PROJECT

By: Nyameaama Gambrah

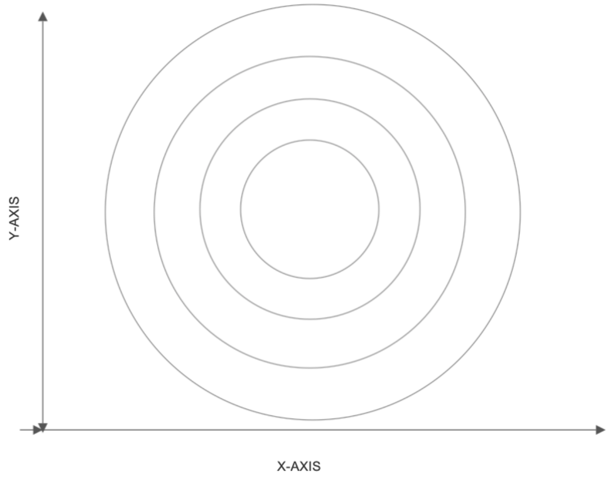
"Measure and model the shortest trajectory a vehicle can travel from earth to a final point between each of the planets at any given time (Gravity Assists should be applied)"

Introduction:

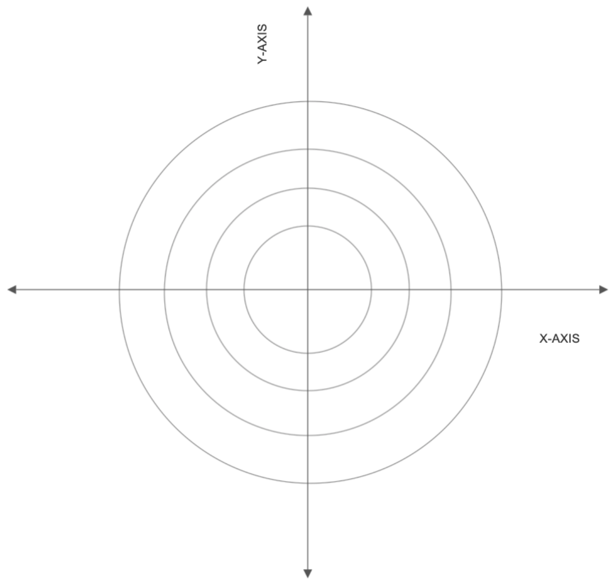
In this project, I wanted to measure the shortest trajectory a vehicle can travel from earth to a final point in space and write a program in order to model this.

Bodies in Space:

Originally we thought we could represent the simple positions of bodies in space with a 2D x-y plane (as shown below) but we quicky encountered some problems. Each body's trajectory in the sun's sphere of influence (The area in which the sun holds gravitational influence) was greatly affected by distance to the sun.



***Not to scale**



***Not to scale**

We could however, derive the equations of the bodies trajectories in the 2d x-y plane by graphing the circular paths (as seen below). The equation of a circle is used

$$(x-h)^2+(y-v)^2=r^2.$$

Distance from Sun (0,0)

Mercury	0.39 AU
Venus	0.723 AU
Earth	1 AU
Mars	1.524 AU
Jupiter	5.203 AU
Saturn	9.539 AU
Uranus	19.18 AU
Neptune	30.06 AU

With the sun being at the midpoint of (0,0) we can simplify the standard circle equation to look like $x^2 + y^2 = r^2$.

Mercury(Equation):

$$r^2 = 0.39^2$$

$$x^2 + y^2 = 0.1521$$

Venus(Equation):

$$r^2 = 0.723^2$$
$$x^2 + y^2 = 0.5227$$

Earth (Equation):

$$r^2 = 1^2$$
$$x^2 + y^2 = 1$$

Mars(Equation):

$$r^2 = 1.524^2$$
$$x^2 + y^2 = 2.322$$

Jupiter(Equation):

$$r^2 = 5.203^2$$
$$x^2 + y^2 = 27.07$$

Saturn(Equation):

$$r^2 = 9.539^2$$
$$x^2 + y^2 = 90.99$$

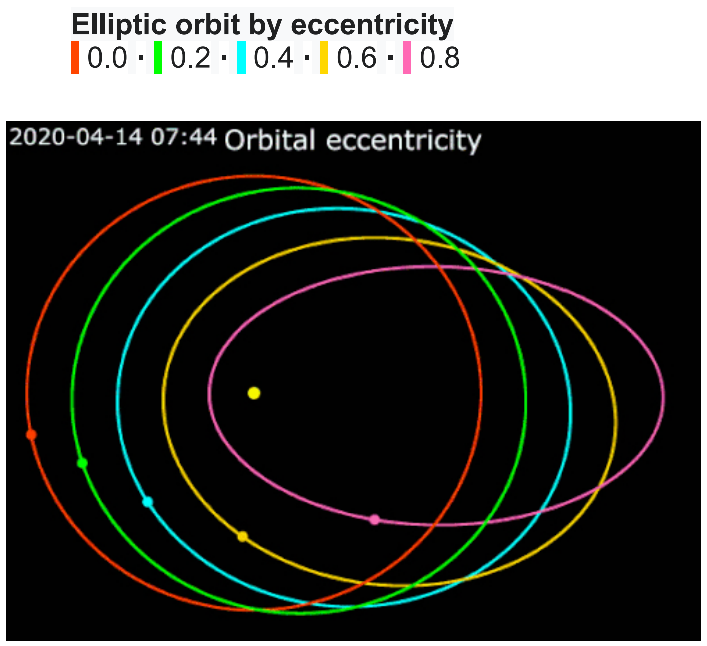
Uranus(Equation):

$$r^2 = 19.18^2$$
$$x^2 + y^2 = 367.872$$

Neptune(Equation):

$$r^2 = 30.06^2$$
$$x^2 + y^2 = 903.6036$$

Although, all bodies in space rarely ever have a perfectly circular path as described, therefore each orbital path has to be characterized by its orbital eccentricity. The orbital eccentricity of an astronomical object is a dimensionless parameter that determines the amount by which its orbit around another body deviates from a perfect circle.


Elliptic orbit by eccentricity
0.0 · 0.2 · 0.4 · 0.6 · 0.8
2020-04-14 07:44 Orbital eccentricity

Planet	Orbital Eccentricity	**Perihelion** (Point in Orbit Closest to Sun) AU	**Aphelion** (Point in Orbit Farthest from Sun) AU
Mercury	0.206	0.31	0.47
Venus	0.007	0.718	0.728
Earth	0.017	0.98	1.02
Mars	0.093	1.38	1.67
Jupiter	0.048	4.95	5.45
Saturn	0.056	9.02	10.0
Uranus	0.047	18.3	20.1
Neptune	0.009	30.0	30.3
Pluto	0.248	29.7	49.9

The orbital eccentricity of a planet can be calculated by the result of:

$$e = \frac{a-p}{a+p}$$

- where ***e*** is the eccentricity,
- ***a*** is the aphelion distance (Semi – major axis), and
- ***p*** is the perihelion distance (Semi – minor axis).

In order to draw the orbit paths for each planetary body in space, the midpoint ellipse drawing algorithm is used:

Mid-Point Ellipse Algorithm :

1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : $(x, y_0) = (0, r_y)$.
3. Obtain the initial decision parameter for region 1 as: $p1_0 = r_y^2 + 1 / 4\, r_x^2 - r_x\,^2 r_y$
4. For every x_k position in region 1 :
   If $p1_k < 0$ then the next point along the is (x_{k+1} , y_k) and
   $p1_{k+1} = p1_k + 2r_y^2\, x_{k+1} + r_y^2$
   Else, the next point is (x_{k+1}, y_{k-1})
   And $p1_{k+1} = p1_k + 2r_y^2 x_{k+1} – 2r_x^2 y_{k+1} + r_y^2$
5. Obtain the initial value in region 2 using the last point (x_0, y_0) of region 1 as: $p2_0 = r_y^2(x_0 + 1/2)^2 + r_x^2\,(y_0 - 1)^2 - r_x^2 r_y^2$
6. At each y_k in region 2 starting at $k = 0$ perform the following task.

If $p2_k > 0$ the next point is (x_k, y_{k-1}) and $p2_{k+1} = p2_k - 2r_x^2y_{k+1} + r_x^2$

7. Else, the next point is (x_{k+1}, y_{k-1}) and $p2_{k+1} = p2_k + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_x^2$
8. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x = x + xc, y = y + yc
9. Repeat the steps for region 1 until $2r_y^2x$&gt = $2r_x^2y$

The implementation is written in python

```
program for implementing
# Mid-Point Ellipse Drawing Algorithm

def midptellipse(rx, ry, xc, yc):

    x = 0;
    y = ry;

    # Initial decision parameter of region 1
    d1 = ((ry * ry) - (rx * rx * ry) +
                      (0.25 * rx * rx));
    dx = 2 * ry * ry * x;
    dy = 2 * rx * rx * y;

    # For region 1
    while (dx < dy):

        # Print points based on 4-way symmetry
        print("(", x + xc, ",", y + yc, ")");
        print("(",-x + xc,",", y + yc, ")");
        print("(",x + xc,",", -y + yc ,")");
        print("(",-x + xc, ",", -y + yc, ")");

        # Checking and updating value of
        # decision parameter based on algorithm
        if (d1 < 0):
            x += 1;
            dx = dx + (2 * ry * ry);
            d1 = d1 + dx + (ry * ry);
        else:
            x += 1;
            y -= 1;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d1 = d1 + dx - dy + (ry * ry);

# Decision parameter of region 2
d2 = (((ry * ry) * ((x + 0.5) * (x + 0.5))) +
      ((rx * rx) * ((y - 1) * (y - 1))) -
       (rx * rx * ry * ry));

# Plotting points of region 2
while (y >= 0):

    # printing points based on 4-way symmetry
    print("(", x + xc, ",", y + yc, ")");
    print("(", -x + xc, ",", y + yc, ")");
    print("(", x + xc, ",", -y + yc, ")");
    print("(", -x + xc, ",", -y + yc, ")");

    # Checking and updating parameter
    # value based on algorithm
    if (d2 > 0):
        y -= 1;
        dy = dy - (2 * rx * rx);
        d2 = d2 + (rx * rx) - dy;
    else:
        y -= 1;
        x += 1;
        dx = dx + (2 * ry * ry);
        dy = dy - (2 * rx * rx);
        d2 = d2 + dx - dy + (rx * rx);
```

With the paths of each body represented in the 2D canvas,

<Picture>

the positions of each body will need to be determined at any time the program will be run. Initially, I had thought to pull position data from a source host every time the program was initialised, although this held problems. With this approach:

- Since the data has to pulled from an external source the program will not run if there is no connection.
- The program speed would vary as it will be at the mercy of a sustainable connection
- No offline use
- Computations will rely solely on data fed from the external source which may reduce integrity

Instead, I used an alternative method in which I would only need reference data to all planets positions at a single point in time, and then using that data to approximate the current position at run time. To approximate this, we need to compile root data.

Planets	**Radius of Orbit Relative to that of Earth's**	**Length of Year Relative to Earth's Year**	**Orbital Velocity Relative to That of Earth's**
Mercury	0.387	0.2409	1.607
Venus	0.723	0.616	1.174
Earth	1.0	1.0	1.000
Mars	1.524	1.9	0.802
Jupiter	5.203	12.0	0.434
Saturn	9.539	29.5	0.323
Uranus	19.18	84	0.228
Neptune	30.06	165	0.182
Pluto	39.52	248	0.159

The table consists of data such as the Radius of Orbit Relative to that of Earth's, the Length of Year Relative to Earth's Year and the Orbital Velocity Relative to That of Earth's. This data