Computationally Determining the Dimensions of the Homology Groups of Directed Graphs

Ariya R. Shajii
Weston High School, MA

Coach: Gabor Lippner
Department of Mathematics, Harvard University, Cambridge MA

September 30, 2012

Abstract

In this paper, we begin by expanding the notion of homology to graphs. We then present a new computational method for computing the dimensions of the homology groups of these graphs. For the first time, we show detailed results for the first homology group H_1 of random directed graphs.

Contents

1 Introduction

In general algebraic topology, homology refers to the association of a sequence of abelian groups with a certain mathematical object.

Given the chain complex

$$... \xrightarrow{\partial_{n+2}} C_{n+1} \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \xrightarrow{\partial_{n-1}} ... \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0$$

with the property that $\partial_n \circ \partial_{n-1} = 0$ at each level, the *homologies* are defined by:

$$H_n(C) = \ker \partial_n \,/\, \mathrm{Im}\, \partial_{n+1}.$$

Each $\partial_n : C_n \to C_{n-1}$ is referred to as a *boundary map*. See [1] for further details.

In this paper, we will focus on computing $\dim H_n$ for a special type of homology on directed graphs. The following lemma will be pivotal in this regard.

Lemma 1.1

$$\forall\, n \geq 0,\ \dim H_n(C) = \dim C_n - \dim \mathrm{Im}\, \partial_n - \dim \mathrm{Im}\, \partial_{n-1} = \dim \ker \partial_n + \dim \ker \partial_{n+1} - \dim C_n$$

Proof. By definition,

$$\dim H_n(C) = \dim \ker \partial_n - \dim \mathrm{Im}\, \partial_{n+1}.$$

By the rank-nullity theorem,

$$\dim \ker \partial_n = \dim C_n - \dim \mathrm{Im}\, \partial_n.$$

Moreover, we have the identity

$$\dim \mathrm{Im}\, \partial_{n-1} = \dim C_{n-1} - \dim \ker \partial_{n-1}.$$

We can arrive at the original equality through substitution [2]. □

With this result at hand, we now proceed to applications to directed graphs.

2 Application to Directed Graphs

2.1 Defining Homologies on Digraphs

Grigor'yan et. al. defined a version of homology for directed graphs (see [2]) which we now explain. A directed graph or *digraph* can be represented as $G = (V, E)$ where $E \subseteq \{ij \mid (i,j) \in V^2, i \neq j\}$ is the set of edges and $0 < |V| < \aleph_0$ is the set of vertices.

We will define a *vertex monomial* $v = i_0 i_1 ... i_n$ to be any member of V^{n+1}, and we will define a *vertex polynomial* to be any linear combination of vertex monomials of the same n, i.e. any member of $span\{V^{n+1}\}$.

We can then define two vector spaces V_n and $A_n \subseteq V_n$ as follows:

$$V_n = span\{v \in V^{n+1} \mid v \text{ is valid}\},$$
$$A_n = \{v \in V_n \mid v \text{ is allowed}\}.$$

A vertex monomial $v = i_0 i_1 ... i_n \in V^{n+1}$ is *valid* if for each pair of consecutive vertices $i_p i_{p+1}$, $i_p \neq i_{p+1}$ for $0 \leq p \leq n-1$. A vertex polynomial is valid if all of its constituent monomials are valid.

A vertex monomial $v = i_0 i_1 ... i_n \in V_n$ is *allowed* if for each concecutive pair of vertices $i_p i_{p+1}$, $i_p i_{p+1} \in E$ for $0 \leq p \leq n-1$. A polynomial is allowed if each of its constituent monomials are allowed.

The boundary map $\partial_n : V_n \to V_{n-1}$ of the vertex monomial $v = i_0 i_1 ... i_n \in V_n$ will be defined as

$$\partial v = \sum_{q=0}^{n} (-1)^q (i_0 i_1 ... \hat{i_q} ... i_n),$$

where $\hat{i_q}$ indicates the exclusion of i_q from the expression. For the linear combination $v = c_1 v_1 + c_2 v_2 + \ ... \ + c_j v_j$ of monomials, the boundary map will simply be

$$\partial v = \sum_{k=1}^{j} c_k \partial v_k.$$

If the boundary map results in a term that is not valid, then that particular term is simply discarded [2].

Lemma 2.1

$$\forall\, v \in V_n,\ (\partial_n \circ \partial_{n-1})(v) = 0$$

Proof. Since ∂ is a linear map, it suffices to check the statement for vertex monomials. We have,

$$\begin{aligned}
(\partial_n \circ \partial_{n-1})(i_0...i_n) &= \sum_{k=0}^{n}(-1)^k \partial_{n-1}(i_0...\hat{i_q}...i_n) \\
&= \sum_{k=0}^{n}(-1)^k(\sum_{j=0}^{k-1}(-1)^j(i_0...\hat{i_j}...\hat{i_k}...i_n) + \sum_{j=k+1}^{n}(-1)^{j-1}(i_0...\hat{i_k}...\hat{i_j}...i_n)) \\
&= \sum_{0\le j<k\le n}(-1)^{k+j}(i_0...\hat{i_j}...\hat{i_k}...i_n) - \sum_{0\le k<j\le n}(-1)^{j+k}(i_0...\hat{i_k}...\hat{i_j}...i_n) \\
&= \sum_{0\le j<k\le n}(-1)^{k+j}(i_0...\hat{i_j}...\hat{i_k}...i_n) - \sum_{0\le j<k\le n}(-1)^{j+k}(i_0...\hat{i_j}...\hat{i_k}...i_n) \\
&= 0.
\end{aligned}$$

□

We can now define yet another vector space: $\Omega_n = \{v \in A_n \mid \partial v \in A_{n-1}\}$.

It is important to note that ∂_n would in fact map Ω_n directly to $\Omega_{n-1} \subseteq A_{n-1}$ since, if $v \in \Omega_n$ and $\partial_n v = w$, then $w \in A_{n-1}$ and $\partial_{n-1} w = (\partial_n \circ \partial_{n-1})(v) = 0 \in A_{n-1} \to w \in \Omega_{n-1}$.

From this, we arrive at the chain complex

$$... \xrightarrow{\partial_{n+2}} \Omega_{n+1} \xrightarrow{\partial_{n+1}} \Omega_n \xrightarrow{\partial_n} \Omega_{n-1} \xrightarrow{\partial_{n-1}} ... \xrightarrow{\partial_1} \Omega_0 \xrightarrow{\partial_0} 0.$$

We can now define the the homologies of a directed graph to be

$$H_n = \ker \partial_n \,/\, \operatorname{Im} \partial_{n-1}.$$

We will now proceed to describe a computational method for calculating the dimensions of the homology groups of directed graphs.

2.2 H_0 Space

Lemma 2.2
For a digraph $G = (V, E)$ with C undirected, connected components,

$$\dim H_0 = C.$$

Proof. By definition we have,

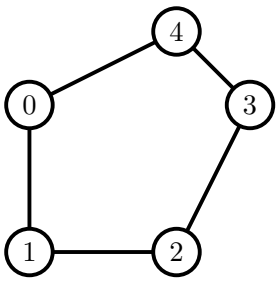


Figure 1: Example cycle-graph (without orientation)

$$H_0 = \ker \partial_0 = \{v \in \Omega_0 \mid \partial_0 v = 0\}.$$

The condition $\partial_0 v = 0$ implies that $\partial_0(ij) = j - i = 0$ or equivalently, $i = j$. This signifies that $\partial = \text{const}$ on any connected component of (V, E), and the dimension of this space of this space is clearly C [2]. □

2.3 Cycle-Graphs

We say that a digraph (V, E) is a *cycle-graph* if it is connected (as undirected) and every vertex has a degree of 2.

For each cycle-graph, $\dim H_0 = 1$ and $\dim \Omega_0 = |V| = |E| = \dim \Omega_1$. Furthermore, for such graphs, we note that

$$\dim \Omega_n = 0 \ \ \forall n \geq 3,$$
$$\dim H_n = 0 \ \ \forall n \geq 2.$$

If our graph is a triangle or square:

$$\dim \Omega_2 = 1, \ \dim H_1 = 0,$$

otherwise:

$$\dim \Omega_2 = 0, \ \dim H_1 = 1.$$

A detailed proof can be found in [2].

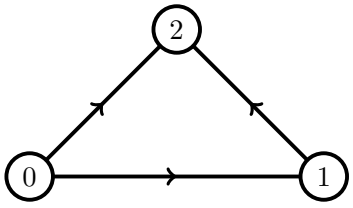


Figure 2: Digraph with 3 vertices and 3 edges.

3 Computationally Determining $\dim H_n$

In order to devise a computational method with which to compute the dimensions of the homology groups of digraphs, object oriented programming will be utilized in conjunction with a mathematics package, which will be used to carry out the more intensive tasks. In this case, Java and *Mathematica* will be used. Other alternatives are Python and MATLAB, which are currently being considered for future work in this area.

3.1 Required Classes

Firstly, a `Digraph` class will be created. The actual graph will be represented through a binary adjacency matrix with 0s along its diagonal. The example shown in Figure 2 corresponds to

$$G = (V, E) = (\{(0), (1), (2)\}, \{(0)(1), (0)(2), (1)(2)\}) = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

This class will resemble the following:

```
public class Digraph {
        private boolean[][] matrix;

        public Digraph(boolean[][] matrix) {
                this.matrix = matrix;
        }

        public VectorSpace vSpace(int n) {
                // implementation discussed later
        }

        public VectorSpace aSpace(int n) {
```

```
                // implementation discussed later
        }

        public VectorSpace omegaSpace(int n) {
                // implementation discussed later
        }

        ...
}
```

The next component is representing the elements of the various vector spaces associated with each digraph. The example shown above, for instance, would have an A_1 of $< (0)(1), (0)(2), (1)(2) >$; i.e. the space would be comprised of all linear combinations of these three elements. Therefore, we require three specific classes:

- `Algebraic` abstract base-class or interface, containing a `boundary` method (in addition to other possible methods) that must be overridden by all subclasses. The implementation could potentially look like this:

  ```
  public interface Algebraic {
          public Algebraic boundary();
          public Algebraic sum(Algebraic v);
          public Algebraic scalarMultiply(double c);
          public boolean isValid();
          ...
  }
  ```

- `VertexMonomial` class (extending/implementing `Algebraic`) containing a real coefficient and a list of vertices. For example, $3(0)(1)$ would be represented by a single 3 and by a list containing 0, 1 in that order.

  ```
  public class VertexMonomial implements Algebraic {
          private double coefficient;
          private int[] vertexList;

          public VertexMonomial(double coefficient, int[] vertexList) {
                  this.coefficient = coefficient;
                  this.vertexList = vertexList;
          }

          public VertexMonomial(int[] vertexList) { this(1, vertexList); }

          ...
  ```

```
}
```

- `VertexPolynomial` class (extending/implementing `Algebraic`) containing a set of `VertexMonomial` instances, the sum of which results in the polynomial that this class represents. For example, $2(0)(1) - 3(1)(2)$ would be represented by a set of two monomials: $2(0)(1)$ and $3(1)(2)$.

```
public class VertexPolynomial implements Algebraic {
        private VertexMonomial[] monomialList;

        public VertexPolynomial(VertexMonomial[] monomialList) {
                this.monomialList = monomialList;
        }
        ...
}
```

Of course, the base-class is not an absolute necessity, but is beneficial as it serves as a connection between its subclasses, which are directly related to one another.

Next, we will create a `VectorSpace` class, which will be used to represent V_n, A_n and Ω_n. This class will hold a set of `Algebraic` objects representing the basis for a specific vector space. One significant feature that must be implemented is the ability to reduce an arbitrary set of such objects to a genuine basis such that each element is linearly independent of the others. This will allow us to determine the dimension of our vector space. For example, if we wished to compute ∂X where $X =< (0)(1), (1)(0) >$, by simply looping through the generators, taking the boundary of each, and adding each result to a new set of generators, we would obtain $\{(1) - (0), (0) - (1)\}$. Hence, this must be reduced to obtain the basis $< (1) - (0) >=< (0) - (1) >$, with which we can deduce that the dimension of this particular vector space is in fact 1.

```
public class VectorSpace {
        private Algebraic[] basis;

        public VectorSpace(Algebraic[] listOfTerms) {
                basis = reduceToBasis(listOfTerms);
        }

        private static Algebraic[] reduceToBasis(Algebraic[] listOfTerms) {
                // implementation discussed later
        }

        public boolean contains(Algebraic v) {
                // implementation discussed later
```

```
        }

        public int dim() { return basis.length; }

        ...
}
```

3.2 Forming a Basis for the Vector Spaces

To determine if $u \in span\{v_1, v_2, ..., v_n\}$, we must determine if there exists coefficients $c_1...c_n$ such that $\sum_{i=1}^{n} c_i v_i = u$. To do this computationally we shall

1. Ensure that each monomial appearing in u appears somewhere in $v_1...v_n$. If one is found that does not, $u \notin span\{v_1, v_2, ..., v_n\}$.
2. List each *unique* monomial of $v_1...v_n$ (ignoring their coefficients). For example, $< 3(0) - 2(1), (0) + 4(2) >$ would produce the list consisting of 1(0), 1(1) and 1(2). We will denote the members of this list by $m_1, m_2, ..., m_j$.
3. From the expression $c_1v_1 + c_2v_2 + ... + c_nv_n = u$, we can form a system of linear equations by equating the coefficient of each m_i with that of the corresponding monomial in u: $(k_{i,1}c_1 + k_{i,2}c_2 + ... + k_{i,n}c_n)m_i = r_i m_i \rightarrow k_{i,1}c_1 + k_{i,2}c_2 + ... + k_{i,n}c_n = r_i, 1 \leq i \leq j$, where each $k_{a,b}$ will be known from the rearrangement of the terms of $c_1v_1 + ... + c_nv_n$.
4. At this point, we will have

$$\begin{bmatrix} k_{1,1} & k_{1,2} & ... & k_{1,n} \\ k_{2,1} & k_{2,2} & ... & k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{j,1} & k_{j,2} & ... & k_{j,n} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_j \end{bmatrix} \tag{3.1}$$

   where each k and r will be known from $v_1...v_n$ and u, respectively. To check if there extists $c_1...c_n$ such that the above holds, we will use the mathematically oriented program mentioned eariler. *Mathematica* offers a `LinearSolve` function which can be utilized.

With this method, we are able to reduce an arbitrary list of `Algebraic` instances to a basis by looping through the list and, for each element, determining whether it can be expressed as a linear combination of the other elements (i.e. determining if a solution to 3.1 exists). Furthermore, we can check if a term is in a certain vector space in general.

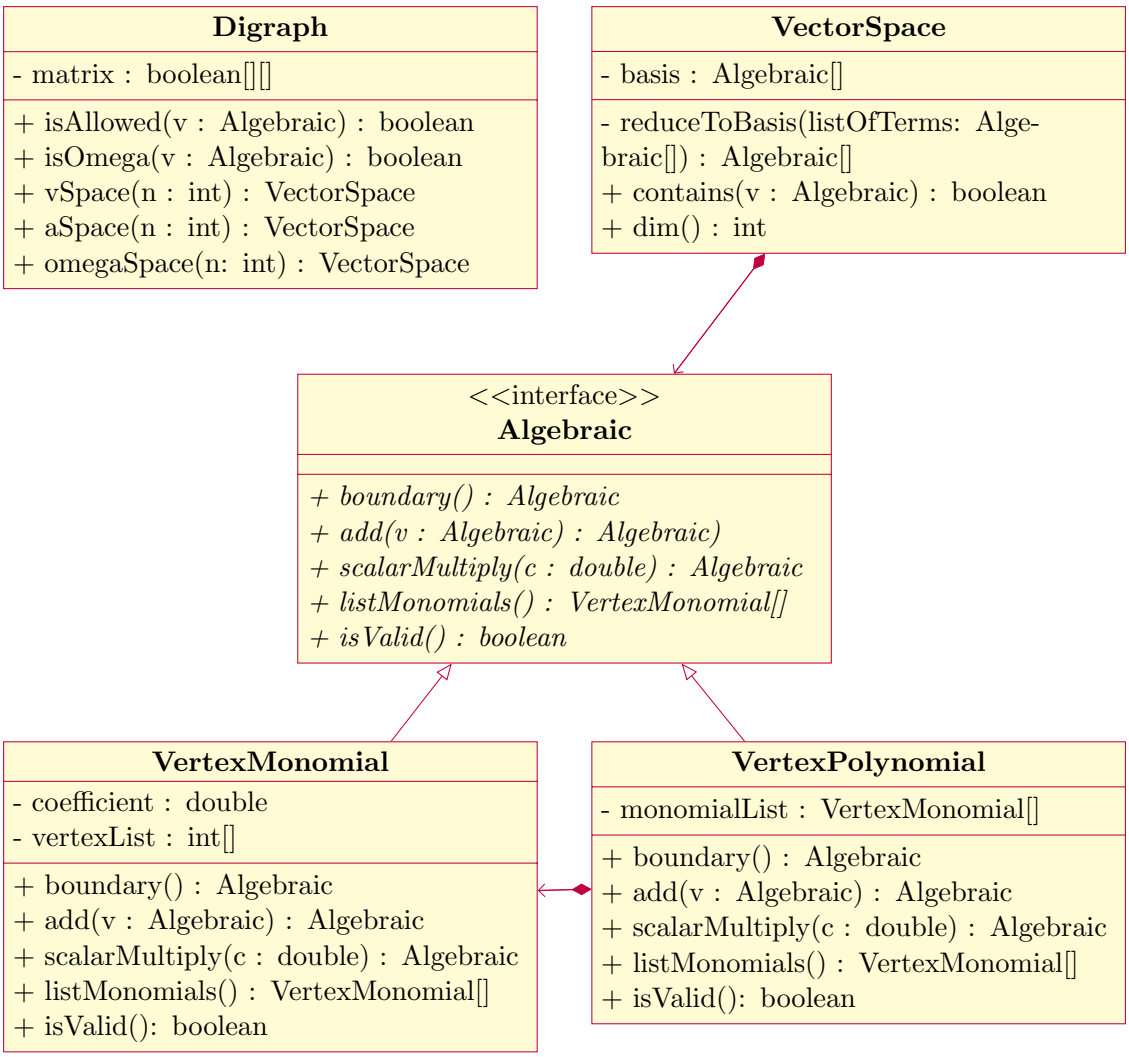


Figure 3: Possible class diagram in Java showing what is required for digraph computations.

3.3 Computing the Vector Spaces

In order to actually compute the several chains of vector spaces associated with each digraph $G = (V, E)$, we will start with $V_n \subset V^{n+1}$. To compute this space, we will list all combinations of $n + 1$ items from V. We will subsequently filter out any elements from this list that are invalid to arrive at a basis of V_n. From here, it will not be difficult to compute a basis of A_n by looping through the monomial basis of V_n and selecting only the terms that are allowed. In other words, for each member $v = i_0 i_1 ... i_n$ of the list, checking that $i_p i_{p+1} \in E$ for $0 \leq p \leq n - 1$.

We present an example by computing V_2, A_2 and Ω_2 for the following graph (depicted in Figure 4):

$$G = (V, E) = (\{0, 1, 2, 3\}, \{(0)(1), (0)(2), (1)(3), (2)(4)\}) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

This will illustrate how the Ω_n spaces are not trivial to compute, unlike the V_n and A_n spaces.

V_2 is fairly trivial to determine:

$$V_2 =< (0)(1)(0), (0)(1)(2), ..., (3)(2)(3) > .$$

From V_2, we can determine A_2 by filtering out all non-allowed terms from the V_2 basis. Recall that abc is allowed only if $ab, bc \in E$. We have,

$$A_2 =< (0)(1)(3), (0)(2)(3) >= span\{S\}.$$

Now to compute Ω_2, we start by checking if any single member of S is a member of Ω_2:

$$\partial[(0)(1)(3)] = (1)(3) - (0)(3) + (0)(1) \notin A_1 \rightarrow (0)(1)(3) \notin \Omega_2,$$
$$\partial[(0)(2)(3)] = (2)(3) - (0)(3) + (0)(2) \notin A_1 \rightarrow (0)(2)(3) \notin \Omega_2.$$

Since no individual member of S is a member of Ω_2, the next step is to determine all sets of coefficients c_1, c_2 such that,

$$\partial[c_1(0)(1)(3) + c_2(0)(2)(3)] = c_1\partial[(0)(1)(3)] + c_2\partial[(0)(2)(3)] \in A_1$$
$$\downarrow$$
$$c_1(1)(3) - c_1(0)(3) + c_1(0)(1) + c_2(2)(3) - c_2(0)(3) + c_2(0)(2) \in A_1.$$

Since we want all non-allowed terms (in this case $(0)(3)$) to vanish, this translates into:

$$-c_1(0)(3) - c_2(0)(3) = 0$$
$$\downarrow$$

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \begin{bmatrix} -1 & -1 \end{bmatrix} = 0.$$

So we have reduced the problem to finding the nullspace of $\begin{bmatrix} -1 & -1 \end{bmatrix}$ which is $\begin{bmatrix} c & -c \end{bmatrix}$ where c is a free real variable. With this, we can deduce that

$$\Omega_2 =< (0)(1)(3) - (0)(2)(3) > .$$

This example pertains to a graph with $|V| = |E| = 4$. Evidently, as these values increase it becomes nearly impossible to determine the Ω_n spaces by hand due to the large nullspace calculation towards the end of the process.

Hence, the greatest challenge evidently lies in computing the Ω_n spaces. This will be done generally as follows:

1. Create an empty set to hold the basis of Ω_n.
2. Form a set S from the basis of A_n by the process described above. Loop through this set; for each v for which $\partial v \in A_{n-1}$, remove v from S and add it to the basis of Ω_n.
3. Let $T = \{\partial v | v \in S\}$. List the *unique, non-allowed* monomials that appear somewhere within the elements of T: $m_1, m_2, ..., m_j$.
4. We wish to find sets of coefficients $c_1...c_p$ such that $c_1t_1 + c_2t_2 + ... + c_pt_p \in A_{n-1}$, where $T = \{t_1, t_2, ..., t_p\}$. In other words, we require coefficients that would make each non-allowed m_i vanish. By rearranging the terms of this expression and discarding any allowed monomials, we would obtain an expression of the form

   $$m_1(k_{1,1}c_1 + ... + k_{1,p}c_p) + m_2(k_{2,1}c_1 + ... + k_{2,p}c_p) + ... + m_j(k_{j,1}c_1 + ... + k_{j,p}c_p).$$

   Since we want each m_i to vanish, we have

   $$\begin{bmatrix} k_{1,1} & k_{1,2} & ... & k_{1,p} \\ k_{2,1} & k_{2,2} & ... & k_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ k_{j,1} & k_{j,2} & ... & k_{j,p} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.2}$$

5. For each set of coefficients $c_1...c_n$ for which 3.2 holds, add $c_1s_1 + c_2s_2 + ... + c_ps_p$ (where each s_i is a member of S and $\partial s_i = t_i$) to the basis of Ω_n created in step 1. To obtain these solutions, we take the nullspace of the k-matrix of 3.2. Consequently, *Mathematica's* `NullSpace` function can be utilized.

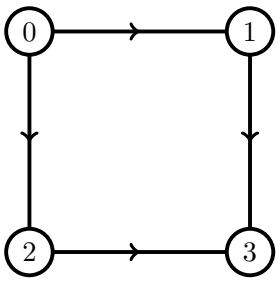


Figure 4: Digraph with 4 vertices and 4 edges.

3.4 Computing the Kernel Dimension

To compute the dimension of the kernel of a vector space X (with basis S) with respect to the boundary map, we first let $T = \{\partial v \mid v \in S\}$. Next, we follow a very similar process to the one outlined in 3.3, with the only difference that this time we want *all* of the monomials to vanish. Hence we have the following:

$$\dim \ker X|_{\partial} = \dim \ker \begin{bmatrix} k_{1,1} & k_{1,2} & \dots & k_{1,p} \\ k_{2,1} & k_{2,2} & \dots & k_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ k_{j,1} & k_{j,2} & \dots & k_{j,p} \end{bmatrix}. \tag{3.3}$$

By the rank-nullity theorem, we have

$$\dim \ker X|_{\partial} = |S| - \operatorname{rank} \begin{bmatrix} k_{1,1} & k_{1,2} & \dots & k_{1,p} \\ k_{2,1} & k_{2,2} & \dots & k_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ k_{j,1} & k_{j,2} & \dots & k_{j,p} \end{bmatrix}. \tag{3.4}$$

Again, *Mathematica* can be used for such a computation; specifically, the `MatrixRank` function that is offered.

3.5 Determining $\dim H_n$

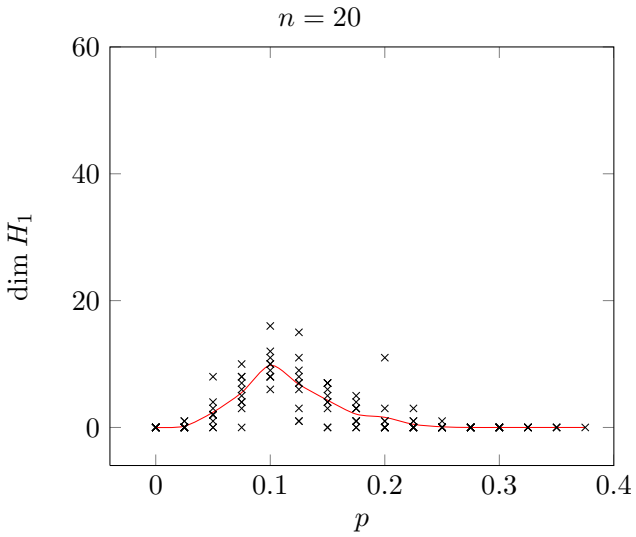
At this stage, we are able to use Lemma 1.1 in order to compute $\dim H_n$. After computing Ω_n and Ω_{n+1} through the methods outlined above, we utilize the equaility

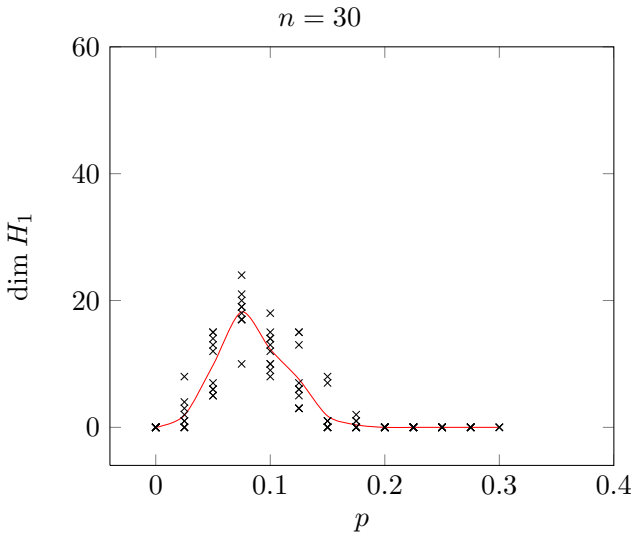
$$\dim H_n = \dim \ker \partial_n + \dim \ker \partial_{n+1} - \dim \Omega_n.$$

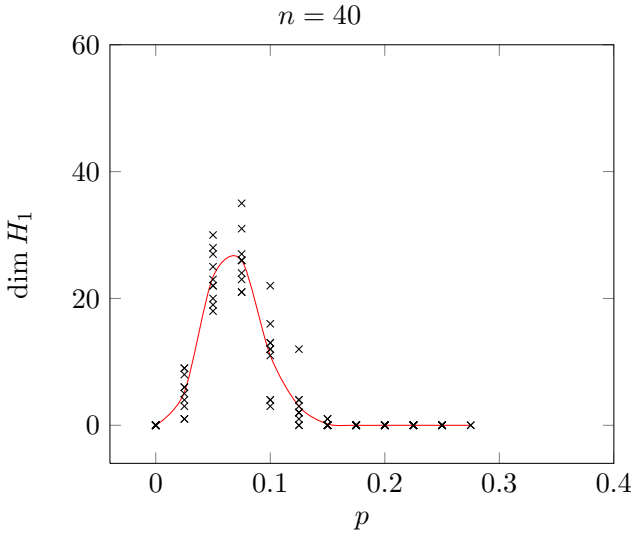
4 Results

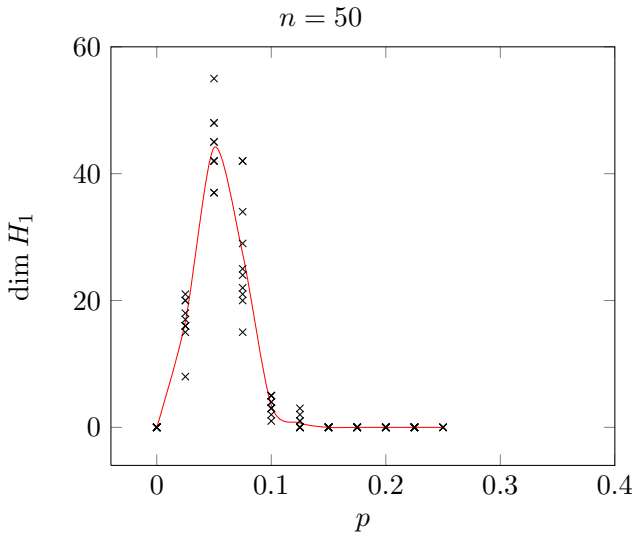
The method described in the previous section was employed to calculate dim H_1 for a collection of random digraphs with a given number of nodes as a function of the edge probability. The figures below show dim H_1 of the random graph $G(n, p)$ as a function of the edge probabality p for a specific number of nodes n. Each figure displays the results of 10 independent trials in addition to their average (exhibited by the red line) so as to obtain a statistically meaningful curve.

Each trial consisted of iterating upwards from $p = 0$ with step-size $\Delta p = 0.025$ and calculating dim $H_1(G(n, p))$ for fixed n in each iteration. If the homology dimension was determined to be 0 for five consecutive iterations after some initial non-zero value, the trial auto-terminated with the assumption that no further non-zero homology dimensions would be encountered.




$n = 30$
60
40
20
0
dim H_1
0
0.1
0.2
0.3
0.4
p


$n = 40$
60
40
20
0
dim H_1
0
0.1
0.2
0.3
0.4
p



In these figures we note that dim H_1 starts at 0, increases to some maximum value, and subsequently begins declining. The strings of 0s towards the start and end of the curve can be explained intuitively, but the center of the curve remains to be a phenomenon that can not yet be explained. The peak of the curve moves closer to $p = 0$ as n increases. Furthermore, the actual value of the peak seems to grow steadily with n.

Some of these results for higher n can be very CPU-intensive to compute. In particular, the total runtime proves to increase drastically as we try to compute the dimensions of the higher homology groups, H_2, H_3, etc.

Future work consists of fully understanding the center regions of the figures above as well as computing the dimensions of these higher homology groups via the methods discussed in this paper.

References

[1] Hatcher, Allen. *Algebraic Topology*. New York: Cambridge UP, 2001. Print.

[2] Alexander Grigor'yan, Yong Lin, Yuri Muranov, and Shing-Tung Yau. *Homologies of Digraphs*. Tech. N.p., n.d. Web. <http://arxiv.org/pdf/1207.2834.pdf>.