Vamshi Jandhyala

Beside the Point

✠

November 2024

A geometric-probability puzzle from Jane Street.

Problem

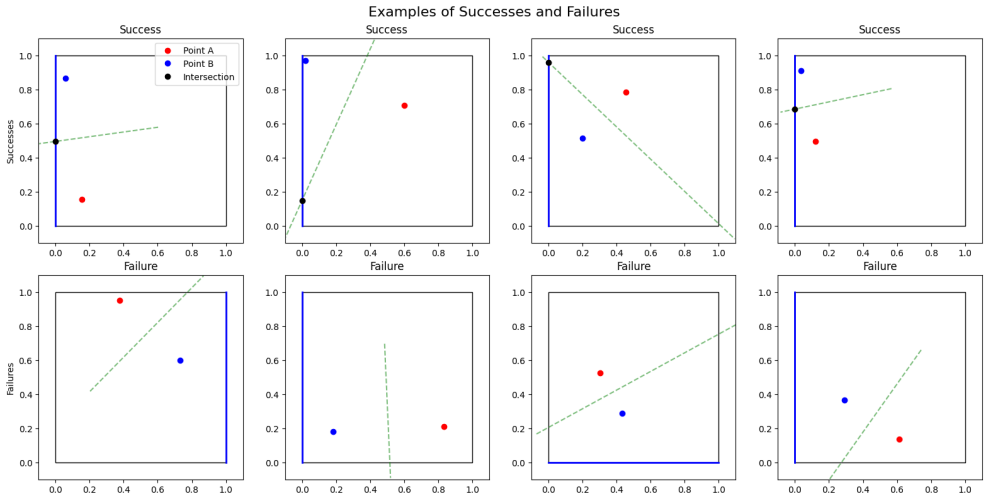
Two random points, one red and one blue, are chosen uniformly and independently from the interior of a square. To ten decimal places, what is the probability that there exists a point on the side of the square closest to the blue point that is equidistant to both the blue point and the red point?

Computational solution

We use Monte Carlo simulation to estimate the probability. From elementary geometry, the point on the blue-nearest side which is equidistant from the two points must lie on the perpendicular bisector of the segment AB. The procedure:

- Choose two points uniformly at random in the unit square; label one red, one blue.
- Identify the side of the square nearest to the blue point.
- Find the equation of the perpendicular bisector of segment AB and test whether it intersects the blue-nearest side.
- Repeat a million times and count the successful intersections.

The simulation gives the required probability as **0.4917**, slightly under one half.



Python code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from typing import Tuple

def check_intersection(A: np.ndarray, B: np.ndarray
                       ) -> Tuple[bool, str, np.ndarray]:
    distances = [
        (abs(B[1] - 0), 'bottom'),
        (abs(B[1] - 1), 'top'),
        (abs(B[0] - 0), 'left'),
        (abs(B[0] - 1), 'right'),
    ]
    nearest_side = min(distances, key=lambda x: x[0])[1]

    midpoint = (A + B) / 2
    direction = B - A
    perp_vector = np.array([-direction[1], direction[0]])

    length = np.linalg.norm(perp_vector)
    if length == 0:
        return False, nearest_side, None
    perp_vector = perp_vector / length

    intersection = None
    success = False

    if nearest_side == 'bottom':
        if perp_vector[1] != 0:
            t = (0 - midpoint[1]) / perp_vector[1]
            x = midpoint[0] + t * perp_vector[0]
            intersection = np.array([x, 0])
            success = 0 <= x <= 1
    elif nearest_side == 'top':
        if perp_vector[1] != 0:
            t = (1 - midpoint[1]) / perp_vector[1]
            x = midpoint[0] + t * perp_vector[0]
```

```
            intersection = np.array([x, 1])
            success = 0 <= x <= 1
    elif nearest_side == 'left':
        if perp_vector[0] != 0:
            t = (0 - midpoint[0]) / perp_vector[0]
            y = midpoint[1] + t * perp_vector[1]
            intersection = np.array([0, y])
            success = 0 <= y <= 1
    else: # right
        if perp_vector[0] != 0:
            t = (1 - midpoint[0]) / perp_vector[0]
            y = midpoint[1] + t * perp_vector[1]
            intersection = np.array([1, y])
            success = 0 <= y <= 1

    return success, nearest_side, intersection

np.random.seed(42)
num_trials = 1_000_000
all_points = np.random.random((2, num_trials, 2))
A_points, B_points = all_points[0], all_points[1]

total_successes = sum(check_intersection(A_points[i], B_points[i])[0]
                      for i in range(num_trials))
probability = total_successes / num_trials
print(f"Probability: {probability:.6f}")
```