EECS 10: Assignment 5

Prof. Rainer Doemer

October 23, 2009

Due Monday 11/02/2008 12:00pm

1 Monte Carlo Calculation of Pi [20 points]

Monte Carlo (MC) methods are stochastic techniques, meaning they are based on the use of random numbers and probability statistics to investigate problems. In this part of the homework, you are asked to write a program to implement a simple geometric MC experiment which calculates the value of Pi based on a "hit and miss" integration.

The figure below shows a unit circle circumscribed by a square. The radius of the circle r equals to 1/2 of the side of the square. Furthermore, the center of the circle and the center of the square are identical.

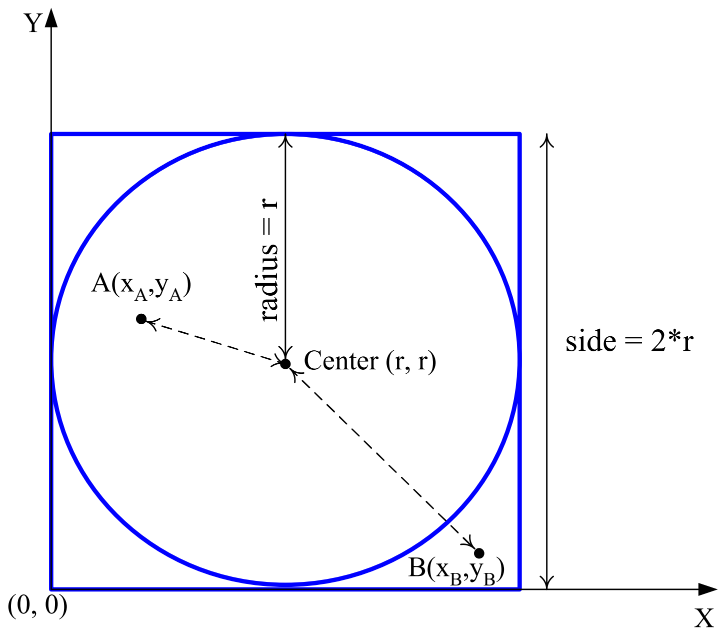

Figure 1: A Circle Circumscribed by a Square

Imagine we can throw points randomly at the above figure. If we can throw infinite random points, it should be apparent that of the total number of points that hit the circle divded by the the number of points that hit within the square is proportional to the area of that part.
In other words:

$$\frac{number\ of\ points\ hitting\ circle\ area}{number\ of\ points\ hitting\ square\ area} = \frac{area\ of\ circle}{area\ of\ square} = \frac{\pi\times r\times r}{(2\times r)^2} = \frac{\pi\times r\times r}{4\times r\times r} = \frac{\pi}{4}$$

Therefore, we get the formula to calculate π using Monte Carlo method:

$$\pi = 4 \times \frac{number\ of\ points\ hitting\ circle\ area}{number\ of\ points\ hitting\ square\ area}$$

In the real world, we can only throw a finite number of random points, therefore, the π calculated using the above formula is an approximation of the exact value of π.

We can have our computer generate random numbers to simulate the throwing of points. For each point, we can have computer to generate two random floating point numbers to be the x and y coordinates of the point, where $0 \leq x \leq 2r$ and $0 \leq y \leq 2r$ so that (x,y) must fall within the square area. However, this randomly generated point could fall within the circle area or fall out of the circle area.

To decide if the randomly generated point (x,y) is within the area of the circle, we can compare the distance of the point to the center with the radius r. For example, the point A in the above figure is in the circle area since its distance to center is less than r. However, the point B in the above figure is not in the circle area since its distance to center is greater than r.

Note: If the distance of point to the center equals to radius r, then that point is considered within the circle area.

Assume the radius of the circle is r and the coordinates of the randomly generated point P is (x,y), then the distance of P to the center is:
$Distance(P,Center) = \sqrt{(x-r)\times(x-r)+(y-r)\times(y-r)}$

To avoid the square root calculation, you can compare $Distance(P,Center) \times Distance(P,Center)$ with the radius squared $r \times r$ in order to decide if the randomly generated point is within the circle area.

At the beginning, your program should ask for the input of radius r and the number of random points N the computer needs to generate. The output should like this:

```
Enter the radius of circle:  5
Enter the number of points:  10
```

During the generation, whenever a random point is generated, your program should print out the coordinates of the point and whether the point is *In* or *Out* the circle area like this:

```
Point No.1(x=0.000000,y=6.551714):  OUT
Point No.2(x=3.048189,y=6.749779):  IN
Point No.3(x=1.067537,y=5.165868):  IN
Point No.4(x=4.896695,y=6.024659):  IN
Point No.5(x=3.699454,y=2.566607):  IN
Point No.6(x=3.741874,y=8.255867):  IN
Point No.7(x=1.727042,y=2.977996):  IN
Point No.8(x=6.435438,y=7.896664):  IN
Point No.9(x=9.878231,y=8.005921):  OUT
Point No.10(x=4.642476,y=5.389874):  IN
```

At the end, your program should output the number of points within and out of the circle, together with the approximation value of π like this:

```
/******In Summary******/
Points within circle area:  8
Points out of circle area:  2
Pi= 3.200000
```

To show that your program works correctly, run it once with the radius = 10 and the number of points = 20. Submit the output as your script file (**mc.script**). Please compile your C code using **-ansi -Wall** options as below to specify ANSI code with all warnings:

```
gcc -o mc -ansi -Wall mc.c
```

The files that you should submit for this part of the assignment are:

- **mc.c**: the source code file.
- **mc.txt**: the brief text file to explain what the program does and why you chose your method of implementation.
- **mc.script**: the typescript file to show that your program works with the radius = 10 and the number of points = 20.

HINT In assignment 4, you have learned how to generate random integer numbers within the range of $[0,n)$ (n is exclusive). However, in this homework, you need to generate floating point numbers with the range of $[0,n]$ (n is inclusive). Therefore, there are some modification of the code described in homework4.

You need to replace the following line in assignment 4
int randomNumber = rand() % n;
with
*double randomNumber = ((double)rand())/((double)RAND_MAX)*s;* /* s is the side of the square */
Note: please do not forget to include the *stdlib.h* at the beginning of your program:
#include ⟨stdlib.h ⟩

Furthermore, in assignment 5, we want you to use the same seed for the random numbers generation in order to generate the same series of random numbers. Therefore, take out the following line:
#include ⟨time.h⟩

and replace the following line in homework4
srand(time(NULL)); with
srand(0);

2 Square root approximation [20 points + 5 points (extra credit)]

Write a program to calculate the square root of any positive floating-point value.
At the beginning, the program should ask the user to input a positive number N in the range between 0 and 10000.

```
Please input a positive number (1 to 10000):
```

We will use a binary search approximation technique for this assignment. In particular, the program will always keep a range of a left bound L and a right bound R, where the actual square root S lies somewhere between L and R: $L \leq S \leq R$ Consequently, it follows that $L*L \leq N = S*S \leq R*R$. Thus, to find S, we can compare $L*L$ or $R*R$ with N. The binary approximation then works as follows. First, we compute a value M that lies in the middle between the left bound L and the right bound $R : M = L + (R-L)/2$. Then, if $M*M$ is less than N, the square root obviously lies somewhere in the right half of the current range (i.e. within M to R), otherwise in the left half of the current range (i.e. within L to M). The program then can use the proper half of the range as the new range and repeat the whole process. With each iteration, the search range is effectively reduced to half its previous size. Because of this, this technique is called binary search. To start the search, we will use the range from 0 to N (which is guaranteed to contain the square root of N). We will stop the iteration, once we have reached a range that is smaller than 0.0000000001 so that we reach a precision of 10 digits after the decimal point for our approximation. (HINT: We will need long double variables for all variables in order to achieve this precision.)

The pseudo-code of the algorithm can be written as follows:

```
Start with a range of 0 to N
As long as the range is not accurate enough, repeat the following steps:
Compute the middle of the range
Compare the square of the middle value with N
If the middle value is less than the square root
              Use middle-to-right as the new range
Otherwise
              Use left-to-middle as the new range
Output the middle of the latest range as result
```

For example, to compute the square root of 10, the program will start with 5, which is in the middle between 0 and 10. Since $5*5 = 25$ is larger than 10, the program will try the middle number 2.5 of left bound (0 to 5). Thus, the program compares $2.5*2.5$ with 10. Because the result 6.25 is smaller than 10, it will pick 3.75 (the middle number of 2.5 and 5) as the next guess. By picking the middle number every time and comparing its square with the original number, the program gets closer to the actual square root.

To demonstrate the approximation procedure, your program should print the approximated square root in each iteration, as follows:

```
Please input a positive number (1 to 10000): 42
Iteration 1: the square root of 42.0000000000 is approximately 21.0000000000
Iteration 2: the square root of 42.0000000000 is approximately 10.5000000000
Iteration 3: the square root of 42.0000000000 is approximately 5.2500000000
...
Iteration 39: the square root of 42.0000000000 is approximately 6.4807406985
```

Note that your program should run properly for any real number between 1.0 and 10000.0 (not only for the demo value 42).

The files that you should submit for this part of the assignment are:

- **root.c**: the source code file.
- **root.txt**: the brief text file to explain what the program does and why you chose your method of implementation.
- **root.script**: the typescript file to show that your program works with the numbers shown above, as well as with the number 28.

3 Bonus Problem [5 points]

Improve your program so that it can calculate the n-th root of any value. The value n should be a positive integer input by the user.

For example:

```
Please input a positive number (1 to 10000): 42
Please input the value of integer n (n>0): 5
Iteration 1: the 5th root of 42.0000000000 is approximately 21.0000000000
Iteration 2: the 5th root of 42.0000000000 is approximately 10.5000000000
Iteration 3: the 5th root of 42.0000000000 is approximately 5.2500000000
...
Iteration 39: the 5th root of 42.0000000000 is approximately 2.1117857650
```

To submit, use the same files as in Part 2, i.e. root.c, root.txt, and root.script. The script file should show the output of your program when the user inputs 42 and $n = 5$.

4 Submission

Submission for these files will be similar to last week's assignment. The only difference is that you need to create a directory called **`hw5/`**. Put all the files for assignment 5 in that directory and run the **`/ecelib/bin/turnin`** command to submit your homework. **And please pay attention to any announcements on the course noteboard.**