PROJECT 1: ESTIMATING π

Edoardo Piovesana

Abstract

Abstract. The scope of the present project is to illustrate how Monte Carlo simulation can be used to estimate the value of π.

1. Introduction

This code is a simulation of a method to estimate the value of π using Monte Carlo methods. The simulation generates random points within a square and checks how many of them fall inside a circle inscribed in the square. By comparing the number of points inside the circle with the total number of generated points, we can estimate the value of π.

2. Code

In this section there is the used procedure to estimate the value of π using the Python language.

2.1. Import the libraries

The first thing to do is to import the necessary libraries:

```
import numpy as np
import matplotlib.pyplot as plt
```

2.2. Generate the Poisson process

The function *generate_poisson_process(λ, T)* generates a Poisson process of points within a square of length T. In particular this function:

- Takes two parameters λ and T.
  - λ: the intensity parameter of the Poisson process, which determines the expected number of points per unit area
  - T: the length of the square where the points are generated.
- Generates the number of points, N, in the Poisson process, using:
  - *np.random.poisson()*: generates a Poisson distributed random variable
  - λ * T**2: expected number of points in a square of length T, computed as the product of the intensity parameter and the area of the square.
- Generates two arrays of N random numbers using *np.random.uniform()* function. These arrays represent the *x-coordinates* and *y-coordinates* of the N points in the Poisson process, respectively. So, *np.random.uniform(0, T, N)* generates a random array of N numbers uniformly distributed between *0* and T.

- Finally, this function returns the *x-coordinates* and *y-coordinates* of the points as two arrays X and Y, respectively.

Here's the code:

```
def generate_poisson_process(λ, T):
    N = np.random.poisson(λ * T**2)
    X = np.random.uniform(0, T, N)
    Y = np.random.uniform(0, T, N)
    return X, Y
```

2.3. Estimates π

The *estimate_pi(X, Y, circle_center, circle_radius)* function estimates the value of π using Monte Carlo simulation.

- This function takes four parameters:
  - X and Y: are the arrays containing the *x-coordinates* and *y-coordinates* of the random points generated in the Poisson process
  - *circle_center*: is a tuple containing the (x,y) coordinates of the center of a circle
  - *circle_radius*: is the radius of the circle.
- The function first checks if the arrays X and Y are empty. If either one is empty, then the function returns *NaN* to indicate that the estimate cannot be computed.
- Then determines which points in the Poisson process fall inside the circle of radius *circle_radius* centered at *circle_center*. It does this by computing the Euclidean distance between each point *(x,y)* in the Poisson process and the circle center *(cx,cy)*. If the distance is less than the circle radius, then the point is inside the circle.
- Then, this function computes the estimate of π as the ratio of the number of points inside the circle to the total number of points generated: *4 * sum(points_in_circle) / len(X)*.
  - *points_in_circle*: Boolean array that is True for points inside the circle and False for points outside the circle.
  - *sum(points_in_circle)*: total number of points inside the circle.
  - *len(X)*: the total number of points generated in the Poisson process.
- Finally, the function returns the estimate of π.

Here's the code:

```
def estimate_pi(X, Y, circle_center, circle_radius):
    if len(X) == 0:
        return np.nan
    points_in_circle = np.sqrt((X - circle_center[0])**2
        + (Y - circle_center[1])**2) < circle_radius
    return 4 * np.sum(points_in_circle) / len(X)
```

2.4. Run Simulation

The *run_simulation* function is responsible for running a Monte Carlo simulation to estimate the value of π using the *generate_poisson_process* and *estimate_pi* functions. This function takes three parameters: λ, T and M (the number of times to run the simulation).

- Firstly, *run_simulation* initializes an array called *pi_estimates* with length M to store the M estimates of π.

- Then, the function loops M times and does the following for each iteration:
  - It calls the *generate_poisson_process* function with the λ and T parameters to generate a Poisson process of N points within the square domain of side T.
  - It then calls the *estimate_pi* function with the X, Y, *circle_center*, and *circle_radius* parameters to estimate the value of π using the generated Poisson process.
  - It stores the estimate of π in the *pi_estimates* array.
- Once the loop is complete, the function returns the array *pi_estimates* containing the M estimates of π.

Here's the code:

```
def run_simulation(λ, T, M):
    pi_estimates = np.zeros(M)
    for i in range(M):
        X, Y = generate_poisson_process(λ, T)
        pi_estimates[i] = estimate_pi(X, Y, [1/2, 1/2], 1/2)
    return pi_estimates
```

2.5. Setting the parameters and run the simulation

At this point, there is the initialisation of the parameter T (the length of the side of the square) and of the parameter M (number of simulation to run). Then, there is the initialisation of the list *lambdas* that contains the values 5^k for k from 1 to 10.

```
pi_estimates = [run_simulation(λ, T, M) for λ in lambdas]
```

The code above creates a list, *pi_estimates* that will store the estimates of π generated by the simulation for each value of λ (initially this list is empty).

3. Plots

3.1. π Estimates for Different Values of Lambda

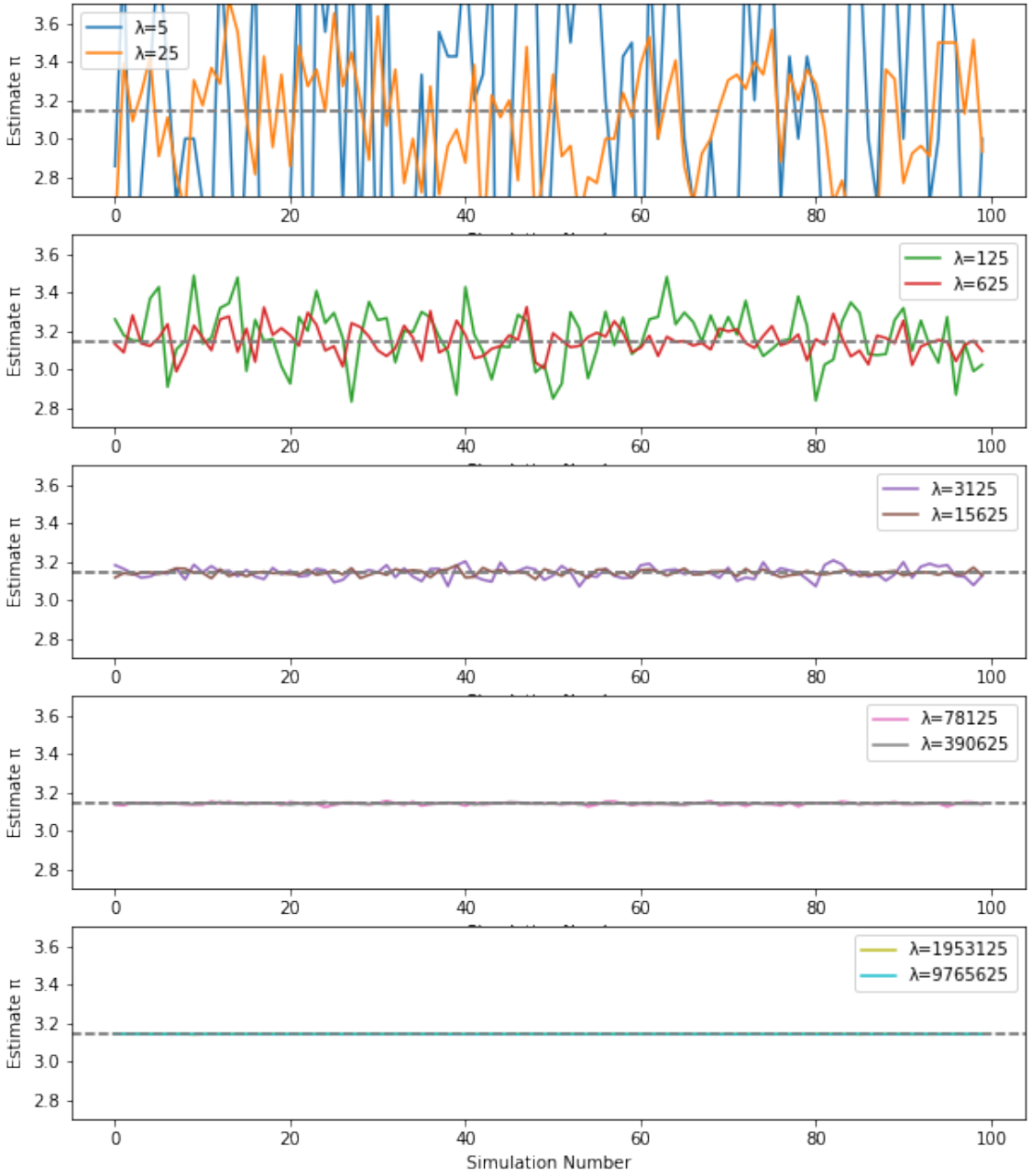

Figure 1. Estimates of π for different values of λ

3.2. Standard Deviation of π estimates and λ

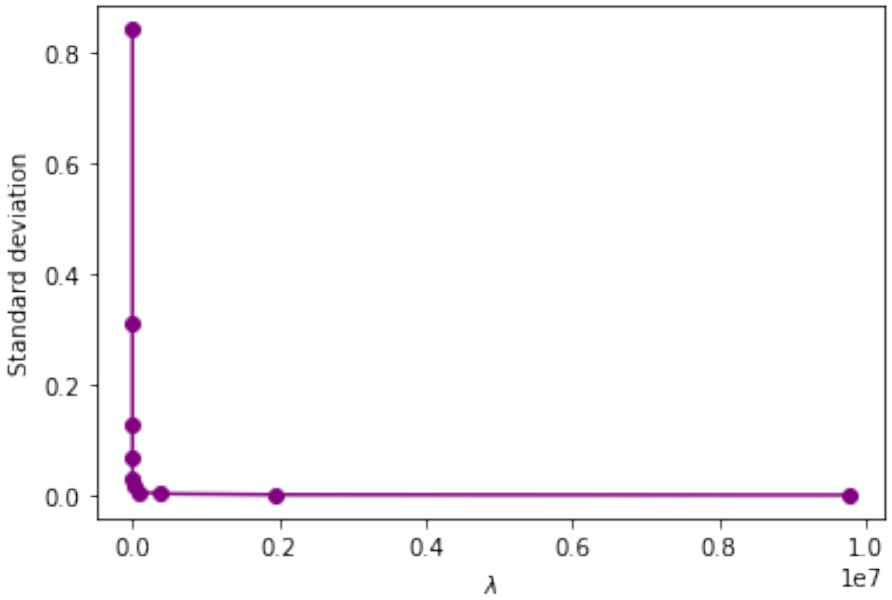

Figure 2. Standard Deviation of π estimates obtained from a simulation for different values of λ, ranging from 5^1 to 5^{10}.

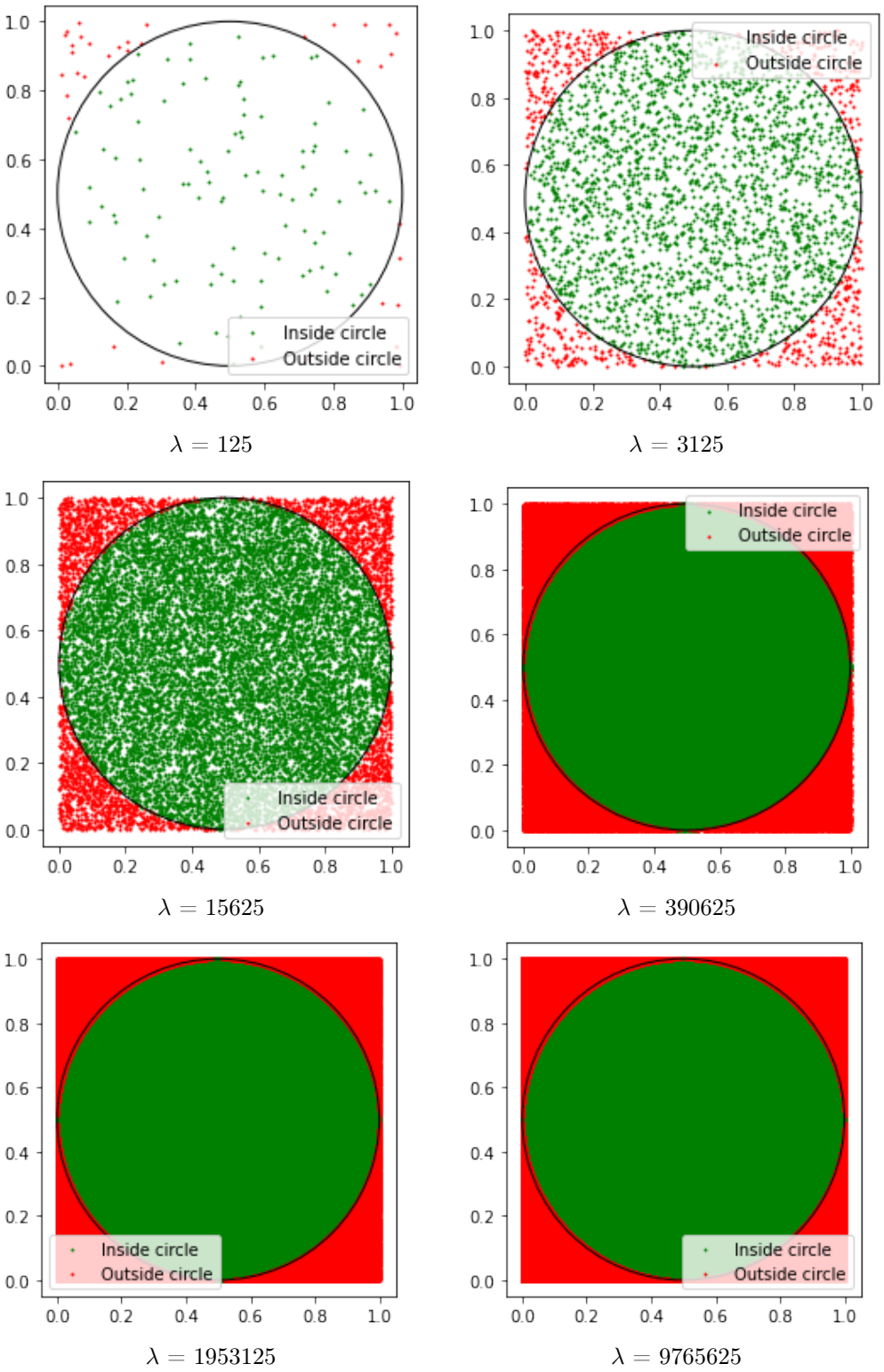

Figure 3. Visual representation of the points in and outside the circle