Introduction to Random Numbers and The Monte Carlo Method

Computer simulations play a very important role in scientific experiments, as well as in determining solutions to complicated mathematical questions. For example, if the computer can be made to imitate an experiment, then by repeating the simulation with different data, scientists can draw statistical conclusions. These computer simulations are often much less expensive than physical experiments performed in a laboratory, and in some cases are the only way to solve certain mathematical problems. When the mathematical questions, or the scientific experiment, can be replaced by simulating a random process, the approach is called a *Monte Carlo method*, or a *method of statistical trials*. We will provide an introduction to the Monte Carlo method, but first we need to discuss random numbers.

1 Random Numbers

In order to use the Monte Carlo method, we need to be able to generate random numbers; that is, a sequence of numbers with the property that it is not possible to predict the next number knowing all of the previous values. Random numbers can come from a variety of *probability distributions*. A probability distribution is a formula, table or graph that provides the probability associated with each value of a random variable. Two most commonly used distributions are *uniform* and *normal*.

1.1 Uniform Distribution

The *uniform distribution* can be defined informally as follows. Suppose $x_1, x_2, \ldots, x_n$ are numbers contained in the interval $(0, 1)$. This sequence is uniformly distributed if no subset of the interval contains more than its fair share of the numbers. We can see an illustration of this using MATLAB's `rand` and `hist` functions. The function `rand` can be used to generate a sequence of random numbers uniformly distributed in $(0, 1)$. For example, if we want to generate 100 values, we use the MATLAB statement:

```
>> x = rand(100,1);
```

A useful way to visualize random numbers is to use a histogram plot of the values. The MATLAB function `hist` is used for this purpose:

```
>> N = hist(x);
```

This bins the elements of `x` into 10 equally spaced containers, and returns in `N` the number of elements in each container. One can use more (or less) than 10 containers. For example:

```
>> N = hist(x, 8);
```

bins the elements of `x` into 8 equally spaced containers, and returns in `N` the number of elements in each container.

Note that with only 100 random numbers, we do not see the "uniform" distribution (i.e., the heights of each container in the histogram plot are **not** approximately the same). However, by constructing a larger set of random numbers, the uniform distribution becomes more apparent. Try:

```
>> x = rand(10000,1);
>> hist(x)
```

Note that in this case, the heights of the containers are now approximately equal.

In some cases we may want the random numbers to be in an interval other than $(0, 1)$, or we may want to generate a sequence of random integers. Here are some examples:

- To generate a sequence of n random numbers uniformly distributed in the interval (a, b) use `(b-a)*rand(n,1)+a`. For example,

  ```
  >> x = 3*rand(100,1) + 7;
  ```

  creates a vector containing 100 entries uniformly distributed in the interval $(7, 10)$.

- To generate a sequence of n uniformly distributed random integers in the set $\{0, 1, \ldots, k\}$, use `floor((k+1)*rand(n,1))`. For example,

  ```
  >> x = floor(11*rand(20,1));
  ```

  creates a vector containing 20 random integers between 0 and 10.

- To generate a sequence of n uniformly distributed random integers in the set $\{j, j+1, \ldots, k\}$, use `floor((k-j+1)*rand(n,1)+j)`. For example,

  ```
  >> x = floor(51*rand(20,1)+100);
  ```

  creates a vector containing 20 random integers between 100 and 150.

1.2 Probability Density Function

Every probability distribution has an associated *probability density function* (PDF), $f(x) \geq 0$. The area under the graph of $f(x)$ between a and b is the probability that a random number lies between a and b. The PDF is scaled so that the total area under the graph is 1.

For example, the uniform distribution on $(0, 1)$ has the PDF

$$f(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

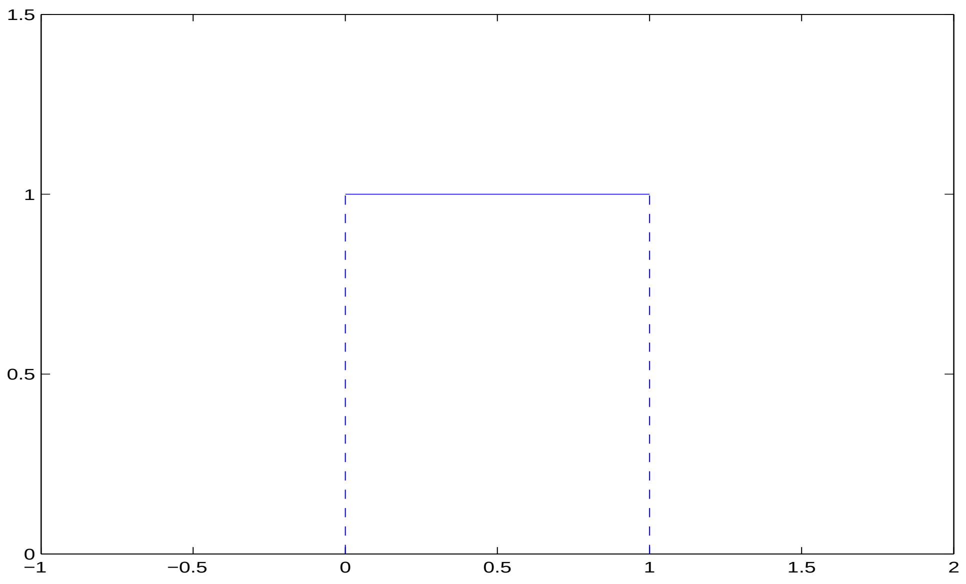

Figure 1: Plot of the PDF for a uniform probability distribution.

A graph of this PDF is shown in figure **??**.

Note that the total area under the graph of $f(x)$ is the area of the square with sides having length 1. The probability of choosing a random number between 0.02 and 0.5 is the area of the rectangle with height 1 and width 0.5 - 0.02 = 0.48. Thus the probability is 0.048.

Problem 1 *What is the relationship between the plot of the PDF and the histogram plots?*

Problem 2 *As another example, consider the graph of the function $f(x)$ shown in figure* **??**. *Show that $f(x)$ is a PDF. Calculate the probability that a random variable is between 3 and 8.*

1.3 Normal Distributions

Many important random phenomena are modeled by a *normal* (or Gaussian) PDF:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} ,$$

where μ is the *mean*, and σ is the *standard deviation*. Figure **??** shows plots of the normal PDF for various values of μ and σ.

The MATLAB function `randn` can be used to generate a sequence of random numbers with a normal distribution, with mean 0 and standard deviation 1.

Problem 3 *Generate a vector containing 1000 random entries from a normal distribution. Make a histogram plot of these random numbers. How is the histogram plot related to the PDF?*

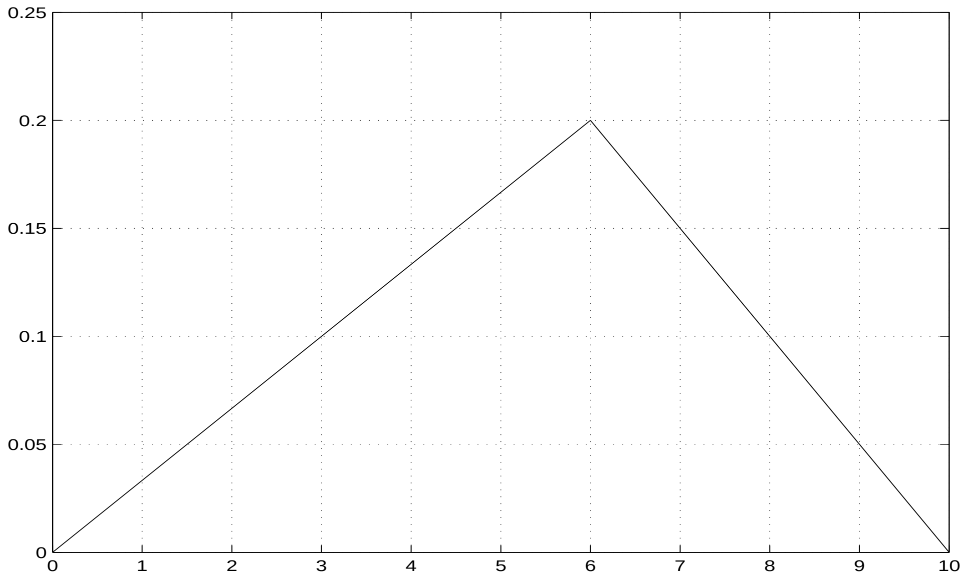

Figure 2: Example of a simple PDF.

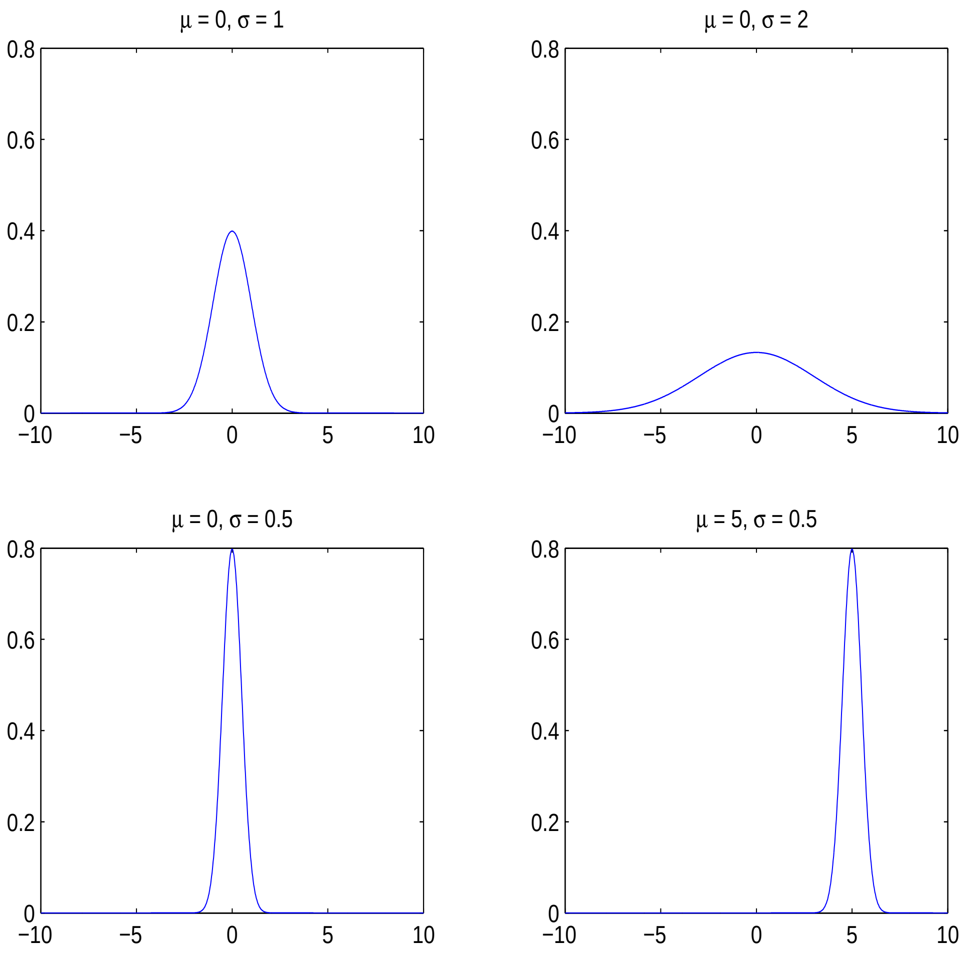

Figure 3: Plot of the PDF for a normal probability distribution for various values of μ and σ.

2 Monte Carlo Method

The Monte Carlo method can be used to determine an unknown quantity through experimentations involving random processes. We illustrate the approach through an example that can be used to estimate the value of π. Suppose we have a square target with vertices (1,1), (1,-1), (-1,1) and (-1,1), with a circle having radius equal to 1 inside the square. That is:

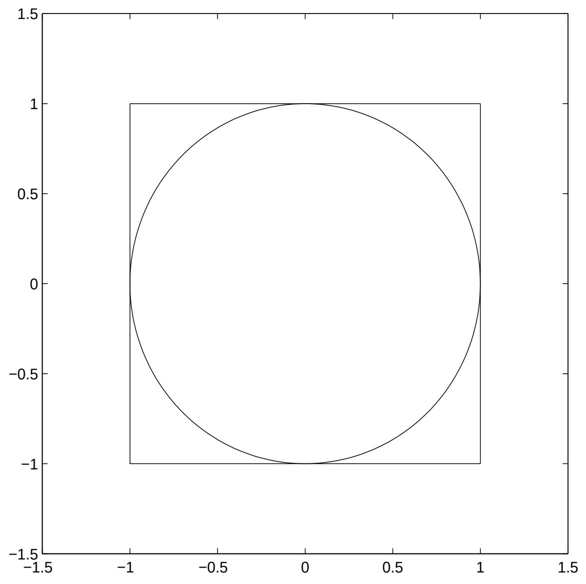

Figure 4: Dart board.

Suppose we now throw darts randomly at the target, and that they can land anywhere on the square with equal probability (i.e., a uniform distribution). Then if we throw enough darts, it is reasonable to use the approximation:

$$\frac{\text{Number of throws inside circle}}{\text{Number of throws}} \approx \frac{\text{Area of circle}}{\text{Area of square}}\,.$$

Problem 4 *Show that this experiment can be used to approximate the value of π as:*

$$\pi \approx 4\frac{\textit{Number of throws inside circle}}{\textit{Number of throws}}\,. \tag{1}$$

We will design a Matlab experiment to simulate this experiment, and hence compute an approximation of π. First, the following code can be use to construct the dart board:

```
figure
plot([-1,1,1,-1],[-1,-1,1,1])
plot([-1,1,1,-1,-1],[-1,-1,1,1,-1])
```

```
axis([-1.5,1.5,-1.5,1.5])
axis square
hold on
theta = linspace(0,2*pi,200);
plot(cos(theta), sin(theta),'r')
```

We now want to generate random dart throws at the board. That is, we want to generate random points (x, y) with $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. We can do this (for, say, 100 dart throws) as follows:

```
x = 2*rand(100,1) - 1;
y = 2*rand(100,1) - 1;
```

If we want to visualize where these dart throws hit the board, we simply plot the points:

```
plot(x, y, 'o')
```

We can now get an estimate of π using the formula given in (**??**):

```
NumberInside = sum(x.^2 + y.^2 <= 1);
PiEstimate = 4 * (NumberInside / 100)
```

Since only 100 random throws were used, it is unlikely that this estimate for π will be very accurate. Better accuracy can be obtained by increasing the number of throws.

Problem 5 *Try increasing the number of throws to 1000, 10000 and 100000. What is the estimated value of π that you compute? What do you conclude about using a Monte Carlo method for computing an approximation of π?*