University of Antwerp

Scientific Programming

Numerical Integration Exercise 5

Armin Halilovic - s0122210

December 11, 2015

Contents

1 Problem

We are given the following set of inequalities:

$$\begin{cases} 1 \leq x \leq 3 \\ -1 \leq y \leq 4 \\ x^3 + y^3 \leq 29 \\ y \geq e^x - 2 \end{cases}$$

These inequalities describe an irregular 2D figure. We will find the area of this figure using the Monte Carlo method to estimate areas and volumes, which in general looks like:

$$\int_A f \approx (measure\ of\ A) * (average\ of\ f\ over\ n\ random\ points\ in\ A)$$

This will be done using C++ and the GNU Scientific Library. In section 3, we will describe how we reached each solution, using the most important parts of the code.

2 Using the program

All of the C++ code for the program can be found in the file main.cpp and in appendix A of this document.

To compile and run the program, execute the following commands in the build/ directory:

```
cmake ..
make
./numerical_integration.bin
```

Do not forget the .bin extension. Output can be found in the console output of the program and in appendix B.1.

3 Solutions

3.1 Find the figure

From the given set of inequations, we can deduce that the figure is bounded by the functions $\sqrt[3]{29 - x^3}$, $e^x - 2$, and by the rectangle defined by the points $(1, -1), (3, -1), (3, 4), (1, 4)$. The functions and rectangle are plotted in figure 1. The marked area is the figure we want to know the area of.

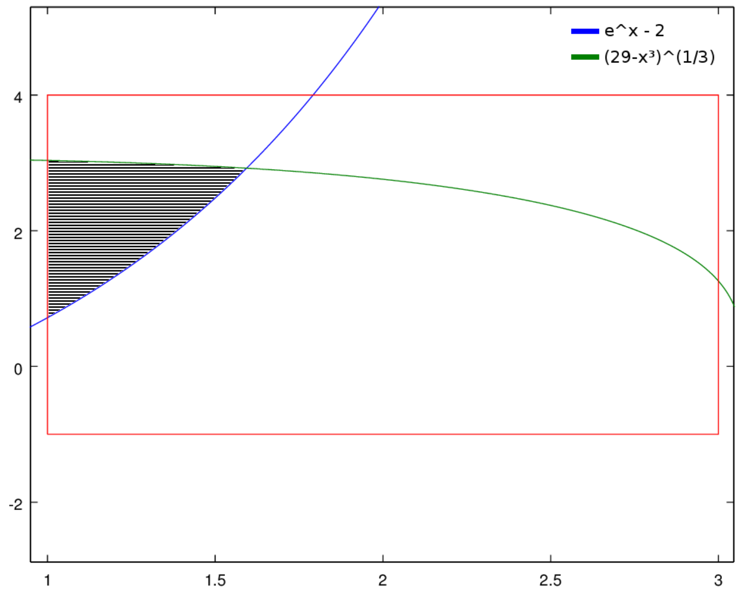

Figure 1: The functions and rectangle defined by the given inequalities

3.2 Choose a decent random number generator

We chose to use ranlxs0 as the random number generator for these solutions. ranlxs0 is based on the RANLUX algorithm. Generators based on the RANLUX algorithm offer the best mathematically-proven quality of randomness. Besides this, ranlxs0 is got a pretty high spot on GSL's preformance table[1]. These facts made ranlxs0 seem like a good choice.
We use GSL's random number generation library as follows:

[1]https://www.gnu.org/software/gsl/manual/html_node/Random-Number-Generator-Performance.html

```
//Use c++11's random_device to create a random unsigned int to use as seed
    for the generator
std::random_device rd;
//Initialize the ranlxs0 random number generator
gsl_rng *r = gsl_rng_alloc(gsl_rng_ranlxs0);
//Seed the generator with a random number
gsl_rng_set(r, rd());

//Get a random number in [0, 1]
double randomNumber = gsl_rng_uniform(r);
```

3.3 Calculate the area

We are now ready to use the Monte Carlo estimation method. The red rectangle in figure 1 is the A in:

$$\int_A f \approx (measure\ of\ A) * (average\ of\ f\ over\ n\ random\ points\ in\ A)$$

How the algorithm is used is explained with the following code excerpt.

```
1  double calculateArea(int n) {
2      ...
3      for (int i = 0; i < n; i++) {
4          //gsl_rng_uniform(r) uniformly gets a random number in [0, 1]
5          x = 2 * gsl_rng_uniform (r) + 1;
6          y = 5 * gsl_rng_uniform (r) - 1;
7
8          if ((gsl_pow_int(x, 3) + gsl_pow_int(y, 3) <= 29) and (y >= exp(x)
       - 2)) result++;
9      }
10     result /= n;
11     result *= 2.0 * 5.0;
12     ...
13     return result;
14 }
```

Lines 3 to 10 do the $(average\ of\ f\ over\ n\ random\ points\ in\ A)$ part of the equation. Points (x, y) are chosen to lie within the given bounds of the rectangle. If they then lie in the marked part of figure 1, a counter is incremented. Then, at line 10, the counter is divided by the total amount of points generated.
The result of this then has to be multiplied by $(measure\ of\ A)$, which is simply the area of the rectangle.
That's all, we can now calculate approximations of the area of the given figure using the function calculateArea, which takes as argument the amount of random points to generate.

3.4 Results

By calculating the area manually by integration, we found that the area of the figure is approximately 0.758.
While experimenting with calculateArea with various values for n, we noticed that there is a high variance in the results we get.
For example, when generating 1000 points, the resulting area can be somewhere between 0.65 and 0.85.

3.4.1 More points

One way to improve the results is to simply generate (many) more points. Filling the figure with as many points as possible will lead to a better result, at the cost of more processing power.
In the function calculateUntil(double e), an area is calculated for an n_0, where n_0 is randomly chosen to lie in [2000, 5000]. Then, an area is calculated for $n_{i+1} = 2n_i$ and compared to the previous result. This is repeated until the difference between two sequential results is smaller than e.
Then, in calcAvgOfN(int n, double e), this is done n times, and an average of the n calculations is returned.
This solution improved the results, but is is not really a practical solution. To give an example, calcAvgOfN(10, 0.01) narrowed the resulting area down to range from around 0.745 to 0.765

3.4.2 Smaller rectangle

A better way to improve the results is to use a rectangle that wraps the figure tighter. The rectangle we used previously left a lot of open space around the figure. Because of this, we need many more points to get a result that we could get with a smaller rectangle.
To construct the smaller rectangle:

- The left x value, x_0, stays the same.
- The right x value, x_1, is the cross point of $\sqrt[3]{29 - x^3}$ and $e^x - 2$
- The bottom y value is the minimum of $\sqrt[3]{29 - x^3}$ and $e^x - 2$ on the interval $[x_0, x_1]$
- The top y value is the maximum of $\sqrt[3]{29 - x^3}$ and $e^x - 2$ on the interval $[x_0, x_1]$

For these values, we find 1, 1.59374, 0.718282 and 3.03659, respectively.

In figure 2, this new rectangle is depicted. We can clearly see that the ratio $\frac{area\ figure}{area\ rectangle}$ is larger than the one with the previous rectangle. The result of this is that the same results as before can be achieved with fewer points generated.
This can be confirmed by using the function calcAvgOfN(10, 0.01, true)[2]. The output of this function can be seen appendix in B.1. We can confirm that when using the smaller rectangle, fewer points are generated to achieve approximately the same result.

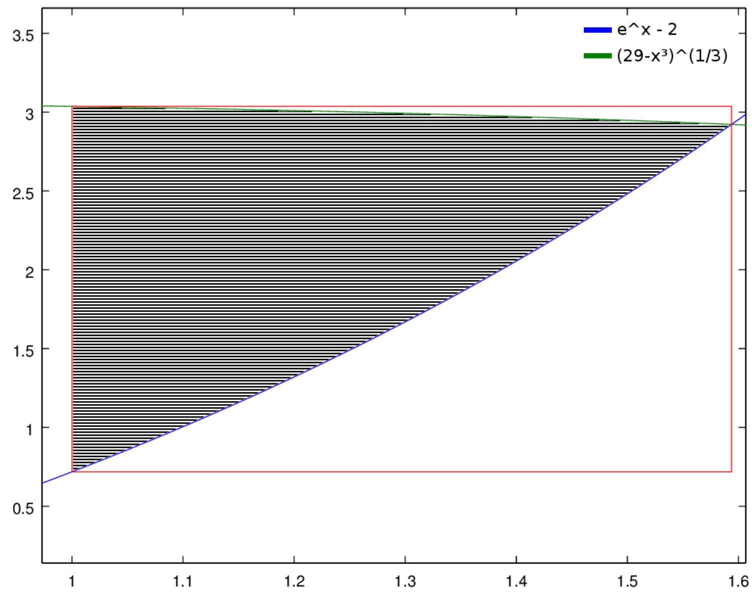

Figure 2: The given functions and the smaller rectangle

[2]In the code, the boolean smallerRectangle will make the calculcations use the smaller rectangle instead of the given one.

Appendices

A Code

A.1 main.cpp

```
#include <iostream>
#include <random>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_math.h>

//Calculate the area of the figure using n points
//smallerRectangle toggles which rectangle is used for the calculation
double calculateArea(int n, bool smallerRectangle) {
    std::random_device rd;
    //Use the ranlxs0 random number generator
    gsl_rng *r = gsl_rng_alloc(gsl_rng_ranlxs0);
    //Seed the generator with a random number
    gsl_rng_set(r, rd());

    double result = 0, x = 0, y = 0;
    for (int i = 0; i < n; i++) {
        // rescale the points with: x' = (b - a) x + a
        if (!smallerRectangle) {
            // 1 <= x <= 3 , -1 <= y <= 4
            x = 2 * gsl_rng_uniform (r) + 1;
            y = 5 * gsl_rng_uniform (r) - 1;
        } else {
            // 1 <= x <= 1.59374 , 0.71828 <= y <= 3.03659
            x = 0.59374 * gsl_rng_uniform(r) + 1;
            y = 2.347618 * gsl_rng_uniform(r) + 0.718282;
        }

        if ((gsl_pow_int(x, 3) + gsl_pow_int(y, 3) <= 29) and (y >= exp(x) - 2)) result++;
    }
    if (!smallerRectangle) result *= 2.0 * 5.0 / n;
    else                   result *= 0.59374 * 2.347618 / n;

    gsl_rng_free(r);
    return result;
}

//Calculate the area of the figure with an increasing n until the difference between 2
     sequential areas <= e
//smallerRectangle toggles which rectangle is used for the calculation
double calculateUntil(double e, bool smallerRectangle) {
    std::random_device rd;
    //Use the ranlxs0 random number generator
    gsl_rng *r = gsl_rng_alloc(gsl_rng_ranlxs0);
    //Seed the generator with a random number
    gsl_rng_set(r, rd());
    //Chose a random n_0 ranging from 2000 to 5000
    int n = (int) (3000 * gsl_rng_uniform (r) + 2000);

    double result = calculateArea(n, smallerRectangle);
    double nextResult = calculateArea(n * 2, smallerRectangle);

    while (fabs(result - nextResult) > e) {
        n *= 2;
```

```
        result = nextResult;
        nextResult = calculateArea(n, smallerRectangle);
    }
    std::cout << "Area: " << result << ", Number of points: " << n << std::endl;
    return result;
}

//Calculate the area of the figure n times using calculateUntil, then print the average
//smallerRectangle toggles which rectangle is used for the calculation
void calcAvgOfN(int n, double e, bool smallerRectangle) {
    double avgArea = 0;
    std::cout << "Calculating area using " << (smallerRectangle ? "a smaller" : "the given")
     << " rectangle" << std::endl;
    for (int i = 0; i < n; i++) {
        avgArea += calculateUntil(e, smallerRectangle);
    }
    avgArea /= n;
    std::cout << "Average Area: " << avgArea << std::endl << std::endl;
}



int main (int argc, char *argv[]) {
    calcAvgOfN(15, 0.01, true);
    calcAvgOfN(15, 0.01, false);

    return 0;
}
```

B Output

B.1 Console output

```
Calculating area using a smaller rectangle
Area: 0.753569, Number of points: 3975
Area: 0.759656, Number of points: 7934
Area: 0.741523, Number of points: 3267
Area: 0.765329, Number of points: 6746
Area: 0.769334, Number of points: 4977
Area: 0.758072, Number of points: 2736
Area: 0.756304, Number of points: 9016
Area: 0.751566, Number of points: 3687
Area: 0.762812, Number of points: 18832
Area: 0.754261, Number of points: 2772
Area: 0.744161, Number of points: 4885
Area: 0.761588, Number of points: 9702
Area: 0.759893, Number of points: 5978
Area: 0.757709, Number of points: 25872
Area: 0.775812, Number of points: 3340
Average Area: 0.758106

Calculating area using the given rectangle
Area: 0.750862, Number of points: 8124
Area: 0.770569, Number of points: 65536
Area: 0.758506, Number of points: 601472
Area: 0.773399, Number of points: 8120
Area: 0.770863, Number of points: 4242
Area: 0.763246, Number of points: 221056
Area: 0.743337, Number of points: 51632
Area: 0.77885,  Number of points: 11196
Area: 0.745841, Number of points: 14668
Area: 0.809414, Number of points: 3484
Area: 0.763514, Number of points: 71040
Area: 0.75474,  Number of points: 145984
Area: 0.74328,  Number of points: 29464
Area: 0.76558,  Number of points: 245696
Area: 0.765242, Number of points: 44352
Average Area: 0.763816
```