PHYS511L Lab 3: Binomial Distribution Monte Carlo Simulation

Spring 2016

1 Introduction

The binomial distribution is of fundamental importance in probability and statistics. It has as its limits the Gaussian and Poisson distributions, and itself is directly useful for describing various everyday phenomena. The goal of this lab is to create a computer simulation which generates data distributed according to the binomial distribution (known as a Monte Carlo simulation) and analyze the generated data to better understand the behavior of the binomial distribution in different limiting cases.

1.1 Bernoulli Distribution

The binomial distribution can be constructed by first considering a much simpler distribution, the *Bernoulli distribution.* The Bernoulli distribution governs simple yes-or-no random events, such as flipping a coin. If the outcomes of a Bernoulli random event are given by 0 and 1, then the Bernoulli distribution can be defined as follows:

$$P_{\text{Bernoulli}}(t;p) = \left\{ \begin{array}{ll} 1-p, & \text{for } t=0 \\ p, & \text{for } t=1 \end{array} \right\} \tag{1}$$

So for example, the probability of getting either heads or tails when tossing a fair coin is governed by the Bernoulli distribution with $p = 1/2$. In general, any process where there are two outcomes or only two kinds of outcomes are considered which have fixed probabilities is governed by the Bernoulli distribution. Typically the outcome labelled with $t = 1$ is called *success* and $t = 0$ is called *failure.*

1.2 Binomial Distribution

Let's say we have a process which is governed by the Bernoulli distribution and we are interested in counting the number of successes we encounter within a given sample size n. Defining $x = \sum_{k=1}^{n} t_k$, the distribution of x is given by the binomial distribution,

$$P_{\text{Binomial}}(x;p,n) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, \tag{2}$$

where $x \in \{0, 1, ..., n\}$.

2 Lab Tasks

1. Create ROOT named script which generates binomially-distributed data and fills it into a histogram, taking the number of binomial events N, the Bernoulli sample size n, and the Bernoulli probability p as parameters. Model the problem directly by generating Bernoulli events, counting the number of successes in a sample, and filling this number into a histogram; don't use the shaping approach from last lab.

2. Run the named script for various N, n and p to simulate the following scenarios:

   - Flipping 2, 5, and 10 coins simultaneously ten, one hundred and one million times.
   - Fix $p = 0.2, N = 10000$ and vary n among 10, 100, and 10000. Fit against a Gaussian to see at which n a Gaussian distribution seems to describe the data.
   - Fix $N = 10000$ and $pn = m = 5$, vary n among 10, 100, and 10000. Fit against a Poisson to see at which n a Poisson distribution seems to describe the data.

3 Helpful Information

- 1-D histograms are provided by the TH1D.h header. The Fill method fills data into a histogram. For example:

```
int nbins = 100;
double x_minimum = -5;
double x_maximum = 5;

TH1D demo_hist("demo_hist","demo_hist",
               nbins,x_minimum,x_maximum);

//Fill it with some data:

demo_hist.Fill(0);
demo_hist.Fill(1);
//this data is outside the range and isn't filled:
demo_hist.Fill(100);
```

- Histograms have a method called Fit which fits them against a formula or a pointer to a TF1 function object.

- The header TMath.h can be included to provide the Gaussian distribution via TMath::Gaus.

- The Poisson distribution is given by

$$P_{\text{Poisson}}(x; m) = \frac{m^x}{x!} e^{-m} \tag{3}$$

where m is the mean of the distribution. Clearly the Poisson distribution as defined above can only take non-negative integers as x arguments, which causes problems when trying to plot with ROOT. The Γ function is an extension of the factorial into the complex plane, and is provided by TMath::Gamma. The Γ function satisfies

$$\Gamma(x) = (x-1)! \;\; \forall x \in \mathbb{N}, \tag{4}$$

so if we add a normalization parameter and use Γ instead of factorial we can fit data with a Poisson distribution:

```
//ROOT uses [param_number] in a formula string to denote a fit
//parameter
TF1 *poisson
= new TF1("poisson",
          //We can split long strings like this:
          "[0]*TMath::Power([1],x)*"
          "TMath::Exp(-[1])/TMath::Gamma(x+1)",
          0,
          1e9);
//So if we had a histogram h ready to be fitted, we could do
h.Fit(poisson);
//Note that if your histogram has very large x values, this fit
//function may fail and generate nan (not a number), so make
//sure your histogram ranges make sense.
```

- Random number generation is provided by TRandom.h, and there is a global random number generator pointer object gRandom provided by this header. gRandom->Uniform(0,1) generates a random number between 0 and 1 with uniform probability. So, a function that throws a Bernoulli variable might look like this:

```
int throw_bernoulli(double p) {
  double uniform = gRandom->Uniform(0,1);
  if(p > uniform) {
    return 1;
  }
  else {
    return 0;
  }
}
```

- Try to structure your code according to how the problem is conceptually. For example, you could have a function throw_bernoulli which throws a Bernoulli variable, and use this to define throw_binomial which would throw a binomial variable. Finally, you could have a function bin_binomial which would create a histogram of multiple binomial trials.