3.4: Numerical Approximation of Multiple Integrals

As you have seen, calculating multiple integrals is tricky even for simple functions and regions. For complicated functions, it may not be possible to evaluate one of the iterated integrals in a simple closed form. Luckily there are numerical methods for approximating the value of a multiple integral. The method we will discuss is called the Monte Carlo method. The idea behind it is based on the concept of the average value of a function, which you learned in single-variable calculus. Recall that for a continuous function \( f(x) \), the average value \( \bar{f} \) of \( f \) over an interval \([a,b]\) is defined as

\[
\bar{f} = \frac{1}{b-a} \int_a^b f(x) \, dx \quad \text{(Eq3.11)}
\]

The quantity \( b - a \) is the length of the interval \([a,b]\), which can be thought of as the “volume” of the interval. Applying the same reasoning to functions of two or three variables, we define the average value of \( f(x,y) \) over a region \( R \) to be

\[
\bar{f} = \frac{1}{A(R)} \iint_R f(x,y) \, dA \quad \text{(Eq3.12)}
\]

where \( A(R) \) is the area of the region \( R \), and we define the average value of \( f(x,y,z) \) over a solid \( S \) to be

\[
\bar{f} = \frac{1}{V(S)} \iiint_S f(x,y,z) \, dV \quad \text{(Eq3.13)}
\]

where \( V(S) \) is the volume of the solid \( S \). Thus, for example, we have

\[
\iiint_R f(x,y) \, dA = A(R) \bar{f} \quad \text{(Eq3.14)}
\]

The average value of \( f(x,y) \) over \( R \) can be thought of as representing the sum of all the values of \( f \) divided by the number of points in \( R \). Unfortunately there are an infinite number (in fact, uncountably many) points in any region, i.e. they cannot be listed in a discrete sequence. But what if we took a very large number \( N \) of random points in the region \( R \) (which can be generated by a computer) and then took the average of the values of \( f \) for those points, and used that average...
as the value of \(\bar{f}\)? This is exactly what the Monte Carlo method does. So in Formula \ref{Eq3.14} the approximation we get is

\[
\iint_R f(x, y) \, dA \approx A(R) \bar{f} \pm A(R) \sqrt{\frac{\overline{f^2} - (\bar{f})^2}{N}} \tag{Eq3.15}
\]

where

\[
\bar{f} = \frac{\sum_{i=1}^{N} f(x_i, y_i)}{N} \quad \text{and} \quad \overline{f^2} = \frac{\sum_{i=1}^{N} (f(x_i, y_i))^2}{N} \tag{Eq3.16}
\]

with the sums taken over the \(N\) random points \((x_1, y_1), \ldots, (x_N, y_N)\). The \(\pm\) “error term” in Formula \ref{Eq3.15} does not really provide hard bounds on the approximation. It represents a single standard deviation from the expected value of the integral. That is, it provides a likely bound on the error. Due to its use of random points, the Monte Carlo method is an example of a probabilistic method (as opposed to deterministic methods such as Newton’s method, which use a specific formula for generating points).

For example, we can use Formula \ref{Eq3.15} to approximate the volume \(V\) under the plane \(z = 8x + 6y\) over the rectangle \(R = [0,1] \times [0,2]\). In Example 3.1 in Section 3.1, we showed that the actual volume is 20. Below is a code listing (montecarlo.java) for a Java program that calculates the volume, using a number of points \(N\) that is passed on the command line as a parameter.

```java
//Program to approximate the double integral of f(x,y)=8x+6y
//over the rectangle [0,1]x[0,2].
public class montecarlo { 

    public static void main(String[] args) { 

        //Get the number N of random points as a command-line parameter 
        int N = Integer.parseInt(args[0]);

        double x = 0; //x-coordinate of a random point 
        double y = 0; //y-coordinate of a random point 
        double f = 0; //Value of f at a random point 
        double m = 0.8; //Mean of the values of f 
        double m2 = 0.0; //Mean of the values of f^2 

        for (int i=0; i<N; i++) { //Get the random coordinates 
            x = Math.random(); //x is between 0 and 1 
            y = 2 * Math.random(); //y is between 0 and 1 
            f = 8*x + 6*y; //Value of the function 
            m = m + f; //Add to the sum of the f values 
            m2 = m2 + f*f; //Add to the sum of the f^2 values 
        }

        m = m/N; //Compute the mean of the f values 
        m2 = m2/N; //Compute the mean of the f^2 values 

        System.out.println("\n\n\nThe volume of the rectangle [0,1]x[0,2] \n\n\nreturn v;");
    }
}
```

**Listing 3.1** Program listing for montecarlo.java
The results of running this program with various numbers of random points (e.g. java montecarlo 100) are shown below:

As you can see, the approximation is fairly good. As \( (N \to \infty) \), it can be shown that the Monte Carlo approximation converges to the actual volume (on the order of \( O(\sqrt{N}) \), in computational complexity terminology).

In the above example the region \( \text{(R)} \) was a rectangle. To use the Monte Carlo method for a nonrectangular (bounded) region \( \text{(R)} \), only a slight modification is needed. Pick a rectangle \( \text{\tilde{R}} \) that encloses \( \text{(R)} \), and generate random points in that rectangle as before. Then use those points in the calculation of \( \text{\bar{f}} \) only if they are inside \( \text{(R)} \). There is no need to calculate the area of \( \text{(R)} \) for Equation \ref{Eq3.15} in this case, since the exclusion of points not inside \( \text{(R)} \) allows you to use the area of the rectangle \( \text{\tilde{R}} \) instead, similar to before.

For instance, in Example 3.4 we showed that the volume under the surface \( z = 8x + 6y \) over the nonrectangular region \( \text{(R)} = \{(x, y) : 0 \leq x \leq 1, 0 \leq y \leq 2x^2 \} \) is 6.4. Since the rectangle \( \text{\tilde{R}} = [0,1] \times [0,2] \) contains \( \text{(R)} \), we can use the same program as before, with the only change being a check to see if \( y < 2x^2 \) for a random point \( (x, y) \) in \([0,1] \times [0,2]\). Listing 3.2 below contains the code (montecarlo2.java):

```
//Program to approximate the double integral of f(x,y)=8x+6y over the
//region bound by x=0, x=1, y=0, and y=2x^2
public class montecarlo2 {  
    public static void main(String[] args) { 
        //Get the number N of random points as a command-line parameter 
        int N = Integer.parseInt(args[0]); 
        double x = 0; //x-coordinate of a random point 
        double y = 0; //y-coordinate of a random point 
        double f = 0.0; //Value of f at a random point 
        double m = 0.0; //Max of the values of f 
        double m2 = 0.0; //Max of the square of f 
        double i = 0.0; //Get the random coordinates 
        x = Math.random(); //x is between 0 and 1 
        y = Math.random(); //y is between 0 and 2 
        if (y < 2 Math.pow(x,2)) { //The point is in the region 
            f = 8x + 6y; //Value of the function 
            m = m + f; //Add to the sum of the f values 
            m2 = m2 + Math.pow(f,2); //Add to the sum of the f^2 values 
        } 
        m = m/N; //Compute the mean of the f values 
        m2 = m2/N; //Compute the mean of the f^2 values 
        double vol1 = m2-m; //Integral = \% \text{vol(1)f} + 
        System.out.println("\n\text{The volume of the rectangle [0,1]x[0,2]}\n\text{public static double vol1()} {\n\text{ \quad return vol1;}}\n\); 
    } 
}
```

UC Davis ChemWiki is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License.
The results of running the program with various numbers of random points (e.g. java montecarlo2 1000) are shown below:

<table>
<thead>
<tr>
<th>N</th>
<th>integral</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6.95747529014694 +/- 2.9105131555120552</td>
</tr>
<tr>
<td>100</td>
<td>6.3149656296555355 +/- 0.9549029662159609</td>
</tr>
<tr>
<td>1000</td>
<td>6.47702813858756 +/- 0.3191583726973624</td>
</tr>
<tr>
<td>10000</td>
<td>6.3499758080155889 +/- 0.10648685345895105</td>
</tr>
<tr>
<td>100000</td>
<td>6.440184132811864 +/- 0.012047879881392</td>
</tr>
<tr>
<td>1000000</td>
<td>6.417050897922222 +/- 0.0109454409769472</td>
</tr>
</tbody>
</table>

To use the Monte Carlo method to evaluate triple integrals, you will need to generate random triples \((x, y, z)\) in a parallelepiped, instead of random pairs \((x, y)\) in a rectangle, and use the volume of the parallelepiped instead of the area of a rectangle in Equation \ref{Eq3.15} (see Exercise 2). For a more detailed discussion of numerical integration methods, see PRESS et al.

**Contributors and Attributions**

- Michael Corral (Schoolcraft College). The content of this page is distributed under the terms of the GNU Free Documentation License, Version 1.2.