Randomly Generated Numerical Semigroups

By
Zachary J. Spaulding

Abstract.

Numerical semigroups are subsemigroups of $\mathbb{N}$, that is, subsets of the natural numbers closed under addition. In this paper, we will look at a particular way to randomly generate numerical semigroups given certain parameters. We will then discuss a new software package which generates and plots data about these objects. Examples of usage will be highlighted.

Contents

1 Introduction **3**

2 Background **5**
2.1 Semigroups . . . 5
2.2 Numerical Semigroups . . . 6
2.2.1 Generating Set . . . 6
2.2.2 Properties of Numerical Semigroups . . . 6
2.2.3 Apéry Set . . . 7

3 Algorithms for Randomly Selecting Numerical Semigroups **7**
3.1 `GenerateNumericalSemigroup()` . . . 8
3.2 `GenerateNumericalSemigroupAp()` . . . 9

4 External Packages **11**
4.1 GAP System . . . 11
4.2 SQL . . . 12
4.3 Sage . . . 13

5 Usage **13**
5.1 `RunExperiment()` . . . 13
5.2 `RunExperimentP()` . . . 13
5.3 Generating Data . . . 14
5.4 Storing Data . . . 15
5.5 Plotting Data . . . 16
5.6 Example . . . 17
5.7 Example Data . . . 18

References **19**

1 Introduction

Numerical semigroups are subsemigroups of $\mathbb{N}$. We are interested in randomly generating these algebraic objects in order to better understand their expected behavior. In a recent paper [1], the average behavior of random numerical semigroups was studied and relatively loose theoretical bounds were given for the expected values of certain attributes. In addition to the theoretical approach used in [1], random semigroups can be studied via computations and experiments. To find more detail about the bounds for certain attributes, we created rns-db-plot to randomly generate numerical semigroups and plot their behavior.

Our software generates random numerical semigroups in the following way. Fix a probability $p \in [0, 1]$ and upper bound $M \in \mathbb{N}$ as input. For all integers $n \in 1, \ldots, M$, we choose with probability p whether or not to include n as a generator of the semigroup. We

then create a database which stores data about large collections of numerical semigroups generated in this fashion. This database can then be used to plot expected invariant values of a random semigroup given fixed parameters via Sage. Example plots of average Frobenius number (Definition 2.6) and average number of minimal generators (Definition 2.4) are found in Figure 1 and Figure 2, respectively.

Figure 1: Average Frobenius Number

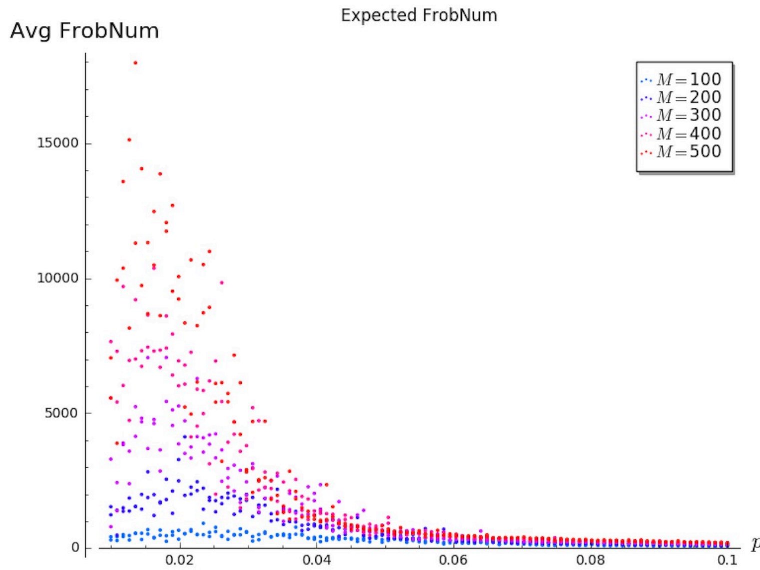
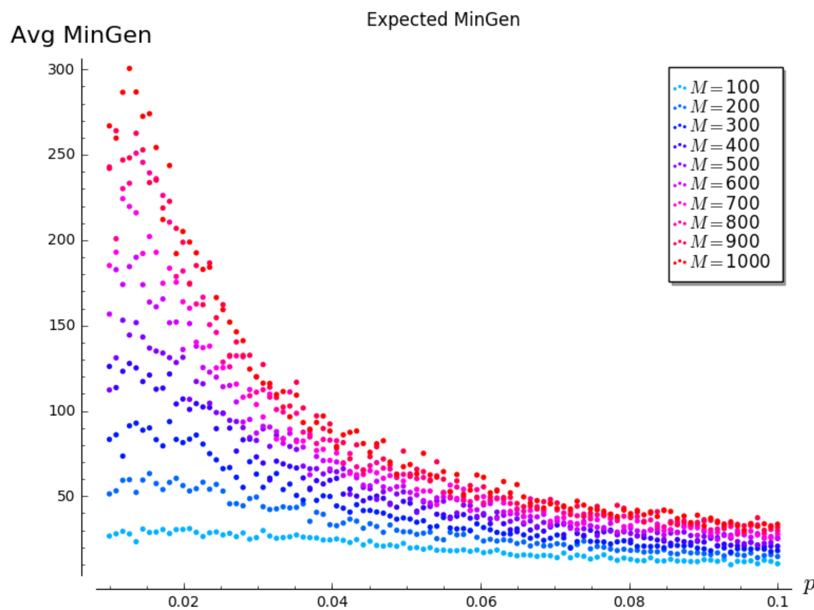


Figure 2: Average Number of Minimal Generators



In this paper, we demonstrate the usage of rns-db-plot, as well as the dependent packages GAP, SQL, and Sage. We will begin by introducing necessary definitions, theorems,

and examples to discuss numerical semigroups in Section 2. In Section 3, we will discuss the GAP system for numerical semigroup computations. In the following section, we describe two algorithms for randomly generating numerical semigroups, the second of which simultaneously generates the semigroup's Apéry set (Definition 2.8). This function is intended to be used when calculating invariants related to the Apéry set. In the final section, we describe usage of rns-db-plot along with a full example.

2 Background

We begin with some preliminary definitions and examples to build familiarity with the relevant algebraic objects and their attributes.

2.1 Semigroups

We begin with our attention on semigroups which are groups without the requirement of inverses. More, precisely, we have the following definition.

Definition 2.1. A pair $(M, *)$ is said to be a *semigroup* if M is a set and the following is satisfied:

$$* : M \times M \to M \text{ is an associative binary operation on } M$$

That is, a semigroup is a set closed under a binary operation. We often refer to the semigroup $(M, *)$ as M when the binary operation is clear from context.

Example 2.2. Consider $(\mathbb{N}, +)$, the set of nonnegative integers under addition. We know that addition of integer satisfies associativity. Moreover, 0 acts as the identity element since for all $n \in N$, we have $0 + n = n + 0 = n$. Hence, $\mathbb{N}$ is a semigroup.

As with other algebraic objects, we can restrict our view to a subset of a given semigroup which also has a semigroup structure.

Definition 2.3. We say that $(N, *)$ is a *subsemigroup* of $(M, *)$ if $N \subseteq M$, N contains the idenitity e of M, and N is closed under $*$; that is,

$$n_1, n_2 \in N \implies n_1 + n_2 \in N.$$

Example 2.4. Going back to the semigroup $\mathbb{N}$, we can consider the set of even nonnegative integers

$$E := \{2n \mid n \in \mathbb{N}\} \subseteq \mathbb{N}.$$

Here, 0 is even, so $0 \in E$. Moreover, the sum of even integers is even, so E is closed under addition. Thus, E is a subsemigroup of $\mathbb{N}$.

2.2 Numerical Semigroups

Definition 2.5. A *numerical semigroup* is a subsemigroup of $\mathbb{N}$ which is cofinite in $\mathbb{N}$, that is, $|\mathbb{N} \setminus S| < \infty$.

Example 2.6. Consider the following semigroup under addition:

$$\begin{aligned} M &= \{3a_1 + 8a_2 \mid a_i \in \mathbb{N}\} \\ &= \{0, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, \dots\}. \end{aligned}$$

Note that the complement of M is $\mathbb{N}$ is finite, that is

$$|\mathbb{N} \setminus M| = \Big|\{1, 2, 4, 5, 7, 10, 13\}\Big| < \infty.$$

Since M is a subsemigroup of $\mathbb{N}$ and cofinite in $\mathbb{N}$, M is a numerical semigroup.

2.2.1 Generating Set

Definition 2.7. A *generating set* of a numerical semigroup S is a set $G = \{n_1, n_2, \dots, n_p\} \subseteq \mathbb{N}$ for which every element of S is a non-negative integer combination of elements of G. That is,

$$S = \langle G \rangle = \langle n_1, n_2, \dots, n_p \rangle = \Big\{a_1 n_1 + a_2 n_2 + \dots + a_p n_p \mid n_i \in \mathbb{N}\Big\}.$$

A generating set G is *minimal* if $|G| \leq |H|$ for all generating sets H of S.

Example 2.8. Continuing Example 2.3, we can write

$$\begin{aligned} M &= \{3a_1 + 8a_2 \mid a_i \in \mathbb{N}\} \\ &= \{0, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, \dots\}. \end{aligned}$$

As such, $M = \langle 3, 8 \rangle$, and so $\{3, 8\}$ is a generating set for M. In fact, it is the minimal generating set for M. Note that $\{3, 8, 14\}$ is also a generating set for M. We can see that this generating set is not minimal because it contains a redundant element, namely $14 = 3 + 3 + 8$.

Remark 2.9. It will not be shown, but we will make use of the fact that every numerical semigroup has a unique minimal generating set and that this set is always finite.

2.2.2 Properties of Numerical Semigroups

Definition 2.10. The *gaps* of a numerical semigroup S are the natural numbers not included in S, that is, the elements of $\mathbb{N} \setminus S$. The *genus* of S is $g(S) = |\mathbb{N} \setminus S|$.

Definition 2.11. The *Frobenius number* of S is $F(S) = \max(\mathbb{N} \setminus S)$. That is, the Frobenius number is the largest gap of S.

2.2.3 Apéry Set

Definition 2.12. The *Apéry set* of $n \in S$ is

$$Ap(S, n) = \{s \in S \mid s - n \notin S\}.$$

When n isn't specified, we define $Ap(S) = Ap(S, n_1)$ where n_1 is the smallest non-zero element of S. We have an equivalent definition of Apéry set which will be used throughout this paper.

Lemma 2.13 (4, Lemma 1.4)**.** *Let S be a numerical semigroup and n be a nonzero element. Then*

$$Ap(S, n) = \{0 = w(0), w(1), \ldots, w(n-1)\},$$

where $w(i)$ is the least element of S congruent with i modulo n.

Example 2.14. The numerical semigroup:

$$\begin{aligned} S &= \langle 5, 8, 9 \rangle \\ &= \{0, 5, 8, 9, 10, 13, 14, 15, 16, 17, \ldots\}. \end{aligned}$$

has gap set

$$\mathbb{N} \setminus S = \{1, 2, 3, 4, 6, 7, 11, 12\}.$$

Hence, the Frobenius number of S is 12. Lastly, let's look at the Apéry set of S. Lemma 2.1 tells us that we should expect five elements in $Ap(S, 5)$. The minimal generators and 0 are always elements of the Apéry set. Obviously, $0 - n \notin \mathbb{N}$. Suppose $k \neq 5$ is a minimal generator such that $k - 5 \in S$. Then $k - 5$ could replace k as a generator while still generating S, but this contradicts minimality. Hence, minimal generators (other than the smallest) are always in the Apéry set. So $0, 8, 9 \in Ap(S)$. Also, note

$$16 - 5 = 11 \notin S \quad \text{and} \quad 17 - 5 = 12 \notin S.$$

So

$$Ap(S, 5) = \{0, 8, 9, 16, 17\} = \{0, 16, 17, 8, 9\}.$$

The second expression above shows the elements in order modulo 5.

Remark 2.15. Some properties of a numerical semigroup can be easily calculated from its Apéry set. In particular, the Frobenius number, the number of gaps, and the number of minimal generators can be calculated in linear time given the Apéry set.

3 Algorithms for Randomly Selecting Numerical Semigroups

This section will discuss two new functions for generating numerical semigroups. Both methods, `GenerateNumericalSemigroup()` and `GenerateNumericalSemigroupAp()`, generate semigroups according to the same random model, but the second method also calculates the semigroup's Apéry set while generating the semigroup. Because certain numerical semigroup invariants are calculated via the Apéry set, this method is more efficient when the goal is to then calculate such invariants.

As referenced earlier, we consider the following method of randomly generating semigroups, used in [1], and then study the expected attributes of semigroups generated in this fashion. Our generation algorithm is outlined as follows:

1. Fix input parameters $M \in \mathbb{N}$ and $p \in [0,1]$.
2. Initialize $\mathcal{G} = \emptyset$ to be a generating set for the semigroup.
3. For each $n \in \mathbb{N}$ where $n \leq M$, independently choose with probability p whether or not to include n in the generating set $\mathcal{G}$.

So, given parameters M and p, we construct a generating set for S where each positive integer less than M has probability p of being a generator for S.

3.1 `GenerateNumericalSemigroup()`

The most fundamental function from rns-db-plot is `GenerateNumericalSemigroup()` which implements the described algorithm at the beginning of this section. In particular, this function takes nonnegative integer parameters m, a, and $b \neq 0$. Here, we take m to be the upper bound M and $\frac{a}{b}$ to be the probability p in the above procedure. Pseudocode for the implementation of this functions follows:

Algorithm 3.1. Given m, a, and b, randomly generate a numerical semigroup.

function GENERATENUMERICALSEMIGROUP(m, a, b)
    $G \leftarrow \{\}$
    $p \leftarrow a/b$
    for all $n \in [1, m]$ **do**
        if Random$([0,1]) < p$ **then**
            $G \leftarrow G \cup \{n\}$
        end if
    end for
    return NumericalSemigroup(G)
end function

Note that this implementation is relatively inefficient in terms of operation complexity. In particular, the algorithm often considers appending generators which are already spanned by the current generating set at some given point.

Example 3.2. Suppose we run `GenerateNumericalSemigroup()` and $3, 7, 8$, and 14 are chosen as generators. Note that

$$3 + 3 + 8 = 14.$$

So inlcuding 14 as a generator is redundant and does not change the elements of the semigroup given that 3 and 8 are already generators. Hence, considering to include 14, or any other linear combinations of already chosen generators, is inefficient.

3.2 `GenerateNumericalSemigroupAp()`

As previously mentioned, numerical semigroups possess many invariants that have been heavily studied in literature, and GAP has functions which calculate many of these invariants. However, the complexity of these functions vary depending on the property in question. Remark 2.2 states that certain properties can be easily read from the Apéry set. Hence, GAP's implementation of the calculation of properties like these often first requires the calculation of the Apéry set, and then goes on to use this set to calculate the desired property.

Recall that the Apéry set of a numerical semigroup S can be expressed as:

$$Ap(S,n) = \{0 = w(0), w(1), \dots, w(n-1)\}$$

where $w(k) \in S$ is the minimal element such that $w(k) \equiv k \mod n$. So, we see that as the smallest generator of S increases, the size of the Apéry set increases as well. This in turn increases the runtime for the calculation of $Ap(S)$ and hence the runtime for calculation of invariants like the Frobenius number. So, given that we intend on calculating several such invariants, we call `GenerateNumericalSemigroupAp()` when randomly generating semigroups to reduce overall runtime for experiments. This method is especially effective when we expect to have a relatively large first generator (i.e. for small p).

As we saw in Section 3, one can create a numerical semigroup object in GAP by passing the Apéry set of a semigroup to the function `NumericalSemigroupByAperyList()`. Our function `GenerateNumericalSemigroupAp()` randomly generates a semigroup according to the same parameters passed to `GenerateNumericalSemigroup()`, but does so by constructing the Apéry set, rather than just the generating set.

Algorithm 3.3. Given $n \in S = \langle n_1, \dots, n_k \rangle$, computes $\mathsf{Z}(m)$ for all $m \in [0, n] \cap S$.

function GENERATENUMERICALSEMIGROUPAP(m, a, b)
  $G \leftarrow \{\}$
  $p \leftarrow a/b$
  for all $n \in [1, m]$ **do**
    if Random$([0,1]) < p$ **then**
      $n_1 \leftarrow n$
      $G \leftarrow G \cup \{n_1\}$
      $A \leftarrow (0, 0, \dots, 0)$
      Break
    end if
  end for
  $n \leftarrow n_1$
  while A has at least two 0 entries **do**
    $n \leftarrow n + 1$
    if $n \neq g \mod n_1$ for all $g \in G$ **then**
      for all $g \in G$ **do**

```
                if n − g ∈ A then
                    A[n mod n1] ← n
                else
                    if n ≤ M and Random([0,1]) < p then
                        A[n mod n1] ← n
                    end if
                end if
            end for
        end if
    end while
    return NumericalSemigroup(G)
end function
```

In this algorithm, we have $A = Ap(S, n_1) = Ap(S)$ where S is the semigroup that is constructed when the algorithm finishes. After picking the first generator, we continue iterating through larger integers as before but we now construct the Apéry set along the way. In addition, once the kth entry in A is appropriately filled, we no longer consider integers with the same residue as k modulo n_1 as new generators or as possible elements of A. We do this because in such a case, a natural number $n = k \mod n_1$ would be a redundant generator and could not be the least positive residue modulo n_1 of its class since obviously $k < n$.

In step 5 of the algorithm, the criteria used to check whether or not n is in the Apéry set is to see if $n - g$ is in the Apéry set for any generator $g \in \mathcal{G}$. Equivalently, we are checking to see if n is the sum of two elements of $Ap(S)$, since minimal generators are always in $Ap(S)$. The use of this reduction in the algorithm is crucial in terms of complexity. Moreover, this statement is nontrivial, hence its proof is given below.

Proposition 4.2.1 If $a \in A = Ap(S, n_1)$ and a is not a minimal generator of S, then $a = a' + a''$ for some $a', a'' \in Ap(S)$.

Proof. Let a be defined as above. Then $a = m' + m''$ for some $m', m'' \in S$. Since $Ap(S, n_1)$ contains one representative for each equivalence class modulo n_1, there exist $a', a'' \in A$ such that

$$m' \equiv a' \mod n_1$$
$$m'' \equiv a'' \mod n_1.$$

Hence, $a = m' + m'' \equiv a' + a'' \mod n_1$. Since $Ap(S, n_1)$ contains the minimal elements of S which belong to the equivalence classes, it must be that $a = a' + a''$, else there would be a smaller element in S and in the same equivalence class as a. □

Remark 3.4. The second function, `GenerateNumericalSemigroupAp()`, has higher complexity than the first function, `GenerateNumericalSemigroup()`, and thus should only be used if the Apéry set is to be used in subsequent computation.

4 External Packages

Here, we describe the other functionality of rns-db-plot and the general procedure of its use. As previously mentioned, GAP was used to implement the random generation of numerical semigroups. This was often done in large quantities with varying paramaters. In addition, we made use of SQL to keep a database of the data generated over the course of the project as well as Sage to give visual expressions of the data produced.

4.1 GAP System

This section discusses GAP, a standard language for computations involving numerical semigroups. All of the code from this section is executed in GAP.

Example 4.1. We can create a numerical semigroup generated by a subset of $\mathbb{N}$ by passing this set into `NumericalSemigroup()`. The statement below assigns `s` to be a numerical semigroup object generated by $5, 8$, and 9.

```
gap> s := NumericalSemigroup([5,8,9]);
<Numerical semigroup with 3 generators>
gap>
```

GAP also has its own random numerical semigroup function. However, its method of generation is different from the procedure used in [1]. GAP's function takes positive integer parameters k and M where k is the maximum number of generators and M is an upper bound for the generators. The function then uniformly picks k integers from $\{1, 2, \ldots, M\}$ and returns the numerical semigroup generated by these integers.

Example 4.2. As an example, the following statement assigns `s` to be a numerical semigroup with 10 generators randomly chosen from $\{1, 2, \ldots, 500\}$.

```
gap> s := RandomNumericalSemigroup(10, 500);
<Numerical semigroup with 5 generators>
```

Example 4.3. We can determine the minimal generating set of a numerical semigroup by calling

```
gap> s := NumericalSemigroup([5,8,9,13]);
<Numerical semigroup with 4 generators>
gap> MinimalGeneratingSystem(s);
[5,8,9]
```

Example 4.4. We can also calculate the gaps, Frobenius number, and Apéry set of a numerical semigroup via GAP, and use the Apéry set to create its corresponding numerical semigroup.

```
gap> s := NumericalSemigroup([5,8,9]);
<Numerical semigroup with 4 generators>
gap> GapsOfNumericalSemigroup(s);
[1,2,3,4,6,7,11,12]
gap> FrobeniusNumber(s);
12
gap> ApSet := AperyListOfNumericalSemigroup(s);
[0,16,17,8,9]
gap> t := NumericalSemigroupByAperyList(ApSet);
<Numerical semigroup>
gap> s = t;
true
```

This agrees with our calculations from Example 2.5.

Remark 4.5. GAP is a language designed for compuations in discrete mathematics. Because of this, the language features little support for the use of real numbers. To avoid problems with the system's representation of real numbers, the inputs of functions are always rational. In particular, probabilities are entered as two integers, a numerator a and a denominator b. For example, a probability of 0.05 could be entered as $a = 1$ and $b = 20$.

The functions `RunExperiment()` and `RunExperimentP()`, Sections 4.2.1 and 4.2.2, were also often used in GAP. The former randomly generates a desired amount of semigroups with desired paramaters and calculates for each semigroup generated specific invariants.

4.2 SQL

SQL is a language used for creating and managing databases.

Example 4.6. We will demonstrate creating, inserting data into, and pulling data from a table in SQL. Within the terminal, we launch SQL and create a table named `Table_1` with columns labeled `a`, `b`, and `str`. The first two columns will hold integer values while the third will hold a string value.

```
zacharyspaulding$ sqlite3
sqlite> create table Table_1(a integer, b integer, str string);
```

We can now insert and select entries from `Table_1`. Let us insert three rows into and select some data from `Table_1`.

```
sqlite> insert into Table_1(a,b,str) values (1, 2, "hello");
sqlite> insert into Table_1(a,b,str) values (30, 2, "world");
sqlite> insert into Table_1(a,b,str) values (1, -45, "goodbye")
sqlite> select a from Table_1;
hello
world
goodbye
```

```
sqlite> select * from Table_1;
1|2|hello
30|2|world
1|-45|goodbye
sqlite> select * from Table_1 where a = 1;
1|2|hello
1|-45|goodbye
```

The documents created by the previously mentioned functions are actually written as commands for SQL. Initial experiments contain commands for creating a new database. Subsequent experiments add to existing databases by appending entries of data which include the paramaters M and p chosen and various invariants of the particular semigroup generated. As of now, these documents need be copied from the Docker container to some local directory in order to make use of them with SQL, a local program.

4.3 Sage

Sage is an opensource computer algebra system written in Python. This was used for plotting data from SQL databases to give a visual description of expected behavior of the randomly generated semigroups. In short, the functions written in Sage read data from an SQL database created by the GAP functions. It then averages the values of the numerical invariants for the semigroups with fixed M and p. These averages are then plotted against each other for varying M or p.

5 Usage

In this section, we describe the process of running an experiment in GAP using rns-db-plot and plotting its results using Sage. A full example is featured in Section 6.6.

5.1 `RunExperiment()`

`RunExperiment()` takes the same parameters as `GenerateNumericalSemigroup()` along with a positive integer parameter called `len`. This will generate `len` many numerical semigroups. For each of these generated semigroups, `RunExperiment()` will write an SQL insert statement to a file in the Docker shell. Note that this function only makes calculations for a fixed probability p and fixed upper bound M.

5.2 `RunExperimentP()`

This function is similar to `RunExperiment()` but will make calculations for varying values of p. Because of this, `RunExperimentP()` takes all the same parameters as `RunExperiment()` as well as a second probability and a positive integer p_{len}. We will call the two probabilities p_1 and p_2.

`RunExperimentP()` will consider the interval $[p_1, p_2]$. This interval will then be partitioned into p_{len} components of equal length. Then for each endpoint of the partition,

$$P_k = p_1 + \frac{p_2 - p_1}{p_{len}} k \text{ for } k \in \{0, 1, 2, \ldots, p_{len}\},$$

`RunExperiment()` will be called with probability P_k.

This allows us to generate several datapoints with varying probability parameters using a single function.

The former of these allows the user to randomly generate N semigroups, with fixed M and p, using either our implementation or GAP's. The function will also calculate any number of desired invariants for each semigroup as they are generated and store the data in a document inside of the active Docker container. The latter function does the same but allows for the probability p to vary over a given interval.

5.3 Generating Data

The first step in running an experiment with rns-db-plot is to generate the desired data. This is done in GAP through Docker.

There are several parameters to be chosen when running an experiment. We choose an upper bound M for the minimal generators of the semigroup, a rational probability $p \in [0, 1]$ for the probability that a positive integer less than M is to be a generator, and some N for the number of datapoints. We must also choose which attributes we would like to be calculated for each of the datapoints, i.e. Frobenius number, number of (minimal) generators, whether or not the semigroup is symmetric, etc.

Depending on the chosen attributes, we will call different functions to generate the data. In both cases, we must also make a choice for a filename and tablename which will be elaborated upon in the following sections.

Example. To launch Docker, open the terminal and execute the following commands

```
zacharyspaulding$ docker start rns
zacharyspaulding$ docker attach rns
```

Enter the following commands to lanuch GAP and load required packages.

```
[homalg@494436f5876c ~]$ gapL
gap> LoadPackage("num");
gap> NumSgpsUse4ti2();
gap> NumSgpsUse4ti2gap();
gap> NumSgpsUseSingular();
gap> NumSgpsUseNormaliz();
```

After this, import rns-db-plot so that the generating functions may be used. You can now generate data. As an example, execute the following to generate 1000 random numerical semigroups with upper bound $M = 500$ and probability $p = .01$ with data to be stored into `table_1` in the file named `file.txt`.

```
gap> RunExperiment(1000, "file.txt", "table_1", [["FrobNum", FrobeniusNumber,
   "integer"], ["NumMinGens", NumMinGens, "integer"]], 500, 1, 100, GenNumSemAp);
```

Note that this command will calculate the Frobenius number and minimal generating set for each generated semigroup. These values are given the names `FrobNum` and `MinGens`, respectively. And their data types in SQL are integer and string, respectively. The last command says to use the function `GenNumSemAp()` to generate the semigroups. We could also generate data for varying probabilites as follows.

```
gap> RunExperimentP(1000, "file.txt", "table_1", [["FrobNum", FrobeniusNumber,
   "integer"], ["NumMinGens", NumMinGens, "integer"]], 500, 1, 100, 11, 100, 5,
   GenNumSemAp);
```

This will generate data for $M = 500$ and $p = \frac{1}{100}, \frac{2}{100}, \ldots, \frac{11}{100}$.

5.4 Storing Data

Data is stored in tables in a local directory via SQL. When the data is generated in GAP, strings *filename* and *tablename* are passed when calling either of the experiment functions. As data is generated, these functions will also write out to the file named *filename* lines of SQL commands. For each data point generated, the functions write out one line of SQL code which, when called in SQL, will insert into the table *tablename* the generated point along with its corresponding attributes. If *filename* is an empty file, then the experiment function will write out a statement that creates the table with name *tablename* and appropriate columns for attributes. Here is an example of SQL output.

```
sqlite> select * from table_1;
1|100|1|100|55|109|12|0|0
2|100|1|100|0|-1|34|0|0
3|100|1|100|0|0|0|0|1
4|100|1|100|0|-1|12|0|0
5|100|1|100|0|-1|73|0|0
6|100|1|100|0|-1|76|0|0
7|100|1|100|0|-1|41|0|0
8|100|1|100|0|-1|71|0|0
9|100|1|100|0|-1|12|0|0
10|100|1|100|171|341|10|0|1
11|100|1|100|0|-1|64|0|0
12|100|1|100|0|-1|11|0|0
13|100|1|100|0|-1|82|0|0
14|100|1|100|0|-1|18|0|0
```

```
15|100|1|100|0|-1|12|0|0
16|100|1|100|945|1889|43|0|1
17|100|1|100|0|-1|88|0|0
```

The first column is the entry ID for the randomly generated numerical semigroup, the second column is the M value, and the third and fourth columns are integers a and b whose ratio defines the probability p. The remaining columns are attribute values of the generated semigroup, in this case Frobenius number, number of generators, number of minimal generators, and smallest generator.

After the experiment function finishes its execution, we must move `file.txt` from the Docker shell where it was created to a local directory. This can be done by executing a command simlar to

```
zacharyspaulding$ docker cp 494436f5876c:home/homalg/file.txt .
```

in a Linux based environment. The initial directory will vary between users.

Next, launch SQL and execute `.read file.txt` inside the database. This will generate the table called `table_1` and insert all datapoints from the experiment. Make sure to save this SQL database.

```
zacharyspaulding$ sqlite3 Database
sqlite> .read file.txt
sqlite> .save Database
sqlite> .quit
```

Note that this will save the table into and SQL database with filename `Database`.

5.5 Plotting Data

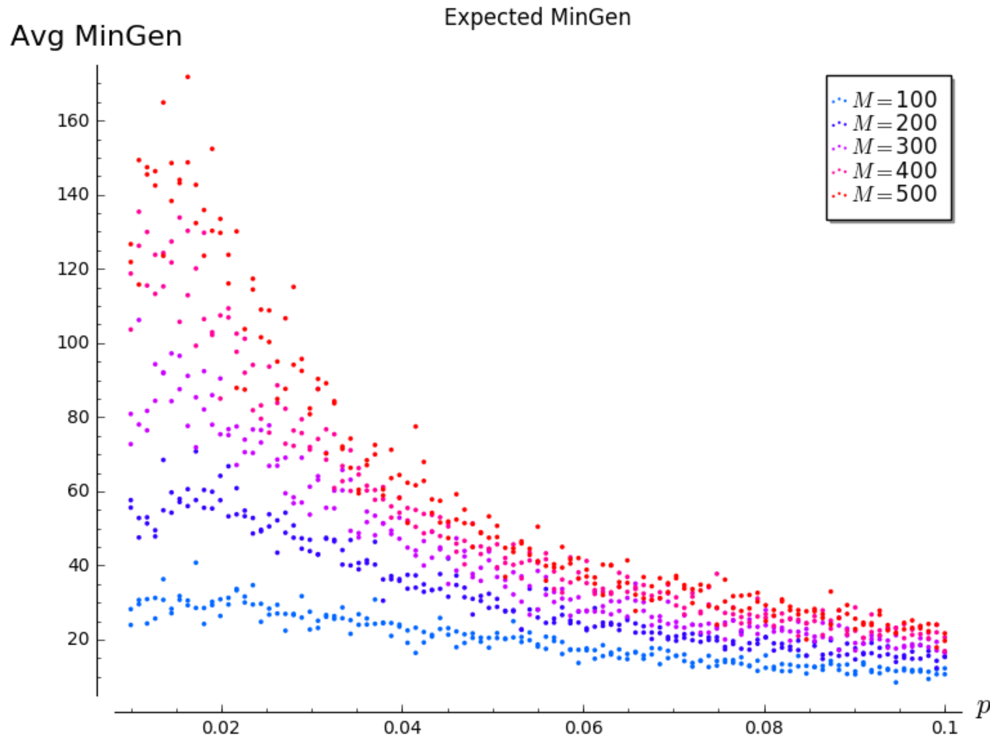
Data is plotted via functions in Sage.

To plot functions in Sage, we need the strings *database* and *tablename*. The former is a local path which ends in the name of the SQL database where the data is stored. The latter is the name of the specific table in this database. We then pass these values into `expectedAttributePlot()`. This will return plots of average values of chosen attributes against M values. In Sage, execute the following to plot average Frobenius number for $M = 500, 1000$ using data located in `table_1` of an SQL database with pathname `/zacharyspaulding/NumSemGp/Database`.

```
sage: import sqlite3
sage: expectedAttributePlot("~/zacharyspaulding/NumSemGp/Database", "table_1",
    "NumMinGens", [500, 1000])
```

This will plot the average number of generators for M values 500 and 1000 from table_1 from the SQL database with the path `/zacharyspaulding/NumSemGp/Database`. Note that this function can only be used to plot integer valued functions. See Figures 3-5.

Figure 3: Average Number Minimum Generators I



5.6 Example

Here we will walk through an example of running a small experiment using rns-db-plot. This example used the function `RunExperimentPM()` which calls `RunExperimentP()` for each entry in a list of M values. We first launch GAP and generate data in the terminal.

```
zacharyspaulding$ docker start rns
zacharyspaulding$ docker attach rns
[homalg@494436f5876c ~]$ gapL
gap> LoadPackage("num");
gap> NumSgpsUse4ti2();
gap> NumSgpsUse4ti2gap();
gap> NumSgpsUseSingular();
gap> NumSgpsUseNormaliz();
```

After loading these packages, paste in rns-db-plot and execute the following.

```
gap> RunExperimentPM(1000, "file2.txt", "table\_2", [["NumMinGens", NumMinGens,
   "integer"]], [500, 1000, 5000, 10000], 1, 100, 1, 10, 15,
   GenerateNumericalSemigroupAp);
gap> quit;
```

Here, enter Ctrl + p then Ctrl + q to return to the local directory or open a new terminal window. When copying a file from the Docker container, use your unique container

pathname.

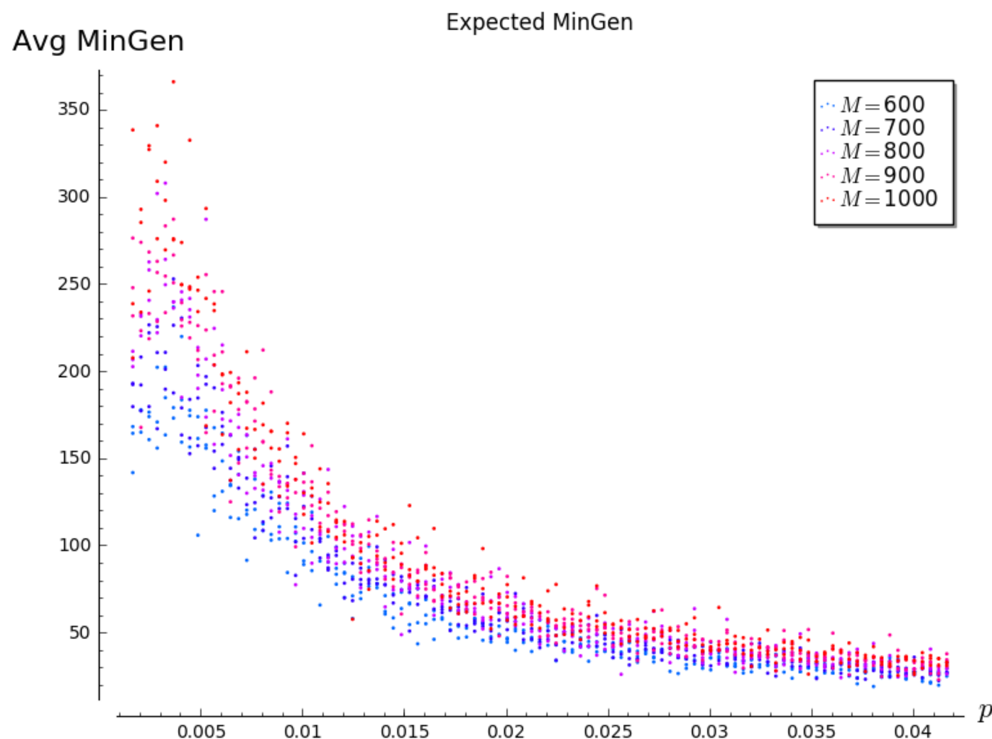
```
zacharyspaulding$ cd NumSgps
zacharyspaulding$ docker cp 494436f5876c:home/homalg/file_2.txt .
zacharyspaulding$ sqlite3 Database
sqlite> .read file_2.txt
sqlite> .save Database
sqlite> .quit
```

Now launch Sage and execute the following to generate the plots from this experiment.

```
sage: import sqlite3
sage: expectedAttributePlot("~/zacharyspaulding/NumSgps/Database", "table_2",
    "NumMinGens", [500, 1000, 5000, 10000])
```

See Figures 3-5 for plots similar to ones which would be generated from this experiment.

Figure 4: Average Number Minimum Generators II



5.7 Example Data

The table in Figure 6 displays runtimes for the code above for generating 1000 semigroups. Varying M values and different generating functions are used.

The following page shows examples of output from `expectedAttributePlot()`. The

Figure 5: Average Number Minimum Generators III

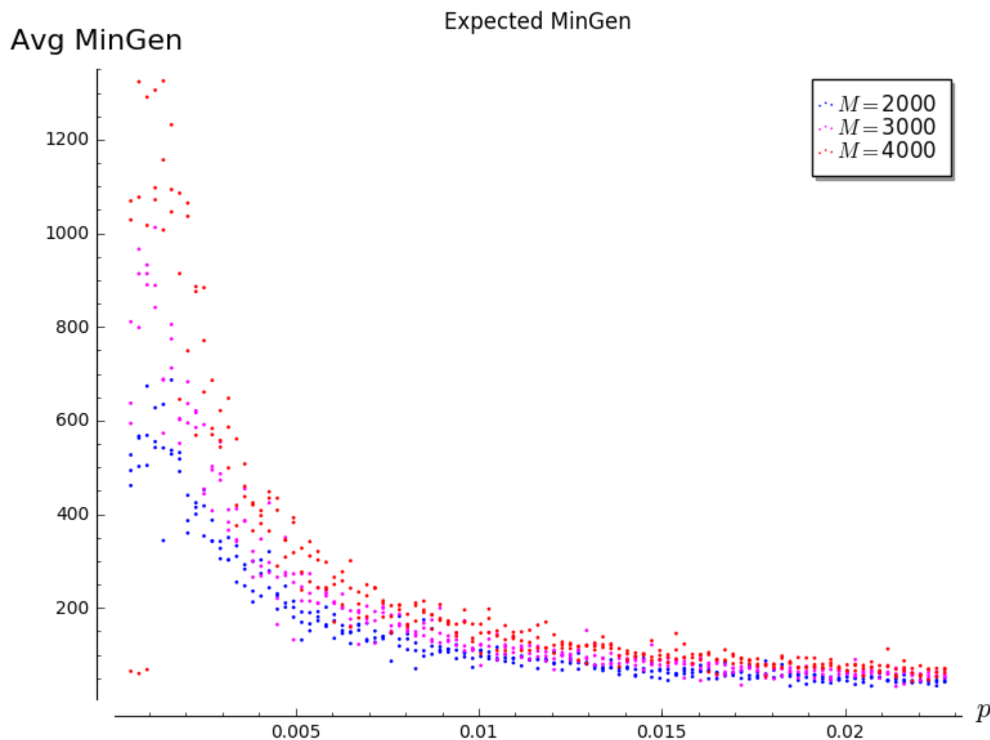


Figure 6: Run Times for GenerateNumericalSemigroup and GenNumSemAp

Time (s)	$M = 500$	$M = 1000$	$M = 5000$	$M = 10000$
GenNumSemAp	1802.0	35.7	16.4	17.6
GenNumSem	2.3	4.3	59.5	310.6

horizontal axis measures the probability p used to generate the semigroups and the vertical axis measures the average of the attribute.

References

[1] J.A. De Loera, C. O'Neill, D. Wilburne, *Random Numerical Semigroups and a Simplicial Complex of Irreducible Semigroups*.

[2] M. Delgado, P. García-Sánchez, and J. Morais, *GAP Numerical Semigroups Package*, `http://www.gap-system.org/Manuals/pkg/numericalsgps/doc/manual.pdf`.

[3] Sage: Open Source Mathematics Software, available at `www.sagemath.org`.

[4] Rosales, J. C. and García-Sánchez, P. A., *Numerical semigroups*.