

```
1: import numpy as np
2: from matplotlib import animation
3: import pylab
4:
5:
6: def RungeKutta(f1, f2, f3, f4, a_init, b_init, c_init, d_init, t0, tf, h=0.1):
7:     A = []
8:     B = []
9:     C = []
10:    D = []
11:    T = np.arange(t0, tf, h)
12:    a = a_init
13:    b = b_init
14:    c = c_init
15:    d = d_init
16:    for t in T:
17:        A.append(a)
18:        B.append(b)
19:        C.append(c)
20:        D.append(d)
21:        k1 = h * f1(a, b, c, d, t)
22:        l1 = h * f2(a, b, c, d, t)
23:        m1 = h * f3(a, b, c, d, t)
24:        n1 = h * f4(a, b, c, d, t)
25:        k2 = h * f1(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
26:        l2 = h * f2(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
27:        m2 = h * f3(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
28:        n2 = h * f4(a + k1 / 2, b + l1 / 2, c + m1 / 2, d + n1 / 2, t + h / 2)
29:        k3 = h * f1(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
30:        l3 = h * f2(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
31:        m3 = h * f3(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
32:        n3 = h * f4(a + k2 / 2, b + l2 / 2, c + m2 / 2, d + n2 / 2, t + h / 2)
33:        k4 = h * f1(a + k3, b + l3, c + m3, d + n3, t + h)
34:        l4 = h * f2(a + k3, b + l3, c + m3, d + n3, t + h)
35:        m4 = h * f3(a + k3, b + l3, c + m3, d + n3, t + h)
36:        n4 = h * f4(a + k3, b + l3, c + m3, d + n3, t + h)
37:        a += (k1 + 2 * k2 + 2 * k3 + k4) / 6
38:        b += (l1 + 2 * l2 + 2 * l3 + l4) / 6
39:        c += (m1 + 2 * m2 + 2 * m3 + m4) / 6
40:        d += (n1 + 2 * n2 + 2 * n3 + n4) / 6
41:    return T, A, B, C, D
42:
43:
44: def minimum(T, X, Y, Vx, Vy, h):
45:     delta = 10*h
46:     V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
47:     Theta = [np.fabs(y/ x) * 57.2958 for x, y in zip(X, Y)]
48:     theta = []
49:     times = []
50:     vel = []
51:     xpos = []
52:     ypos = []
53:     step = int(0.24 / h)
54:     for i in range(5):
55:         v_max = max(V)
56:         index = V.index(v_max)
57:         vel.append(v_max)
58:         theta.append(Theta[index])
59:         times.append(T[index])
60:         xpos.append(X[index])
61:         ypos.append(Y[index])
62:         V = V[:max(index - 20, 0)] + V[min(index + 20, len(V)):]
63:         Theta = Theta[:max(index - 20, 0)] + Theta[min(index + 20, len(Theta)):]
```

```

64:         T = T[:max(index - 20, 0)] + T[min(index + 20, len(T)):]
65:         X = X[:max(index - 20, 0)] + X[min(index + 20, len(X)):]
66:         Y = Y[:max(index - 20, 0)] + Y[min(index + 20, len(Y)):]
67:         return vel, theta, times, xpos, ypos
68:
69: def get_slope(alpha):
70:     MsG = 4 * np.pi**2
71:     f1 = lambda x, y, vx, vy, t: vx
72:     f2 = lambda x, y, vx, vy, t: vy
73:     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)
74:     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)
75:     T, X, Y, Vx, Vy = RungeKutta(f1, f2, f3, f4, 0.3897, 0, 0, 8.166, 0.0, 1,
76:                                   0.0001)
77:     v, y, x, xp, yp= list(minimum(list(T), X, Y, Vx, Vy, 0.0001))
78:     V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
79:     # pylab.plot(xp, yp, 'bo')
80:     # pylab.plot(X, Y, 'k.', markersize=0.1)
81:     # pylab.show()
82:     # pylab.plot(x, v, 'bo')
83:     # pylab.plot(T, V, 'k.', markersize=0.1)
84:     # pylab.show()
85:     E_x = 0.0
86:     E_y = 0.0
87:     E_xx = 0.0
88:     E_xy = 0.0
89:     for pt in range(len(y)):
90:         E_x += x[pt]
91:         E_y += y[pt]
92:         E_xx += (x[pt]**2)
93:         E_xy += (x[pt]*y[pt])
94:     E_x /= len(y)
95:     E_y /= len(y)
96:     E_xx /= len(y)
97:     E_xy /= len(y)
98:     m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
99:     c = (E_xx * E_y - E_x * E_xy) / (E_xx - E_x**2)
100:    # pylab.plot(x, y, 'k.')
101:    # pylab.plot(np.linspace(min(x), max(x)), [m*x+c for x in np.linspace(min(x),max(x)
)])
102:    # pylab.show()
103:    return m
104:
105:
106: def main():
107:     # MsG = 4 * np.pi**2
108:     # alpha = 1e-2
109:     # alpha = 0.0008
110:     # f1 = lambda x, y, vx, vy, t: vx
111:     # f2 = lambda x, y, vx, vy, t: vy
112:     # 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)
113:     # 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)
114:     # T, X, Y, Vx, Vy = RungeKutta(f1, f2, f3, f4, 0.3897, 0, 0, 8.166, 0.0, 1,
115:                                   0.0001)
116:     # print("Calculated")
117:     # V = [np.sqrt(x**2+y**2) for x, y in zip(Vx, Vy)]
118:     # R = [np.sqrt(x**2+y**2) for x, y in zip(X, Y)]
119:     # y, x = list(minimum(list(T), X, Y, Vx, Vy, 0.0001))
120:     # pylab.plot(x, y, 'k.')
121:     # pylab.show()

```

```
122:     slopes = []
123:     alphas = np.linspace(1e-3, 1e-4)
124:     for alpha in alphas:
125:         slopes.append(get_slope(alpha))
126:         print(alpha, slopes[-1])
127:     E_x = 0.0
128:     E_y = 0.0
129:     E_xx = 0.0
130:     E_xy = 0.0
131:     for pt in range(len(slopes)):
132:         E_x += alphas[pt]
133:         E_y += slopes[pt]
134:         E_xx += (alphas[pt]**2)
135:         E_xy += (alphas[pt]*slopes[pt])
136:     E_x /= len(slopes)
137:     E_y /= len(slopes)
138:     E_xx /= len(slopes)
139:     E_xy /= len(slopes)
140:     m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
141:     c = (E_xx * E_y - E_x * E_xy) / (E_xx - E_x**2)
142:     print("{}*x+{}".format(m, c))
143:     mer_alpha = 1.1e-8
144:     print("\u0251={}".format(mer_alpha))
145:     print("d\u03b8/dt={}".format((m*mer_alpha + c) * 3600 * 0.01))
146:     pylab.plot(alphas, slopes)
147:     pylab.plot(alphas, [m*x+c for x in alphas])
148:     pylab.show()
149:     # E_x = 0.0
150:     # E_y = 0.0
151:     # E_xx = 0.0
152:     # E_xy = 0.0
153:     # for pt in range(len(y)):
154:     #     E_x += x[pt]
155:     #     E_y += y[pt]
156:     #     E_xx += (x[pt]**2)
157:     #     E_xy += (x[pt]*y[pt])
158:     # E_x /= len(y)
159:     # E_y /= len(y)
160:     # E_xx /= len(y)
161:     # E_xy /= len(y)
162:     # m = (E_xy - E_x * E_y) / (E_xx - E_x**2)
163:     # x = points[0]
164:     # y = points[1]
165:     # pylab.plot(x, y, 'bo')
166:     # pylab.plot(T, V, 'k.', markersize=0.1)
167:     # pylab.show()
168:     # pylab.ylim((-0.5, 0.5))
169:     # pylab.xlim((-0.5, 0.5))
170:     # pylab.plot(X, Y, 'k.')
171:     # pylab.show()
172:     # pylab.show()
173:     # pylab.plot(T, R, 'k.')
174:     # pylab.show()
175:
176:
177: if __name__ == "__main__":
178:     main()
```