THE HEBREW UNIVERSITY OF JERUSALEM

Waves and Optics

Numerical Analysis of Gravitational Lensing

Amit Roth

1. Numerical Integration For a Single Ray

Given the Ray equation

$$\frac{d}{d\tau}\left(n(\vec{r}(\tau))\frac{\dot{\vec{r}(\tau)}}{\|\dot{\vec{r}(\tau)}\|}\right) = \|\dot{\vec{r}(\tau)}\|\vec{\nabla}n(\vec{r})\Big|_{\vec{r}=\vec{r}(\tau)}$$

And the refractive index

$$n(\vec{r}) = 1 + \frac{2Gm}{c^2 r}, \quad \frac{2Gm}{c^2} = 1$$

We will calculate the a ray's trajectory given the initial values:

1. Light source poistion $(0,z_i)$
2. Mass position $(0,0)$

After assigning $n(\vec{r})$ in the Ray Equation we got:

$$n\ddot{\vec{r}} = \nabla n - (\nabla n \cdot \dot{\vec{r}})\vec{r}$$

$$\ddot{\vec{r}} = \nabla ln(n) - (\nabla ln(n) \cdot \dot{\vec{r}})\vec{r}$$

$$\ddot{\vec{r}} = \frac{-\alpha(x\hat{x} + y\hat{y})}{r^2(\alpha + r)} + \frac{\alpha(x\dot{x}\hat{x} + y\dot{y}\hat{y})}{r^3}$$

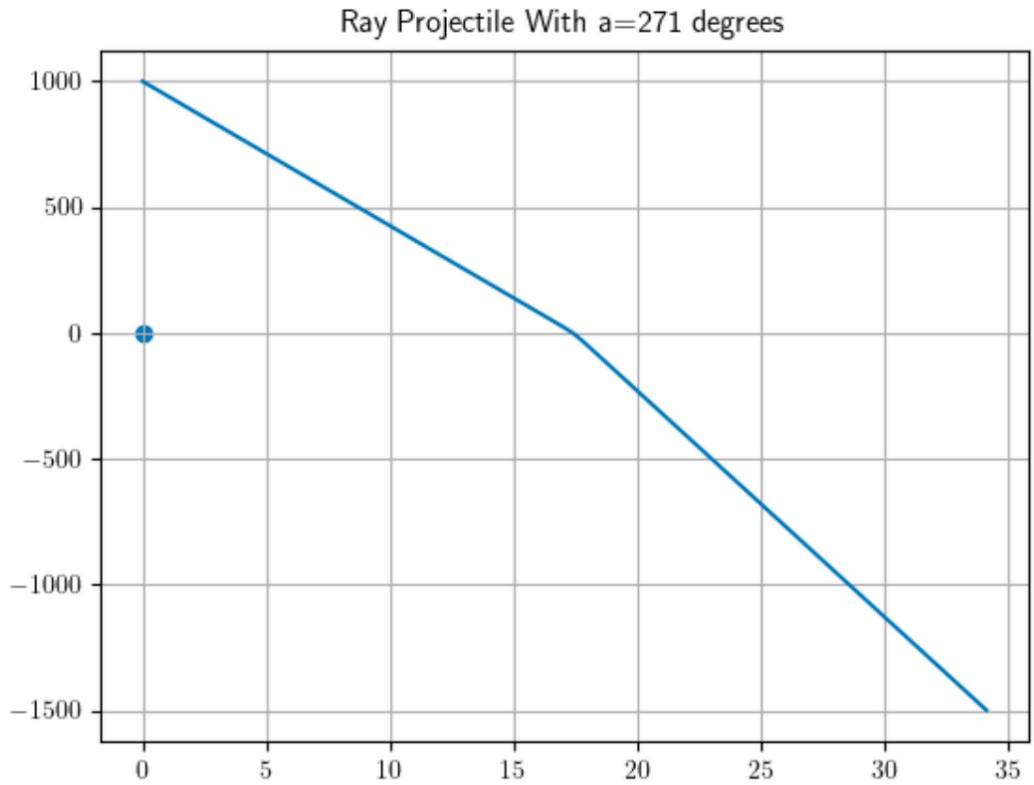
We assumed that the ray is contained in the $x - z$ plane

Explanation

$\dot{\vec{r}}$ and $\ddot{\vec{r}}$ construct a plane which contains the ray.

$\ddot{\vec{r}}$ defined to be orthogonal derivative of $ln(n(\vec{r}))$ hence will always point in the $\dot{\vec{r}}$ - $\ddot{\vec{r}}$ plane.

For a ray with an angle $\alpha = 271^{\circ}$ we can see the trajectory



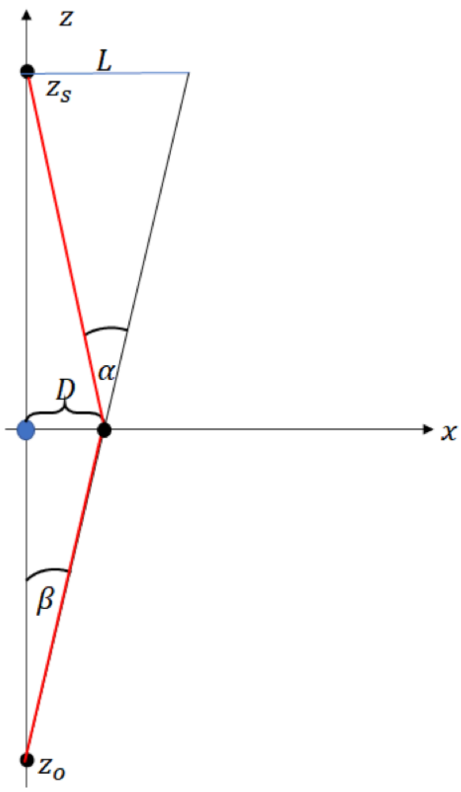
The deflection angle with the following initial values is 6.8°

The approximated analytical solution is 6.57°

The Analytical solution is accurate up to $10\,\%$ of error in the range of small angles - $[-40^{\circ}, 40^{\circ}]$.

2. Einstein Ring

Assume we have a viewer standing at z_0.



We will express β with small angles approximation.

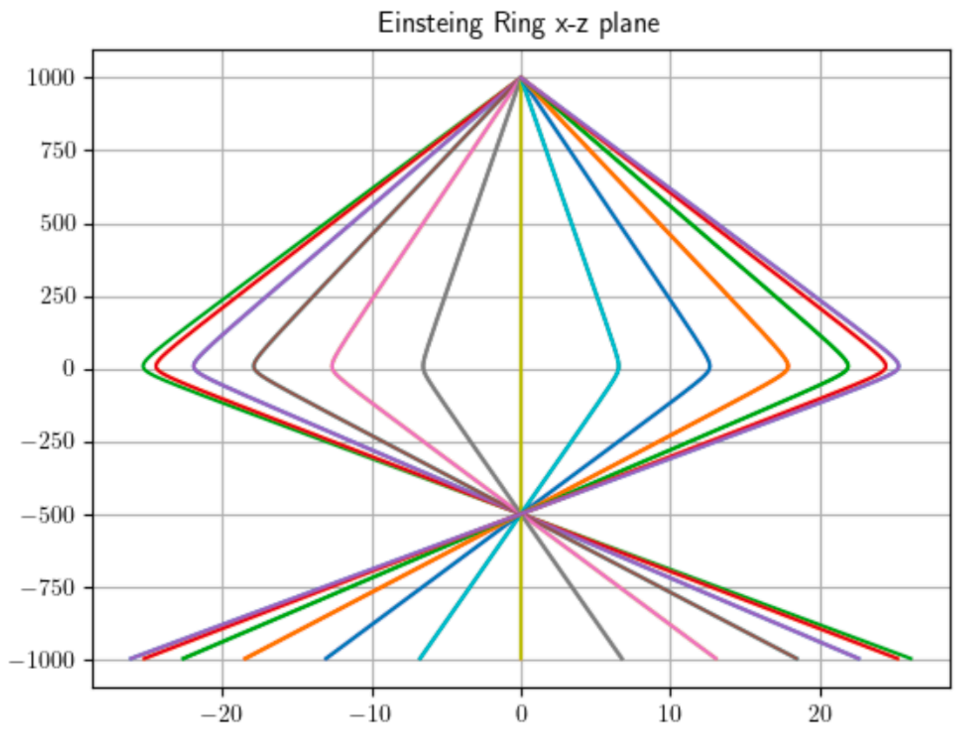
$$tan(\alpha - \beta) \cdot z_s = D = z_o \cdot \tan(\beta)$$

$$\alpha z_s = \beta(z_0 + z_s)$$

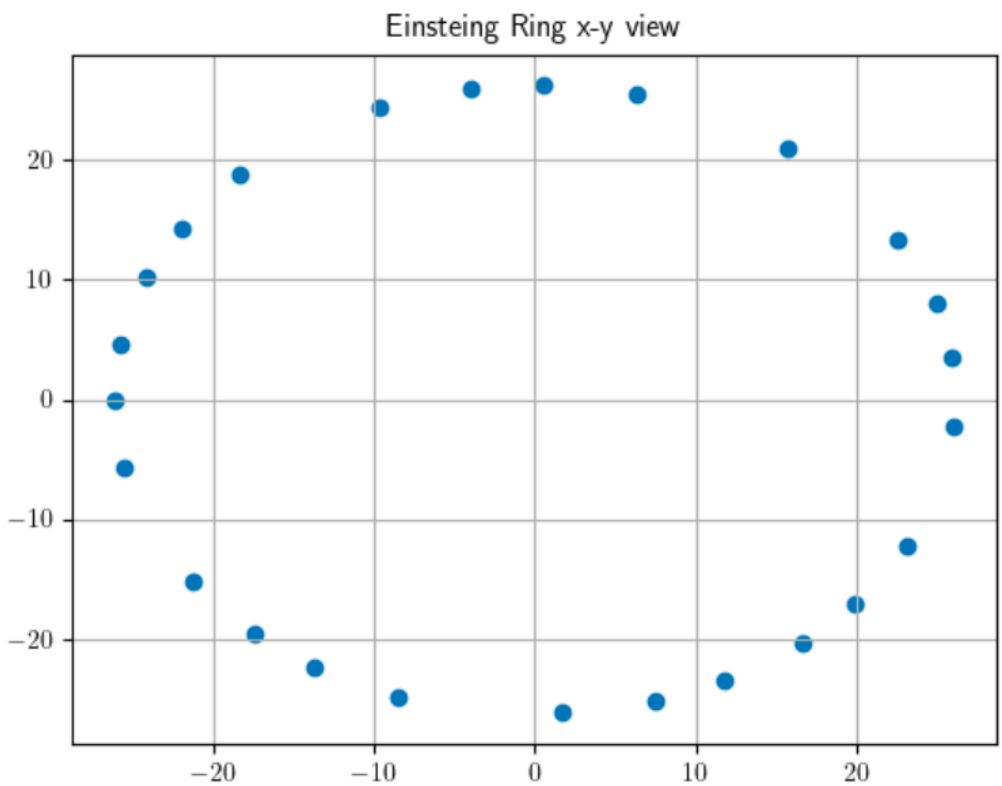
$$\Rightarrow \beta = \frac{z_s}{z_0 + z_s}\alpha$$

$$\Rightarrow D = \beta z_0$$

In order to find all of the rays which arrives z_0 we will calculate the trajectories for the rays for every θ.



And by plotting the rays in $x - y$ plane we would get the following view, A circle with radius $26.27[m]$ and the approximated analytical radius is $25.01[m]$.



3. Two Masses Gravitational Lensing

We would now solve a more general problem with the following refractive index

$$n(\vec{r}) = 1 + \frac{2Gm_1}{c^2|\vec{r} - \vec{r}_1|} + \frac{2Gm_2}{c^2|\vec{r} - \vec{r}_2|}$$

We will use the following initial values in the program

$$\begin{cases} \frac{2Gm}{c^2} = 1 \\ m_1 = m_2 = m \\ \vec{r}_1 = (20{,}0{,}0) \\ \vec{r}_2 = (-20{,}0{,}0) \end{cases}$$

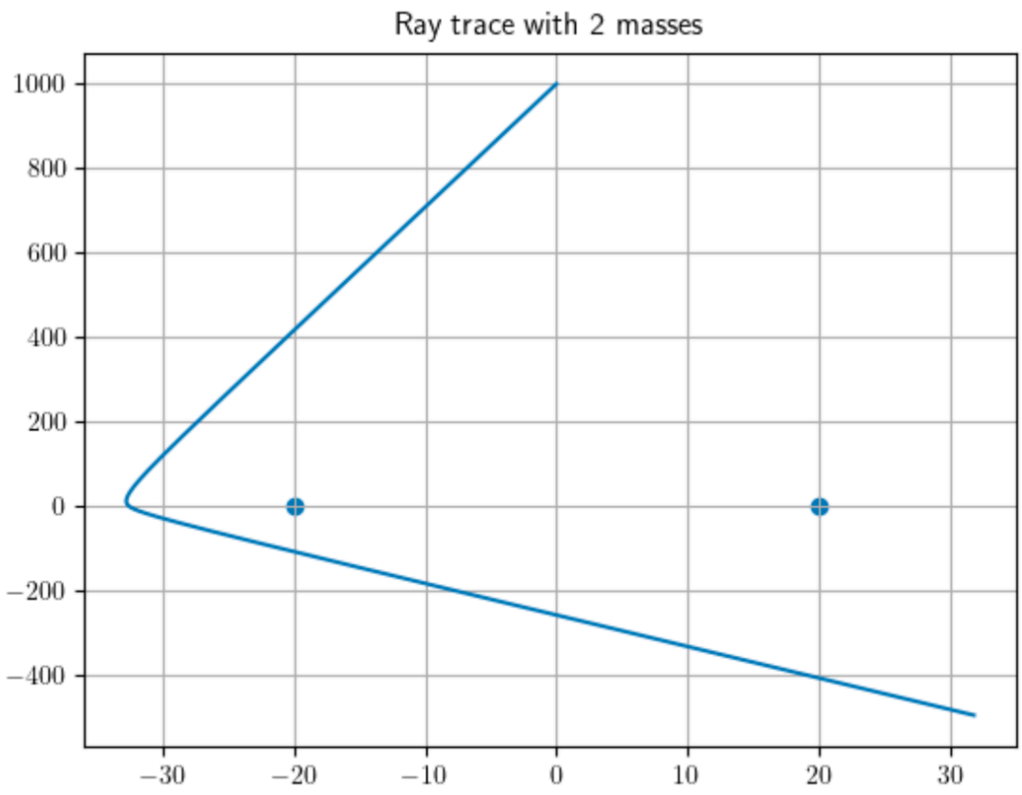
We just need to recalculate $\nabla ln(n)$ and assign it in the previous expression.

$$\nabla ln(n) = \nabla ln(1 + \frac{2Gm_1}{c^2|\vec{r} - \vec{r}_1|} + \frac{2Gm_2}{c^2|\vec{r} - \vec{r}_2|}) =$$

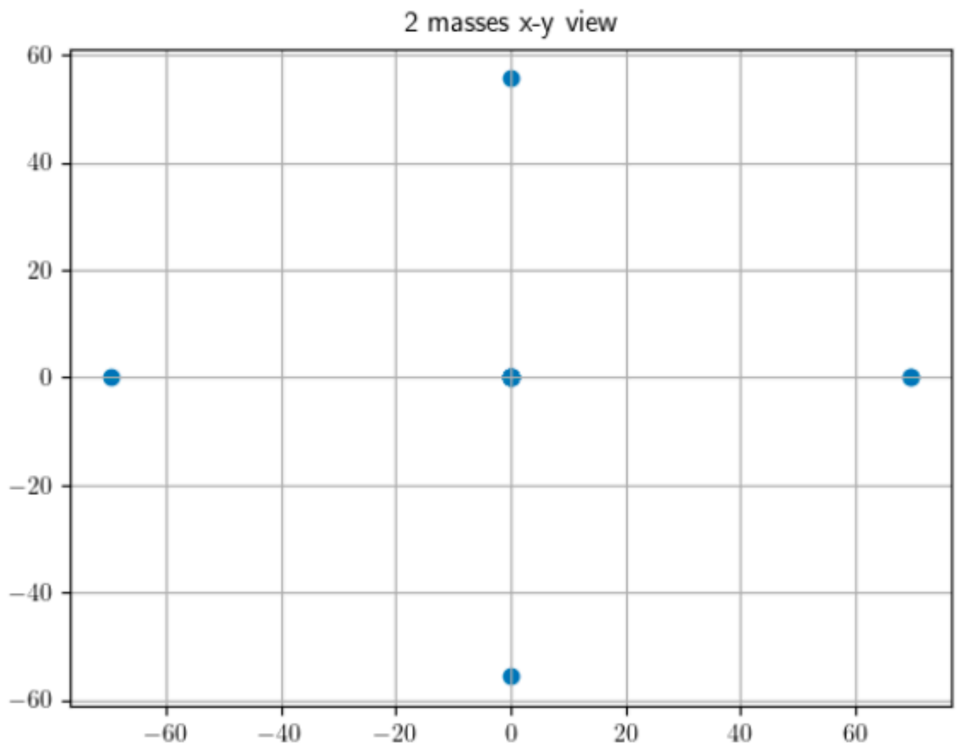
$$\frac{\frac{-\alpha(\vec{r} - \vec{r}_1)}{|\vec{r} - \vec{r}_1|^3} + \frac{-\alpha(\vec{r} - \vec{r}_2)}{|\vec{r} - \vec{r}_2|^3}}{1 + \frac{\alpha}{|\vec{r} - \vec{r}_1|} + \frac{\alpha}{|\vec{r} - \vec{r}_2|}} = \frac{\frac{-\alpha(x - x_1)}{r_1^3} + \frac{-\alpha(x - x_2)}{r_2^3}}{1 + \frac{\alpha}{r_1} + \frac{\alpha}{r_2}}\hat{x} + \frac{\frac{-\alpha(z - z_1)}{r_1^3} + \frac{-\alpha(z - z_2)}{r_2^3}}{1 + \frac{\alpha}{r_1} + \frac{\alpha}{r_2}}\hat{z}$$

We will assign this result in $\ddot{\vec{r}} = \nabla ln(n) - (\nabla ln(n) \cdot \dot{\vec{r}})\vec{r}$ and will get the result.

A trace of a ray with an angle of $\theta = 268°$ is:



By running over the rays which come towards $(0,0,-1300)$ we would get 5 rays arriving from different angles, hence constructing the following image for a viewer:



```python
import math

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import scipy

alpha = 1


def ode2D(x0, z0, theta, s=2000):
    theta = np.deg2rad(theta)

    y0 = [
        x0,
        z0,
        np.cos(theta),
        np.sin(theta)
    ]

    def f(y, t):
        def grad_ln_n(x, z):
            r_squared = x ** 2 + z ** 2
            return - alpha / (r_squared * (np.sqrt(r_squared) + alpha))

        grad_x_z = grad_ln_n(y[0], y[1])
        position_vec = np.array([y[0], y[1]])
        velocity_vec = np.array([y[2], y[3]])
        vx, vy = grad_x_z * position_vec - np.dot(grad_x_z * position_vec, velocity_vec) * velocity_vec

        return [y[2], y[3], vx, vy]

    t = np.arange(0, s, 1)
    y = odeint(f, y0, t)

    return y


```

```
def ode3D(x0, y0, z0, phi, theta, s=2000):
    theta = np.deg2rad(theta)
    phi = np.deg2rad(phi)

    y0 = [
        x0,
        y0,
        z0,
        np.sin(theta - math.pi / 2) * np.cos(phi),
        np.sin(theta - math.pi / 2) * np.sin(phi),
        np.cos(theta - math.pi / 2)
    ]

    def f(y, t):
        def grad_ln_n_2_masses(x, z, y, x1, y1, z1, x2, y2, z2):
            r1 = np.sqrt((x - x1) ** 2 + (z - z1) ** 2 + (y - y1) ** 2)
            r2 = np.sqrt((x - x2) ** 2 + (z - z2) ** 2 + (y - y2) ** 2)

            x_numerator = - alpha * ((x - x1) / r1 ** 3 + (x - x2) / r2 ** 3)
            y_numerator = - alpha * ((y - y1) / r1 ** 3 + (y - y2) / r2 ** 3)
            z_numerator = - alpha * ((z - z1) / r1 ** 3 + (z - z2) / r2 ** 3)
            denominator = (1 + alpha * (1 / r1 + 1 / r2))

            return np.array(
                [x_numerator / denominator, y_numerator / denominator, z_numerator / denominator]
            )

        grad_x_z = grad_ln_n_2_masses(x=y[0], y=y[1], z=y[2], x1=-20, y1=0, z1=0, x2=20, y2=0, z2=0)
        velocity_vec = np.array([y[3], y[4], y[5]])
        pos = grad_x_z - np.dot(grad_x_z, velocity_vec) * velocity_vec
```

```
        return [y[3], y[4], y[5],
                pos[0], pos[1], pos[2]]

    t = np.arange(0, s, 1)
    y = odeint(f, y0, t)

    return y


def approximated_analytical_angle(angle):
    b = 1000 * np.tan(np.deg2rad(angle))
    theta = 2 * alpha / b
    print("analytical theta {} and {} deg".format(theta, np.rad2deg(theta)))
    return theta


def error(real, approx):
    return 100 * abs(real - approx) / abs(real)


def calculate_angles(x, z):
    angles = np.arctan2(
        np.array([z[1] - z[0], z[-1] - z[-2]]),
        np.array([x[1] - x[0], x[-1] - x[-2]])
    )
    mid_angle = abs(angles[1]) - abs(angles[0])
    return angles[0], angles[1] - math.pi / 2, mid_angle


def einstein_ring_2D(x0=0, z0=1000, x1=0, z1=-500, theta_range=[]):
    def calculate_distance_route_from_point_2D(x, z, x1, z1):
        for p in zip(x, z):
            if (p[0] - x1) ** 2 + (p[1] - z1) ** 2 < 1:
                return True

        return False
```

```

    routes = list()
    for theta in theta_range:
        y = ode2D(x0=x0, z0=z0, theta=theta, s=2000)
        x, z, _, _ = zip(*y)
        if calculate_distance_route_from_point_2D(x, z, x1, z1):
            routes.append(
                [x, z]
            )
    return routes


def einstein_ring_3D(x0=0, y0=0, z0=1000, x1=0, y1=0, z1=-1300, theta_range=[], phi_range=[]):
    def calculate_distance_route_from_point_3D(x, y, z, x1, y1, z1):
        for p in zip(x, y, z):
            if (p[0] - x1) ** 2 + (p[1] - y1) ** 2 + (p[2] - z1) ** 2 < 1:
                return True
        return False

    routes = list()

    for theta in theta_range:
        for phi in phi_range:
            route = ode3D(x0=x0, y0=y0, z0=z0, theta=theta, phi=phi, s=4000)
            x, y, z, _, _, _ = zip(*route)

            if calculate_distance_route_from_point_3D(x, y, z, x1, y1, z1):
                routes.append(
                    [x, y, z]
                )

    return routes

```

```python

def create_x_z_rays(routes, phi_range):
    new_routes = []
    for route in routes:
        for phi in phi_range:
            x = list(route[0]) * np.full(len(route[0]), np.cos(np.deg2rad(phi)))
            z = route[1]
            new_routes.append([x, z])

    return new_routes


def create_x_y_points_2D(routes, phi_range):
    circle_points_x = []
    circle_points_y = []
    for route in routes:
        in_angle, out_angle, mid_angle = calculate_angles(route[0], route[1])
        for phi in phi_range:
            x = np.tan(out_angle) * 500 * np.cos(phi)
            y = np.tan(out_angle) * 500 * np.sin(phi)
            circle_points_x.append(x)
            circle_points_y.append(y)

    return circle_points_x, circle_points_y


def create_x_y_points_3D(routes):
    def point_in_z_plane(route):
        """
        returns the x,y of the neerest point to z=0 plane
        :param route:
        :return:
        """
        min_dist = 100
        min_point = None
        for point in zip(*route):
```

```
            if point[2] < min_dist:
                min_dist = point[2]
                min_point = point
        return min_point[0], min_point[1]

    circle_points_x = []
    circle_points_y = []
    for route in routes:
        x, y = point_in_z_plane(route)
        circle_points_x.append(x)
        circle_points_y.append(y)

    return circle_points_x, circle_points_y


def section_1():
    angle = 275
    y = ode2D(x0=0, z0=1000, theta=angle, s=2500)
    x, z, _, _ = zip(*y)
    plt.rcParams['text.usetex'] = True
    plt.plot(x, z)

    theta_in, theta_out, theta_mid = calculate_angles(x, z)
    theta_approx = approximated_analytical_angle(angle - 270)
    print("The error of the approximation is {} %".format(error(theta_mid, theta_approx)))
    plt.scatter([0], [0])
    plt.title("Ray Projectile With a={} degrees".format(angle))
    plt.grid(True)
    plt.savefig('Ray trace')
    plt.show()


def section_2():
    phi_range = np.linspace(0, 360, 25)
    theta_range = np.linspace(271, 272, 11)

    routes = einstein_ring_2D(theta_range=
```

```
theta_range)
    x_z_rays = create_x_z_rays(routes, phi_range)
    x_y_points = create_x_y_points_2D(routes, phi_range)

    for ray in x_z_rays:
        plt.plot(ray[0], ray[1])
    plt.grid(True)
    plt.title("Einsteing Ring x-z plane")
    plt.savefig('Einsteing Ring x-z plane')
    plt.show()

    plt.scatter(x=x_y_points[0], y=x_y_points[1])
    plt.grid(True)
    plt.title("Einsteing Ring x-y view")
    plt.savefig('x-y circle')
    plt.show()

    r1 = np.sqrt(x_y_points[0][0] ** 2 + x_y_points[1][0] ** 2)
    beta = approximated_analytical_angle(1.5) - np.deg2rad(1.5)  # alpha - theta
    r2 = 500 * beta  # z_0 * beta
    print("radius of circle is {} and approximated anayltical radius is {}".format(r1, r2))


def section_3():
    y = ode3D(x0=0, y0=0, z0=1000, theta=272, phi=10, s=1500)
    x, y, z, _, _, _ = zip(*y)
    plt.rcParams['text.usetex'] = True
    plt.plot(x, z)
    plt.scatter([-20, 20], [0, 0])
    plt.grid(True)
    plt.title('Ray trace with 2 masses')
    plt.show()

    theta_range = np.linspace(270, 280, 21)
    phi_range = np.linspace(0, 360, 9)

```

```
    routes = einstein_ring_3D(theta_range=theta_range, phi_range=phi_range)
    for ray in routes:
        plt.plot(ray[0], ray[2])

    plt.title('2 masses x-z view')
    plt.show()

    x_points, y_points = create_x_y_points_3D(routes)
    plt.scatter(x=x_points, y=y_points)
    plt.grid(True)
    plt.title("2 masses x-y view")
    plt.show()


if __name__ == '__main__':
    section_1()
    section_2()
    section_3()

```