

Monte Carlo Integration Technique

In addition to Gaussian Quadrature, there is another technique that is widely used for evaluating complicated integrals numerically. The technique discussed here is known as the *Monte Carlo method*. This relies on drawing n uniformly distributed points in the interval $[a, b]$ and determining the average value of the function that is to be integrated over that interval. The algorithm is as follows:

1. Generate n points x_1, x_2, \dots, x_n from a uniform distribution on the interval $[a, b]$.
2. Find average value of the function. That is, find

$$\hat{f} = \frac{1}{n} \sum_{i=1}^n f(x_i)$$

3. Approximate the integral

$$\int_a^b f(x) dx \approx (b - a) \hat{f}$$

Below is the C++ implementation of this algorithm.

```
//Approximation of integral using the Monte Calo Method

#include<iostream>
#include<math.h>
#include <assert.h>

double random_uniform_0_1()
{
    return double(rand())/double(RAND_MAX);
}

using namespace std;

double (*func)(double); //integrand

double MonteCarloIntegral1D(double f, double a, double b, int n)
{
    double u, x;
    double sum = 0.0;
    double avgValue;

    for (int i = 1; i <= n; i++)
    {
```

```

u = random_uniform_0_1();
x = a + (b - a)*u;
sum += func(x);
avgValue = sum/n;
}

double answer = (b-a)*avgValue;

    return answer;
}

```

This method can also be used to approximate two dimensional integrals. The algorithm is as follows:

1. Generate n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ from a uniform distribution on the interval $[a, b] \times [c, d]$.
2. Find average value of the function. That is, find

$$\hat{f} = \frac{1}{n} \sum_{i=1}^n f(x_i, y_i)$$

3. Approximate the integral

$$\int_a^b \int_c^d f(x, y) dx \approx (b-a)(d-c)\hat{f}$$

Below is the C++ implementation of this algorithm.

```

//Approximation of double integral using the Monte Calo Method

#include<iostream>
#include<math.h>
#include <assert.h>

double random_uniform_0_1()
{
    return double(rand())/double(RAND_MAX);
}

using namespace std;

double (*func)(double, double); //integrand

```

```

double MonteCarloIntegral2D(double f, double a, double b, double c, double d, int n)
{
    double u, v, x, y;
    double sum = 0.0;
    double avgValue;

    for (int i = 1; i <= n; i++)
    {
        u = random_uniform_0_1();
        x = a + (b - a)*u;
        v = random_uniform_0_1();
        y = c + (d - c)*v;
        sum += func(x, y);
        avgValue = sum/n;
    }

    double answer = (b-a)*(d-c)*avgValue;

    return answer;
}

```

It can also be used to approximate three dimensional integrals. The algorithm is as follows:

1. Generate n points $(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n)$ from a uniform distribution on the interval $[a, b] \times [c, d] \times [g, h]$.
2. Find average value of the function. That is, find

$$\hat{f} = \frac{1}{n} \sum_{i=1}^n f(x_i, y_i, z_i)$$

3. Approximate the integral

$$\int_a^b \int_c^d \int_g^h f(x, y, z) dx \approx (b-a)(d-c)(h-g)\hat{f}$$

Below is the C++ implementation of this algorithm.

```

//Approximation of triple integral using the Monte Calo Method

#include<iostream>
#include<math.h>
#include <assert.h>

```

```

double random_uniform_0_1()
{
    return double(rand())/double(RAND_MAX);
}

using namespace std;

double (*func)(double, double, double); //integrand

double MonteCarloIntegral3D(double f, double a, double b, double c, double d, double g, double h, double n)
{
    double u, x, y, z;
    double sum = 0.0;
    double avgValue;

    for (int i = 1; i <= n; i++)
    {
        u = random_uniform_0_1();
        x = a + (b - a)*u;
        y = c + (d - c)*u;
        z = g + (h - g)*u;
        sum += func(x, y, z);
        avgValue = sum/n;
    }

    double answer = (b-a)*(d-c)*(h-g)*avgValue;

    return answer;
}

```