Project 1

Estimating the value of Pi using Monte Carlo Integration

In belated honor of π day, this project will introduce you to a novel way to write a computer program to estimate the value of π. We will do this using little more than a random number generator.

Methodology

Monte Carlo Integration

Consider the two rectangles in Figure 1. I want to randomly select a point $< x, y >$ from anywhere within the outer rectangle (including the inside of the inner rectangle), so that every point is equally likely to be selected. What is the probability that this point will *also* lie within the inner rectangle? It turns out that it depends on the ratio of the area of the two rectangles:

$$P(< x, y > \text{inside inner rectangle}) = \frac{\text{area of inner}}{\text{area of outer}}$$

Our intuition supports this. If the inner rectangle is very small compared to the outer one, then it is highly unlikely that the randomly selected point will also appear within the inner rectangle. Conversely, if the inner rectangle is nearly the same size as the outer one, it is very likely.

This logic is also useful the other way around. Suppose I know the area of the outer rectangle but not the area of the rectangle within. If I randomly select a point from anywhere within the outer rectangle, the probability that the point is also drawn from the inner rectangle is $P = A_{\text{inner}}/A_{\text{outer}}$. Even without knowing A_{inner}, I can estimate P by selecting not one but many points. Suppose I generate 100 different random $< x, y >$ points, and 75 of them originate from the inner rectangle. This tells me that $P(< x, y > \text{inside inner rectangle}) = 75/100 = 0.75$. Since $P = A_{\text{inner}}/A_{\text{outer}}$, I can therefore say that the area of the inner rectangle is 0.75 times the area of the outer one, which I know. I have used a random number generator to find the area of the shape. This is an example of a computational technique known as Monte Carlo integration (in reference to the Monte Carlo casino in Monaco).

This example is rather trivial, since it is not hard to just calculate the area of the inner rectangle. The beauty of this technique is that it works for *any* shape. Give me some weird unknown shape, and I can place it within a rectangle and use random numbers to estimate its area (Figure 2).

An obvious limitation of this method is that we can only ever *estimate* $P = A_{\text{inner}}/A_{\text{outer}}$; we cannot use random numbers to calculate it exactly. Therefore, the value we find for A_{inner} is not exact. If you generate N many points, then the estimate of A_{inner} is likely to be within $\frac{A_{\text{outer}}}{\sqrt{N}}$ of the true value.

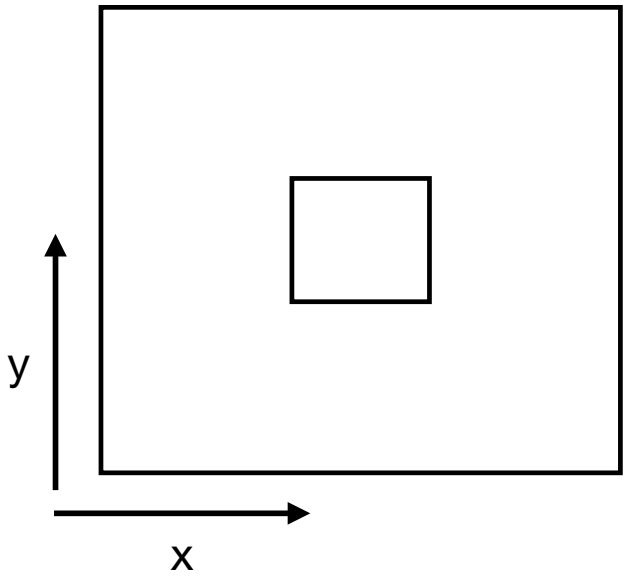

Figure 1:

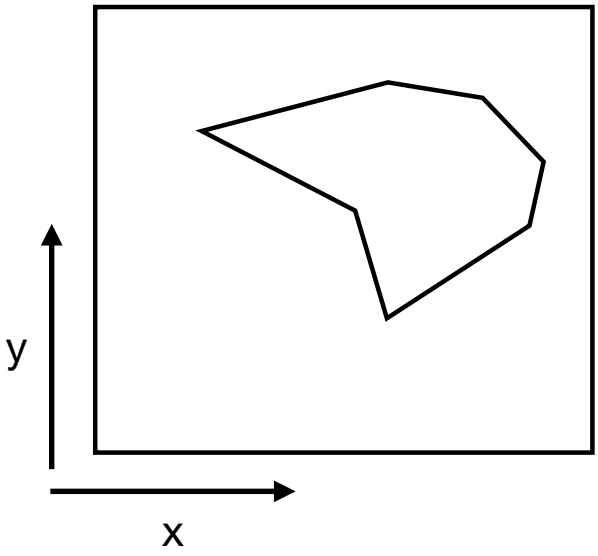

Figure 2:

Project Description

The goal of this project is use the Monte Carlo integration technique discussed in the previous section to estimate the area of a circle. If you then divide this area by the square of the radius of the circle, you will have an estimate of π. You will write code to do this several repeatedly while increasing the number of random points to explore how your estimate gets better with more random points.

To estimate the area, use the `random.uniform` function to generate a number of $< x, y >$ points drawn from a rectangle (or square). Then write code to determine what fraction of those points lie within the circle of radius r. Multiply this fraction by the area of the encompassing rectangle to estimate the area of the circle.

Details

Your program should do the following:

- Write a function called `inside_circle(x,y,radius)` which returns `True` if the point `x,y` is located within the circle of radius `radius` (centered at the origin), and `False` otherwise
- Write a function called `estimate_area(radius,N,l,w)` which generates N-many random points, drawn from a rectangle with width `l` and width `w`. The function should call `inside_circle(x,y,radius)` on each point to determine if it is within the circle, and then estimate the area as discussed in the previous section. The function should return both the area and the quantity $\Delta A = \frac{lw}{\sqrt{N}}$, which is the uncertainty of your area estimate
- Finally, write the function `estimate_pi(N,radius,l,w)` which calls `estimate_area(radius,N,l,w)` and returns two things: a) the area divided by radius squared, and b) $\Delta A/r^2$, the approximate uncertainty on your estimate of π.
- In `main()`, ask the user which mode they want the program to run
  - If the first mode is chosen, the program will then ask the user how many points to use for the estimation, and then use this value for N to estimate pi and print the result and uncertainty. Note that the precision of your answer is limited to $1/N$; that is: the number of digits in your answer should equal the number of digits in N. Python's built in `round` function can help you with this.
  - If the second mode is chosen, the program will estimate π for several different values of N and then plot π vs N. The user will provide three numbers: N_{min}, N_{max}, and n, the number of points to include on the plot. Your program will estimate π at N_{min} and N_{max}, and $n-2$ many points in between, so that the number of points on the graph is n.