```python
import numpy as np
from matplotlib import animation
import pylab


def RungeKutta(f1, f2, f3, f4, a_init, b_init, c_init, d_init, t0, tf, h=0.1):
    A = []
    B = []
    C = []
    D = []
    T = np.arange(t0, tf, h)
    a = a_init
    b = b_init
    c = c_init
    d = d_init
    for t in T:
        A.append(a)
        B.append(b)
        C.append(c)
        D.append(d)
        k1 = h * f1(a, b, c, d, t)
        l1 = h * f2(a, b, c, d, t)
        m1 = h * f3(a, b, c, d, t)
        n1 = h * f4(a, b, c, d, t)
        k2 = h * f1(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
        l2 = h * f2(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
        m2 = h * f3(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
        n2 = h * f4(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
        k3 = h * f1(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
        l3 = h * f2(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
        m3 = h * f3(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
        n3 = h * f4(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
        k4 = h * f1(a + k3, b + l3, c + m3, d + n3, t + h)
        l4 = h * f2(a + k3, b + l3, c + m3, d + n3, t + h)
        m4 = h * f3(a + k3, b + l3, c + m3, d + n3, t + h)
        n4 = h * f4(a + k3, b + l3, c + m3, d + n3, t + h)
        a += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        b += (l1 + 2 * l2 + 2 * l3 + l4) / 6
        c += (m1 + 2 * m2 + 2 * m3 + m4) / 6
        d += (n1 + 2 * n2 + 2 * n3 + n4) / 6
    return T, A, B, C, D


def minimum(T, X, Y, Vx, Vy, h):
    delta = 10*h
    V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
    Theta = [np.fabs(y/ x) * 57.2958 for x, y in zip(X, Y)]
    theta = []
    times = []
    vel = []
    xpos = []
    ypos = []
    step = int(0.24 / h)
    for i in range(5):
        v_max = max(V)
        index = V.index(v_max)
        vel.append(v_max)
        theta.append(Theta[index])
        times.append(T[index])
        xpos.append(X[index])
        ypos.append(Y[index])
        V = V[:max(index - 20, 0)] + V[min(index + 20, len(V)):]
        Theta = Theta[:max(index - 20, 0)] + Theta[min(index + 20, len(Theta)):]
```

```
        T = T[:max(index - 20, 0)] + T[min(index + 20, len(T)):]
        X = X[:max(index - 20, 0)] + X[min(index + 20, len(X)):]
        Y = Y[:max(index - 20, 0)] + Y[min(index + 20, len(Y)):]
    return vel, theta, times, xpos, ypos

def get_slope(alpha):
    MsG = 4 * np.pi**2
    f1 = lambda x, y, vx, vy, t: vx
    f2 = lambda x, y, vx, vy, t: vy
    f3 = lambda x, y, vx, vy, t: -MsG * x / pow(x**2 + y**2, 3/2)-MsG*x*alpha/pow(x**2+y**2,5/2)
    f4 = lambda x, y, vx, vy, t: -MsG * y / pow(x**2 + y**2, 3/2)-MsG*y*alpha/pow(x**2+y**2,5/2)
    T, X, Y, Vx, Vy = RungeKutta(f1, f2, f3, f4, 0.3897, 0, 0, 8.166, 0.0, 1,
                                 0.0001)
    v, y, x, xp, yp= list(minimum(list(T), X, Y, Vx, Vy, 0.0001))
    V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
    # pylab.plot(xp, yp, 'bo')
    # pylab.plot(X, Y, 'k.', markersize=0.1)
    # pylab.show()
    # pylab.plot(x, v, 'bo')
    # pylab.plot(T, V, 'k.', markersize=0.1)
    # pylab.show()
    E_x = 0.0
    E_y = 0.0
    E_xx = 0.0
    E_xy = 0.0
    for pt in range(len(y)):
        E_x += x[pt]
        E_y += y[pt]
        E_xx += (x[pt]**2)
        E_xy += (x[pt]*y[pt])
    E_x /= len(y)
    E_y /= len(y)
    E_xx /= len(y)
    E_xy /= len(y)
    m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
    c = (E_xx * E_y - E_x * E_xy)/(E_xx - E_x**2)
    # pylab.plot(x, y, 'k.')
    # pylab.plot(np.linspace(min(x), max(x)), [m*x+c for x in np.linspace(min(x),max(x))])
    # pylab.show()
    return m


def main():
    # MsG = 4 * np.pi**2
    # alpha = 1e-2
    # alpha = 0.0008
    # f1 = lambda x, y, vx, vy, t: vx
    # f2 = lambda x, y, vx, vy, t: vy
    # f3 = lambda x, y, vx, vy, t: -MsG * x / pow(x**2 + y**2, 3/2)-MsG*x*alpha/pow(x**2+y**2,5/2)
    # f4 = lambda x, y, vx, vy, t: -MsG * y / pow(x**2 + y**2, 3/2)-MsG*y*alpha/pow(x**2+y**2,5/2)
    # T, X, Y, Vx, Vy = RungeKutta(f1, f2, f3, f4, 0.3897, 0, 0, 8.166, 0.0, 1,
    #                              0.0001)
    # print("Calculated")
    # V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
    # R = [np.sqrt(x**2+y**2) for x, y in zip(X, Y)]
    # y, x = list(minimum(list(T), X, Y, Vx, Vy, 0.0001))
    # pylab.plot(x, y, 'k.')
    # pylab.show()
```

```
    slopes = []
    alphas = np.linspace(1e-3, 1e-4)
    for alpha in alphas:
        slopes.append(get_slope(alpha))
        print(alpha, slopes[-1])
    E_x = 0.0
    E_y = 0.0
    E_xx = 0.0
    E_xy = 0.0
    for pt in range(len(slopes)):
        E_x += alphas[pt]
        E_y += slopes[pt]
        E_xx += (alphas[pt]**2)
        E_xy += (alphas[pt]*slopes[pt])
    E_x /= len(slopes)
    E_y /= len(slopes)
    E_xx /= len(slopes)
    E_xy /= len(slopes)
    m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
    c = (E_xx * E_y - E_x * E_xy)/(E_xx - E_x**2)
    print("{}*x+{}".format(m, c))
    mer_alpha = 1.1e-8
    print("\u0251={}".format(mer_alpha))
    print("d\u03b8/dt={}".format((m*mer_alpha + c) * 3600 * 0.01))
    pylab.plot(alphas, slopes)
    pylab.plot(alphas, [m*x+c for x in alphas])
    pylab.show()
    # E_x = 0.0
    # E_y = 0.0
    # E_xx = 0.0
    # E_xy = 0.0
    # for pt in range(len(y)):
    #     E_x += x[pt]
    #     E_y += y[pt]
    #     E_xx += (x[pt]**2)
    #     E_xy += (x[pt]*y[pt])
    # E_x /= len(y)
    # E_y /= len(y)
    # E_xx /= len(y)
    # E_xy /= len(y)
    # m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
    # x = points[0]
    # y = points[1]
    # pylab.plot(x, y, ’bo’)
    # pylab.plot(T, V, ’k.’, markersize=0.1)
    # pylab.show()
    # pylab.ylim((-0.5, 0.5))
    # pylab.xlim((-0.5, 0.5))
    # pylab.plot(X, Y, ’k.’)
    # pylab.show()
    # pylab.show()
    # pylab.plot(T, R, ’k.’)
    # pylab.show()


if __name__ == "__main__":
    main()
```