Spring 5-2011

Row Reduction of Macaulay Matrices

Lorrin Debenport
University of Southern Mississippi

The University of Southern Mississippi

ROW REDUCTION OF MACAULAY MATRICES

by

Lorrin Debenport

A Thesis

Submitted to the Honors College
of The University of Southern Mississippi
in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science
in the Department of Mathematics

May 2011

Approved by

John Perry
Assistant Professor of Mathematics

Joseph Kolibal
Chair, Department of Mathematics

Dave Davies, Dean
Honors College

Contents

CHAPTER 1

Description of the Problem

A computer can use a matrix to represent a system of non-linear multivariate polynomial equations. The fastest known ways to transform this system into a form with desirable computational properties rely on transforming its matrix into upper-triangular form [8, 9]. The matrix for such a system will have mostly zero entries, which we call **sparse** [7]. We propose to analyze several methods of performing row-reduction, the process by which matrices are reduced into upper-triangular form [2].

What is special about row-reducing matrices in this context? When row-reducing a matrix, swapping rows or columns is typically acceptable. However, if the order of the terms in the polynomials must be preserved as in nonlinear systems, swapping columns of its matrix would be unacceptable. Another thing to consider is the “almost” upper-triangular structure of these matrices. Therefore, we want to adapt methods of performing row-reduction to allow swapping rows but not columns and perform the least amount of additions and multiplications..

CHAPTER 2

Background Material

We want to find an efficient way to transform a sparse matrix into upper-triangular form without swapping columns. What does this mean?

1. Sparse and Dense Matrices

A **vector** is a list of numbers, written horizontally between parentheses and without commas. For example,

$$(5\ 3\ 8\ 1\ 0)$$

is a vector of length 5. A **matrix** is a list of vectors of the same length written vertically between parentheses. For example,

$$A = \begin{pmatrix} -5 & 8 & 7 \\ 2 & -10 & 9 \\ 3 & 9 & 8 \end{pmatrix}$$

is a matrix of 3 vectors of length 3. Each vector corresponds to a **row** of the matrix. The second row of A is

$$(2\ -10\ 9)$$

which corresponds to the second vector in A. The list of ith entries in each vector is the ith **column** of the matrix. For example, the third column of A is

$$\begin{pmatrix} 7 \\ 9 \\ 8 \end{pmatrix}.$$

The **size** of a matrix is the number of rows by the number of columns. For A, the size would be 3×3.

A **sparse matrix** is a matrix with almost all zero entries. More precisely, a sparse matrix can be defined as having approximately $c \cdot m$ nonzero entries (where c is significantly smaller than m). This is different from a dense matrix that has closer to m^2 nonzero entries.

EXAMPLE 1. The matrix

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \end{pmatrix}$$

is sparse. On the other hand, the matrix

$$\begin{pmatrix} -5 & 8 & 7 \\ 2 & -10 & 9 \\ 3 & 9 & 8 \end{pmatrix}$$

is dense.

It is known that sparse matrix computations can be more efficient than dense matrix computations [7]. An especially important computation is transformation into upper-triangular form.

2. Row-Reduction

A matrix is in **upper-triangular form** if all entries below the main diagonal are zero entries. For example,

$$\begin{pmatrix} 1 & 7 & 2 \\ 0 & -5 & 1 \\ 0 & 0 & 10 \end{pmatrix}$$

is a matrix in upper-triangular form. To transform a matrix into upper-triangular form, we use the process of **row-reduction**, a method to reduce a matrix into upper-triangular form using **elementary row operations** [2]:

- swapping rows or columns,
- adding a multiple of one row to another, and
- multiplying the entries of a row by a number.

When adding a multiple of one row to another, we are trying to "clear" a column. The entry chosen to clear other entries in the column for row-reduction is called the **pivot**.

EXAMPLE 2. The origin of reducing matrices into upper-triangular form is the need to solve a system of linear equations. As an example, consider the system of linear equations

$$2x + 2y + 12z = -22$$

$$6x + 8y + z = 49$$

$$-8x + 10y + 2z = 28.$$

This corresponds to the matrix

$$\begin{pmatrix} 2 & 2 & 12 & -22 \\ 6 & 8 & 1 & 49 \\ -8 & 10 & 2 & 28 \end{pmatrix}.$$

With the matrix representation of this system of linear equations, we can use row reduction to solve the system. By multiplying the first row by -3 and adding it to the second row (the pivot is the entry in row 1 and column 1), we get

$$\begin{pmatrix} 2 & 2 & 12 & -22 \\ 0 & 2 & -35 & 115 \\ -8 & 10 & 2 & 28 \end{pmatrix}.$$

Then we multiply the first row by 4 and add it to the third row (the pivot is the entry in row 1 and column 2) to get

$$\begin{pmatrix} 2 & 2 & 12 & -22 \\ 0 & 2 & -35 & 115 \\ 0 & 18 & 50 & -60 \end{pmatrix}.$$

Finally, we multiply the second row by -9 and add it to the third row (the pivot is the entry in row 2 and column 2) to get

$$\begin{pmatrix} 2 & 2 & 12 & -22 \\ 0 & 2 & -35 & 115 \\ 0 & 0 & 365 & -1095 \end{pmatrix}.$$

Thus, we get a matrix that has been row-reduced into upper-triangular form. This corresponds to the system of equations

$$2x + 2y + 12z = -22$$

$$2y - 35z = 115$$

$$365z = -1095.$$

Using matrices, we have rewritten the system in a nice form that is easy to solve. Solving the third equation gives

$$z = -3.$$

Substituting into the second equation gives

$$\begin{aligned} 2y - 35\,(-3) &= 115 \\ 2y + 105 &= 115 \\ 2y &= 10 \\ y &= 5. \end{aligned}$$

Substituting into the first equation gives

$$x = 2.$$

3. Avoiding column swaps

As mentioned before, we use row-reduction to transform matrices into upper-triangular form, thus making solving the system of linear equations easier. The operations used to achieve this form include being able to swap rows or columns of the matrix; however, some applications of matrix triangularization do not allow for the swapping of columns. Since we need to preserve the order of the terms, we need to forbid swapping columns.

Avoiding swapping columns, however, forces us to perform more operations.

EXAMPLE 3. Consider the matrix

$$\begin{pmatrix} 3 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Swapping columns 1 and 3 gives

$$\begin{pmatrix} 0 & 1 & 3 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Then, if we swap rows 1 and 2 we get

$$\begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Swapping columns 2 and 4 gives us

$$\begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

which is in upper-triangular form.

Row-reducing the matrix using no column or row swaps significantly increases the number of operations from four swaps to ten arithmetic operations.

If there are more calculations, then what is the advantage of avoiding column swaps? An important application of this work is computing Gröbner bases [13, 3], where reducing sparse matrices to upper-triangular form reveals new basis elements. If column swaps were allowed, these new basis elements would not be revealed.

EXAMPLE 4. Each entry of the matrix below corresponds to the coefficients of the monomials of the equations.

$$\begin{array}{r} x^2y^3 + y^5 - 4y^3 = 0 \\ x^2y^3 + x^3 - 4x^2y = 0 \\ y^5 + y^2x - 4y^3 = 0 \\ x^3 + x^2y - 4x = 0 \end{array} \longleftrightarrow \left(\begin{array}{ccccccc} x^2y^3 & y^5 & x^3 & x^2y & xy^2 & y^3 & x \\ 1 & 1 & & & & -4 & \\ 1 & & 1 & -4 & & & \\ & 1 & & & 1 & -4 & \\ & & 1 & 1 & & & -4 \end{array}\right)$$

If this matrix were reduced to upper triangular form using no column swaps, we would have

$$\left(\begin{array}{ccccccc} x^2y^3 & y^5 & x^3 & x^2y & xy^2 & y^3 & x \\ 1 & 1 & & & & -4 & \\ & 1 & & & 1 & -4 & \\ & & 1 & 1 & & & -4 \\ & & & -5 & 1 & & 4 \end{array}\right).$$

The first rows correspond to three of the polynomial equations given, and the last row corresponds to a new equation: $-5x^2y+xy^2+4x$. This process has revealed a new basis element that is necessary for the Gröbner basis.

If the matrix were reduced and column swaps were allowed, we would have

$$\left(\begin{array}{ccccccc} xy^2 & y^5 & y^3 & x & x^2y^3 & x^3 & x^2y \\ 1 & 1 & -4 & & & & \\ & 1 & -4 & & 1 & & \\ & & & -4 & & 1 & 1 \\ & & & & 1 & 1 & -4 \end{array}\right) \longleftrightarrow \begin{array}{r} y^5 + y^2x - 4y^3 = 0 \\ x^2y^3 + y^5 - 4y^3 = 0 \\ x^3 + xy^2 - 4x = 0 \\ x^2y^3 + x^3 - 4x^2y = 0 \end{array},$$

which corresponds to the polynomial equations from the original matrix. This has failed to reveal the new basis element necessary for the Gröbner basis.

The goal of this project is to compare different methods of row-reduction of sparse matrices without swapping the columns.

4. Exact Computation

In computational algebra, we are concerned with exact computations and not with floating point numbers. Because of this, we are not concerned with the stability of the information contained in the matrices. To keep the integers from growing too large in computation, the coefficients in the systems correspond to elements of **finite fields** [12]. In a finite field, arithmetic behaves as it does on a clock: if the numbers grow too large or too small, we divide and take the remainder.

EXAMPLE 5. Consider the finite field $\mathbb{Z}_7$. The elements of $\mathbb{Z}_7$ are

$$\{0, 1, 2, 3, 4, 5, 6\}.$$

Within this finite field, for example,

$$2 + 3 = 5,$$

$$2 \times 3 = 6,$$

and for numbers that grow too large,

$$6 + 1 = 7 \to 0,$$

$$3 \times 4 = 12 \to 5.$$

Division corresponds to multiplication of the multiplicative inverse of the element. For example,

$$3 \div 4 = 3 \times 2 = 6$$

because 2 is the multiplicative inverse of 4:

$$2 \times 4 = 8 \to 1.$$

It is beyond the scope of this work to describe the reasons finite fields are preferred in computational algebra, and how we reconstruct the correct answers in a different field when necessary. However, this is an important problem with several important applications.

5. Applications

Some applications of the research include

- cryptology [1, 10]
- decoding gene expression [11]
- F4 algorithm, which uses sparse matrix reduction with fixed columns to compute Gröbner bases [8].

6. Macaulay Matrices

6.1. Ideals and Gröbner Bases.

As mentioned in Example 4, an important application of avoiding swapping columns during the triangularization of a matrix is computing Gröbner bases. In order to understand Gröbner bases, an understanding of ideals is needed. Let $\mathbb{F}[x_1, \ldots, x_n]$ be the set of all polynomials in $x_1, \ldots, x_n$ with real coefficients. Let $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$. The **ideal** generated by $f_1, \ldots, f_m$, written $\langle f_1, \ldots, f_m \rangle$, is the set I consisting of all expressions $h_1 f_1 + \ldots + h_m f_m$ where each h_i is some element of $\mathbb{F}[x_1, \ldots, x_n]$.

EXAMPLE 6. Consider $F = \left\{xy - 1, x^2 + y^2 - 4\right\}$. Appropriately, we label the polynomials $f_1 = xy - 1$ and $f_2 = x^2 + y^2 - 4$. An element of the ideal generated by F and $h_1 = x^2$ and $h_2 = -xy$ is

$$\begin{aligned} h_1 (xy - 1) + h_2 \left(x^2 + y^2 - 4\right) &= \left(x^2\right)(xy - 1) + (-xy)\left(x^2 + y^2 - 4\right) \\ &= x^3y - x^2 - x^3y - xy^3 + 4xy \\ &= -x^2 - xy^3 + 4xy. \end{aligned}$$

A **Gröbner basis** is a "nice form" for the generators of an ideal. By "nice form," I mean it can answer important questions of commutative algebra and algebraic geometry easily and quickly [5].

Before we go any further in the explanation of Gröbner bases, we need to discuss ordering and leading terms. Normally, when we deal with monomials with one variable, we order the terms of a polynomial in order of decreasing degree. However, we can see that this is more difficult when there are several variables involved. There are two commonly use orderings for monomials with more than one variable. **Lexicographic** ordering puts the variables in order alphabetically first, then by degree. The other commonly used ordering for polynomials is **total degree** (or graded reverse lexicographic). This ordering involves putting monomials of highest total degree first. If there are two monomials of the same degree, the variable in the monomial that is last alphabetically is "removed" and the degree is determined again.

For the purpose of these examples, I will use total degree ordering. This leads to the definition of the leading term. The leading term of a polynomial is simply the term that is first in the polynomial after an ordering has been applied to the polynomial.

EXAMPLE 7. One might expect that every element of an ideal would have a leading term divisible by the leading term of the generator. In reality this is not always true. For example, consider F from the previous example and $h_1 = x$ and $h_2 = y$, so

$$\begin{aligned} h_1(xy-1) - h_2\left(x^2+y^2-4\right) &= x^2y - x - x^2y - y^3 - 4y \\ &= -x - y^3 - 4y. \end{aligned}$$

Notice the leading term y^3 is not divisible by the leading terms of the generator F.

Gröbner bases are often computed using a matrix. In the case of this research, this is the Gröbner basis computation of concern.

The most common algorithm used to compute Gröbner is the Buchberger algorithm [3] . The Buchberger algorithm takes a finite set of polynomials and outputs a Gröbner basis of the ideal generated by a generator called F. Let $G = F$. Then repeat the following until all distinct pairs (f, g) in the finite set of polynomials has been considered:

- Pick an unconsidered pair
- Reduce its s-polynomial with respect to G (comparable to taking a row of a matrix and performing elimination on it)
- If result is non-zero, add it to G

The s-polynomial is the sum of the smallest term multiples of the polynomials in the pair that cancel the leading terms.

Here is an example of using Buchberger's algorithm with a matrix

EXAMPLE 8. Consider $F = \left\{x^2 + 2x + y^2 - 3, xy - 1\right\}$. Each time a step is taken to compute the Gröbner basis, the following question must be asked: What must be multiplied to the polynomials in F in order to cancel the leading terms? The first terms to be multiplied to the polynomials must be $h_1 = y$ and $h_2 = -x$. Then,

$$\begin{aligned} h_1 f_1 + h_2 f_2 &= y\left(x^2 + 2x + y^2 - 3\right) - x\left(xy - 1\right) \\ &= x^2y + 2xy + y^3 - 3y - x^2y + x \\ &= 2xy + y^3 - 3y + x. \end{aligned}$$

When we multiply h_1 and h_2 to f_1 and f_2 respectively, we represent the resulting polynomials as follows: yf_1 and $-xf_2$. Each time we multiply the polynomials in F by different elements to try to cancel the leading terms, we look at each of the terms produced during the multiplication. If it is a term that is not already in F, we add it to F. Ultimately, we cease to get any new polynomials. When this happens, we put each different monomial included in the polynomials in F into the list T. The following terms are included in all the polynomials generated:

$$T = \left\{x^2y^3, xy^3, y^5, y^3, x^3y, x^3, x^2y, x^2, y^2, xy, x, y, xy^2, 1\right\}.$$

In the matrix representation of the set of polynomials in F, the columns are labeled by the monomials in T in the order deter mind by total degree ordering. The rows are labeled according to the polynomials formed when multiplying values of h_m to values of f_m to include all the resulting polynomials without repeating any two identical polynomials. The entries of the matrix are the coefficients of the monomials listed in T that are included in each of the polynomials listed as row labels. The polynomials containing these terms are represented by the following matrix, called a

Macaulay matrix [14]:

$$\left(\begin{array}{rrrrrrrrrrrrrrr}
 & x^2y^3 & xy^3 & x^3y & xy^3 & x^3 & x^2y & xy^2 & y^3 & x^2 & xy & y^2 & x & y & 1 \\
y^3f_1 & 1 & 1 & 0 & 2 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\
-x^2f_3 & -1 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
y^2f_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
-xf_3 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 & 0 & 0 \\
y^2f_3 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & -3 & 0 & 0 & 1 & 0 & 0 & 0 \\
f_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & -3 & 1 \\
x^2f_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
xf_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 2 & 0 & -3 & 0 & 0 \\
xf_2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
f_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & -3 \\
f_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\
yf_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0
\end{array}\right)$$

Buchberger's algorithm used was the first to explain Gröbner basis computation; however, it was not the most efficient way of calculating them. Since Gröbner computation is a very lengthy process, a more efficient process was needed. The **normal strategy** was one solution to this problem. The normal strategy involves taking the pair (f, g) with $lcm\,(lm\,(f)\,, lm\,(g))$ (where lcm is the least common multiple and lm is the leading monomial). This strategy was understood to work well because of the Buchberger's lcm criterion [4] which allows one to skip many s-polynomials. The normal strategy was experimentally very effective and was the most used strategy until very recently.

Notice the structure of the matrix in Example 8. Since the goal of each step of computing the Gröbner basis is to eliminate the leading terms of the generator, putting the polynomials in the order they are found shows that the matrix is in a form very similar to upper-triangular form. Performing Gaussian elimination from right-to-left might be more effective because of this structure. This could be due to the normal strategy. When going right to left, we go from smallest monomial to largest and any elimination is the lcm of two monomials. Instead of looking at lm's we are looking at non-leading monomials, which references the normal strategy.

EXAMPLE 9. Here is a matrix, which, by performing Gaussian elimination from right-to-left, takes many calculations to be triangularized:

$$\begin{pmatrix} 1 & -1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 2 & 0 & 4 \\ 1 & -1 & 0 & 3 & 0 & 5 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

When reduced, the resulting matrix is

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

However, the number of multiplications and additions for the left-to-right method is 43, compared to 69 for the right-to-left method. This illustrates how the type of matrix I am dealing with (specifically Macaulay matrices) is a very special case such that the structure suggests that choosing a pivot from right-to-left would be more efficient.

CHAPTER 3

Strategy

To look at the different methods used to transform a matrix into upper-triangular form, we will produce several computer programs that implement well-known methods of row-reducing sparse matrices. We will then analyze the programs' behavior, and adapt them to perform row-reduction without swapping columns. Then we will analyze the programs' behavior again, and determine the best algorithm to use in this context.

Table 3.1 describes the typical strategy used in numerical methods to solve a linear system. We, however will not use every step described there.

Following the steps of Table 3.1, we will implement the Analyze and Factorize phases for each algorithm. The Analyze phase of the program will create the matrix structure and determine what

TABLE 3.1. Solving sparse matrices.[7].

Phase	Key Features
ANALYZE	1. Transfer user data to internal data structures. 2. Determine a good pivotal sequence. 3. Prepare data structures for the efficient execution of the other phases. 4. Output of statistics.
FACTORIZE	1. Transfer new numerical values to internal data structure format. 2. Factorize the matrix using the chosen pivotal sequence. 3. Estimate condition number of the matrix. 4. Monitor stability to ensure reliable factorization.
SOLVE	1. Perform the appropriate permutation from user order to internal order. 2. Perform forward substitution and back-substitution using stored **L\U** factors. 3. Perform the appropriate permutations from internal order to user order and return the solution in user order.

approach needs to be taken to reduce it. This approach will help to determine how the next step—factorize—is done. One reason we need the analyze phase is because sparse matrices tend to be very large. Some of these sparse matrices could take a very long time to factorize if we do not first analyze their structure. This reduces the possibility of unnecessary calculations. Some methods are more efficient with certain shapes than other methods are.

The Analyze phase has four main steps. The first step involves putting the given matrix into a structure appropriate to the computer language. Most of the algorithms we will use will help with the next step: “determining a good pivotal sequence.” Determining the pivotal sequence will involve analyzing the matrix structure to see if it fits with a specific algorithm. Then the next step will involve preparing this matrix for the algorithm to perform reduction. Finally, the Analyze phase ends with the output of the statistics collected while determining the pivotal sequence.

The Factorize phase is the most computationally intensive. This is where the row-reduction will take place to transform the matrix into upper-triangular form. The second phase also involves four steps. The first step is nearly identical to the first of the Analyze phase except that the matrix will be copied to a new structure. At the end of the Factorize phase, we have a matrix in upper-triangular form. We do not need to determine the condition number of the matrix since condition numbers are not needed with exact mathematics [15]. The final step is not needed in my research.

Since the long-term application of my research is computing Gröbner bases, we will only be dealing with exact computations and not the approximation common in numerical methods so stability is not a concern. Moreover, we only need to reduce the matrices into upper-triangular form and not solve them.

Some well-known algorithms we will study are CSparse, the Doolittle Algorithm, Markowitz Criterion, the Sargent and Westerberg algorithm, Tarjan’s algorithm, Tinney Scheme 2, and an algorithm for band matrices [7, 6].

When looking at the CSparse algorithm for reducing matrices into upper-triangular form I found that it is an implementation of several different algorithms. The part of CSparse I am concerned with is the implementation of the LU factorization algorithm. LU factorization is similar to Gaussian elimination in that part of LU factorization puts the matrix in upper triangular form (what we want). However, it also involves putting the matrix in lower triangular form, but we are not concerned with this part. While looking at the code for this algorithm, I found that LU factorization assumes that the matrices are square, or $n \times n$ matrices [6]. Then I found that all the algorithms I will be researching assume square matrices [7] as inputs. At first glance, this seems to be a problem because these algorithms would have to be adapted to allow for the input of $n \times m$ matrices because, in general, the matrices I am concerned with will not be square matrices. However, I have determined that the algorithm can still work with a square matrix by taking the non-square matrices and partitioning them into a square matrix and the remaining part of the matrix. The basic structure of the matrix would be similar to this:

$$\left(\begin{array}{c|c} A & B \end{array} \right)$$

where A is the $n \times n$ component of the matrix and B is the remaining $n \times (m - n)$ part of the matrix. This partition would require only a slight adaptation to the algorithms. The B partition of the matrix would still require all the calculations, but is not needed in the major implementation of the algorithms. Also, since I am adapting the algorithms to avoid swapping columns, the calculations and row swaps will not affect the structures of the partitions since it is partitioned by the column only. These algorithms can all be used to transform matrices into upper-triangular form. Each uses different methods of reduction, so we may not be able to adapt all of them since some may rely fundamentally on swapping columns. Here are brief descriptions of the previously listed algorithms:

- CSparse [6]
  - CSparse performs Gaussian elimination on a matrix that has been analyzed by another function within the CSparse package.
- The Doolittle algorithm
  - The Doolittle algorithm is an algorithm that performs Gaussian elimination and has no analysis function.
- Markowitz Criterion

  - Markowitz criterion chooses a pivot in the following way. For each of the remaining columns to clear in the matrix, the algorithm checks each of the possible pivots using the following equation:

    $$(r_i - 1)(c_j - 1),$$

    where r_i is the number of entries in row i and c_j is the number of entries in column j. The pivot will be the entry that minimizes that expression. The purpose of the previous expression is to reduce fill-in when row-reducing by choosing the row and column with the smallest possible calculations to be performed.
- The Sargent and Westerberg
  - The Sargent and Westerberg algorithm involves looking at matrices that are symmetric permutations of triangular matrices.
- Tarjan's Algorithm
  - Tarjan's algorithm is similar to the Sargent and Westerberg, making only one change in the storage of data.
- Tinney Scheme 2
  - The Tinney Scheme 2 algorithm deals with a special case of the Markowitz Criterion when the matrix is symmetric.
- Band matrices
  - A band matrix is a matrix whose bandwidth (defined in the results section) is significantly smaller than the number of rows or columns in the matrix.

1. Computer Languages

When implementing these algorithms, we will first work in the Sage computer algebra system. Sage has built-in commands for building matrices and performing the different techniques of row-reduction [16].

CHAPTER 4

Results

1. Abandoned Algorithms

After looking at each of the algorithms mentioned in the previous chapter, I have determined, for several reasons, that some of them are incompatible with the Macaulay matrices I am working with. CSparse [6] uses Gaussian elimination from left-to-right to row-reduce a matrix whose structure is indicated by a parameter passed to the function. Also, the main concern of [6] is to store the matrices most efficiently, which is not the point of this research. The Doolittle algorithm is also essentially an algorithm written for Gaussian elimination. The Sargent and Westerberg algorithm and Tarjan's algorithm both involve working with symmetric matrices, and, therefore, are not applicable to this research. Although the Tinney Scheme 2 is an algorithm that uses Markowitz Criterion, it only works with symmetric matrices; again, this is not applicable to this research. Due to the structure of the Macaulay matrices, I initially thought some methods of reducing band matrices may be applicable. However, after computing the bandwidth of the matrices of concern, I determined any method of reducing band matrices is not applicable to this research.

Table 4.1 shows the bandwidths of each of the matrices I am working with. According to [7], for a matrix to be considered a band matrix, the bandwidth must be significantly lower than the

TABLE 4.1. Bandwidths

Matrix	Bandwidth
8×11	9
13×16	13
15×14	12
13×16	15
11×14	13
2×5	4

number of rows in the matrix, where the bandwidth is $2m + 1$ where m is the smallest integer such that $a_{ij} = 0$ whenever $|i - j| > m$ [7]. In Table 4.1, it is shown that the bandwidths of the matrices are significantly larger than the number of rows in the matrices! Thus, any methods of reducing band matrices are not applicable to this research.

The methods of choosing a pivot I will discuss are the "textbook" left-to-right method, a right-to-left method, and Markowitz Criterion. The algorithms I wrote to implement these methods are discussed in the next section.

2. Description of Algorithms

Each algorithm implemented uses the method of Gaussian elimination to triangularize an $n \times m$ matrix. The algorithm to choose a pivot from left-to-right or right-to-left takes the matrix and the direction to choose a pivot specified by a Boolean value as inputs. The `true` value will make the algorithm triangularize from left to right, and `false` will make the algorithm triangularize from right to left. When choosing the pivot, if there is more than one row eligible to be the pivot, the one with the fewest non-zero entries, and thus the fewest calculations is chosen. If left-to-right is the direction, the pivot is chosen by searching rows for the leading nonzero numbers. This is shown in Algorithm 1. For the right-to-left method, the last column is searched first and the first is searched last. This is shown in Algorithm 2. The direction, in other words, determines how the pivot is chosen. Once the pivot is determined, the way the calculations are done is the same for both methods.

Algorithm 3 involves Gaussian elimination using Markowitz criterion (Algorithm 4) to choose a pivot.

Markowitz criterion is implemented according to the description in Chapter 3.

Algorithm 5 and Algorithm 6 are both functions used within the Gaussian elimination algorithms. Algorithm 5 is used to find the entries that must be cleared within the right-to-left and left-to-right algorithms. Algorithm 6 is used to eliminate the entries in the column of the pivot using the pivot.

ALGORITHM 1.

```
algorithm Gaussian Elimination Left-to-Right
  inputs
    $M \in \mathbb{F}^{m \times n}$
  outputs
    $N$, upper-triangular form
    $c$, number of calculations
    $s$, number of row swaps
  do
    let $N = M$
    let $s = c = 0$
    for $j \in \{1, \ldots, n\}$ do
      let $i$ be the row whose first nonzero entry is in column $j$
      — Get a one in the first entry
      if $N_{i,j} \neq 1$ then
        let $inv = N_{i,j}^{-1}$
        for $\ell \in \{j, \ldots, n\}$ do
          let $N_{i,\ell} = N_{i,\ell} \cdot inv$
          increment $c$
      — Clear this column
      let $N, c = Clear\ Column(N, c, i, j)$
    for $i \in \{1, \ldots, m-1\}$ do
      let $j = \min\{j : N_{ij} \neq 0\}$
      if $\exists k$ such that $j > \min\{\ell : N_{k\ell} \neq 0\}$ then
        swap rows $i, k$
        increment $s$
    return $N, c, s$
```

The reason for developing the algorithm for both directions was to determine which direction would produce fewer calculations. Since the matrices representing Gröbner bases have a structure very close to a matrix that is already in upper triangular form, working from right to left appeared to produce fewer calculations than working from left to right. I also wanted to include the Markowitz criterion algorithm to determine if it is more efficient than the other two for computing Gröbner bases.

To test this hypothesis, I implemented and tested the three algorithms using 6 different matrices. The matrices are part of a system, called Cyclic-4, that is based on the relationship between the roots and the coefficients of the polynomial $x^4 + 1$. Tables 4.2 and 4.3 show the results of the test, over the fields specified. The numbers for each direction and heuristic for pivot choice are the number of additions and multiplications and the number of row swaps, respectively.

ALGORITHM 2.

```
algorithm Gaussian Elimination Right-to-Left
  inputs
    $M \in \mathbb{F}^{m\times n}$
  outputs
    $N$, upper-triangular form
    $c$, number of calculations
    $s$, number of row swaps
  do
    let $N = M$
    let $s = c = 0$
    let $j = n$
    while $j \neq 0$ do
      let $next\_j = j - 1$
      if $\exists i$ such that the first nonzero entry of row $i$ is in column $j$ then
        if $N_{ij} \neq 1$ then
          let $inv = N_{ij}^{-1}$
          for $\ell \in \{j, \ldots, m\}$ do
            if $N_{i\ell} \neq 0$ then
              let $N_{i\ell} = N_{i\ell} \cdot N_{ij}^{-1}$
              increment $c$
        let $N, c = Clear\ Column(N, c, i, j)$
        let $next\_j = n$
      let $j = next\_j$
    for $i \in \{1, \ldots, m-1\}$ do
      let $j = \min\{j : N_{ij} \neq 0\}$
      if $\exists k$ such that $j > \min\{\ell : N_{k\ell} \neq 0\}$ then
        swap rows $i, k$
        increment $s$
    return $N, c, s$
```

TABLE 4.2. Additions and Subtractions over the Field $\mathbb{Q}$

	$\mathbb{Q}$		
Matrix	**Left to Right**	**Right to Left**	**Markowitz Criterion**
8×11	84, 10	66, 10	66, 7
13×16	375, 36	344, 31	362, 35
15×14	286, 20	230, 19	232, 12
13×16	313, 19	193, 26	197, 28
11×14	156, 10	151, 9	156, 10
2×5	8, 1	8, 1	8, 1

3. Conclusions

The success of the right-to-left algorithm in $\mathbb{Q}$ (compared to the other algorithms) can possibly be explained by the success of the normal strategy (using Buchberger's lcm criterion) and the structure

ALGORITHM 3.

```
algorithm Gaussian Elimination using Markowitz Criterion
  inputs
    $M \in \mathbb{F}^{m \times n}$
  outputs
    $N$, upper-triangular form
    $c$, number of calculations
    $s$, number of row swaps
  do
    let $N = M$
    let $s = c = 0$
    let $C = Find\ Columns\ (N, 1)$
    while $C \neq \emptyset$ do
      let $i, j = Markowitz\ Criterion(N, C)$
      — Get a one in the first entry
      if $N_{i,j} \neq 1$ then
        let $inv = N_{i,j}^{-1}$
        for $\ell \in \{j, \ldots, n\}$ do
          if $N_{i,\ell} \neq 0$ then
            let $N_{i,\ell} = N_{i,\ell} \cdot inv$
            increment $c$
      — Clear this column
      let $N, c = Clear\ Column(N, c, i, j)$
      let $C = (C \cap \{1, \ldots, j-1\}) \cup Find\ Columns(N, j+1)$
    return $N, c, s$
```

ALGORITHM 4.

```
algorithm Markowitz Criterion
  inputs
    $M \in \mathbb{F}^{mxn}$
    $C \subseteq \{1, \ldots, n\}$
    — columns that need to be reduced
  outputs
    $i, j \in \{1, \ldots, n\}$
  do
    let $(i, j) = (0, 0)$
    let $score = \infty$
    for $\ell \in C$ do
      let $c_\ell =$ number of non-zero entries in column $\ell$
      for $k \in \{1, \ldots, m\}$ do
        if $M_{k\ell}$ is a leading entry then
          let $r_{k\ell} =$ number of entries in row $k$
          if $(r_{k\ell} - 1)(c_\ell - 1) < score$ then
            let $(i, j) = (k, \ell)$
            let $score = (r_{k\ell} - 1)(c_\ell - 1)$
    return $(i, j)$
```

ALGORITHM 5.

```
algorithm Find Columns
  inputs
    $M \in \mathbb{F}^{m \times n}$
    $j \in \{1, \dots, n\}$
  outputs
    $C \subseteq \{1, \dots, n\}$
  do
    let $C = \emptyset$
    for $\ell \in \{j, \dots, n\}$ do
      let $pos = nonzero\ positions\ in\ column$
      if $|pos| > 1$ then
        for $i \in pos$ do
          let $\hat{\ell} = first\ nonzero\ pos\ in\ row$
          if $\hat{\ell} = \ell$ then
            add $\ell$ to $C$
    return $C$
```

ALGORITHM 6.

```
algorithm Clear Column
  inputs
    $N \in \mathbb{F}^{m \times n}$
    — calculation count
    $c \in \mathbb{N}$
    $i \in \mathbb{N}$
    $j \in \mathbb{N}$
  do
    for $k \in \{1, \dots, m\} \setminus \{i\}$ do
      if $N_{k,j} \neq 0$ then
        let $mul = N_{k,j}$
        for $\ell \in \{j, \dots, n\}$ do
          if $N_{k,\ell} \neq 0$ or $N_{i,\ell} \neq 0$ then
            let $N_{k,\ell} = N_{k,\ell} - N_{i,\ell} \cdot mul$
            increment $c$
    return $N, c$
```

TABLE 4.3. Additions and Subtractions over the Field $\mathbb{F}_2$

	$\mathbb{F}_2$		
Matrix	**Left to Right**	**Right to Left**	**Markowitz Criterion**
8×11	72, 10	60, 10	60, 7
13×16	274, 42	294, 36	318, 38
15×14	240, 20	196, 19	192, 12
13×16	256, 19	156, 26	152, 28
11×14	132, 10	120, 9	112, 9
2×5	8, 1	8, 1	8, 1

of Macaulay matrices. This is somewhat surprising because the Markowitz criterion is best for a generic matrix, and as Example 9 showed, right-to-left is spectacularly bad for a generic matrix.

However, this result is not all that surprising due to the structure of the matrix. We can separate the matrix into several partitions, for example,

$$\begin{pmatrix} \multirow{2}{*}{A} & b_1 & \\ & b_2 & C \\ 0 & b_3 & \end{pmatrix}$$

so that A contains rows only up to the column of the last leading entry, the center column (containing b_1, b_2 and b_3) contains the last column with a leading entry (b_3), and C contains the remaining entries. When performing row-reductions, the calculations done in C should be the about the same no matter the method used to choose the pivot. However, the calculations in the center column would be affected by the pivot choice.

Choosing from left-to-right, we would clear the left most columns first. For example, choosing a pivot in A would create the following matrix

$$\begin{pmatrix} \multirow{2}{*}{A'} & b_1' & \\ & b_2' & C' \\ 0 & b_3 & \end{pmatrix}.$$

Each time we performed a reduction, b_1' and b_2' would probably still contain nonzero entries, and next time a reduction is done, more calculations will be performed on the same entries again. The Markowitz criterion can work the same way, repeating calculations on rows.

However, when choosing a pivot from right to left, since there are no nonzero values to the left of the leading entry (the pivot), when reductions are performed, calculations are only performed on entries in the column of the chosen pivot. For example, if b_3 were the pivot, the center column (b-column) would contain only b_3 after the reduction:

$$\begin{pmatrix} \multirow{2}{*}{A} & 0 & \\ & 0 & C' \\ 0 & b_3 & \end{pmatrix}$$

Moving to the next pivot will not perform any extra calculations in column b.

The success of the Markowitz criterion in $\mathbb{F}_2$, however, may not be as easily explained as the previous results. Since the field $\mathbb{F}_2$ could make the matrices I tested more sparse (by make entries divisible by 2 go to zero), this could make the Markowitz criterion method more successful than the other methods.

Ultimately, some possible explanations for the success of the right-to-left method in $\mathbb{Q}$ are the structure of the Macaulay matrices and the normal strategy, while the possible explanation for the success of Markowitz criterion in $\mathbb{F}_2$ is the increasing sparsity when a matrix has entries in $\mathbb{F}_2$. These possible explanations are limited to the matrices in the Cyclic 4 system (the matrices tested); however, since these matrices are relatively small compared to the general case of Macaulay matrices, the success of Markowitz criterion over $\mathbb{F}_2$ may increase as the size (and sparsity) of the matrix increases.

Bibliography

[1] Peter Ackermann and Martin Kreuzer. Gröbner basis cryptosystems. *Applicable Algebra in Engineering, Communication and Computing*, 17(3):173–194, 2006.

[2] Howard Anton and Chris Rorres. *Elementary Linear Algebra: Applications Version*. Anton Textbooks, Inc., ninth edition, 2005.

[3] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalem Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal)*. PhD thesis, Mathematical Insitute, University of Innsbruck, Austria, 1965. English translation published in the Journal of Symbolic Computation (2006) 475–511.

[4] Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner bases. In E. W. Ng, editor, *Proceedings of the EUROSAM 79 Symposium on Symbolic and Algebraic Manipulation, Marseille, June 26-28, 1979*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21, Berlin - Heidelberg - New York, 1979. Springer.

[5] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Spring-Verlag New York, Inc., second edition, 1998.

[6] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[7] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications. Oxford University Press, 1986.

[8] Jean-Charles Faugére. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139:61–88, 1999.

[9] Jean-Charles Faugére. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83. ACM Press, 2002.

[10] Jean-Charles Faugère. Cryptochallenge 11 is broken or an efficient attack of the C* cryptosystem. Technical report, LIP6/Universitè Paris, 2005.

[11] Rienhard Laubenbacher and Brandilyn Stigler. A computational algebra approach to the reverse engineering of gene regulatory networks. *Journal of Theoretical Biology*, (229):523–537, 2004.

[12] Niels Lauritzen. *Concrete Abstract Algebra: From Numbers to Gröbner Bases*. Cambrdge University Press, 2003.

[13] Daniel Lazard. Gröbner bases, Gaussian elimination, and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *EUROCAL '83, European Computer Algebra Conference*, volume 162, pages 146–156. Springer LNCS, 1983.

[14] F. S. Macaulay. On some formulæ in elimination. *Proceedings of the London Mathematical Society*, 33(1):3–27, 1902.

[15] John Perry. Numerical analysis. Unpublished, 2000. Class notes from Dr. Pierre Gremaud's lectures.

[16] William Stein. *Sage: Open Source Mathematical Software (Version 3.1.1)*. The Sage Group, 2008. `http://www.sagemath.org`.

Appendix

Matrices Tested

FIGURE 4.1. M_2 (2×5)

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 0 \end{pmatrix}$$

FIGURE 4.2. M_3 (8×11)

$$\begin{pmatrix}
1 & -1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
-1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 \\
-1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0
\end{pmatrix}$$

FIGURE 4.3. M_4 (14×15)

$$\begin{pmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
-1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Figure 4.4. M_5 (13×16)

$$\begin{pmatrix}
0 & 0 & 0 & -1 & 0 & 1 & -1 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\
-1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
-1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
-2 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Figure 4.5. M_6 (13×16)

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
2 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\
0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\
1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

FIGURE 4.6. M_7 (11×14)

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & -2 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & -1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & -1 & -1 & 0 & 0 & 1 & 1 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 0 & 1 & -2 & 1 & 0 & 0 \\
1 & 0 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 1 & -2 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\
1 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 0 & -1
\end{pmatrix}$$

Matrices Plotted

FIGURE 4.7. M_4 (14×15)

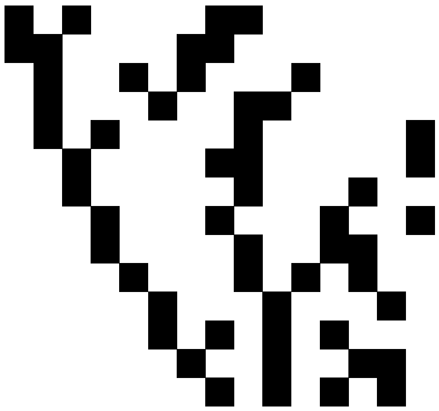


FIGURE 4.8. M_5 (13×16)

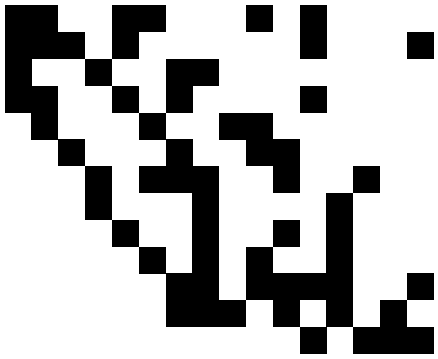


FIGURE 4.9. M_6 (13×16)

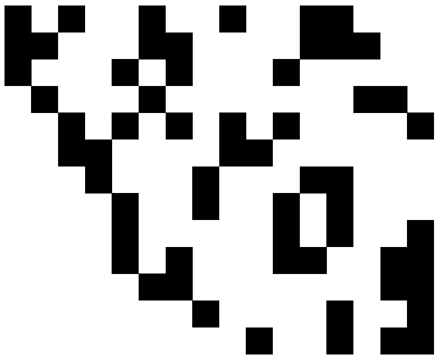
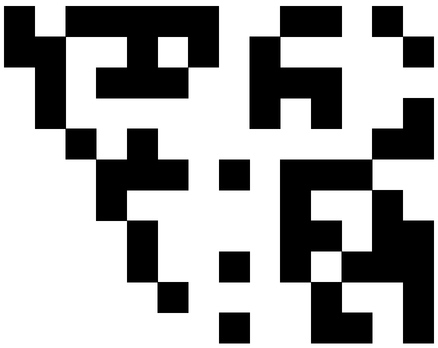


FIGURE 4.10. M_7 (11×14)



Sage Code

LISTING 4.1. markowitz.py

```
# M is an mxn matrix
# direction = True implies triangularize left to right
# direction = False implies triangularize right to left
def mark(M):
  c = 0
  s = 0
  N = M.copy()
  m = M.nrows()
  n = M.ncols()
  C = find_columns(N,0,n)
  while(len(C) != 0):
    i,j = markowitz_criterion(N,C)
    if N[i,j] != 1:
      inv = (N[i,j])**(-1)
      for l in xrange(j,n):
        if N[i,l] != 0:
          N[i,l] = N[i,l]*inv
          c = c + 1
    for k in xrange(m): #not sure how to take out i
      if (k!=i) and (N[k,j] != 0):
        mul = N[k,j]
        for l in xrange(j,n):
          #if (N[k,l] != 0) or (N[i,l] != 0):
          if (N[i,l] != 0):
            N[k,l] = N[k,l] - N[i,l]*mul
            c = c + 2
    find = find_columns(N,j+1,n) #min(pos)
    C = find.union(C.intersection(range(j)))
  for i in xrange(m):
    pos = N.nonzero_positions_in_row(i)
    if (len(pos) != 0):
        j = min(pos)
        for k in xrange(i + 1,m):
          pos = N.nonzero_positions_in_row(k)
          if (len(pos) != 0 and min(pos) < j):
            N.swap_rows(i,k)
            s = s + 1
            j = min(pos)
  return N,c,s

def markowitz_criterion(N,C):
  i = 0
  j = 0
  score = infinity
  for l in C:
    pos = N.nonzero_positions_in_column(l)
    c_l = len(pos)
    for k in pos:
      pos2 = N.nonzero_positions_in_row(k)
      leading_entry = min(pos2)
      if (l == leading_entry):
        r_kl = len(pos2)
        if ((r_kl-1)*(c_l-1) < score):
          i = k
          j = l
          score = (r_kl-1)*(c_l-1)
  return i,j

def find_columns(N,j,n):
  C = set()
  for l in xrange(j,n):
    pos = N.nonzero_positions_in_column(l)
    if len(pos)>1:
      for i in pos:
        pivot = min(N.nonzero_positions_in_row(i))
        if pivot == l:
          C.add(l)
  return C
```

LISTING 4.2. testtriangularize.py

```
# M is an mxn matrix
# direction = True implies triangularize left to right
# direction = False implies triangularize right to left
def test_triangularize(M, direction=True):
  c = 0
  s = 0
  count = 0
  N = M.copy()
  m = M.nrows()
  n = M.ncols()
  if direction:
    for j in xrange(n):
      i = find_pivot_row_for_col(N,m,j,0)
      if i != -1:
        # make row start w/1
        if (N[i,j] != 1):
          mul = N[i,j]**(-1)
          for l in xrange(j,n):
            if (N[i,l] != 0):
              N[i,l] = N[i,l] * mul
              c = c + 1
        # clear this column in other rows
        for k in xrange(m):
          pos = N.nonzero_positions_in_row(k)
          if (i != k) and N[k,j] != 0 and (len(pos) != 0) and (min(pos) <= j):
            mul = N[k,j]
            for l in xrange(j,n):
              if N[i,l] != 0: #or N[k,l] != 0:
                N[k,l] = N[k,l] - mul*N[i,l]
                c = c + 2
    for i in xrange(m):
      pos = N.nonzero_positions_in_row(i)
      if (len(pos) != 0):
        j = min(pos)
        for k in xrange(i + 1,m):
          pos = N.nonzero_positions_in_row(k)
          if (len(pos) != 0 and min(pos) < j):
            N.swap_rows(i,k)
            s = s + 1
            j = min(pos)
  else:
    # triangularize right to left
    j = n - 1
    while j >= 0:
      next_j = j - 1
      i = find_pivot_row_for_col(N,m,j,0)
      if (i != -1):
        if (N[i,j] != 1):
          mul = N[i,j]** (-1)
          for l in xrange(j,n):
            if N[i,l] != 0:
              N[i,l] = N[i,l] * mul
              c = c + 1
        for k in xrange(0,m):
          if (k != i and N[k,j] != 0):
            mul = N[k,j]
            for l in xrange(j,n):
              if N[i,l] != 0: #or N[k,l] != 0:
                N[k,l] = N[k,l] - mul*N[i,l]
                c = c + 2
            next_j = n - 1
      j = next_j
    for i in xrange(m):
      pos = N.nonzero_positions_in_row(i)
      if (len(pos) != 0):
        j = min(pos)
        for k in xrange(i + 1,m):
          pos = N.nonzero_positions_in_row(k)
          if (len(pos) != 0 and min(pos) < j):
            N.swap_rows(i,k)
            s = s + 1
            j = min(pos)
  return N, c, s

def find_pivot_row_for_col(N,m,j,k):
  # N is matrix
  # m is number of rows
  # j is column
  # k is first row to search
  candidates = {}
  for i in xrange(k,m):
```

```
  pos = N.nonzero_positions_in_row(i)
  if (len(pos) != 0) and (min(pos) == j):
    candidates[len(pos)] = i
if len(candidates) != 0:
  shortest_candidate = min(candidates.keys())
  return candidates[shortest_candidate]
else:
  return -1
```