`PHSX815_Project3`: Biased Coins

Yoni Brande

Apr 4, 2021

1 Abstract

In previous examples, we examined the problem of testing two coins and determining whether one of them was biased away from a 50/50 chance for landing on either Heads or Tails. Here, we will be given some data, a sequence of coin flips, and given this data tasked with finding the true coin probability. We will express the likelihood function $\mathcal{L}(p|X)$ (where p is the probability of landing on heads) as some other function $f(p) \propto \mathcal{L}$, and use the function minimization strategies we've learned in class to find the value of p that minimizes $f(p)$ such that $\mathcal{L}$ is maximized. With this measured p value, we then use the Neyman construction to estimate a $\pm 1\sigma$ confidence interval on p, and compare to a true value used to create the data.

2 Introduction

Coin flipping is an extremely ancient method of hardware number generation, and has remained popular for many purposes until the present. In most cases, a coin is a flat disk of metal with two distinctly marked sides, and is flipped into the air using the fingers. The coin rotates in the air, subject to the forces imparted on it by the flipper, making it difficult or impossible to predict which face the coin will land on (assuming the coin is fair). Ideally, these are modeled by processes with equal probabilities for each face, although in practice, coins are impossible to perfectly uniformly manufacture and flip without bias, so a real-world coin may deviate from the 50/50 probability to some extent. Over a large enough number of trials, we expect that the distribution of heads and tails will approach the "true" probabilities of the coin, and will use a set of simulated data to determine whether or not a particular simulated coin is biased.

3 Minimization Methods

Function minimization is the process of finding the minimum value of some function over a particular interval. For convex functions, assuming the interval encompasses all of space, there is a single global minimum. For non-convex functions, there may be many local minima, making the problem of finding any specific minimum difficult without providing initial guesses and bounding the function. Methods exist for 1, 2, and higher dimensional problems, and use varying strategies to find their global or local minima. For this project, we will find a 1-dimensional function to help estimate our maximum likelihood, and we will use Brent's method as implemented in `scipy.optimize` to minimize our function.

Brent's method performs inverse parabolic interpolation to find minima. With two points defining an interval, and a third somewhere inside the interval, the method picks which direction to shrink the interval by making function evaluations at each of the three points and weighting them. This is then subtracted from the point inside the interval to give a new endpoint, either greater than the left interval endpoint or less than the right interval endpoint. Doing this iteratively shrinks the interval until the algorithm finds a point at which the descent is within some determined numerical tolerance and the midpoint is then the function minimum.

4 Maximum Likelihood Estimation for a Bernoulli Process

The various outcomes of a coin flip can be modeled by using a Bernoulli distribution.

$$P(x|p) = p^{x}(1-p)^{1-x}$$

We can write the likelihood function for a series of coin flips as:

$$\mathcal{L}(p) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i}$$

And then the log-likelihood function as:

$$\begin{aligned} log(\mathcal{L}(p)) &= log\left(\prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i}\right) \\ &= \sum_{i=1}^{n} log(p^{x_i}(1-p)^{1-x_i}) \\ &= \sum_{i=1}^{n} [x_i log(p) + (1-x_i) log(1-p)] \end{aligned}$$

In order to find the value of p which maximizes the normal likelihood function, we can define the function $f(p)$:

$$f(p) = log(\frac{1}{\mathcal{L}(p)}) = log(1) - log(\mathcal{L}(p))$$

$$\begin{aligned} f(p) &= -log(\mathcal{L}(p)) \\ &= -\left(\sum_{i=1}^{n} [x_i log(p) + (1-x_i) log(1-p)]\right) \\ &= -\left[\xi log(p) + (n-\xi) log(1-p)\right] \end{aligned}$$

where $\xi = \sum_{i=1}^{n} x_i$.

Now, we can minimize $f(p)$ by one of our minimization methods and this will yield the value of p that will give us our maximum likelihood.

4.1 Simulated Data

Our simulated experiment is incredibly simple. As we only consider a coin flip, we can just select a random probability p_{true} for our true Heads probability, giving us $1 - p_{true}$ for our true Tails probability. Properly, we probably shouldn't look at the true values for the final experiment, but knowing them will be useful in development. With these random probabilities, we can flip our biased coin N times to get our simulated data. We then print the data to a file to be read by the minimization code later on. For the purposes of this experiment, we will conduct four trials with differing $N = 100$, 1000, 10000, and 100000, each with the same true heads probability.

5 Minimization

The process for minimizing our $f(p)$ is now also very simple, given our simulated data. If our x_i flips are either 0 for tails or 1 for heads, then we can fill in the summation terms with $\xi = N_{heads}$ and $n = N_{flips}$. This is a one-dimensional function in p, so when we run a minimization method on this, we can just rely on the minimization routine to test varying values of p until it finds one that minimizes the function, which in turn maximises our likelihood function. This is then our best guess for the true probability of our simulated coin landing on heads. In this case, we use `scipy.optimize.minimize_scalar` to minimize our function of probability over the interval [0, 1]. This is fast and accurate, and since our estimator function is convex, not subject to false local minima. From this minimized value of p, we then proceed to estimating a confidence interval on this value. We will need to use the Neyman construction, as previously demonstrated in the homework. Effectively, we create mass functions for N binomial draws at varying values for the probability, then pick the value closest to our minimized value, and find the standard deviation of that distribution. In this case we adopt a confidence interval of $\pm 1\sigma$.

6 Analysis

In practice, we need to use some specific functions here, and modify our code slightly from the previous Neyman construction homework. As we want to investigate the dependence of our estimates on the total number of coin flips we do, we may be dealing with quite large datasets. If initially we made N coin flips, we will need to draw at least N^2 new Bernoulli random numbers in order to fill in our two-dimensional histogram, and that's only if we decide to do a single "experiment" for each p value. In this case, we decide to forgo using the Bernoulli function we wrote in class, and instead adopt the numpy binomial random number function which removes the need for N^2 Python function calls and significantly speeds up the process. Otherwise, when we hit ~ 100000 iterations, the code runs long enough that execution needs to be manually canceled.

Armed with this efficient method, we can now make our Neyman construction plot. We split the possible range of probabilities into N equally spaced values, starting at 0 and ending at 1. For each p value in this list, we use the `numpy.random.binomial` method to draw N random numbers. We add these values to a 2-dimensional histogram with $N/10$ bins, and save the bins to help find the confidence interval for the measured p value.

In order to find the confidence interval for our p value, we take our minimized p value, and extract the row histogram from our 2-dimensional histogram for which our p is in the bin. As we are looking for $\pm 1\sigma$ confidence intervals, we find the mean p by taking the weighted average of the bin midpoints of the histogram, and the standard deviation by taking the square root of the weighted average of the squared difference of the bin midpoints and the mean p. We can then say that our $p_{meas} = \mu \pm 1\sigma$. Figs. 1, 2, 3, 4 show the analysis plots for N = 100, 1000, 10000, and 100000.

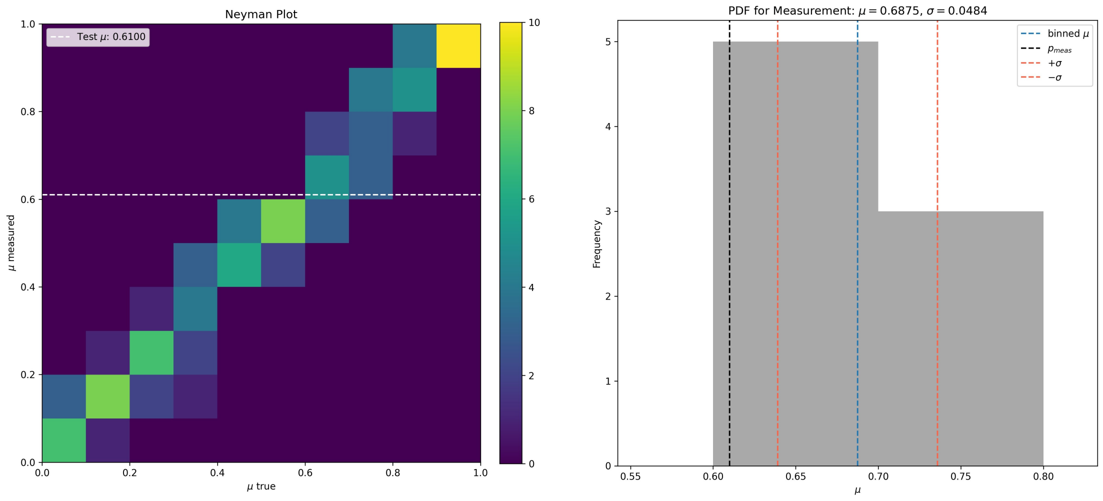

Figure 1: Results for $N = 100$.

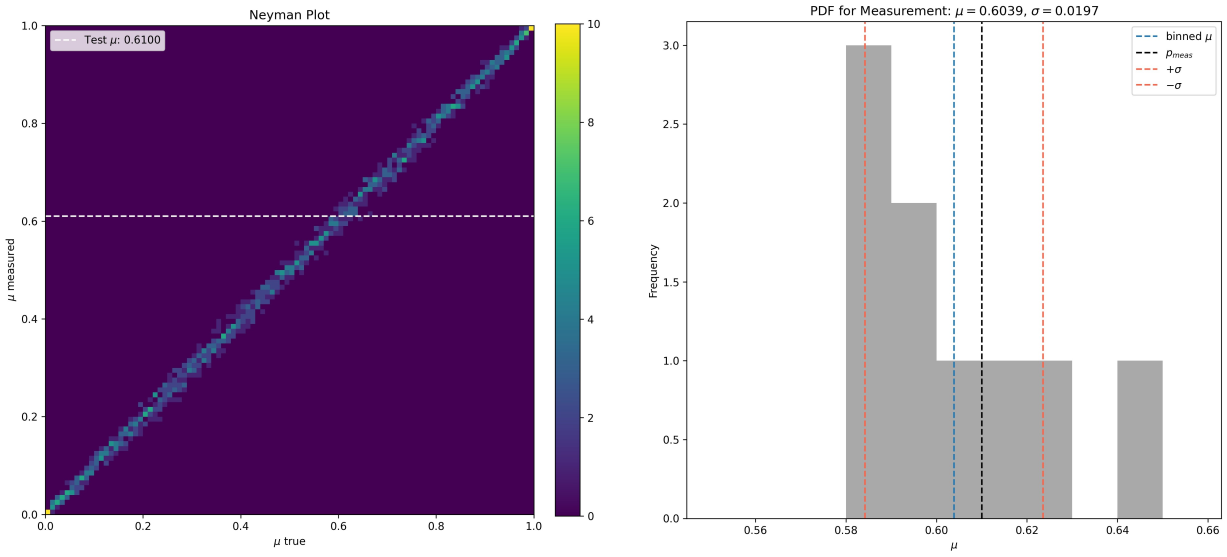

Figure 2: Results for $N = 1000$.

7 Conclusions

For our four trials, we find the following results:

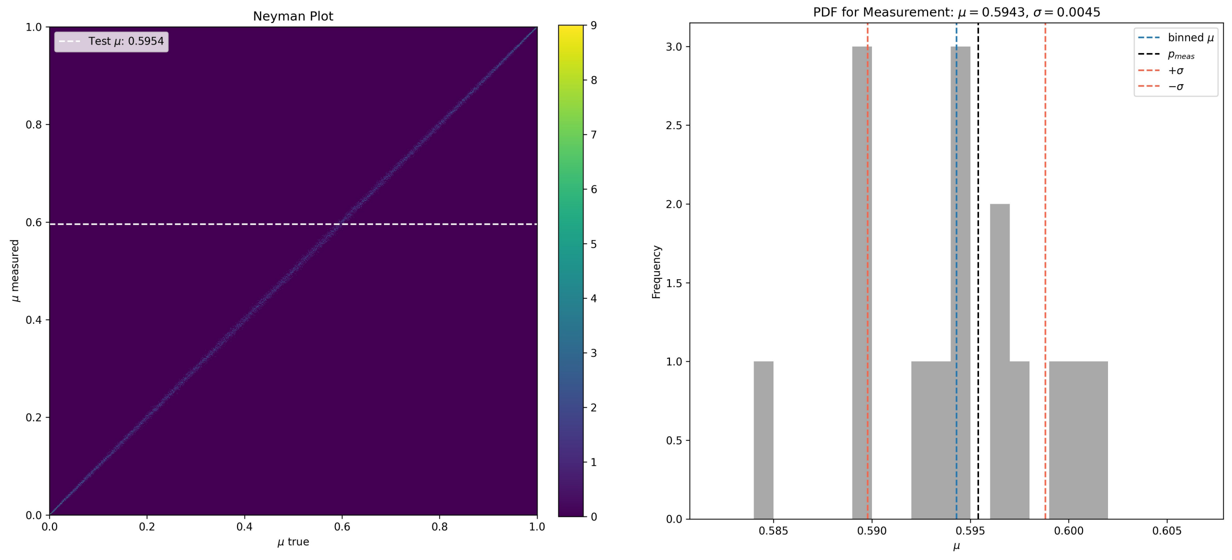

Figure 3: Results for $N = 10000$.

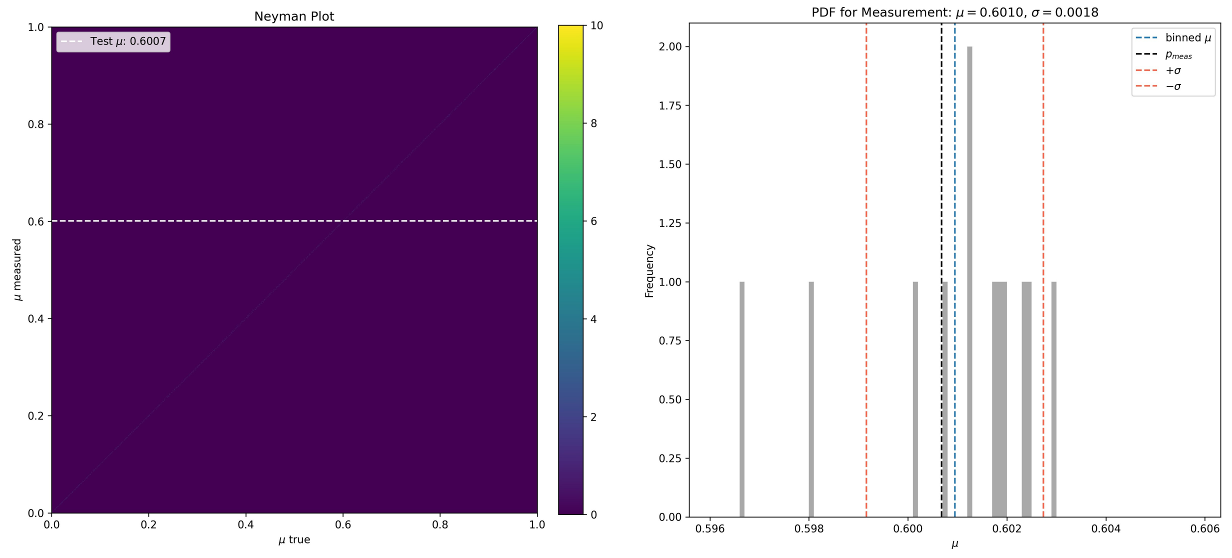

Figure 4: Results for $N = 100000$.

$p_{minimized}$	$\mu_{measured}$	σ
0.610	0.6875	0.0484
0.610	0.6039	0.0197
0.5954	0.5943	0.0045
0.6007	0.6010	0.0018

None of these values are consistent within error to $p = 0.5$, so we are confident in determining that this coin is biased, with approximately a 60% chance of landing on heads.

As we can see from the plots and results, in nearly every case our measured p is consistent to within 1σ of our minimized p. When this isn't the case, as in the first run with only 100 coin flips, this is due to the bin width of the histograms. Since our bin widths scale inversely with the number of coin flips, we find that at N=100 we have the widest confidence interval, ± 0.0484, but since our bins are so coarse, we found μ to be far away from the minimized p. As N increases, our results improve, and by $N = 100000$ we find that μ and p differ by only 0.0003, well within the confidence interval of ± 0.0018. We also note that as N increases, the weights of our histogram bins decrease overall. By $N = 100000$, all the bins in our area of interest have weight 1, except a single one near the middle with weight 2. Perhaps this indicates we have too many bins, and a more intelligent bin width selection process might improve our results.

Either way, we find that function minimization to find the maximum likelihood estimator of our coin flips is a good way to determine whether a coin is biased.