Lab Assignment 1
Statistics of Pseudo-Random Numbers

ACS II
Spring 2019

Assigned January 10
Due January 17

Introduction

In this lab, we will explore the use of pseudo-random number generators and the characteristics of the numbers that they produce. The assignment is divided into two parts: in the first part we will use `MATLAB` to perform some statistical analysis on distributions, and in the second part we will look at goodness-of-fit indicators. For this you will need to produce your own software using `C`, `C++`, or `FORTRAN`, and parallelize it using `OpenMP`.

Statistical Analysis with `MATLAB`

To be able to use a lot of statistical analyses quickly, we'll take advantage of some of the functions offered by `MATLAB`. Your deliverables are as follows:

1. Write a code in `MATLAB` that generates n random floating-point values on the interval $(0, 1)$ (hint: try the `rand()` function). Use $n = 100000$.

2. Produce a histogram of the data that you generated (check out the `histogram()` function). What is the mean of the data? Describe other properties of your data as well.

3. In many cases, we prefer to sample from a different distribution, which we can do by mapping values from one distribution to another. Some languages have high-level routines that will do this for you, but we're going to explore how to do this on our own.

   First, recall the cumulative distribution function (CDF) of a probability distribution function (PDF). The CDF evaluated at a point, x, evaluates to the probability that a random value sampled from the distribution will be less than or equal to x. Therefore, the CDF takes on values in the interval [0,1]. We can invert the CDF to map uniformly distributed data into data following the distribution corresponding to the CDF. Graphically, you can think of this as taking uniformly-distributed data on the y-axis in the CDF plot, tracing them to the CDF function, and taking the corrsponding value on the x-axis as the new data. For a distribution

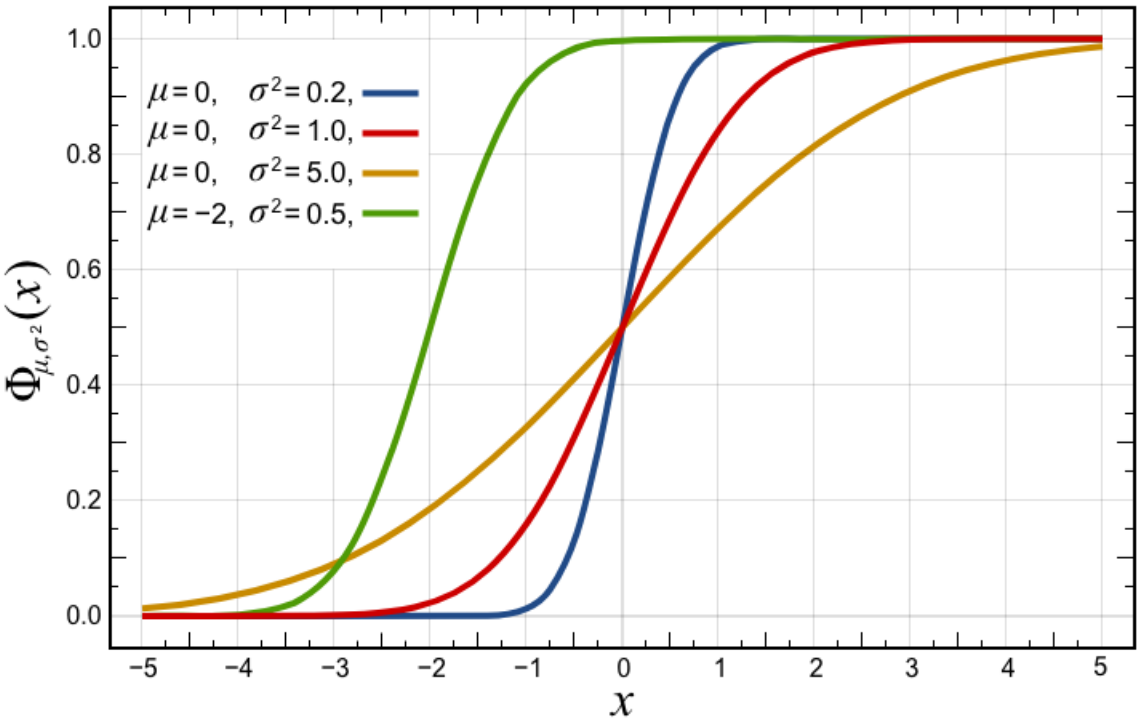

Figure 1: CDFs for some Normal probability distributions

with an analytic PDF this is reasonable, but not so for arbitrary distributions (see Monte Carlo methods).

Transform your random data to look like data drawn from a normal distribution ($\mu = 0$, $\sigma^2 = 2$). Recall that a normal probability distribution is completely determined once we've specified the mean (μ) and the variance (σ). Fortunately for us, `MATLAB` has a built-in function that will compute the inverse CDF of a normal distribution (see `norminv`).

4. Now that we believe our sample has the properties of a normally distributed sample, let's use some statistical measures to test our theory. Create a histogram of the transformed data, and compare what you see with your observations from the histogram of the non-transformed data (which we produced earlier). Also compute the mean and the variance of your sample data (these are the first raw moment and the second central moment, respectively). Do the computed mean and variance match the values used to generate this 'normally distributed' data?

5. Next, let's look at the Quantile-Quantile (Q-Q) plot. This plot compares the sample quantiles to the quantiles we would expect to have if the sample was perfectly drawn from a certain distribution. The `MATLAB` function `qqplot` compares quantiles of the input data to theoretical quantiles of the normal distribution by default. Other probability distributions can be used to compute the theoretical quantiles if specified. If our sample was drawn from a normal distribution, then we expect the points to lie approximately along a straight line. (The straight line will be $y = x$ only if the observed quantiles and theoretical quantiles are from the same distribution. Here we're comparing observed quantiles from a non-standard normal distribution to theoretical quantiles from the standard normal distribution, so the points

should lie in a straight line, but not $y = x$.) `MATLAB` should add a line for you to help judge your data.

6. Not being satisfied to simply say that our data "looks" normal, let us quantify our confidence in the normality of the sample. Here we will use the `kstest` function, which tests a data sample and determines the probability that such a sample would be drawn from a normal distribution. Here, we are actually doing some *hypothesis testing*.

   When we test a hypothesis, we assume a null hypothesis (H_0) to be true and attempt to disprove the null hypothesis using the data at hand. The test will give us a *p-value*, which will be in $[0, 1]$ and which represents the probability of observing our data if the null hypothesis is true. If the p-value is high, then we have no reason to suspect our null hypothesis of being false. If the p-value is low (usually lower than some significance level, say 0.05 or 0.01), then the data we have is very unlikely to have been obtained under then null hypothesis. Therefore, we reject the null hypothesis, in favor of some alternative hypothesis (H_1 or H_A).

   For the Kolmogorov-Smirnov test (`kstest`), what is the null hypothesis? What is the alternative hypothesis? What is the p-value computed when you test your data? What does the result suggest about your data? Are the results conclusive?

Goodness of Fit Testing

Now that we have explored some ways to analyze data using `MATLAB`, you are to write your own code in `C`, `C++`, or `FORTRAN` for this section.

1. Write a code that generates n random numbers in $(0, 1)$. Note that to do this, you may have to do some manipulation and map the random numbers to the correct interval. Use one of the Linear Congruential Generator methods discussed in lecture to generate the random sample. Make sure to set the seed of the random number generator in a way that you don't get the same sequence of random numbers every time the code is executed.

2. We're going to use Pearson's chi-squared test to determine whether these values do indeed follow a uniform probability distribution. To compute the chi-squared statistic, we divide the interval $(0, 1)$ into m subintervals and compute

$$\chi^2 = \sum_{i=1}^{m} \frac{(O_i - E_i)^2}{E_i} \tag{1}$$

   where E_i is the number of points in the sample which we would expect to lie in the i^{th} subinterval and O_i is the actual observed number of points in the sample which lie in the i^{th} interval. For our problem, using equally-sized subintervals, E_i is easy to compute as the number of observations divided equally amongst the subintervals.

$$E_i = \frac{n}{m} \tag{2}$$

   It is also easy to count the number of observations which belong to each subinterval to compute O_i. The number of degrees of freedom of this particular test is $m - 1$.

3. Your code should be printing the value of χ^2 to the screen. The null hypothesis for this test is that the observations follow a uniform distribution. Obtain a p-value for the result of your test, either using software (such as `MATLAB`'s `chi2cdf` function) or using a table of statistical data. Here is a link to a Youtube video that does a nice job of explaining how to interpret the results of your χ^2 test: *https://www.youtube.com/watch?v=HwD7ekD5l0g*

4. Try your code for a small value of n, say $n = 1000$. Once you're convinced that your code is working, modify the code to run in parallel using OpenMP. Specifically, you should be parallelizing the generation of the random numbers, and the counting of observations in each of the m subintervals. This way we can use a large value of n, and divide the increased work among several processes. Once each process has counted the number of its observations in each of the m processes, that information can be aggregated to a single thread, and χ^2 can be computed and printed from that master thread. Be sure to seed the random number generator in a way such that the threads are not all using a copy of the same numbers.

5. Run your code for $n = 1000$, $n = 10000$, $n = 1000000$. Does the outcome of the χ^2 test differ significantly? What does the result of the χ^2 test suggest about your data? For $n = 1000000$, do you see significant speedup when you double the number of processors (say, from 2 to 4)?

Submission and Grading

To get credit for this assignment, you must submit the following information to the lab instructor by 11:59pm, January 17, 2019:

- your `MATLAB` script file
- your parallel χ^2 computation file
- A single `.pdf` file which includes the figures and results from your work, as well as relevant discussion of your observations, results, and conclusions.

This information must be received by 11:59pm, January 17. Upload your material in a compressed folder to Canvas. As stated in the course syllabus, late assignment submissions will be subject to a 10% point penalty per 24 hours past the due date at time of submission, to a maximum reduction of 50%, according to the formula:

*[final score] = [raw score] - min(0.5, 0.1 * [# of days past due]) * [maximum score]*