499 Report on Random Sampling

Brenna Cosio

May 13, 2022

1 Background and Relevant Information on Numerical Semigroups

A numerical semigroup is a subset, S, of the non-negative integers that is closed under addition. This subset of numbers can be described using the form $x_1n_1 + x_2n_2 + ... + x_tn_t$ where each x_i is a positive integer, and each n_i is a fixed number. The set of n_i's are called the generators of the semigroup. Generally, the way that semigroups are described is using their generators, in the form $< n_1, n_2, ..., n_i >$. For example, all positive integers can be generated with $< 1 >$, where each member of this semigroup can be described using $1 * x_1$. A more interesting example is the semigroup generated by $< 6, 9, 20 >$, referred to as the McNuggets semigroup, where each member of the semigroup is of the form $6x_1 + 9x_2 + 20x_3$. The first several elements of the semigroup are $\{0, 6, 9, 12, 15, 18, 20, 21,...\}$. Semigroups have many interesting properties, but the ones we focused on were the embedding dimension, or the number of generators of a semigroups, the Frobenius number, or the largest number excluded from the semigroup, and the genus or number of gaps, which is the total number of integers not included in the semigroup. Given a multiplicity m, the Apéry set is the set of the first numbers in the semigroup that fills each congruence class mod m.

Thus, when one wants to work with random numerical semigroups, one must randomly produce generators. One method of doing so in a systematic way, is referred to as the Erdős-Rényi type model (ER-type model). Where you fix a non-negative integer M, and a probability p, between 1 and 0. Then, in the interval $(0, M]$, you determine independently whether or not to include an integer as a generator with probability p. From there, one must determine whether each random number can be made using a combination of the other generators. For example, if the random generators chosen were $5, 10, 13, 20, 21$ the actual generators of the set would be $< 5, 13, 21 >$ since 10 and 20 are both multiples of 5.

2 Methods

We made alterations to the ER-type model, where instead of being given M as a maximal number, we took in m as a minimal generator, referred to as the multiplicity. We still took p as a probability of choosing each number as a generator. We then choose m numbers greater than or equal to m, with probability p. Those numbers were treated as the set of generators and were then put through the circle of lights algorithm which calculates the Apéry set of a semigroup given its generators. We then stored the minimal numbers from the Apéry set, that is the numbers that cannot be created by any integer combination of the others. These minimal numbers were our generators.

For smaller probabilities, the integers that are selected as generators became very large and spaced out. So, we then made further modifications by allowing for generators more quickly and closer together by increasing the probability every m numbers. We allow for two methods of increasing probability: through addition and multiplication. The user can pass a value between 0 and 1 that they want the probability to increase by for every m numbers. For example, if the user passed a probability of $\frac{1}{6}$, a multiplicity of 4, and an increment of $\frac{1}{6}$, the probability would increase every 4 numbers by $\frac{1}{6}$. So at 8, the probability of selecting a number is $\frac{2}{6}$, and at 24 the probability increases to $\frac{6}{6}$ or 100% chance of selecting a number. Similarly, if the user chooses to increment at a rate higher than 1, then every m numbers the probability increases by being multiplied by the increment.

3 Code Description by Function Name

3.1 er_COL

Takes m, *prob*, and *inc* as parameters. m is the multiplicity of the set, the set lower bound. *prob* is the probability of choosing a number. *inc* is the amount that we want the probability to change for each m numbers. Slight alterations to the Erdős-Rényi model, where we choose members of the semigroups with increasing probability as the numbers increase. *Returns* the original Apéry set, called $origAP$ and the minimal Apéry set, called $minAP$.

3.2 AperySetCOL

Takes m and $setOfn$ as parameters. m is the multiplicity. $setOfn$ is the set of numbers chosen at random from er_COL. Using the Circle of Lights algorithm detailed in PAPER with slight alterations to be able to save the

minimal generators of the Apéry set, the generators, as well as the Apéry set. *Returns* the original Apéry set, called a and the minimal Apéry set, called $minA$.

3.3 massTesting

Takes m, inc,$prob$, and $numTrials$ as parameters. m is the multiplicity. inc is the amount that we want the probability to change for each m numbers. $prob$ is the probability with which we want numbers to be picked. $numTrials$ is the number of different generators we want to produce with the same m, inc, and $prob$ values. Calculates the frobenius number, genus, and embedding dimension for each set of generators chosen from er_COL call, stores generators in $specRandArray$. *Returns* $frobNumsSpecArray$, $genusSpecArray$, $embedSpecArray$, and $specRandArray$. $frobNumsSpecArray$ is the Frobenius number for a set of generators stored at corresponding index in $specRandArray$. Similarly, $genusSpecArray$ and $embedSpecArray$ store the genus and embedding dimension for generators stored at a corresponding index in $specRandArray$.

3.4 testHome

Takes m, $prefix$, $prefixGen$, $prefixConst$, inc , $prob$, $numTrials$ as parameters. m is the multiplicity. $prefix$ is the filename that you want to save the Frobenius number, genus, and embedding dimensions to. $prefixGen$ is the filename you want to save the generator array to (because the dimensions of this array are different, it cannot be saved to the same file). Similarly, $prefixConst$ is the filename you want to save the $prob$, inc, and $numTrials$ values to for future reference. inc is the amount that we want the probability to change for each m numbers, if no value is passed for inc, default is 0. $prob$ is the probability with which we want numbers to be picked, if no value is passed, then a random probability is assigned to a random float between 0 and 0.06. $numTrials$ is the amount of different sets of generators we want to produce with the same m, inc, and $prob$ values. Calls $massTesting$, and saves $frobNumsSpecArray$, $genusSpecArray$, $embedSpecArray$, $specRandArray$. To two csv files, with names decided by $prefix$ and $prefixGens$.

3.5 wilf_check

Takes $frobSpec$, $genusSpec$, $genSpec$, $embedSpec$, and $numTrials$ as parameters. Checks to see if what generators have interesting behavior for the inequality$F(S)+1 \leq e(S)(F(S)+1-g(s))$. If yes, either stops completely, or stores the left and right side of the inequality. *Returns*; *ratio*, a boolean array that states whether the left and right sides are equal (true) or unequal (false). Also outputs scatter plot with embedding dimension on the x axis and (right side of wilf ratio - left side) on the y axis.

3.6 readIn

Takes m,$prefix$, $prefixGen$, and $numTrials$ as parameters. . m is the multiplicity. $prefix$ is the filename that the Frobenius number, genus, and embedding dimensions are saved to. $prefixGen$ is the filename the generators are saved to. $numTrials$ is the amount of different sets of generators that were produced. Uses Pandas to read the CSV from the python file and passes information to wilf_check. *Outputs*: array containing the generators where the Wilf's inequality produced an equality, histogram for frequency of occurrence of numbers in generators, Frobenius number, genus, and embedding dimension.

4 Usage and Sample Output

Here is an example of how to use both files.
In the $ErdosRenyi\_Testers$:

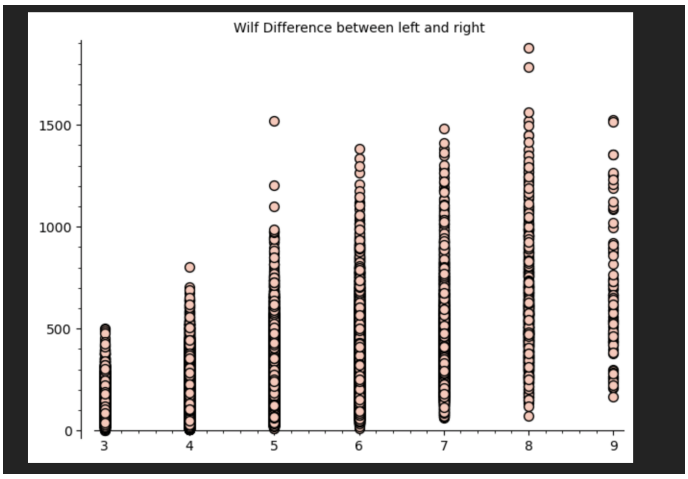
```
probstore = (1-random.uniform(0,.6))/10

testHome(43, ("datamath/Spec" + str(43)),( "datamath/Spec" + str(43) + "trialGens"), ( "datamath/Spec" + str(43) + "trialConst"), .08,
probstore, 10000)
testHome(6, ("datamath/Spec" + str(6)),( "datamath/Spec" + str(6) + "trialGens"), ( "datamath/Spec" + str(6) + "trialConst"), 1.16, probstore,
100000)
testHome(9, ("datamath/Spec" + str(9)),( "datamath/Spec" + str(9) + "trialGens"), ( "datamath/Spec" + str(9) + "trialConst"), numTrials =
100000)
```

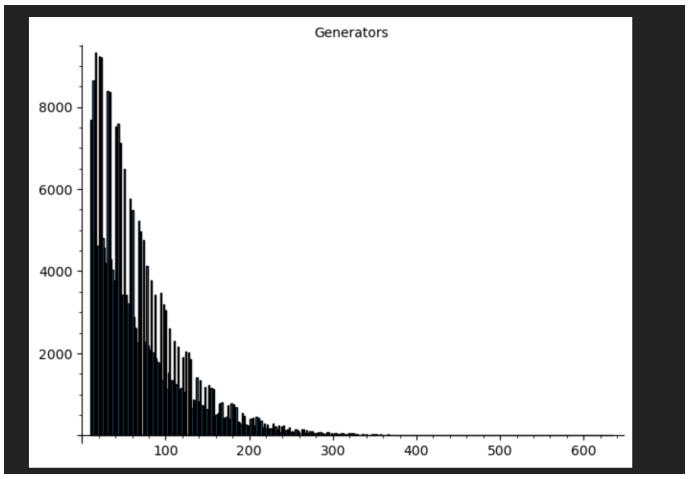
Since inc,$prob$, and $numTrials$ are all optional arguments, it is possible to pass any or no combination of these parameters.
In the $ErdosRenyi\_Visualization$:

```
readIn(9, ("datamath/Spec" + str(9)), ("datamath/Spec" + str(9) + "trialGens"), 100000)
```

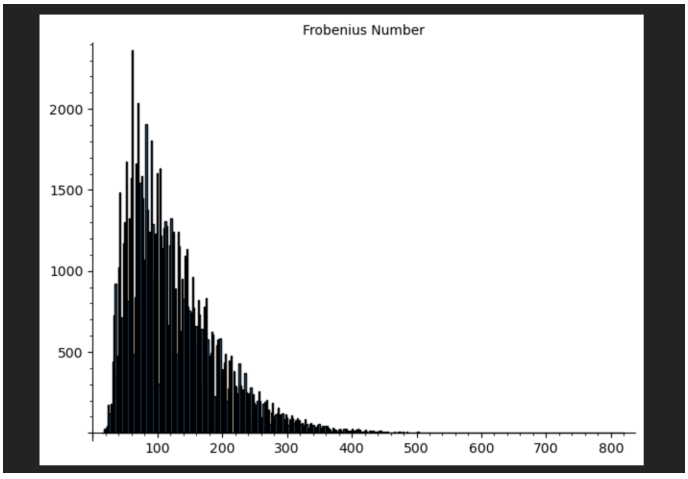
To use the $readIn$ method easily, just pass the same prefix values from $ErdosRenyi\_Testers$ and retrieve the $numTrials$ from the corresponding csv file that stores the constants for that set of trials. Here is the output from this particular $readIn$:



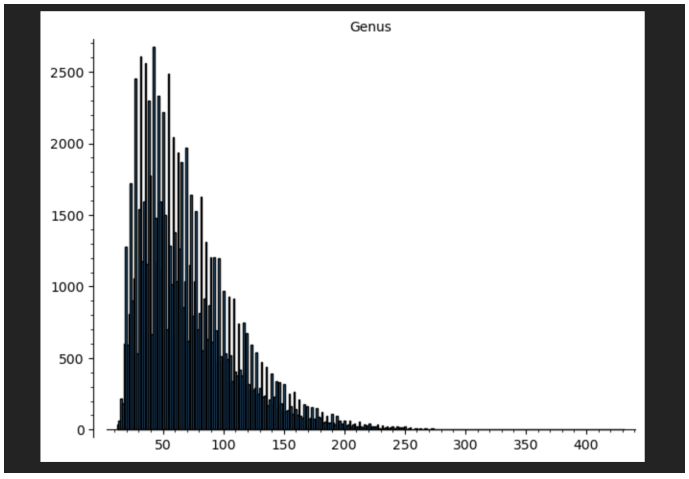
(a) Embedding Dimension vs Wilf Difference



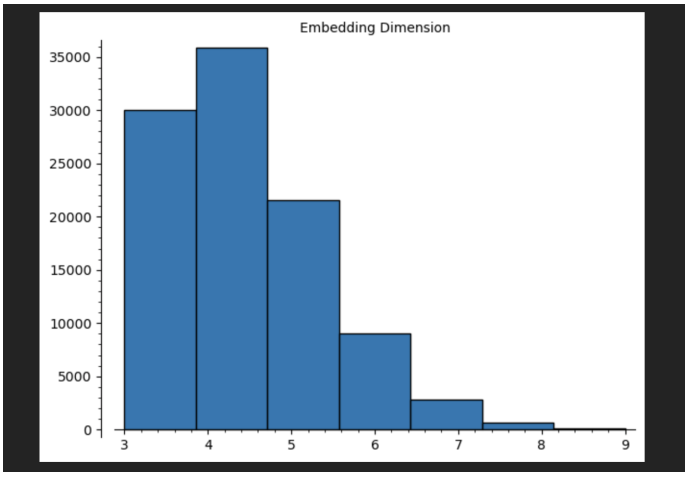
(b) Histogram of Generators



(c) Histogram of Frobenius Number



(d) Histogram of Genus



(e) Histogram of Embedding Dimension