Solving the Trace Length of Mice Problem with Calculus

Winnie Shi and Floyd Liu

Professor Taryn Flock

1. Introduction and Literature Review

Pursuit curves is a fundamental concept in mathematics that have been extensively studied in various fields, including multivariable calculus, differential equations, optimal control theory, mathematical biology, and computational geometry. In this project, we will study and visualize a special pursuit curve–the Mice problem. In the pursuit curve problem, it is assumed that there is a pursuer and a pursued, and the pursuer moves directly towards the pursued at a unit speed. Typically, the analysis goals are getting the path shape and the distance traveled by the pursuer. In our specific scenario, there are n participants. Their starting position is at the vertex of an n polygon with a unit length l. Each participant chases directly toward at a constant speed the next participant in a counterclockwise direction. The movement trajectories of these participants form a set of spiral curves in this n polygon. This problem is called the Mice problem, also known as the Beetle problem.

Problem 1.1 (Mice Problem) *There are n mice starting at the corners of a regular n-gon of unit side length, each heading towards its closest neighboring mouse in a counterclockwise direction at a constant speed.*

This problem was first posed by Edouard Lucas in 1877 in the form of the "three-dog problem". The three-dog problem is the case where $n = 3$ in the Mice problem. For the Mice problem, each of the mice traces out a logarithmic spiral. The length of this path can be calculated.[3]

In Bernhart's 1959 study, the mathematical exploration of pursuit polygons is detailed, focusing on logarithmic spirals and Brocard points. The work investigates harmonic polygons, Lemoine points, and polygonal symmetries, which provides geometric configurations in pursuit problems. [1]

Theorem 1.2 (Mice problem trace length) *For n mice in the mice problem, they meet in the center of the polygon and travel a distance of [4]*

$$d_n = \frac{1}{1 - \cos\left(\frac{2\pi}{n}\right)} \tag{1}$$

"Math Physics Explained" employed a mathematical approach to calculate the trajectory distance of the ants pursuing each other in a polygon. This calculation involved solving a system of nonlinear ordinary differential equations (ODEs) that describe the motion of each ant as they follow each other in a spiral toward the center of the polygon. [2]

We proved theorem 1.2 using multivariable calculus principles, including recursive sequences, iterative methods, and differential calculus. We utilize parametric equations to model each mouse's path over time. We numerically solve differential equations to describe changes in path with Euler's method, and incorporate *limit* processes as the distance decreases, reflecting derivative concepts. Additionally, we derive relationships between successive n-gon side lengths to calculate distances of the trajectory path.

2. Limit method to proof trace length theorem

From the calculus perspective, we can cut the mouse's movement into small segments with a time length of Δt. We can first assume that the mouse does not change direction with the position of the target within this Δt time. Finally, we can take $\Delta t \to 0$ to approximate the true path length. We used a Python program to simulate this process. Figure 1a, Figure 1b, Figure 1c, and Figure 1d show the situation when $n = 3$, $n = 4$, $n = 5$, and $n = 6$ respectively.

2.1. Recursive sequence iteration

The path length can be calculated using the recursive sequence iteration method. In the program we made, we asked the program to draw a new n-gon formed by the mice positions after each Δt. This results in the figure forming an envelope curve.

These n-gons are sequentially inscribed. In each set of inscriptions, each vertex of the smaller n-gon is a fixed distance from the corresponding vertex of the larger n-gon. In a practical sense, this fixed distance is the distance traveled by the mouse in Δt. Let this distance be Δd. Therefore, the side length of the new n-gon obtained each time after Δt is calculable.

We can think of these side lengths as elements in a sequence. Let $\langle a_i \rangle_{i=0}^{k}$, a_0 is the initial distance of the mouse, that is, the unit distance $a_0 = 1$, a_i is the side length of the formed n-gon after the mouse passes through i segments of length Δd. The last term of the sequence is a_k, where k is the smallest natural number such that $a_k \leq \Delta d$.

Suppose the path length we need is D. Therefore, we can calculate the length of the path,

$$D = k \times \Delta d + e \tag{2}$$

Here, e is the error caused by the true length of the path minus the approximate length of the sequence up to a_k. Obviously, because $a_k \leq \Delta d$, when $\Delta t \to 0$ there are $\Delta d \to 0$, and $e \to 0$.

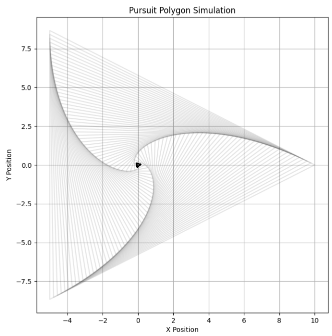
Figure 2 show an example when $n = 3$. In $\triangle ABC$, each angle is $\frac{\pi}{3}$. So we can use the cosine theorem to get the relationship between a_{n+1} and a_n.

$$a_{n+1}^2 = \Delta d^2 + (a_n - \Delta d)^2 - 2\Delta d \cdot (a_n - \Delta d) \cdot \cos\left(\frac{\pi}{3}\right) \tag{3}$$

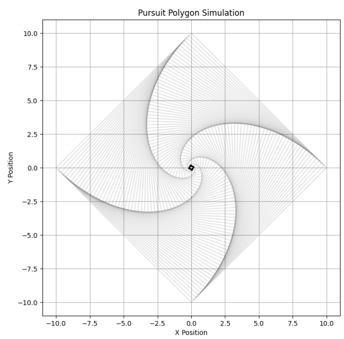
$$a_{n+1}^2 = a_n^2 + 3\Delta d^2 - 3a_n \Delta d \tag{4}$$

Next, we can use iterative calculation in the program to obtain the expression of k for Δd. Then we take $\Delta d \to 0$ to calculate and get the length of D.

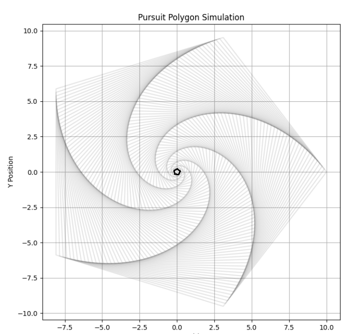
The shortcomings of this method are very obvious, because the close form of this sequence is very difficult to find, and we



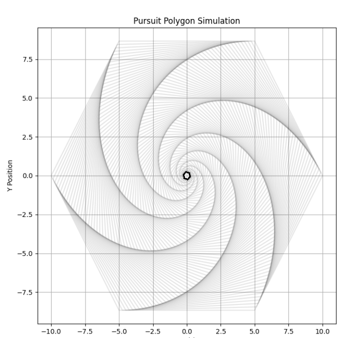
(a) Mice problem simulation, $n = 3$



(b) Mice problem simulation, $n = 4$



(c) Mice problem simulation, $n = 5$



(d) Mice problem simulation, $n = 6$

Figure 1. Mice problem simulations

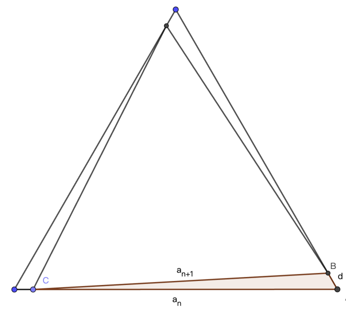


Figure 2. We can use the cosine theorem to find the recursive formula of a sequence.

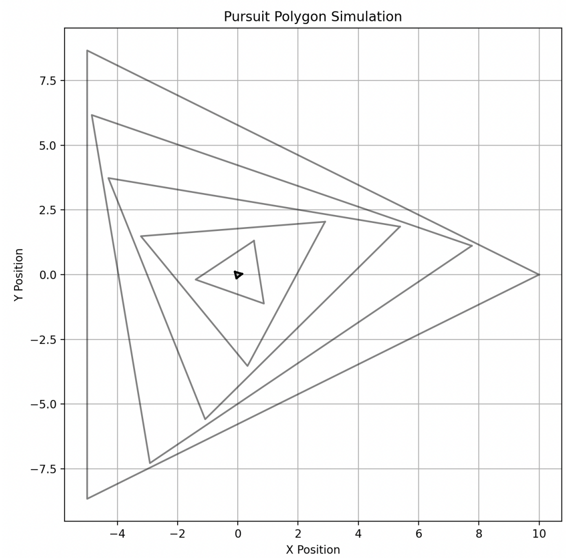


Figure 3. Trajectory of n-gons, n=3

need to use a program to complete the calculation (appendix). The advantage of this method is that it very intuitively displays the process of differentiating and integrating the movement of the mouse.

2.2. Compute paths using similarities

A feature of the Mice problem can help us get the theoretical length of the path faster. It can be noted that the speed of the mouse is unrelated to the shape of the final path. Therefore, the side length of n-gon is unrelated to the final path shape. Therefore, we can conclude that the graphs formed by any two n-gons and the mice action trajectories are similar. Because the trajectory curve and side length are both one-dimensional quantities, we can conclude that the ratio of side lengths is equal to the ratio of trajectory lengths.

Lemma 2.1 (Trajectory similarity lemma) *Assume two regular n-gons have side lengths L_1, L_2, and internal trajectory lengths D_1, D_2 respectively. We have:*

$$\frac{L_1}{L_2} = \frac{D_1}{D_2} \tag{5}$$

We can use the differential method to calculate the trajectory length D by calculating the ratio between a_0 and a_1. In $n = 3$ situation, as Figure 2 shown, in the process from a_0 to a_1, mice has advanced a distance of Δd. Therefore, the new equilateral triangle with a_1 as the side length has a trajectory length $D_2 = D - \Delta d$. According to equation 4,

$$
\begin{aligned}
\frac{a_1}{a_0} &= \frac{D - \Delta d}{D} \\
\frac{\sqrt{a_0^2 + 3\Delta d^2 - 3a_0\Delta d}}{a_0} &= \frac{D - \Delta d}{D} \\
\frac{a_0^2 + 3\Delta d^2 - 3a_0\Delta d}{a_0^2} &= \frac{(D - \Delta d)^2}{D^2} \\
D^2(a_0^2 + 3\Delta d^2 - 3a_0\Delta d) &= a_0^2(D^2 + \Delta d^2 - 2D\Delta d) \\
D^2(3\Delta d^2 - 3a_0\Delta d) &= a_0^2(\Delta d^2 - 2D\Delta d) \\
D^2(3\Delta d - 3a_0) &= a_0^2(\Delta d - 2D) \\
\frac{D^2}{a_0^2} &= \frac{\Delta d - 2D}{3\Delta d - 3a_0} \\
\lim_{\Delta d \to 0} \frac{D^2}{a_0^2} &= \frac{2D}{3a_0} \\
\lim_{\Delta d \to 0} \frac{D}{a_0} &= \frac{2}{3}
\end{aligned}
$$

Here, a_0 is the unit length. By bringing in $a_0 = 1$, we can get $D = \frac{2}{3}$ in $n = 3$ situation. This matches the conclusion in Mice problem trace length theorem 1.2.

For other regular polygons, we only need to use the cosine theorem to adjust the ratio between a_1 and a_0. For n-gon, let $\theta = (1 - \frac{2}{n})\pi$ be the measure of the interior angle of the regular n-gon.

$$
\begin{aligned}
a_1^2 &= \Delta d^2 + (a_0 - \Delta d)^2 - 2\Delta d \cdot (a_0 - \Delta d) \cdot \cos(\theta) \\
a_1^2 &= a_0^2 + (2 + 2\cos(\theta))\Delta d^2 - (2 + 2\cos(\theta))a_0\Delta d
\end{aligned}
$$

Putting this relationship into equation 5, we get:

$$
\begin{aligned}
D^2(2 + 2\cos(\theta))(\Delta d^2 - a_0\Delta d) &= a_0^2(\Delta d^2 - 2D\Delta d) \\
D^2(2 + 2\cos(\theta))(\Delta d - a_0) &= a_0^2(\Delta d - 2D) \\
\frac{D^2}{a_0^2} &= \frac{\Delta d - 2D}{((2 + 2\cos(\theta)))\Delta d - a_0} \\
\lim_{\Delta d \to 0} \frac{D^2}{a_0^2} &= \frac{D}{(1 + \cos(\theta))a_0} \\
\lim_{\Delta d \to 0} \frac{D}{a_0} &= \frac{1}{1 + \cos(\theta)} \\
\lim_{\Delta d \to 0} \frac{D}{a_0} &= \frac{1}{1 + \cos((1 - \frac{2}{n})\pi)} \\
\lim_{\Delta d \to 0} \frac{D}{a_0} &= \frac{1}{1 - \cos(\frac{2\pi}{n})}
\end{aligned}
$$

Input $a_0 = 1, D = \frac{1}{1 - \cos(\frac{2\pi}{n})}$. Thus, we prove the Mice problem trace length theorem 1.2.

3. Conclusion and Further Investigation

Using principles from multivariable calculus, we have successfully proven the trace length of the Mice problem in this study. Through the application of recursive sequence iteration and similarity methods, we derived the length of the trajectory path for any n-gon. As we discover that the mice in pursuit are directly related to the side lengths of the polygons, we formulate a precise mathematical expression for the trace length. By computationally simulating the calculation process, we validated our findings and provided insight into the dynamics of pursuit curves of the polygon.

There are several areas for further investigation. We can enhance our results' generalizability by exploring the behavior of pursuit curves in non-regular n-gons. Additionally, examining optimal pursuit strategies for maximizing or minimizing path lengths in various polygonal configurations could have practical applications in fields such as nature conservation or robotics development.

References

[1] A. Bernhart, "Polygons of pursuit", *Scripta Mathematica*, vol. 24, no. 1, pp. 23–50, 1959.

[2] Math Physics Engineering, *Zeno's mice (ants) problem and the logarithmic spirals*, https://www.youtube.com/watch?v=NdTVvWrD6r0, 2021.

[3] A. H. A. Kalameh, K. B. Komitaki, R. Sharifian, and M. M. Eftekhari, "Investigating the classical problem of pursuit, in two modes", *arXiv preprint arXiv:2309.02471*, 2023.

[4] W. Eric W., *Mice problem*, 2024. [Online]. Available: https://mathworld.wolfram.com/MiceProblem.html.

4. Appendix

```python
# n side Pursuit Curve
import numpy as np
import matplotlib.pyplot as plt


def create_polygon(n_sides, radius=10):
    angles = np.linspace(0, 2 * np.pi, n_sides, endpoint=False)
    return np.column_stack((radius * np.cos(angles), radius * np.sin(angles)))

def pursuit_simulation(vertices, velocity, dt, steps, convergence_threshold=0.01):
    n_sides = len(vertices)
    trajectory = [vertices.copy()]

    for _ in range(steps):
        new_vertices = vertices.copy()
        for i in range(n_sides):
            target_index = (i + 1) % n_sides
            direction = vertices[target_index] - vertices[i]
            norm = np.linalg.norm(direction)
            if norm != 0:
                direction /= norm
            new_vertices[i] += direction * velocity * dt

        vertices = new_vertices
        trajectory.append(vertices.copy())

        if all(np.linalg.norm(vertices[(i + 1) % n_sides] - vertices[i]) < convergence_threshold for i in
                range(n_sides)):
            break

    return np.array(trajectory)


def plot_trajectory(trajectory, title="Pursuit Polygon Simulation"):
    plt.figure(figsize=(8, 8))
    for vertices in trajectory:
        vertices = np.vstack([vertices, vertices[0]])
        plt.plot(vertices[:, 0], vertices[:, 1], 'k-', alpha=0.1)

    plt.title(title)
    plt.grid(True)
    plt.xlabel("X Position")
    plt.ylabel("Y Position")
    plt.show()

n_sides = int(input("Enter the number of sides for the polygon: "))
velocity = 0.5
dt = float(input("Enter the time gap: "))
steps = 1000

vertices = create_polygon(n_sides)
trajectory = pursuit_simulation(vertices, velocity, dt, steps)

plot_trajectory(trajectory)
```

Code 1. Basic model Python code