Monte Carlo Integraion

Rebecca Corley[†]

[1]University of North Georgia, Department of Physics and Astronomy

1. MC INTEGRATION

In this project, we were to use Monte Carlo methods to calculate the area under the curve of the function

$$f(x) = xsin^2(x) \tag{1}$$

within a range of $[0, 2\pi]$. By calculating the area under the curve of equation (1), we are integrating the function.

1.1 The Code

Fig. 1 shows the input code used for estimating the area under the curve of function (1).
The code is structured as follows: Import necessary libraries and define variables and empty arrays that will later be used. The most important part of the code is the for loop. The for loop creates a random number generator for 10000 x-values between 0 and 2π, and the y-values between 0 and 4.8. The domain of the y-values were chose as such because we needed a value just above the local maximum of the function. The if statement inside the for loop says for any y values that are less than or equal to function (1) then send the x- and y-values to a list and increase the counter. Then all other values above the curve are sent to a separate designated list and added to another counter.
We then estimate the area by dividing the number of points under the curve over the total number of points in the area of the square to then be multiplied by the length times width of the total square. The output of the code values can be seen in Fig. 2. The plot is generated for the curve as seen in Fig. 3.

2. SUMMARY AND CONCLUSIONS

The plot generated using Monte Carlo (MC) integration methods is what we expect plotting function (1) in range of 0 to 2π. Solving the equation analytically gives a solution of 9.86960.
The two previous experiments using MC methods solved problems that we can calculate analytically.
Monte Carlo methods are most appropriately applied to problems that cannot be solved analytically or even do not yet have solutions and require numerical approximations. In particular, MC becomes useful for higher dimensional integration or problems with large numbers of n degrees of freedom. This probably becomes especially useful in many fields using statistical analysis, including physics, biology, economics, etc. Perhaps this could be useful for calculating the volume of some complex object, which may be useful for engineers. The first thought that came to mind was having to calculate the volume of a rocket ship in order to know how much breathable oxygen to provide astronauts.
MC methods seem incredibly useful, however, the more I learn about the techniques, I am curious if there is a limit to what MC can do and how we find, if any exist, alternative solutions.

Further author information: E-mail: rlcorl0436@ung.edu

```python
#This code estimates the area under the curve of a function using Monte Carlo methods
#python libraries used for this code
import numpy as np
import matplotlib.pyplot as plt
import random
import math
#start_time = time.time()
N_iterations = 100000 #define n number of iterations - higher number returns closer approximation
Under_curve_x = [] #define empty array called circle to later store (x,y) variables/dots inside circle radius
Under_curve_y = []
In_domain_x = []
In_domain_y = []
area_inside_curve = 0
outside_curve = 0
#create random number generator
for i in range(N_iterations):
    #using random.uniform because it returns random floating point numbers within defined range
    x = random.uniform(0,(2*math.pi)) #random variables for x in range 0 2pi
    y = random.uniform(0, 4.8) #random variables for y in range 0, 4.8
    #adding x,y points to array and counter if pass selection criteria
    if (y) <= (x*math.sin(x))*(math.sin(x)):
       Under_curve_x.append(x)
       Under_curve_y.append(y)
       area_inside_curve += 1.0
    else:
        In_domain_x.append(x)
        In_domain_y.append(y)
    outside_curve += 1.0
Area_estimate = ((float(area_inside_curve)/float(outside_curve)) * 2 * np.pi * 4.8)
print(Area_estimate)
Area = np.pi * np.pi
print(Area)
error = Area_estimate - Area
print(error)
#Plot dots inside the circle
plt.figure(figsize=(5, 5)) #sets size of figure
#plot under curve
plt.scatter(Under_curve_x, Under_curve_y, marker="*", alpha=1.0, color = "purple", label="Inside Area Points")
#plot outside curve
plt.scatter(In_domain_x, In_domain_y, marker="*", alpha=1.0, color = "blue", label="Outside Area Points")
plt.xlabel("Random X-Values")
plt.ylabel("Random Y-Values")
plt.legend(loc="upper left")
#plt.savefig() #set path to save figure
```

Figure 1. Figure showing code used for MC integration assignment.

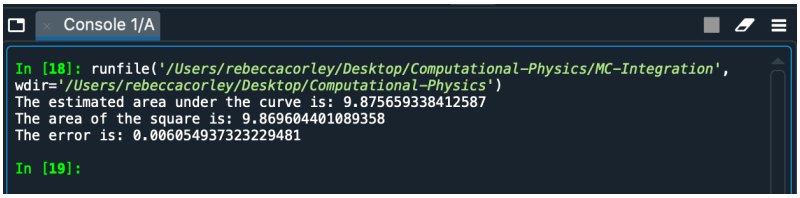

Figure 2. Figure showing the output of the code, including the estimated area, calculated area, and the error.

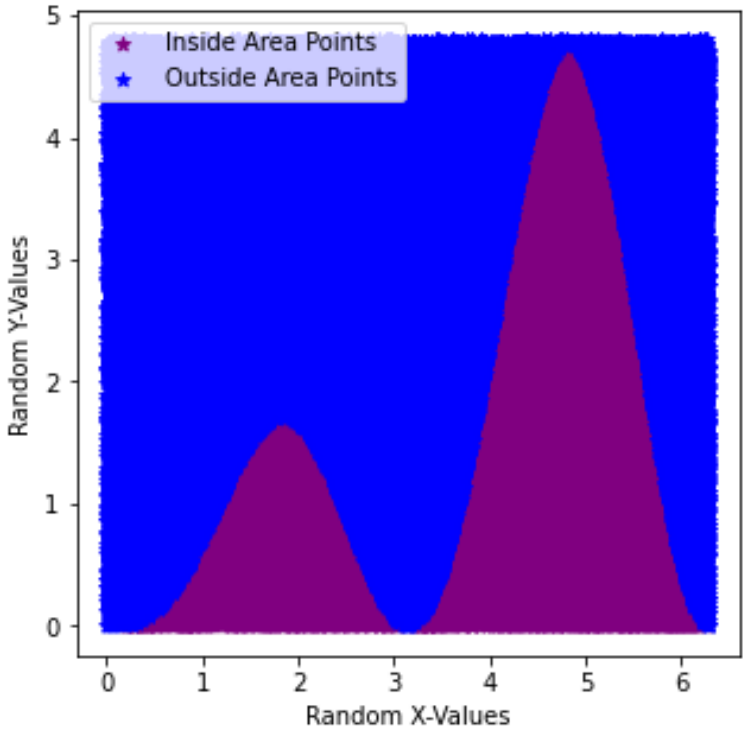

Figure 3. This plot shows the randomly generated (x,y) values that are above the function (blue) and values under the curve (purple).