Discrete Mathematics
Computer Science 2430
Programming project 3

For this assignment, write, test, and execute code to solve the following problems. You should also answer all of the questions.

Here we will be examining the ways that various random numbers interact in systems. Often times, when we are pulling data from physical devices, a small random amount of noise is superimposed on the signal. For this project, we will be looking specifically at two types of random noise, linear random and normally distributed (or Gaussian) random noise. The linear random function takes random variables evenly distributed between two points while the Gaussian distribution is the classic bell curve (http://www.graphpad.com/articles/interpret/principles/gaussian.htm) (don't worry about the mathematics on this page, this is just to illustrate the graph).

Part 1

Use a linear random number generator (such as Java's Math.random()) function to simulate random noise. Write a function that takes a single integer from 0 to 100 as an argument and returns true that percent of the time. Write a driver which calls your function a large number of times and keeps track of what percentage of the time it returns true.

1) If I were to call your random function once with an argument of 70 it would obviously either return true or false, i.e. it would either be true 100% of the time or 0% of the time. If I called it twice it would either return true twice (true 100% of the time), it would return true once and false once (true 50% of the time), or it would return false twice (true 0% of the time). From your data, about how many times would I need to call your function to reliably have the overall percentage returned be about equal to 70% (say somewhere between 69% and 71%)?
Using code from my revised Part1Code.py, if we are trying to acheive that percentage at least once, it took about 10,000 runs.
Using code from CodeForPart1.py original code: If we are trying to get the average percentage of times we acheive the correct values, the number of times that I had to run my function was 20,000. If we assume, which is probably wrong but I should include anyway in case that is what is being asked for, overall percentage means that we do not achieve any percentages outside our boundaries, about 36,500 times. The closest number I could actually achieve percentages only between 69 and 71 was 36,493.

2) How long would I have to run your driver for the overall percentage returned to be between 69.9% and 70.1%?
About 500,000. Closes number I could narrow it down to is 490,000

3) What does this say about the overall accuracy of data with a random linear signal superimposed on it?
Not very good. You have to run it quite a few times to actually get the results you expected.

Part 2

Now we will do much the same with Gaussian random distributions. For this portion you will need to use a Gaurssian random number generator (such as Java's Random.nextGaussian()). (Note: while C# does not implement a normal/Gaussian distribution natively, there are a number of libraries available that can do this for you, see, for example http://www.codeproject.com/KB/recipes/Random.aspx if you want to implement this project in C#).

While a linear random number is evenly distributed over some range, the Gaussian distribution is described by two numbers, the mean (the average value) and the standard deviation (a measure of how spread out the bell curve is). Small standard deviations give very narrow bell curves, broad standard deviations give wide bell curves.

Write a function and/or modify your driver to generate random Gaussian numbers with a specified mean and standard distribution. As an example, if I wanted to generate random numbers with a mean of 70 and a standard deviation of .5, I could code double gausRand = myRand.nextGaussian() * 0.5 + 70. For this round of tests, you should treat a test as true if it returns a value within the specified range and false otherwise. For example, if my function were calculating Gaussian values with a mean of 70 and a standard deviation of .5, it would return true if the number generated were 70.43 but would return false if the number generated were 70.51.

Answer the questions from part 1 using your Gaussian random numbers with a mean of 70 and a standard deviation of .5.

4) **Just two because there is an extremely low chance that we will get anything that strays too far off in either direction (+-1) using a deviation argument of .5 from the mean but you could go with 3 to be a lot more reliable.**

5) **80 is the closest I could narrow it down to.**

6) **This works very well and you can narrow down your results to what you would guess to expect with not too many iterations. In other words, this is a good way to get values around what you want and is a lot more customized. Having never taken a proper statistics course, this is pretty new to me and seems like it could be very useful for some things.**

Answer the questions from part 1 using your Gaussian random numbers with with a mean

of 70 and a standard deviation of 2.

7) **25, which makes sense that we get a relatively much higher number than question 4 because we are using a relatively higher deviation argument, 2.**

8) **About 2500, which makes sense using the same reasonging from question 7; we are trying to get narrower results with a broader spread.**

9) **The results are less accurate than the results when we used a deviation value of .5. This shows that we can be more or less accurate depending on what deviation value we use. We can still acheive pretty random results and they are closer to what we would expect than those from part 1.**

10) Essentially, questions 1, 2, 4, 5, 7, and 8 had you compare the number of times that you needed to run your code to go from an answer accurate to +/- 1% to an answer accurate to +/- .1%. If you did not know the type of noise (or random data) superimposed on a good signal, what rules could you give for helping to determine if your answer is accurate?
If you do not know the type of noise, you can calculate the mean and see how far it strays from the expected value, like we did in this assignment. Calculate the mean of about 100 values at first, and then increase by doubling and see how long it takes to start achieving values you would expect.

Part 3

Now we will use your driver from part 1 to derive some rules about how linear random distributions interact. Use your driver to calculate how often the following two statements return true
(myRandomFunction(70) && myRandomFunction(40))
(myRandomFunction(70) || myRandomFunction(40))
. The first function should be true about 28% of the time (.7*.4), while the second function should be true about 82% of the time (1-(1-.7)*(1-.4)).

11) How many times did you need to run each test to get to an accuracy of about +/- 1%?
(myRandomFunction(70) && myRandomFunction(40))
28%. To get an accuracy of +/- 1% it takes about 8,700 runs.
(myRandomFunction(70) || myRandomFunction(40))
82%. To get an accuracy of +/- 1% it takes about 11,000

12) How many times did you need to get an accuracy of about +/-.1%?
(myRandomFunction(70) && myRandomFunction(40))
28%. To get an accuracy of +/- .1% it takes about 500,000 runs.
(myRandomFunction(70) || myRandomFunction(40))
82%. To get an accuracy of +/- .1% it takes about 700,000 runs.

13) How do these numbers compare to your answers for questions 1 and 2? Why do you think this might be the case?

(myRandomFunction(70) && myRandomFunction(40))

The numbers were very similar for the and function. I think this is because we are essentialy testing the same thing: how long does it take for driver function to return the expected results. We are just performing it on two different percentages but we are performing an and with the same accuracy so it makes sense for it to result in the same percentage.

(myRandomFunction(70) || myRandomFunction(40))

Like the previous question, we got similar results but just a bit higher. I think this is because we are getting more trues because of the or function so we need more noise to get a more defined mean.