Putnam Problem 1992 A6: Probability of Center Inside Tetrahedron

Dimitri Chrysafis

June 11, 2024

1 Introduction

The Putnam problem 1992 A6 states: "Four points are chosen independently and at random on the surface of a sphere (using the uniform distribution). What is the probability that the center of the sphere lies inside the resulting tetrahedron?"

It can be proven by math that the answer is 1/8, or 0.125. In this text, we will present a computational approach to solve this problem. We will generate random points on a sphere, compute the resulting tetrahedron, visualize it, and determine the probability of the center lying inside the tetrahedron.

SOLUTION (NOT MINE) https://dqlin.xyz/post/2018/09/01/putnam/

2 Examples

Here are examples showing both cases: when the center of the sphere lies inside and outside the tetrahedron. Generated using plotly:

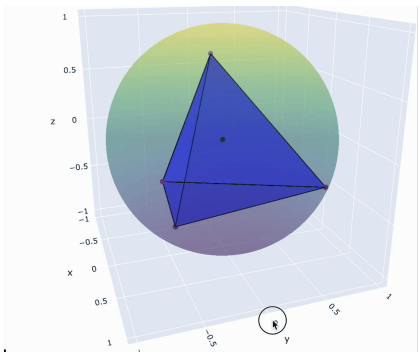

Figure 1: Center inside the tetrahedron.

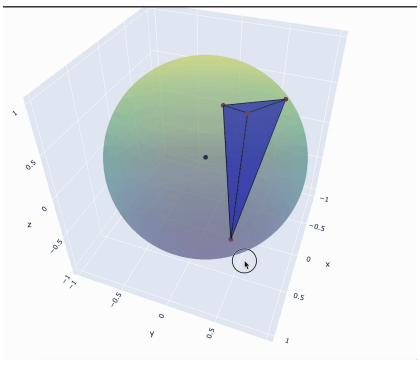

Figure 2: Center outside the tetrahedron.

3 Methodology

To solve the problem, we will follow these steps:

1. Generate four random points on the surface of a sphere.
2. Compute the convex hull of the points to form a tetrahedron rather than just four dots existing.
3. Check if the center of the sphere lies inside the tetrahedron.
4. Repeat the process for multiple simulations to calculate the probability.

4 Implementation Details

We implemented the solution in Python using the following libraries:

- **NumPy**: For numerical computations.
- **SciPy**: For computing the convex hull.
- **Matplotlib**: Running the 5000 image simulation to get the solution.

5 Part 1: Simulation

5.1 Spherical Coordinates and Random Point Generation

Spherical coordinates extend the concept of polar coordinates to represent points on a sphere in three-dimensional space.

In spherical coordinates:

- r represents the distance from the origin.
- θ is the angle from the positive x-axis in the xy-plane.
- ϕ is the angle from the positive z-axis.

To generate random points uniformly distributed on the surface of a sphere:

1. Generate a random azimuthal angle ϕ uniformly between 0 and 2π.

```
phi = np.random.uniform(0, 2*np.pi, num_points)
```

2. Generate a random polar angle θ by sampling $\cos(\theta)$ uniformly between -1 and 1, then calculate $\theta = \arccos(\cos(\theta))$.

```
cos_theta = np.random.uniform(-1, 1, num_points)
theta = np.arccos(cos_theta)
```

3. Convert spherical coordinates to Cartesian coordinates (x, y, z).

```
    points /= np.linalg.norm(points, axis=1)[:, np.newaxis]
```

This method provides us with random points uniformly distributed on the surface of the sphere.

5.2 Checking If Center Is Inside

To determine if the center of the sphere lies inside the tetrahedron, we utilize the convex hull of the generated points. Here's how the process works:

First, let's examine the `isInCenter` function:

```
def isInCenter(points):
    hull = ConvexHull(points)
    center = np.array([0, 0, 0])
    equations = hull.equations
    signs = np.sign(np.dot(equations[:, :-1], center) + equations[:, -1])
    return np.all(signs <= 0) or np.all(signs >= 0)
```

Here's how it works:

1. We compute the convex hull of the points using `scipy.spatial.ConvexHull`.
2. Next, we obtain the equations of the planes defining the convex hull.
3. Then, we calculate the signed distances from the center of the sphere to these planes.
4. If all signs are less than or equal to zero, or all signs are greater than or equal to zero, it indicates that the center lies inside the convex hull (tetrahedron).

This method works because if the center lies inside the tetrahedron, it will have the same sign for all distances to the planes formed by the tetrahedron's faces. Otherwise, if it lies outside, the signs will differ.

5.3 Generating the Images and Multithreading

This function generates a 3D plot of the tetrahedron and the sphere with annotations indicating the result of each simulation iteration. It includes:

- Wireframe of the sphere.
- Convex hull of the points forming the tetrahedron.
- Scatter plot of the random points.
- Annotation for the center of the sphere.

- Annotation showing the current index, probability, and whether the center is inside or outside the tetrahedron.
- Table showing the coordinates of each point.
- Finally, it saves the plot as an image file for later reference and conversion into a video.

We use the multiprocessing module in Python to divide the simulation workload among multiple CPU cores. Here's the code snippet:

```
import multiprocessing

def simulate_single(max_size_bytes):
    # Simulation code
    pass

def compileSize(max_size_bytes=1000000000):
    num_cores = multiprocessing.cpu_count()
    pool = multiprocessing.Pool(processes=num_cores)
    results = pool.map(simulate_single, [max_size_bytes] * num_cores)
    pool.close()
    pool.join()

    probability = sum(results) / len(results)

    print(f'Final Result {probability:.6f}')
```

The `compileSize` function creates a pool of processes, maps the `simulate_single` function to each process, and runs them concurrently. This maximizes CPU utilization and speeds up the simulation.

6 Conclusion/Data

When running this code in python, I could only get to case 33800000.png which will be shown below. The c++ did not have to render images and only captured instances every million or so, as can be seen in the output. It is unknown why the probabilities were so different.

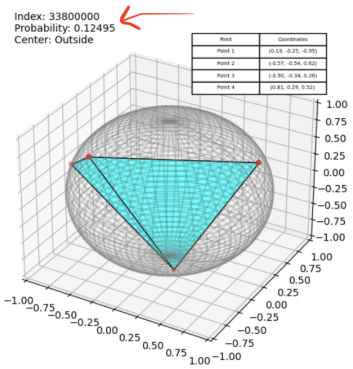

Figure 3: Python Version (Higher accuracy)

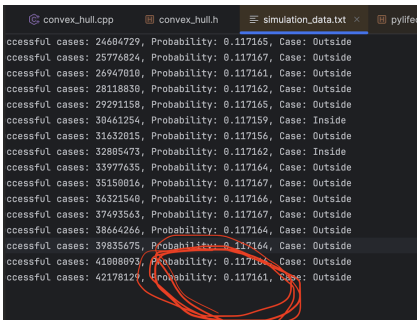

Figure 4: C++ Version (Much faster yet off by 0.008 ish

7 Extras

In addition to the main simulation, I explored two additional visualization methods: one using Plotly for interactive 3D visualization and another using Pygame to render images into a video.

7.1 Plotly 3D Interactive Playground

For interactive visualization, I utilized Plotly, a powerful Python library. Here's how I implemented it:

- Generated random points on the surface of a sphere using polar coordinates.
- Rendered the sphere and the convex hull formed by these points using Plotly.
- Created an interactive 3D playground where the scene could be rotated, zoomed, and explored dynamically. (on page below) (compile code to actually play with it)

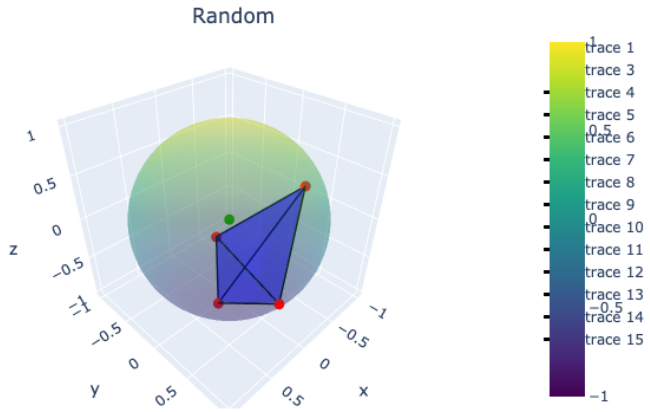

Figure 5: Python Version (Higher accuracy)

This was mostly for me to experiment with the sphere stuff.

7.2 Pygame Image Renderer

To create a video of the simulation, I used Pygame, a popular library for game development. Here's what I did:

- generated random points on the sphere's surface and calculated the convex hull.
- saved each visualization as an image file in a folder.
- had Pygame to render these images sequentially, creating a video of the evolving scene.
- https://github.com/DimitriChrysafis/Putnam1992A6/blob/main/input.gif
- THE GIF IT GENERATES CAN BE FOUND ABOVE

the approach gave the for the generation of a video that visualized the simulation process step by step

8 Code

The code for this project is available on GitHub: `https://github.com/DimitriChrysafis/Putnam1992A6/tre`

9 Final Simulation Youtube

Click here to watch the video