Monte Carlo Simulation

Lab Session 1 - Integration

Objective:
To explore and apply the Monte Carlo simulation for numerical integration by evaluating its effectiveness in approximating definite integrals, especially in multidimensional cases where analytical solutions are impractical. This session will involve implementing Monte Carlo methods to calculate integrals, analyzing the accuracy of the results, and comparing the numerical approximation to exact solutions in simpler cases.

Task 1: Numerical Solution

Goal: To define the function `f(x) = x^2` and set up the parameters for numerical integration over a defined interval.

Steps:
1. Choose a PRNG Algorithm:
Use a built-in PRNG in your preferred programming language (e.g., `rand()` in C, `Random` class in Java, `random` module in Python).

2. Generate Numbers:
Initialize the PRNG with a specific seed (e.g., `seed = 2`)

3. Define the function and set integration limits
- The function to be integrated is `f(x)= x^2`
- The lower limit is `a=0.0` and the upper limit is `b=3.0`.
- Generate `1000000` steps.
- Initialize Lists for Plotting

Task 2: Min-max Detection

Goal: To detect the minimum and maximum values of the function `f(x) = x^2` within the integration range [0, 3].

Steps:
1. Find `ymin` and `ymax`:
Set initial `ymin` and `ymax`, begin by calculating the function value and set both `ymin` and `ymax` to this value.

2. Iterate and update
Loop through the x-values from a to b in steps defined by `NumSteps`. At each step, compute `f(x)`. update `ymin` if the current value is lower, and update `ymax` if it's higher than the previous maximum.

Task 3: Monte Carlo Method

Goal: To perform numerical integration using the Monte Carlo method by generating random points and counting how many points fall under the curve.

Steps:
1. Calculating the Area of a Square:
After getting `ymin` and `ymax`, calculate the area of the square bounding the function using the formula: `A = (b - a) * (ymax - ymin)`

2. Monte Carlo Parameter Initialisation:
Set the number of random points `N=1000000` and initialize the variable `M` to count the number of points under the curve.

3. Generate Random Points:
For each iteration, generate a random value for `x` between `a` and `b`, and `y` between `ymin` and `ymax`.

4. Check if the Point is Under the Curve:
Check if the random point falls under the curve `f(x)= x^2` and store it in counter `M`.

5. Calculate Numerical Integrals:
Calculate the integral by using the formula: `NumericalIntegral = M/N * A`

Task 4: Visualization

Goal: To create a visual representation of the Monte Carlo simulation showing the function curve and the random points.

Steps:

1. Generate Data for Function Curves:

Use `numpy.linspace()` to generate x-values from `a` to `b`, and calculate the corresponding y-values using the function `f(x)= x^2`.

2. Plot the Function Curve and random points:

-Plot the curve of the function `f(x)=x^2` in red color to show the shape of the curve.

-Plot random points that fall under the curve in blue, and points outside the curve in yellow.

Lab Report:

After completing the tasks, students should compile their observations and analysis into a lab report. The report should include:

1. Introduction:
Briefly explain the importance of numerical integration and why the Monte Carlo method is used for solving complex integrals.
Poin: sesuaikan dengan materi yang telah diterima.
2. Methods:
-Describe the function f(x)=x^2, integration limits, and the use of random numbers.
-Outline the steps followed for min-max detection, generating random points, and applying the Monte Carlo method.
-Mention tools used for visualization.
3. Results: Present the results of each task, including charts, statistical test outcomes, and any identified patterns.
4. Discussion: Discuss the accuracy of the results and how it changes with more random points.
5. Conclusion: Summarize key findings.Provide recommendations for improving simulation accuracy, such as increasing random points or optimizing the random number generation process.