Project 1A

Integration of function $\sinh(\cosh(x))$ on the interval $[-4.0,3.0]$.
Method: Monte-Carlo

Abstract

This work is an introduction to classical Monte Carlo method for numerical integration. A short description on the generating of pseudo-random numbers is given. Also included full code listing of working program on C++ as example for implementation algorithm on computer languages.

Introduction

A Monte-Carlo method is any method that solves a problem by generating random numbers and observing fraction of the numbers obeying some property or properties. The term firstly used by Stanslaw Ulam in 1946 and is commonly used in physics and other fields that require solution for problems that are impractical or impossible to solve using traditional analytical or numerical methods.

Generating random sequences

A key point in Monte-Carlo method is generating random numbers. This is why we are looking for a sequence of independent numbers with a specified distribution. In order to be truly useful, such sequences of random numbers must normally be uniformly distributed within some interval (typically $[0,1) \in \Re$).
Truly random sources can be found by measuring unpredictable physical processes such as background radiation or atmospheric noise. This is impractical for Monte-Carlo integration, which must be often calculated on a computer. That is why algebraic methods of generating random sequences are generally preferred. Such algorithms are called ***pseudorandom number generators*** and the sequences generated are called ***pseudorandom sequences***.

Generating real numbers

A useful and easily implemented method of generating pseudorandom sequences is the ***linear congruential method***. The sequence $\langle X_n \rangle$ is given by

$$X_{n+1} = (aX_n + c) \bmod m, n \geq 0 \qquad (1)$$

Where m (modulus), a (multiplier) and c (increment) are magical integers chosen by theoretical analysis of the sequences generated and X_0 is an initial nonrandom "seed" value.

The generator given by equation (1) produces a sequence of natural numbers. But we would prefer to generate real numbers in interval $[0,1)$. This is why we use the formula

$$r_n = \frac{X_n}{m} \qquad (2)$$

for the real sequence $\langle r_n \rangle$, where m is close to a word size of the computer used (i.e. $2^{32}-1$ for the 32-bit computers, but in our case we will use $2^{15}-1$, from the reason of C++ language specifications). To generate sequence $\langle x_n \rangle$ in the interval $[a,b)$, we use formula

$$x_n = r_n(b-a)+a \; . \qquad (3)$$

Monte-Carlo Integration

Consider a circle inscribed in a square. Suppose we threw a large number of points at the square and counted the number of points landing within the square (n) as well as the number of points landing inside the circle (n^*). If the area of the square is S, then the circle's area (I) could be approximated by

$$I \approx \frac{n^*}{n} S \; . \qquad (4)$$

Equation (4) is applied more generally to functions and their supersets. Integration problems are reduced to integrating a simple superset. Like other statistical measures the Monte-Carlo definite integral is only approximation to the area. More generally, the Monte-Carlo integration is only practice if the function is continuous and if it possible to find an appropriate simple superset.

The Monte-Carlo problem

We wish to find definite integral

$$I = \int_a^b f(x)dx \, , \quad (5)$$

where $f(x)$ is continuous and real-valued in interval $[a,b)$.We can find the area S, of a bounding box such that I lies entirely within S (see figure 1).

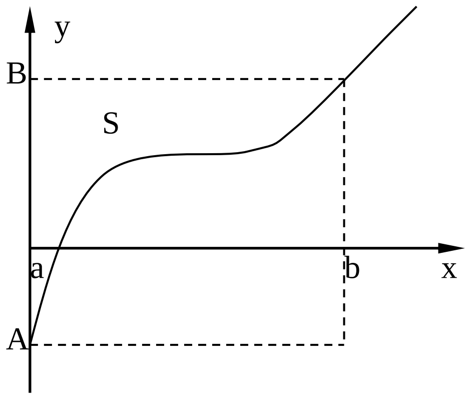

figure 1

$$S = (b-a)(B-A),\ A \le 0 \qquad (6)$$

We can define a second function $g(x)$, such that

$$g(x) = f(x) - A \qquad (7)$$

then always $g(x) \ge 0$.

From equations (5),(6) and (7) we have

$$\int_a^b g(x)dx = \int_a^b (f(x) - A)dx = \int_a^b f(x)dx - \int_a^b Adx = I - Ax\Big|_a^b = I - A(b-a) \Rightarrow$$
$$\Rightarrow I = \int_a^b g(x)dx + A(b-a) \qquad (8)$$

From result of equation (8)

$$I \approx \frac{n^*}{n}S + A(b-a) = \frac{n^*}{n}(b-a)(B-A) + A(b-a) \qquad (9)$$

Error evaluation

Since a, b, A, B and n are constants, thus we can concentrate our attention only on the equation (4). If σ represents the ***standard deviation*** of n then we have

$$\sigma^2 = \overline{(n-\overline{n})^2} \quad (10)$$

which can be also written

$$\sigma^2 = \sum_{i,j=1}^{n} \overline{(x_i - \overline{x})(x_j - \overline{x})}. \qquad (11)$$

However, $(x_i - \overline{x})(x_j - \overline{x})$ equals to $\overline{(x-\overline{x})^2}$ if $i = j$, and zero, otherwise. Hence

$$p = \frac{I}{S}, \text{ and}$$

$$\sigma^2 = n\overline{(x-\overline{x})^2} = np(1-p). \quad (12)$$

Thus

$$\sigma = \sqrt{np(1-p)} \qquad (13)$$

Finally, since values of n lie in the range $n = \bar{n} + \sigma$ and $\bar{n} = \frac{I}{S}n$, we can write

$$n^* = \frac{I}{S}n \pm \sqrt{n\frac{I}{S}\left(1 - \frac{I}{S}\right)} \qquad (14)$$

Thus

$$I = \frac{n^*}{n}S \pm \frac{\sqrt{I(S-I)}}{\sqrt{n}} \quad (15)$$

In other words, the error scales like $\frac{1}{\sqrt{n}}$.

User requirements

An input for our algorithm will be number of points to generate, function $f(x)$ for integration and interval $[a,b]$. Output must be approximated value of integral I and its error. Function $f(x)$ must be continuous and differential in the interval. We will use ***double*** data type for calculation, thus according to Microsoft Specific of ***double*** data type, the output value must be in bounded by $\pm 1.7 \cdot 10^{307}$ with at least 15 digits of precision.

Algorithm

Now we can define algorithm for solving Monte-Carlo problem.

Step 1.
Set $n^* \leftarrow 0$
Set $n \leftarrow$ number of points to be generated.
Step 2.
Generating random number x in the interval $[a,b]$.
Generating random number y in the interval $[0, B-A], A \le 0, B > 0$.
Step 3.
If $g(x) = f(x) - A > y$ then point $p(x,y)$ is in the area of integral I, thus $n^* \leftarrow n^* + 1$
Step 4.
Repeat Steps 2 and 3 n times.
Step 5.
Calculating and output value of integral

$$I \approx \frac{n^*}{n}(b-a)(B-A) + A(b-a).$$

Error of the calculated integral is about $\frac{1}{\sqrt{n}}$.

User information for Code implementing the algorithm

Full code listing can be found in the Appendix.
Program uses functions that are implemented in standard C++ libraries. For better presicion and comparability included functions for streamed input and output. Firstly program waits for user input - number of points k. Then it calculates value of integral and outputs the result with error value.

Testing

Here is a table of results given by the program for simple functions $f(x)$ with known integrals that can be easily calculated by analytical methods with good precision.

Function $f(x)$	Interval $[a,b]$	Value of integral I (k=100000)	Value of integral I (k=1000000)	Value of integral I (k=1000000)	Exact value of integral
$\sin(x)$	$[-4,0]$	-1.65624	-1.652584	-1.6518688	-1.65364362
$\sin(x)$	$[-\pi,\pi]$	0.00030611905	-0.000037230	-0.0000000369	0.0
$\sin(x)$	$[0,4]$	1.65024	1.6535734	1.65361264	1.6536436208
e^x	$[-1,0]$	0.63403	0.63132	0.6321214	0.6321205588
e^x	$[-5,4]$	54.1701005	54.3868005	54.59003674	54.591412086
e^x	$[0,1]$	1.71659497467	1.71828030940	1.7182818403	1.7182818284
x^3	$[-3,4]$	43.48589	43.671894	43.7927966	43.75
x^3	$[-4,4]$	0.0	0.0	0.0	0.0
x^3	$[-4,3]$	-42.77245	-43.72795	-43.7643227	-43.75

We can see that program gives results that are clear to the real values of integrals.

Evaluation of the integral for given function

Our function is $\sinh(\cosh(x))$ in the interval $[-4,3]$.

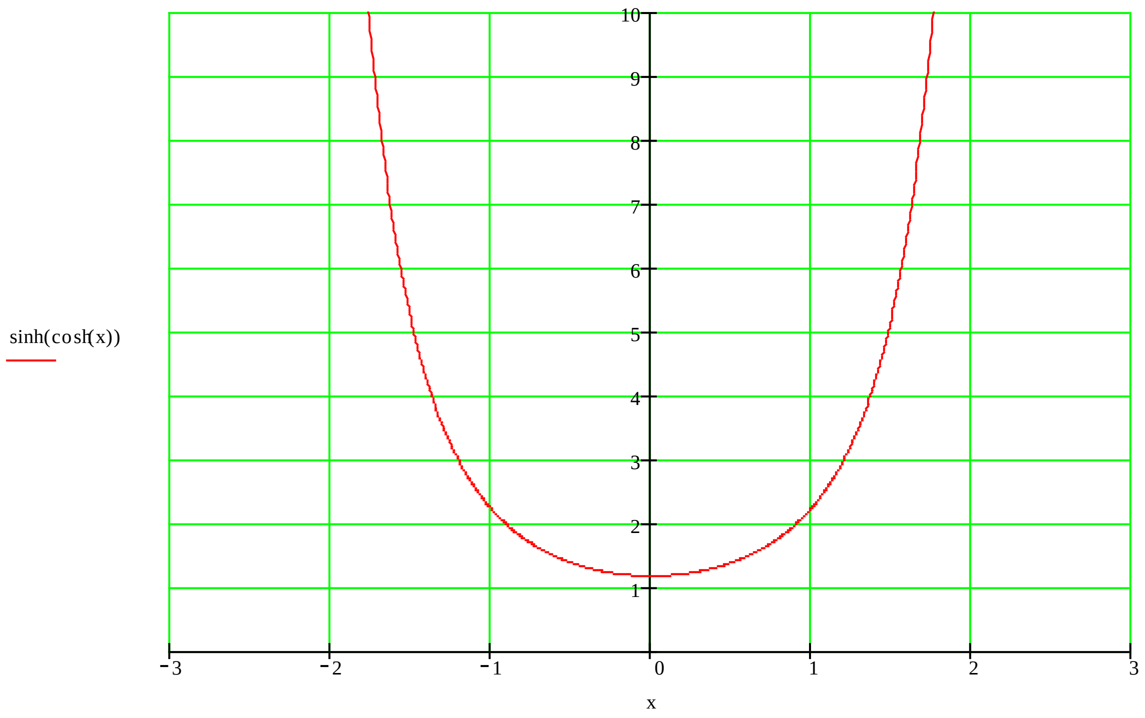

Figure 1. Graph of the function sinh(cosh(x))

Graph of the function has a form that presented above, thus, in our case, following from attributes of function $f(x)$: $a = -4, b = 3, A = 0, B = \sinh(\cosh(-4))$
The program output is presented in the next table:

Number of points k	Integral value	Error
10000	13939447645.2651	0.01
100000	13559280891.3033	0.003162
1000000	13724019818.02	0.001
10000000	13728945842.1135	0.00031622
100000000	13797097068.8737	0.0001

The result calculated by program MathCad is $1.379 \cdot 10^{10}$.
Integral of such function cannot be computed by analytical methods but we can estimate it value by approximation with area of two trapezoids. If $x \le 0$ then function can be inscribed into trapezoid with bases $\sinh(\cosh(0))$ and $\sinh(\cosh(-4))$ and height -4, if $x > 0$ then $f(x)$ can be inscribed into trapezoid with bases $\sinh(\cosh(0))$ and $\sinh(\cosh(3))$ and height 3. Thus, if S – area of two trapezoids then $I < S$.

$$S = \frac{1}{2}((\sinh(\cosh(0)) + \sinh(\cosh(3))) \cdot 3 + (\sinh(\cosh(0)) + \sinh(\cosh(-4))) \cdot 4) \approx 7241 \cdot 10^{11} > 1.397 \cdot 10^{10}$$

Appendix

Code listing of program implemented on C++.

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double function(const double value); /* calculates our function  f(x) */
const double left_value=-4.0; /*  a  value */
const double right_value=3.0; /*  b  value */
const double bottom_value=0.0; /* A  value */
const double top_value=function(left_value); /* B  value */
unsigned long m=0; /* number of points inside the area of integral */
unsigned long k; /* number of points to generate */
/* function that generates real random numbers in the interval [a_value,a_value+range] */
double generator(const double a_value,const double range)
{
	return a_value+((double)rand()/RAND_MAX)*range;
}

/* calculating function  f(x) */
double function(const double value)
{
	return sinh(cosh(value));
}
/* main function */
```

```
void main()
{
        double x,y; /* $x$ and $y$ coordinates of point */
        srand((unsigned)time(NULL)); /* reset seed value */
        cout <<"Enter number of point to generate : ";
        cin>>k; /* reading k value */
        for (unsigned int i=0; i<k; i++) /* main loop */
        {
                /* generating $x$ value in the interval $[a,b]$ */
                x=generator(left_value,right_value-left_value);
                /* generating $y$ value in the interval $[0,B-A]$ */
                y=generator(0.0,top_value-bottom_value);
                /* if $g(x)>y$ then increasing m value */
                if ((function(x)-bottom_value)>y)
                        m++;
        }
        cout.precision(15); /* setting up output precision */
        cout<<"Calculated integral value : ";
        cout<<(right_value-left_value)*(top_value-bottom_value)*((double)m/k)
                +bottom_value*(right_value-left_value)<<endl;
        cout<<"Error is about "<<1/sqrt(k)<<endl;
}
```

References

1. C. Henry Edwards, David E. Penney, ”Calculus with analytic geometry”, Prentice Hall, 1998
2. Guan Yang, “A Monte Carlo Method of Integration”, December 2002
3. Stefan Weinzierl “Introduction to Monte Carlo Methods”, 23 Jun 2000

Help program: MathCad.