates in decimal degrees (e.g. 2.89078, 12.79797)
    lon1, lat1 = coord1
    lon2, lat2 = coord2

    R = 6371000  # radius of Earth in meters
    phi_1 = math.radians(lat1)
    phi_2 = math.radians(lat2)

    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)

    a = math.sin(delta_phi / 2.0) ** 2 + math.cos(phi_1) * math.cos(phi_2) * math.sin(delt

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    meters = R * c  # output distance in meters
    km = meters / 1000.0  # output distance in kilometers

    meters = round(meters, 3)
    km = round(km, 3)
```

7.ACTIVITIES

Module1:

Input: Enter the source and destination.

Output: Mark the locations on a Map, here gmaps is used which fetches the latitude and longitude.

Module2:

Input: The output of module1 is i.e latitude and longitude

Output: The distance between the two places which is computed using the Haversine formula. For reference go to index.

Module3:

Input: The distance between two places.

Output: At different angle intervals the rest parameters are given.

Module4:

Input: Above parameters.

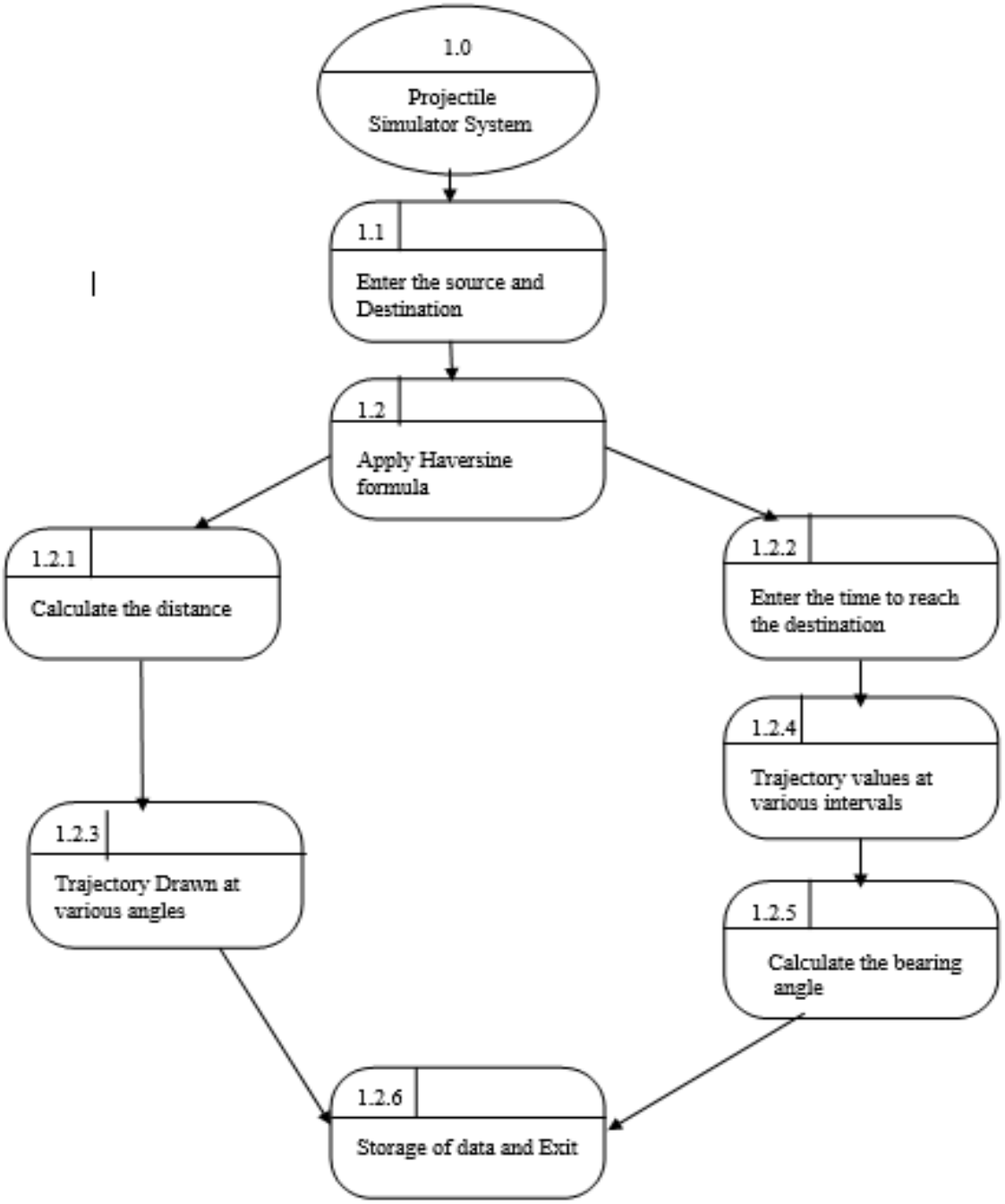
Output: Graph is drawn for each of the input.

Module5:

Input: Time to reach the destination.Parameters i.e velocity and angle of projection

Output: Parameters are given [velocity, angle of projection].At regular intervals of time other parameters are printed with the bearing angle.

DATA FLOW DIAGRAM:



8. IMPLEMENTATION AND SCREEN SHOTS

MODULE1:

```
import gmaps
gmaps.configure(api_key="AIzaSyAvJ7UO_JSS9gyDwMQGFhYX3B-rGMZBa2M")
marker_locations = [
(place['lat'],place['lng']),
(place2['lat'], place2['lng']),
]
fig = gmaps.figure()
markers = gmaps.marker_layer(marker_locations)
fig.add_layer(markers)
fig
```

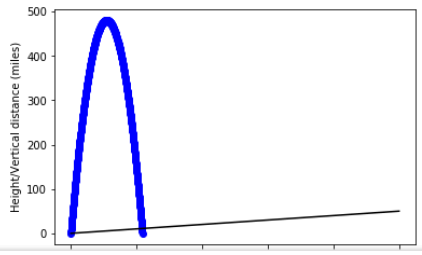
MODULE2:

```
range=haversine(place['lng'],place['lat'],place2['lng'],place2['lat'])
print ('\n\nDistance/Range is: ',range,"\n\n")
```

```
Distance/Range is:  2775.656209767241
```

MODULE3:

Index	Angle	Velocity	Height	Time of Flight
1	10	558.6603776091835	6.194858474488972	3.519902974369882



MODULE4:

Index	Time interval	Height	Range	Velocity at point	Angle_at_Point
1	1.5	209.47500000000005	277.5656209767241	227.47412692748108	35.563481404956406
2	3.0	396.9000000000001	555.1312419534482	219.25088009269146	32.43699476097358
3	4.5	562.2750000000001	832.6968629301723	211.73001303882208	29.077767004101037
4	6.0	705.6000000000003	1110.2624839068965	204.98884950508838	25.484525235036916
5	7.5	826.8750000000002	1387.8281048836204	199.106600647542	21.663142704315447
6	9.0	926.1000000000001	1665.3937258603446	194.1613463628118	17.628315300452698
7	10.5	1003.2750000000002	1942.9593468370688	190.22617701415274	13.40479379004853
8	12.0	1058.4000000000005	2220.524967813793	187.36474700812786	9.027771846409351
9	13.5	1091.4750000000004	2498.090588790517	185.62671796220437	4.54207662279779

MODULE5:

```
from math import radians, cos, sin,tan,atan,asin,atan,pi, sqrt
from math import sin, cos, radians
from math import radians, cos, sin, asin, sqrt

def atan2(y,x):
    return atan((y/x))

lon1=radians(place['lng'])
lon2=radians(place2['lng'])
lat1=radians(place['lat'])
lat2=radians(place['lat'])
ytemp=sin((lon2-lon1))*cos((lat2))
xtemp=cos((lat1))*sin((lat2))-sin((lat1))*cos((lat2))*cos((lon2-lon1))
brng=atan2(ytemp,xtemp)*(180/pi)
print('\nThe Bearing to be considerd is: ',brng,'\n')
```

```
The Bearing to be considerd is:  -79.23753310133198
```

9.FUTURE WORK AND CONCLUSION

The above project is a base work for a vast application for inter ballistic missiles.So I hope I have provided enough information about the project to get started.Up until now it has been the theoretical and computational implementation.

Further improvements would be on designing the hardware implementation of this projects i.e that is presenting this in the real-time environment. The step by procedure is to-

- Design the Hardware
- Build a prototype
- Connecting the hardware and software systems
- Deploying the product

10.REFERENCES

1. https://en.wikipedia.org/wiki/Haversine_formula
2. https://community.esri.com/groups/coordinate-reference-systems/blog/2017/10/05/haversine-formula
3. https://www.maplesoft.com/content/EngineeringFundamentals/1/MapleDocument_1/Projectile%20Motion.pdf
4. http://jupyter-gmaps.readthedocs.io/en/latest/
5. https://www.triton.edu/uploadedFiles/Content/Resources_and_Services/Services/Academic_Success_Center/Student_Resources/Projectile_Motion.pdf