Ball State Undergraduate Mathematics Exchange
http://www.bsu.edu/libraries/beneficencepress/mathexchange
Vol. 13, No. 1 (Fall 2019)
Pages 20 – 35

Sudoku Solutions in MuPAD Using a Gröbner Basis

Arlan (A.J.) Zelenky and Ry Gallagher

Arlan (A.J.) Zelenky graduated from Saint Francis University in 2019 with a degree in Mathematics with a concentration in Secondary Education. He intends to go into the teaching field and hopes to teach at both the high school and college levels.

Ry Gallagher is a senior student with double major in Mathematics and Computer Science at Saint Francis University. He will pursue a career in Information Technology after graduation.

Abstract A Gröbner basis is a construction in algebraic geometry that aids the study of the zero locus of a set of polynomials. We present MuPAD programs that perform a Gröbner basis computation for the ideal describing conditions of a Shidoku (four-by-four puzzle) board or a Sudoku (nine-by-nine puzzle) board. This is implemented using three different strategies : the sum and product method, the binary method, and the quotient method. The paper also includes a theorem guaranteeing the result of the quotient method.

1 Introduction

Solving Sudoku puzzles using a Gröbner basis methodology has been studied by different authors (e.g., [1], [2], and [3]) using various methods and computer algebra systems. For a detailed discussion of multiple methods for solving Sudoku puzzles using Gröbner bases, we refer the reader to [1]. Among these, the functionality and performance of MuPAD (a computer algebra package of MATLAB) on computation in algebraic geometry is seldom explored. In this project, we explore MuPAD functions and efficiency in Gröbner bases computation by solving Sudoku puzzles. In our experiments, the quotient method turns

out to be the most efficient method. In this paper we improve its efficiency by performing the operations in a finite field of integers.

In Section 2 we give a brief introduction to Sudoku and its simplified version, Shidoku. Three methods of constructing sets of polynomials related to Sudoku are explained in detail. The methods are the sum and product method, the binary method, and the quotient method. The first two methods are described in [1], while the latter method is described in Chapter 3 of [3].

Section 3 begins with some relevant concepts in algebraic geometry. Then we prove that the quotient method over a finite field yields solutions to Sudoku boards. The section finishes with a brief discussion of Gröbner bases, without mathematical details.

The construction and performance of MuPAD programs using all three methods is discussed in Section 4. In our experiments, the quotient method turns out to be the most efficient method. We improve its efficiency by performing operations in a finite field.

2 Sudoku Boards

Sudoku boards are number placement puzzles. A Sudoku board of order n^2 is an $n^2 \times n^2$ array of cells with some of the cells containing symbols drawn from the symbol set $S = \{1, 2, ..., n\}$. Further, we require that there be a unique way to place symbols drawn from S in the remaining cells so that there is no repetition of symbols in any row, any column, or any of the canonical $n \times n$ sub-arrays (called blocks) that tile the Sudoku board. Meanwhile, a Sudoku board solution of order n^2 is the $n^2 \times n^2$ array of symbols one obtains by filling in all of the cells in a Sudoku board. All Sudoku puzzles seen in newspapers are Sudoku boards of order 9. The Sudoku puzzle frequently discussed in the paper is seen in Figure 1.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Figure 1: Example of a Sudoku board

Order-4 Sudoku boards with 2×2 blocks are called Shidoku boards. Below

is an example of the Shidoku board commonly discussed in this paper.

			3
3	2	4	
	4	3	2
2			

Figure 2: Example of a Shidoku board

This particular Shidoku board (and any other Shidoku or Sudoku board) can be solved through a process of elimination technique by looking at each individual row, column, and block. In the second row, three elements $3, 2, 4$ are placed in the first three boxes. The number 1 must be placed in the last box since otherwise the rules of Shidoku will be violated. Looking at the fourth column, we have the numbers 3 in the first box and 2 in the third box already given. Since we know that 1 needs to be in the second box in the fourth column due to the previous operation, by process of elimination, 4 needs to be in the fourth box in the fourth column. This process can be extended to find the remaining numbers on this Shidoku board. The solution of this Shidoku board is presented in Figure 3.

4	1	2	3
3	2	4	1
1	4	3	2
2	3	1	4

Figure 3: Solution of the Shidoku board presented in Figure 2

In the following we will formulate the Sudoku problem in terms of solving certain systems of polynomial equations. The variables of a polynomial equation represent the numbers in the cells of a Sudoku puzzle. With the preassigned numbers as given conditions, the solution set of the polynomial equations will provide the solution of the corresponding Sudoku puzzle. This transforms the process of trial and error and logical derivation into the process of solving systems of polynomial equations. With appropriate formulations, one can then solve a Sudoku puzzle by finding the zero locus of the associated polynomial representations.

Method 1: Sum and Product Method

The first method employed by [1] that we applied in MuPAD involves creating equations describing the total sum and product of each row, column, and block. To obtain a solution in a 4×4 Shidoku board, the numbers $1, 2, 3, 4$ are placed such that each number appears exactly once in each row, column, and block. There are 16 cells on a Shidoku board. We define a variable x_i for each cell, arranged left to right, top to bottom. Since they only assume values $1, 2, 3, 4$,

they satisfy the 16 equations

$$(x_i - 1)(x_i - 2)(x_i - 3)(x_i - 4) = 0, \quad i = 1, \ldots, 16.$$

In addition, for any four variables x_i, x_j, x_k, x_l that only assume values $1, 2, 3, 4$, it can be easily checked that the equations

$$x_i + x_j + x_k + x_l = 10,$$
$$x_i \cdot x_j \cdot x_k \cdot x_l = 24,$$

give a unique solution

$$x_i = 1, x_j = 2, x_k = 3, x_l = 4$$

up to order. These equations are assigned to each row, column, and block. For example, the equations for the first row are given by

$$x_1 + x_2 + x_3 + x_4 = 10,$$
$$x_1 \cdot x_2 \cdot x_3 \cdot x_4 = 24.$$

The equations for the first block are given by

$$x_1 + x_2 + x_5 + x_6 = 10,$$
$$x_1 \cdot x_2 \cdot x_5 \cdot x_6 = 24.$$

Since there are four rows, four columns, and four blocks, these equations yield $12{\cdot}2 = 24$ equations. In all, 40 equations are needed to describe a Shidoku board in order to satisfy the rules. In order to obtain a single solution (i.e., a single member of the zero locus of the corresponding set of polynomials) one should also include the linear polynomial equations $x_{i_1} - a_{i_1} = 0, ..., x_{i_t} - a_{i_t} = 0$, where $a_{i_1}, ..., a_{i_t}$ are the initial clues lying in cells $i_1, ..., i_t$. For example, in Figure 3, the linear equations representing the initial clues are $x_4 - 3 = 0, x_5 - 3 = 0, \ldots, x_{13} - 2 = 0$.

This process can be expanded to a Sudoku board. In a 9×9 Sudoku board solution, each of the nine numbers $1, 2, 3, 4, 5, 6, 7, 8, 9$ must appear exactly once in each row, column, and block. We define 81 variables for the 81 cells, arranged left to right, top to bottom. Since they only assume values $1, 2, 3, 4, 5, 6, 7, 8, 9$, the variables satisfy the 81 equations

$$(x_i - 1)(x_i - 2)\cdots(x_i - 9) = 0, \quad i = 1, \ldots, 81. \tag{1}$$

Unfortunately, for any nine variables of these, if we specify that their sum is 45 and product is 362880, there exists more than one solution. For example, both of the following two sets of numbers have a sum of 45 and a product of 362880:

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9\}, \quad \{1, 2, 4, 4, 4, 5, 7, 9, 9\}.$$

Since the values of the numbers are irrelevant, we instead choose the set $\{-2, -1, 1, 2, 3, 4, 5, 6, 7\}$. This set is the unique solution of equations speci-

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Figure 4: Solution of the Sudoku board presented in Figure 1

fying a sum of 25 and a product of 10080, regardless of the order of elements. Consequently, equation (1) is modified to reflect this change:

$$(x_i + 2)(x_i + 1)\cdots(x_i - 7) = 0, \quad i = 1, \ldots, 81.$$

The following two equations are assigned to each row, column, and block:

$$x_{i_1} + \cdots + x_{i_9} = 25 \qquad \text{and} \qquad x_{i_1}\cdots x_{i_9} = 10080.$$

For example, the equations for the first row are given by

$$x_1 + \cdots + x_9 = 25 \qquad \text{and} \qquad x_1\cdots x_9 = 10080.$$

The equations for the first block are given by

$$x_1 + x_2 + x_3 + x_{10} + x_{11} + x_{12} + x_{19} + x_{20} + x_{21} = 25,$$
$$x_1 \cdot x_2 \cdot x_3 \cdot x_{10} \cdot x_{11} \cdot x_{12} \cdot x_{19} \cdot x_{20} \cdot x_{21} = 10080.$$

For the 9 rows, 9 columns, and 9 blocks, these yield $9 \cdot 3 \cdot 2 = 54$ equations. In all, 135 equations are needed to describe a Sudoku board solution in order to satisfy the rules.

Method 2: Binary Method

The second method employed by [1] that we applied in MuPAD involves creating equations using binary variables assuming values of either 0 or 1. For the ith cell of a Shidoku puzzle, 4 variables $x_{i1}, x_{i2}, x_{i3}, x_{i4}$ are defined. They satisfy the equations

$$x_{ij}^2 - x_{ij} = 0, \quad j = 1, 2, 3, 4.$$

Note that this gives 64 variables and 64 equations. The indices $1, 2, 3, 4$ indicate the number placed in the cell. For example, if the first cell has the number 2, then

$$x_{11} = 0, x_{12} = 1, x_{13} = 0, x_{14} = 0.$$

Since these variables only assume values 0 and 1, by requiring

$$x_{i1} + x_{i2} + x_{i3} + x_{i4} = 1, \quad i = 1, \ldots, 16,$$

we make sure the value of the ith cell is well-defined, i.e., only one of the four variables assumes value 1, whose second index is placed in the cell. Now we consider the fact that each number may only appear once in each row, column, or block in a Shidoku board. This is achieved if every pair of numbers sharing a row, column, or block are different. The following equations describe the rule:

$$x_{i1}x_{j1} + x_{i2}x_{j2} + x_{i3}x_{j3} + x_{i4}x_{j4} = 0,$$

for $i, j = 1, \ldots, 16$, where i, j run through cells sharing a row, column, or block. Notice that only one of $x_{i1}, x_{i2}, x_{i3}, x_{i4}$ assumes value 1, and only one of $x_{j1}, x_{j2}, x_{j3}, x_{j4}$ assumes value 1. The above equations guarantee that the two cells take different values. In each row or column, there are $\binom{4}{2} = 6$ pairs. In each block, we need only to consider the diagonal pairs, since other pairs have been considered in rows and columns. There are 4 blocks, each of which contains 2 diagonal pairs. Hence $6 \cdot 4 + 6 \cdot 4 + 2 \cdot 4 = 56$ equations are needed for this rule. All in all, there are 136 polynomial equations. Recall that the linear polynomials representing the initial clues should be included to guarantee a unique solution.

This process can be expanded to a Sudoku board solution. However, since there are 9 variables for each cell and there are 81 cells in a Sudoku board, we need 729 variables. To describe the conditions of a Sudoku board solution, over 800 polynomials are needed! In our experiment, computing a Gröbner basis of this method did not produce a Sudoku board solution in MuPAD as MuPAD "ran out of memory".

Method 3: Quotient Method

In this section, we adapt the set of polynomial equations from [3] for a Sudoku puzzle to the finite field $\mathbb{Z}_{11}$. We choose the field $\mathbb{Z}_{11}$ since it is the smallest finite field containing 9 distinct values. Define a variable x_i for cell i. Define the polynomials $F_i \in \mathbb{Z}_{11}[x_i]$ by

$$F_i(x_i) = (x_i - 1)\cdots(x_i - 9) = x_i^9 + a_8x_i^8 + \cdots + a_0$$

for some $a_\ell \in \mathbb{Z}_{11}$, $\ell = 0, \ldots, 8$. Consequently, for any $i \neq j$,

$$\begin{aligned} F_i(x_i) - F_j(x_j) &= (x_i^9 + a_8x_i^8 + \cdots + a_0) - (x_j^9 + a_8x_j^8 + \cdots + a_0) \\ &= (x_i^9 - x_j^9) + a_8(x_i^8 - x_j^8) + a_7(x_i^7 - x_j^7) + \cdots + a_1(x_i - x_j). \end{aligned}$$

Note that if ℓ is odd,

$$x_i^\ell - x_j^\ell = (x_i - x_j)(x_i^{\ell-1} + x_i^{\ell-2}x_j + \cdots + x_i x_j^{\ell-2} + x_j^{\ell-1}).$$

On the other hand, if ℓ is even,

$$x_i^\ell - x_j^\ell = (x_i^{\ell/2} + x_j^{\ell/2})(x_i^{\ell/2} - x_j^{\ell/2}),$$

in which the factor $(x_i^{\ell/2} - x_j^{\ell/2})$ can be further factorized in the same pattern. It follows that the polynomial $F_i(x_i) - F_j(x_j)$ contains a factor $x_i - x_j$. Hence the functions

$$G_{ij}(x_i, x_j) = \frac{F_i(x_i) - F_j(x_j)}{x_i - x_j} \in \mathbb{Z}_{11}(x_i, x_j)$$

are polynomials. Set

$E = \{(i,j) \mid 1 \le i < j \le 81$ such that i, j are in the same row, column, or block$\}$.

Theorem 1. *Each member of the zero locus of the set of polynomial equations* $\{G_{ij} = 0 \mid (i,j) \in E\}$ *is the solution of a Sudoku board.*

We will prove this theorem after going over some of the concepts involved in Section 3.

3 Proof of Theorem 1

In this section, we prove Theorem 1. Let k be a field and $k[x_1, \ldots, x_n]$ be the ring of polynomials with variables $x_1, x_2, ..., x_n$ and coefficients in k. We will use the following facts from [4].

Definition 1. A subset $I \subseteq k[x_1, \ldots, x_n]$ is an ideal if it satisfies:

(i) $0 \in I$.

(ii) If $f, g \in I$, then $f + g \in I$.

(iii) If $f \in I$ and $h \in k[x_1, ..., x_n]$, then $hf \in I$.

Lemma 2. *If* $f_1, \ldots, f_s \in k[x_1, \ldots, x_n]$, *then*

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i \mid h_1, \ldots, h_s \in k[x_1, \ldots, x_n] \right\}$$

is an ideal of $k[x_1, \ldots, x_n]$.

We will call $\langle f_1, \ldots, f_s \rangle$ the ideal generated by $f_1, \ldots, f_s$. Meanwhile, $f_1, \ldots, f_s$ are the generators of this ideal, and a set of generators is a basis of the ideal.

Definition 3. Let $f_1, \ldots, f_s$ be polynomials in $k[x_1, \ldots, x_n]$. Then we set

$$V(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in k^n \mid f_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \le i \le s\}.$$

We call $V(f_1, \ldots, f_s)$ the affine variety defined by $f_1, \ldots, f_s$.

Lemma 4. *If $f_1, \ldots, f_s$ and $g_1, \ldots, g_t$ are generators of the same ideal in $k[x_1, \ldots, x_n]$, then we have $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$.*

Definition 5. An affine variety $V(I)$ where $I = \langle f_1, \ldots, f_s \rangle$ is defined by

$$V(I) = V(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in k^n \mid f_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq s\}.$$

Using the terminology of ideals we can restate Theorem 1 as follows:

Theorem 1. *Let F_i, G_{ij}, and E be defined as in Section 2. Let $I = \langle G_{ij} \mid (i,j) \in E \rangle$ be the ideal generated by polynomials G_{ij}, such that i, j are in the same row, column, or block. Let $\vec{a} = (a_1, ..., a_{81})$. Then $\vec{a} \in V(I)$ if and only if $a_i = 1, ..., 9$ and $a_i \neq a_j$ whenever i, j are in the same row, block, and column.*

Proof of Theorem 1. The direction "$\Leftarrow$" is shown in [3]. We show the other direction. Let $\vec{a} \in V(I)$. Then $G_{ij}(a_i, a_j) = 0$ for all i, j such that i, j are in the same row, block, or column. Let $(i,j) \in E$. We show the following claims:

1. If $a_i, a_j \in \{1, ..., 9\}$, then $a_i \neq a_j$;

2. $a_i \neq 0$, for all i;

3. $a_i \neq 10$, for all i.

Proof of claim 1: Suppose $(i,j) \in E$ and $a_i = a_j = b \in \{1, ..., 9\}$. Consider the equation

$$F_i(x_i) = F_j(x_j) + (x_i - x_j)G_{ij}(x_i, x_j).$$

Substituting $x_j = b$ into $F_i(x_i)$, we obtain

$$F_i(x_i) = 0 + (x_i - b)G_{ij}(x_i, b) = (x_i - b)G_{ij}(x_i, b).$$

Note that $G_{ij}(b, b) = 0$. This implies that $F_i(x_i)$ has a repeated root at b, a contradiction.

Proof of claim 2: Suppose $(i,j) \in E$ and $a_j = 0$. Similar as above, we have

$$F_i(x_i) = F_j(0) + x_i G_{ij}(x_i, 0).$$

This implies

$$(x_i - 1)\cdots(x_i - 9) - F_j(0) = x_i G_{ij}(x_i, 0)$$

and therefore,

$$(x_i - 1)\cdots(x_i - 9) + 1 = x_i G_{ij}(x_i, 0).$$

Note that $1, \ldots, 10$ are not roots of the left-hand side. So 0 is the only solution, a contradiction to claim 1.

Proof of claim 3: Suppose $(i,j) \in E$ and $a_j = -1$. We then have

$$F_i(x_i) = F_j(-1) + (x_i + 1)G_{ij}(x_j, -1).$$

This implies

$$(x_i - 1)\cdots(x_i - 9) - (-2)\cdots(-10) = (x_i + 1)G_{ij}(x_j, -1)$$

and therefore

$$(x_i - 1)\cdots(x_i - 9) - 1\cdots(-10) = (x_i + 1)G_{ij}(x_j, -1).$$

Again the only root of the left hand side is -1, a contradiction to claim 1.

This proves that $\vec{a} \in V(I)$ if and only if $a_i = 1, \ldots, 9$ and $a_i \neq a_j$ whenever i, j are in the same row, column, or block. □

4 Experiments and MuPAD Computations for Shidoku and Sudoku Board Solutions

Finding a zero locus, and thereby finding a solution to a Sudoku puzzle, would be a lot easier if the original set of polynomials could be traded in for a simpler set of polynomials that has the same zero locus. The Gröbner basis technique provides a method of finding this simpler set of polynomials. It is computationally complex, but computers, using software like MuPAD, can carry out the calculations.

A Gröbner basis G of an ideal I is a basis such that, under an appropriate ordering of the terms of polynomials, every possible leading term of the polynomials in I is divisible by some leading term of the polynomials in G. This leads to the fact that, if the polynomial equations are set up properly for a Shidoku or Sudoku puzzle, the Gröbner basis of the ideal representing the puzzle should consist of polynomials of the form $x_i - a_i$, where a_i is the symbol that lies in the cell i. Note that the row echelon form of a linear system of equations is a special case of a Gröbner basis, where the terms are naturally ordered as $x_1 > x_2 > \cdots > x_n$. Readers are referred to [4] to learn more about the details of Gröbner bases.

For a Shidoku board, the Gröbner basis of each of the polynomial systems constructed by the three methods consists of polynomials of the form $x_i - a$, which form the solution of the Shidoku board.

Figure 5 shows the Shidoku puzzle that was solved in Section 2. We use this example to illustrate our implementation of the three methods in MuPAD. The MuPAD code below implements the sum and product method.

			3
3	2	4	
	4	3	2
2			

Figure 5: Shidoku Puzzle in Figure 1

```
GenerateSudoku := proc( )
begin
    G := [];
    x := x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16;
    // defining the first 16 basic equations which assigns values 1,2,3,4 to each
    of the 16 variables;
    for k from 1 to 16 do
        g := poly((x[k]-1)*(x[k]-2)*(x[k]-3)*(x[k]-4),[x]);
        G := append(G,g);
    end_for:
    // defining two equations (sum and product) for each row; (4 rows)
    for k from 1 to 4 do
        gsum := -10;
        gprod := 1;
        for j from 1 to 4 do
            gsum := gsum+x[4*(k-1)+j];
            gprod := gprod*x[4*(k-1)+j];
        end_for:
        gs := poly(gsum,[x]);
        gp := poly(gprod-24,[x]);
        G := append(G,gs,gp);
    end_for:
    // defining two equations (sum and product) for each column; (4 cols)
    for k from 1 to 4 do
        gsum := -10;
        gprod := 1;
        for j from 1 to 4 do
            gsum := gsum+x[k+4*(j-1)];
            gprod := gprod*x[k+4*(j-1)];
        end_for:
        gs := poly(gsum,[x]);
        gp := poly(gprod-24,[x]);
        G := append(G,gs,gp);
    end_for:
    // defining two equations (sum and product) for each 2x2 block; (4 blocks)
    for i from 1 to 2 do
        for k from 1 to 2 do
            gsum := -10;
            gprod := 1;
            for j from 1 to 2 do
                for l from 1 to 2 do
                    gsum := gsum+x[8*(i-1)+2*(k-1)+4*(j-1)+l];
                    gprod := gprod*x[8*(i-1)+2*(k-1)+4*(j-1)+l];
                end_for:
            end_for:
            gs := poly(gsum,[x]);
            gp := poly(gprod-24,[x]);
            G := append(G,gs,gp);
        end_for:
    end_for:
    G := append(G,poly(x4-3,[x]),poly(x5-3,[x]),poly(x6-2,[x]),poly(x7-4,[x]),
    poly(x10-4,[x]),poly(x11-3,[x]),poly(x12-2,[x]),poly(x13-2,[x]));
end_proc:
```

Notice that the last two lines before the end of the procedure indicate the numbers that are already placed on the board. In less than a second, MuPAD produces the solution of the board, where $x_j - k$ indicates that symbol k lies in cell j.

```
G:=GenerateSudoku()

groebner::gbasis(%,LexOrder)

[poly(x1 - 4, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x2 - 1, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x3 - 2, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x4 - 3, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x5 - 3, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x6 - 2, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x7 - 4, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x8 - 1, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x9 - 1, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x10 - 4, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x11 - 3, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x12 - 2, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x13 - 2, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x14 - 3, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x15 - 1, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16]),
poly(x16 - 4, [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16])
```

The following code implements the binary method.

```
GenerateSudoku := proc( )
begin
    G := [];
    x := x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,
    x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,x33,x34,x35,x36,x37,x38,x39,
    x40,x41,x42,x43,x44,x45,x46,x47,x48,x49,x50,x51,x52,x53,x54,x55,x56,x57,x58,
    x59,x60,x61,x62,x63,x64;
    // defining the first 64 basic equations which assigns values 0,1 to each
    of the 64 variables;
    for k from 1 to 64 do
        g := poly((x[k]*(x[k]-1),[x]));
        G := append(G,g);
    end_for:
    // defining two equations (sum and product) for each row; (4 rows)
    for k from 1 to 16 do
        gsum := -1;
        gprod := 0;
        for j from 1 to 4 do
            gsum := gsum+x[4*(k-1)+j];
        end_for:
        gs := poly(gsum,[x]);
        gp := poly(gprod,[x]);
        G := append(G,gs)
    end_for:
for i from 1 to 4 do
            for j from 1 to 3 do
            gs := x[16*(i-1)+4*(j-1)+1]*x[16*(i-1)+4*(j-1)+5]+ x[16*(i-1)
            +4*(j-1)+2]*x[16*(i-1)+4*(j-1)+6]+ x[16*(i-1)+4*(j-1)+3]*x[16*(i-1)
            +4*(j-1)+7]+ x[16*(i-1)+4*(j-1)+4]*x[16*(i-1)+4*(j-1)+8];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
for i from 1 to 4 do
            for j from 1 to 2 do
            gs := x[16*(i-1)+4*(j-1)+1]*x[16*(i-1)+4*(j-1)+9]+ x[16*(i-1)
            +4*(j-1)+2]*x[16*(i-1)+4*(j-1)+10]+ x[16*(i-1)+4*(j-1)+3]*x[16*(i-1)
```

```
            +4*(j-1)+11]+ x[16*(i-1)+4*(j-1)+4]*x[16*(i-1)+4*(j-1)+12];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
for i from 1 to 4 do
            for j from 1 to 1 do
            gs := x[16*(i-1)+4*(j-1)+1]*x[16*(i-1)+4*(j-1)+13]+ x[16*(i-1)
            +4*(j-1)+2]*x[16*(i-1)+4*(j-1)+14]+ x[16*(i-1)+4*(j-1)+3]*x[16*(i-1)
            +4*(j-1)+15]+ x[16*(i-1)+4*(j-1)+4]*x[16*(i-1)+4*(j-1)+16];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
//Columns below
for i from 1 to 4 do
            for j from 1 to 3 do
            gs := x[16*(j-1)+4*(i-1)+1]*x[16*(j-1)+4*(i-1)+17]+ x[16*(j-1)
            +4*(i-1)+2]*x[16*(j-1)+4*(i-1)+18]+ x[16*(j-1)+4*(i-1)+3]*x[16*(j-1)
            +4*(i-1)+19]+ x[16*(j-1)+4*(i-1)+4]*x[16*(j-1)+4*(i-1)+20];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
for i from 1 to 4 do
            for j from 1 to 2 do
            gs := x[16*(j-1)+4*(i-1)+1]*x[16*(j-1)+4*(i-1)+33]+ x[16*(j-1)
            +4*(i-1)+2]*x[16*(j-1)+4*(i-1)+34]+ x[16*(j-1)+4*(i-1)+3]*x[16*(j-1)
            +4*(i-1)+35]+ x[16*(j-1)+4*(i-1)+4]*x[16*(j-1)+4*(i-1)+36];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
for i from 1 to 4 do
            for j from 1 to 1 do
            gs := x[16*(j-1)+4*(i-1)+1]*x[16*(j-1)+4*(i-1)+49]+ x[16*(j-1)
            +4*(i-1)+2]*x[16*(j-1)+4*(i-1)+50]+ x[16*(j-1)+4*(i-1)+3]*x[16*(j-1)
            +4*(i-1)+51]+ x[16*(j-1)+4*(i-1)+4]*x[16*(j-1)+4*(i-1)+52];
            gl := poly(gs,[x]);
            G := append(G,gl);
            end_for:
end_for:
//Diagonals below
for i from 1 to 2 do
            for j from 1 to 2 do
            gs := x[8*(j-1)+32*(i-1)+1]*x[8*(j-1)+32*(i-1)+21]+ x[8*(j-1)
            +32*(i-1)+2]*x[8*(j-1)+32*(i-1)+22]+ x[8*(j-1)+32*(i-1)+3]*x[8*(j-1)
            +32*(i-1)+23]+ x[8*(j-1)+32*(i-1)+4]*x[8*(j-1)+32*(i-1)+24];
            gs2:= x[8*(j-1)+32*(i-1)+5]*x[8*(j-1)+32*(i-1)+17]+ x[8*(j-1)
            +32*(i-1)+6]*x[8*(j-1)+32*(i-1)+18]+ x[8*(j-1)+32*(i-1)+7]*x[8*(j-1)
            +32*(i-1)+19]+ x[8*(j-1)+32*(i-1)+8]*x[8*(j-1)+32*(i-1)+20];
            gl := poly(gs,[x]);
            gl2:= poly(gs2, [x]);
            G := append(G,gl,gl2);
            end_for:
end_for:
G := append(G,poly(x15-1,[x]),poly(x19-1,[x]),poly(x22-1,[x]),poly(x28-1,[x]),
poly(x40-1,[x]),poly(x43-1,[x]),poly(x46-1,[x]),poly(x50-1,[x]));
end_proc:
```

By default, MuPAD computes a Gröbner basis in the rational number field $\mathbb{Q}$. To reduce the complexity of operations and the number of possible values, we can perform the operations in the finite field $\mathbb{Z}_5$ for a Shidoku puzzle. MuPAD produced the result much more efficiently. The following code implements the quotient method in the finite field $\mathbb{Z}_5$.

```
Sudoku4by4inZ5 := proc( )
begin
    // Set up polynomials in Z5 field;
    G := [];
    x := x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,
    x21,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32,x33,x34,x35,x36,x37,x38,x39,
    x40,x41,x42,x43,x44,x45,x46,x47,x48,x49,x50,x51,x52,x53,x54,x55,x56,x57,x58,
    x59,x60,x61,x62,x63,x64,x65,x66,x67,x68,x69,x70,x71,x72,x73,x74,x75,x76,x77,
    x78,x79,x80,x81;
    // Since the underlying field is Z5, the variables assume value 0 to 4;

    // Defining two equations (sum and product) for each exclusive set;

    // Numbers in each row and column are different; (2x4+2x4=16 equations)
    for i from 1 to 4 do
        // 2 equations in each row;
        gsum := x[4*(i-1)+1]+x[4*(i-1)+2]+x[4*(i-1)+3]+x[4*(i-1)+4];
        gpsum := x[4*(i-1)+1]*x[4*(i-1)+2]+x[4*(i-1)+1]*x[4*(i-1)+3]
        +x[4*(i-1)+1]*x[4*(i-1)+4]+x[4*(i-1)+2]*x[4*(i-1)+3]+x[4*(i-1)+2]
        *x[4*(i-1)+4]+x[4*(i-1)+3]*x[4*(i-1)+4];
        gs := poly(gsum,[x],Dom::IntegerMod(5));
        gps := poly(gpsum,[x],Dom::IntegerMod(5));
        G := append(G,gs,gps);
        // 2 equations in each column;
        gsum := x[(i-1)+1]+x[(i-1)+5]+x[(i-1)+9]+x[(i-1)+13];
        gpsum := x[(i-1)+1]*x[(i-1)+5]+x[(i-1)+1]*x[(i-1)+9]+x[(i-1)+1]
        *x[(i-1)+13]+x[(i-1)+5]*x[(i-1)+9]+x[(i-1)+5]*x[(i-1)+13]+x[(i-1)+9]
        *x[(i-1)+13];
        gs := poly(gsum,[x],Dom::IntegerMod(5));
        gps := poly(gpsum,[x],Dom::IntegerMod(5));
        G := append(G,gs,gps);
    end_for:
    // Numbers in each block are different; (32 equations)
    for i from 1 to 2 do
        for k from 1 to 2 do
            gsum := x[2*(i-1)+8*(k-1)+1]+x[2*(i-1)+8*(k-1)+2]+x[2*(i-1)
            +8*(k-1)+5]+x[2*(i-1)+8*(k-1)+6];
            gpsum := x[2*(i-1)+8*(k-1)+1]*x[2*(i-1)+8*(k-1)+2]+x[2*(i-1)
            +8*(k-1)+1]*x[2*(i-1)+8*(k-1)+5]+x[2*(i-1)+8*(k-1)+1]*x[2*(i-1)
            +8*(k-1)+6]+x[2*(i-1)+8*(k-1)+2]*x[2*(i-1)+8*(k-1)+5]+x[2*(i-1)
            +8*(k-1)+2]*x[2*(i-1)+8*(k-1)+6]+x[2*(i-1)+8*(k-1)+5]*x[2*(i-1)
            +8*(k-1)+6];
            gs := poly(gsum,[x],Dom::IntegerMod(5));
            gps := poly(gpsum,[x],Dom::IntegerMod(5));
            G := append(G,gs,gps);
        end_for:
    end_for:
G := append(G,poly(x4+3,[x],Dom::IntegerMod(5)),poly(x5+3,[x],Dom::IntegerMod(5)),
poly(x6+2,[x],Dom::IntegerMod(5)),poly(x7+4,[x],Dom::IntegerMod(5)),poly(x10+4,
[x],Dom::IntegerMod(5)),poly(x11+3,[x],Dom::IntegerMod(5)),poly(x12+2,[x],
Dom::IntegerMod(5)),poly(x13+2,[x],Dom::IntegerMod(5)));
end_proc:
```

In our experiments, MuPAD produced solutions for Shidoku puzzles with ease for any of the above methods. However, with the sum and product method and binary method, it was not able to produce a solution for the simplest level of Sudoku puzzle [3]. Using the quotient method in the finite field $\mathbb{Z}_{11}$, MuPAD was able to produce a solution in seventeen seconds. The following shows the result of the Gröbner basis computation of the Sudoku puzzle in Figure 4 using the quotient method:

```
[poly(x1+2, x1,x2,...,x81,IntMod(11)), poly(x2+5, x1,x2,...,x81,IntMod(11)),
poly(x3-3,x1,x2,...,x81,IntMod(11)), poly(x4-1,x1,x2,...,x81,IntMod(11)),
poly(x5+4,x1,x2,...,x81,IntMod(11)), poly(x6-4,x1,x2,...,x81,IntMod(11)),
poly(x7-2,x1,x2,...,x81,IntMod(11)), poly(x8-5,x1,x2,...,x81,IntMod(11)),
poly(x9+3,x1,x2,...,x81,IntMod(11)), poly(x10-1,x1,x2,...,x81,IntMod(11)),
poly(x11+4,x1,x2,...,x81,IntMod(11)), poly(x12+3,x1,x2,...,x81,IntMod(11)),
poly(x13-3,x1,x2,...,x81,IntMod(11)), poly(x14-2,x1,x2,...,x81,IntMod(11)),
poly(x15-5,x1,x2,...,x81,IntMod(11)), poly(x16+5,x1,x2,...,x81,IntMod(11)),
poly(x17-4,x1,x2,...,x81,IntMod(11)), poly(x18+2,x1,x2,...,x81,IntMod(11)),
poly(x19-2,x1,x2,...,x81,IntMod(11)), poly(x20-5,x1,x2,...,x81,IntMod(11)),
poly(x21-4,x1,x2,...,x81,IntMod(11)), poly(x22+5,x1,x2,...,x81,IntMod(11)),
poly(x23+3,x1,x2,...,x81,IntMod(11)), poly(x24+2,x1,x2,...,x81,IntMod(11)),
poly(x25+4,x1,x2,...,x81,IntMod(11)), poly(x26-3,x1,x2,...,x81,IntMod(11)),
poly(x27-1,x1,x2,...,x81,IntMod(11)), poly(x28+3,x1,x2,...,x81,IntMod(11)),
poly(x29-2,x1,x2,...,x81,IntMod(11)), poly(x30-1,x1,x2,...,x81,IntMod(11)),
poly(x31-4,x1,x2,...,x81,IntMod(11)), poly(x32-3,x1,x2,...,x81,IntMod(11)),
poly(x33+4,x1,x2,...,x81,IntMod(11)), poly(x34-5,x1,x2,...,x81,IntMod(11)),
poly(x35+2,x1,x2,...,x81,IntMod(11)), poly(x36+5,x1,x2,...,x81,IntMod(11)),
poly(x37-4,x1,x2,...,x81,IntMod(11)), poly(x38+2,x1,x2,...,x81,IntMod(11)),
poly(x39+5,x1,x2,...,x81,IntMod(11)), poly(x40+3,x1,x2,...,x81,IntMod(11)),
poly(x41-5,x1,x2,...,x81,IntMod(11)), poly(x42-2,x1,x2,...,x81,IntMod(11)),
poly(x43-3,x1,x2,...,x81,IntMod(11)), poly(x44-1,x1,x2,...,x81,IntMod(11)),
poly(x45+4,x1,x2,...,x81,IntMod(11)), poly(x46+4,x1,x2,...,x81,IntMod(11)),
poly(x47-3,x1,x2,...,x81,IntMod(11)), poly(x48-5,x1,x2,...,x81,IntMod(11)),
poly(x49+2,x1,x2,...,x81,IntMod(11)), poly(x50+5,x1,x2,...,x81,IntMod(11)),
poly(x51-1,x1,x2,...,x81,IntMod(11)), poly(x52+3,x1,x2,...,x81,IntMod(11)),
poly(x53-2,x1,x2,...,x81,IntMod(11)), poly(x54-4,x1,x2,...,x81,IntMod(11)),
poly(x55-5,x1,x2,...,x81,IntMod(11)), poly(x56+3,x1,x2,...,x81,IntMod(11)),
poly(x57+2,x1,x2,...,x81,IntMod(11)), poly(x58+4,x1,x2,...,x81,IntMod(11)),
poly(x59-1,x1,x2,...,x81,IntMod(11)), poly(x60-3,x1,x2,...,x81,IntMod(11)),
poly(x61-4,x1,x2,...,x81,IntMod(11)), poly(x62+5,x1,x2,...,x81,IntMod(11)),
poly(x63-2,x1,x2,...,x81,IntMod(11)), poly(x64-3,x1,x2,...,x81,IntMod(11)),
poly(x65-1,x1,x2,...,x81,IntMod(11)), poly(x66+4,x1,x2,...,x81,IntMod(11)),
poly(x67-2,x1,x2,...,x81,IntMod(11)), poly(x68-4,x1,x2,...,x81,IntMod(11)),
poly(x69+5,x1,x2,...,x81,IntMod(11)), poly(x70+2,x1,x2,...,x81,IntMod(11)),
poly(x71+3,x1,x2,...,x81,IntMod(11)), poly(x72-5,x1,x2,...,x81,IntMod(11)),
poly(x73+5,x1,x2,...,x81,IntMod(11)), poly(x74-4,x1,x2,...,x81,IntMod(11)),
poly(x75-2,x1,x2,...,x81,IntMod(11)), poly(x76-5,x1,x2,...,x81,IntMod(11)),
poly(x77+2,x1,x2,...,x81,IntMod(11)), poly(x78+3,x1,x2,...,x81,IntMod(11)),
poly(x79-1,x1,x2,...,x81,IntMod(11)), poly(x80+4,x1,x2,...,x81,IntMod(11)),
poly(x81-3,x1,x2,...,x81,IntMod(11))]
```

Figure 6 gives a visual representation of the code generated above for the quotient method. Remember that in the finite integer field $\mathbb{Z}_{11}$ we have $x_j + k \equiv x_j - (11-k) \pmod{11}$. For example, $x_{80}+4 \equiv x_{80}-7 \pmod{11}$. This congruence yields each number on the Sudoku board! Try to match up all of the equations with each entry on the completed solution to see for yourself.

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Figure 6: Solution of the Sudoku board presented in Figure 1

5 Conclusion

In this project we explored MuPAD's performance on a Gröbner basis computation by solving Shidoku and Sudoku puzzles. With all three methods we investigated, MuPAD produced unique solutions for a well-defined Shidoku puzzle in less than 1 second. For a Sudoku puzzle, even with an easy puzzle, MuPAD spent 17 seconds to produce the solution, with much worse efficiency than that of Singular [3]. However, there is much more to explore in this problem. The paper [5] presents an efficient method of computing binary polynomials. Incorporating this method to our binary method for Sudoku puzzles could be a future direction. The efficient algorithms F_4 [6] and F_5 [7] for computing a Gröbner basis have not been implemented in MuPAD. Considering MATLAB's powerful matrix computations the implementation of F_4 and its variations could be profitable.

6 Acknowledgements

We thank Dr. Ying Li, Associate Professor of Mathematics at Saint Francis University, for all of her support in helping us with writing this paper, for giving us countless amount of insight on the material in abstract algebra and algebraic geometry, and for allowing both of us to become involved with this project. We also thank Dr. Brendon LaBuz, Associate Professor of Mathematics at Saint Francis University, for his valuable reviews of the manuscipt.

Bibliography

[1] E. Arnold, S. Lucas, L. Taalman, *Gröbner basis representations of Sudoku*, College Math. J. 41(2)(2010), pp. 101-112.

[2] J. Gago-Vargas, I. Hartillo-Hermoso, J. Martín-Morales, and J. M. Ucha-Enríquez, *Sudokus and Gröbner bases: not only a divertimento*, in Computer Algebra in Scientific Computing, Lecture Notes in Comput. Sci., 4194, Springer, Berlin, 2006, pp. 155-165.

[3] W. Decker, G. Pfister, *A First Course in Computational Algebraic Geometry*, Cambridge, United Kingdom, 2013.

[4] D. A. Cox, J. B. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, New York, 2008.

[5] Y. Sato, A. Nagai, and S. Inoue, *On the computation of elimination ideals of boolean polynomial rings*, in Computer Mathematics, Lecture Notes in Computer Science, 5081, Springer, Berlin, 2008, pp. 334-348.

[6] J. C. Faugère, *A new efficient algorithm for computing Gröbner bases (F_4)*, J. Pure Appl. Algebr., 139(1-3) (1999), pp. 61-88.

[7] J. C. Faugère, *A new efficient algorithm for computing Gröbner bases without reduction to zero F_5*. in International Symposium on Symbolic and Algebraic Computation Symposium, ISSAC (2002), France, pp. 75–82.