Outline: PRNGs and Permutations

Kellie Ottoboni

Draft May 13, 2016

- The first order of business is to investigate how R and Python generate pseudo-random numbers and what algorithms they use to sample, permute, shuffle, etc. This may require looking at the raw code since the R documentation doesn't say how `sample` works.
- **Permutation testing with one sample:**
  We have N observations and we're doing some permutation test with them. To approximate the null distribution, we want to sample uniformly at random from all $N!$ permutations of the observations. Is it possible to obtain all of these permutations, or are we constrained by the period of the PRNG?
  - I am totally open to changing notation! This is temporary.
  - Suppose the period of the PRNG is $\mathcal{P}$.
  - Suppose the permutation algorithm takes K operations.
  - If $K \equiv 0 \mod \mathcal{P}$, then the PRNG will start over at some point. If $\mathcal{P}/K < N!$, then we can't reach all possible permutations. Otherwise, we're in good shape and we will just start to repeat permutations before the PRNG reaches the end of its period.
  - What happens if $\mathcal{P}/K < N!$ but K does not divide $\mathcal{P}$?
- What is the "best" way to do permutations to avoid reaching the end of the period? There are two issues at tension: the period of the PRNG and the computational complexity of the PRNG and shuffling algorithm. We want to balance computational efficiency with correctness.
- What happens if we generate pseudo-random numbers in a distributed fashion? Obviously one has to set the seed differently for each thread, but does this improve the risk of repeating?