Counting Pandiagonal Magic Squares

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirement for the Degree
Bachelor of Arts

Joshua L. Levenberg

May 1994

Approved for the Division
(Mathematics)

Tom Roby

Acknowledgements

I would especially like to thank my family and friends for being so helpful and supportive during the writing of this thesis. Thanks also Joe Buhler, the Reed College T-Watchers and the Computer User Services staff who were an invaluable aid throughout the year. Of course, this would not be the thesis it is if it were not for the constant guidance of my advisor Tom Roby.

This document was typeset using Plain TEX, version 3.0, with the eplain format, version 2.3: 1 March 1993. The three-dimensional illustrations were rendered using the Persistence of Vision Raytracer (available free via FTP at `alfred.ccs.carleton.ca` in `/pub/pov-ray/POV-Ray2.2`) on Paul Coyne's Dell 486DX-33 computer. The labeling and other figures were done with Aldus' SuperPaint 3.0. The programs developed as a part of this thesis were developed under Borland C++ 3.0 by Borland International and the GNU project C++ Compiler (v2.4), g++. The programs were executed on Romulus and Brahma. I would like to thank the gurus in D-Lab for their maintenance of these two computers. The text processing necessary to manipulate the output of the programs was performed with QEdit Advanced, the "quick editor," version 2.1 by SemWare of Applied Systems Technologies, Inc. The TEX and raytracer source files were written using QEdit and GNU project Emacs, by the Free Software Foundation, Inc.

Table of Contents

Abstract

The number of pandiagonal magic squares of a particular size is considered using the methods of linear homogeneous diophantine equations as described in Richard Stanley's *Enumerative Combinatorics, Volume 1.* Pandiagonal magic squares are square matrices of nonnegative integers where the sum of every row, column, and broken diagonal is the same constant. The broken diagonals start at any element of the top row of the square, and continue downwards diagonally, wrapping around the right-left borders as necessary. The completely fundamental elements are the quasi-generators for the set of magic squares; every magic square is expressible as a not necessarily unique nonnegative linear combination of these elements. A method for computing these is given. Results include polynomial functions which give the number of magic squares of row sum k as a function of k (the size of the square being fixed). So far the method has only provided results for squares of size 7 and smaller.

Notation

$\mathbf{Z}$	the set of integers, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbf{P}$	the set of positive integers, $\{1, 2, 3, \ldots\}$
$\mathbf{N}$	the set of natural or nonnegative integers, $\{0, 1, 2, \ldots\}$
$\mathbf{Q}$	the set or rational numbers, $\{p/q\}$ with p and q integers
$\mathbf{R}$	the set of real numbers
$\mathbf{C}$	the set of complex numbers, $\{a + bi \mid a, b \in \mathbf{R}\}$
$\boldsymbol{\alpha}$	a vector
α_i	the i^{th} component of the vector $\boldsymbol{\alpha}$
$\boldsymbol{\rho}$	a magic square
ρ_{ij}	the element of $\boldsymbol{\rho}$ in the i^{th} row and j^{th} column
$\mathbf{\Phi}$	matrix representing a system of linear homogeneous diophantine equations
r	the number of (linearly independent) rows of $\mathbf{\Phi}$, the number of dependent variables in $\mathbf{\Phi}$
m	the number of columns of $\mathbf{\Phi}$, the number of unknowns
$\dim S$	the dimension of a subspace S
n	the dimension of the solution subspace, $n = m - r$; also used to represent the size of a magic square, i.e., $n \times n$
E	the monoid of nonnegative solutions to $\mathbf{\Phi}$
$\mathcal{C}$	the pointed convex polyhedral cone corresponding to E
$CF(E)$	the set of completely fundamental elements of E
$\boldsymbol{\beta}$	a completely fundamental element of E
$\boldsymbol{\beta}_i$	the completely fundamental element of E numbered i
β_i	the i^{th} component of the completely fundamental element $\boldsymbol{\beta}$
$\operatorname{supp} \boldsymbol{\alpha}$	the support of a vector $\boldsymbol{\alpha}$, the set of indices of the nonzero components of the vector
Γ	a triangulation of $\mathcal{C}$
σ, τ	simplicial cones, usually members of Γ
$\mu(\sigma)$	the Möbius function of the lattice of elements in Γ
$\mathbf{x}$	an m-dimensional vector of purely formal variables

D_σ	The set of vectors of the simplicial monoid E_σ with no integer multiples of the quasi-generators
ω	a vector in D_σ

Introduction

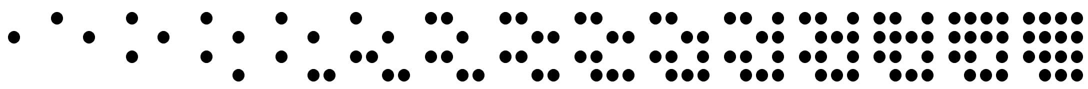
Magic squares have intrigued people for centuries. W. S. Andrews notes in [1] that they were used in the ninth century by Arabian astrologers, and have been found in Chinese literature written about A.D. 1125.[1]

Pandiagonal, or continuous, magic squares are a more specific type of magic square than the standard simple magic square. They are of interest because the conditions on them are very symmetric. Pandiagonal magic squares are considered among the more "magic" of types of magic squares.

Mathematicians normally do not study magic squares with any practical use in mind; however in modern times they have been used in computer graphics to perform an *ordered dither*.[2] In an ordered dither, a square matrix called the *dither matrix* filled with the first n positive integers, such as

2	7	14	11
13	12	1	8
3	6	15	10
16	9	4	5

is used to simulate grayscale on a bilevel (monochrome) device. To display an intensity i in the range $\{0, \ldots, n\}$ on a monochrome monitor in a 4×4 area, turn on those pixels on the display which correspond to numbers less than or equal to i in the dither matrix. The above matrix simulates the intensities 1 through 15 with the following patterns:



These patterns would then be used repeatedly to convey an image with these intermediate intensities. While there are many criteria used for different output

[1] See page 19 of *Chinese Philosophy* by Paul Carus.

[2] Any good computer graphics book should cover this. For example, Foley, et al. *Computer Graphics–Principles and Practice.* Massachusetts: Addison-Wesley, 1987.

mediums to determine the dither matrix, using a pandiagonal magic square as the dither matrix insures that intensity is evenly divided among the rows, columns, and diagonals of a rendering of a picture.

Much of the study of magic squares manifests itself as systems of rules for making magic squares. These rules are not sufficient to reliably generate all magic squares of a particular size, or provide theorems which will give the number of magic squares as a function of row sum. The theory of linear homogeneous diophantine equations is employed here to provide the mathematical structure necessary to address these questions.

The presentation of the techniques here is ostensibly for the actual counting of pandiagonal magic squares, but the path is littered with beautiful mathematics. An important goal of this thesis is to make this material accessible to a junior in mathematics. Much of this presentation follows Stanley [5], though the organization is different.

Chapter 1.

Magic Squares as Systems of Equations

1.1. What are Magic Squares?

Magic squares are square matrices of natural numbers with the property that each individual row and column sums to the same constant k, called the *magic constant* or *magic number*. In general, if an $n \times n$ matrix has this property we will call it a *semi-magic square*. The more traditional *simple magic square* is a semi-magic square with the additional constraint that the two main diagonals must also sum to k. The *order* of a magic square is the number of rows (or columns) of its matrix. A magic square of order n is said to be *normal* if its matrix elements are precisely the first n^2 natural numbers. For example, the only normal simple 3 by 3 magic squares have magic number 15 and are:

$$\boxed{\begin{matrix}2&7&6\\9&5&1\\4&3&8\end{matrix}} \quad \boxed{\begin{matrix}2&9&4\\7&5&3\\6&1&8\end{matrix}} \quad \boxed{\begin{matrix}4&3&8\\9&5&1\\2&7&6\end{matrix}} \quad \boxed{\begin{matrix}4&9&2\\3&5&7\\8&1&6\end{matrix}}$$

$$\boxed{\begin{matrix}6&1&8\\7&5&3\\2&9&4\end{matrix}} \quad \boxed{\begin{matrix}6&7&2\\1&5&9\\8&3&4\end{matrix}} \quad \boxed{\begin{matrix}8&1&6\\3&5&7\\4&9&2\end{matrix}} \quad \boxed{\begin{matrix}8&3&4\\1&5&9\\6&7&2\end{matrix}}$$

Of primary interest in this thesis are *pandiagonal magic squares*. In these matrices, the *broken diagonals* sum to the magic number of the square. The broken diagonals start at any element of the top row of the square, and continue downwards diagonally (either $\searrow$ or $\swarrow$), wrapping around the right-left borders as necessary. The elements $\{a_{ij}\}$ on the broken diagonals going down and to the right each satisfy $j - i \equiv c \pmod p$ for some integer constant c.[3] Similarly, the elements $\{a_{ij}\}$ on the broken diagonals going down and to the left each satisfy $i + j \equiv c \pmod p$ for some integer constant c. The broken diagonals for a 5×5 pandiagonal magic square are shown in figure 1.

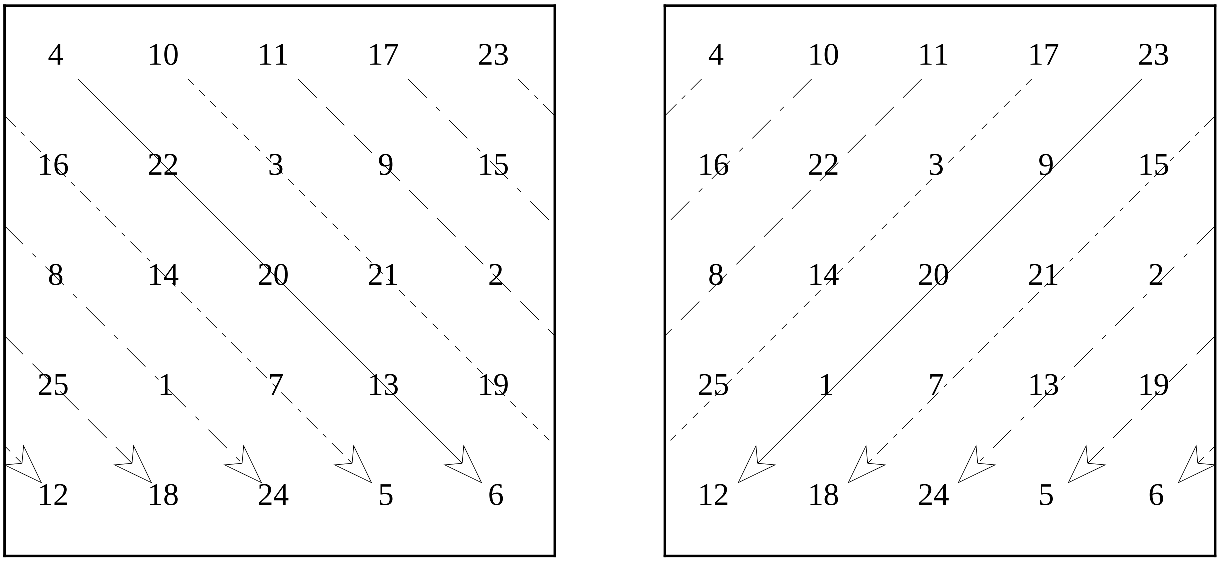


Figure 1. *Broken Diagonals*

Definitions of *symmetric magic squares* vary. When examining normal magic squares, it is more natural to define symmetric to be when every pair a_{ij}, a_{kl} with $i + k = j + l = n + 1$ positioned on opposite sides of the center of the magic square has the same sum (this type of symmetric magic square is also called an *associated* or *regular* magic square). In fact, that sum has to be twice the average value of the elements of the magic square, $\frac{2k}{n}$. These arise from several constructions of particular normal magic squares. All the 3×3 magic squares listed above are

[3] If this notation is unfamiliar, see the definition on page 20.

symmetric. On the other hand, for non-normal magic squares, the usual definition (used in [5]) is that for all $0 \leq i, j < n$, $\rho_{i,j} = \rho_{j,i}$ holds. This may be rephrased, $\boldsymbol{\rho} = \boldsymbol{\rho}^t$.

To actually work with magic squares, a significant amount of mathematical machinery will be needed. Notice that there are several linear relationships between the elements of a pandiagonal magic square. In particular, the difference between the sum of the elements in two different rows, columns, or diagonals is always zero. There is a large class of problems which have similar conditions. In those problems, the interest is in finding the nonnegative solutions for a system of several variables between which there exist linear relationships. In order to get a better grip on magic squares, we will now examine the theory of linear homogeneous diophantine equations. Afterwards, we will be able to say something about the number of pandiagonal magic squares of particular sizes.

1.2. Linear Homogeneous Diophantine Equations

The general theory of linear homogeneous diophantine equations attempts to describe the nonnegative integer solutions to a system of homogeneous linear equations. This problem is equivalent to finding column vectors $\boldsymbol{\alpha}$ in $\mathbf{N}^m$, where m is the number of unknowns, which satisfy the matrix equation

$$\boldsymbol{\Phi}\alpha = \mathbf{0},$$

where $\mathbf{0}$ is the r-dimensional zero vector, r being the number of equations, and $\boldsymbol{\Phi}$ is an $r \times m$ matrix of integers. Without the restriction to nonnegative solutions, the problem simplifies and yields to well-known techniques of finitely generated abelian groups. In the case at hand, however, the solution set does not form a group as it does in the $\mathbf{Z}^n$ case. The nonnegativity condition eliminates the possibility of additive inverse elements existing in the solution, leaving a commutative *monoid* which we shall call E. Monoids are *semigroups with identity.* Since E is a semigroup, given any two elements $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ of E, $\boldsymbol{\alpha} + \boldsymbol{\rho} \in E$, where addition is standard addition of vectors inherited from $\mathbf{Z}^n$. The additive identity, the zero vector, will always be a solution to $\mathbf{\Phi}\boldsymbol{\alpha} = \mathbf{0}$, though in general if E contains $\boldsymbol{\alpha}$ then it will not contain $-\boldsymbol{\alpha}$ (with the exception of $\boldsymbol{\alpha} = \mathbf{0}$). We must be careful when performing subtraction in order to insure that the result has nonnegative components.

In general, we cannot expect E to be a *free commutative monoid.* In a free commutative monoid, there exist a set of elements which act as basis vectors in the sense that every element may be uniquely decomposed into those basis vectors. For example, consider a system with six equations ($r = 6$), nine unknowns ($m = 9$), and equation coefficients defined by

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -3 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 2 & -3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -3 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & -3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & -3 \end{bmatrix}.$$

This is a row reduced version of a matrix that might occur in practice. The following four solutions (which are actually column vectors but it is more convenient to represent them as row vectors)

$$\boldsymbol{\beta}_1 = [\,0 \quad 5 \quad 0 \quad 5 \quad 3 \quad 3 \quad 2 \quad 3 \quad 3\,]$$

$$\boldsymbol{\beta}_2 = [\,5 \quad 0 \quad 5 \quad 0 \quad 2 \quad 2 \quad 3 \quad 2 \quad 2\,]$$

$$\boldsymbol{\beta}_3 = [\,0 \quad 5 \quad 2 \quad 2 \quad 5 \quad 0 \quad 2 \quad 3 \quad 2\,]$$

$$\boldsymbol{\beta}_4 = [\,5 \quad 0 \quad 3 \quad 3 \quad 0 \quad 5 \quad 3 \quad 2 \quad 3\,]$$

cannot be written as the sum of other nonzero vectors in E, but $\boldsymbol{\beta}_1 + \boldsymbol{\beta}_2 = \boldsymbol{\beta}_3 + \boldsymbol{\beta}_4$.

How can we show $\boldsymbol{\beta}_1$ cannot be decomposed into two other elements in E? How may we find vectors like $\boldsymbol{\beta}_1$? What exactly is the structure of E? Consider a generic element $\boldsymbol{\alpha}$ of E. The first six elements of $\boldsymbol{\alpha}$ are dependent upon the last three, so consider just α_7, α_8, and α_9. These may be chosen freely as long as the nonnegativity condition is satisfied. This condition gives us nine inequalities to satisfy:

$$\begin{aligned}
\alpha_1 &= 3\alpha_7 - 2\alpha_8 \geq 0 \\
\alpha_2 &= -2\alpha_7 + 3\alpha_8 \geq 0 \\
\alpha_3 &= 3\alpha_7 - 2\alpha_9 \geq 0 \\
\alpha_4 &= -2\alpha_7 + 3\alpha_9 \geq 0 \\
\alpha_5 &= 3\alpha_8 - 2\alpha_9 \geq 0 \\
\alpha_6 &= -2\alpha_8 + 3\alpha_9 \geq 0 \\
\alpha_7 &\geq 0 \\
\alpha_8 &\geq 0 \\
\alpha_9 &\geq 0
\end{aligned}$$

The last three are redundant. We may rederive $\alpha_7 \geq 0$ by noting $5\alpha_7 = 3\alpha_1 + 2\alpha_2 \geq 0$, and similarly for α_8 and α_9. This yields the form of the solution set, the intersection of six *linear half-spaces* and the *integer lattice*. A linear half-space is simply a hyperplane, a $(n-1)$-dimensional linear subspace of $\mathbf{R}^n$, and all points on one side of the hyperplane; they may be characterized as $\{\mathbf{v} \in \mathbf{R}^m \mid \mathbf{v} \cdot \mathbf{w} \geq 0\}$ for some fixed $\mathbf{w} \in \mathbf{R}^m$. The integer lattice is the set of all points with integer coordinates. So consider the graph of the six inequalities with respect to α_7, α_8, and α_9 which is shown in figure 2. (The set has been truncated for illustration purposes.) Strictly speaking, the solution set is a subset of $\mathbf{Z}^9$, but the linear dependence between the variables indicates that the solution set lives in a three-dimensional subspace of $\mathbf{Z}^9$. If we were to label the axes in figure 2 with the vectors $[\,3 \quad -2 \quad 3 \quad -2 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0\,]$, $[\,-2 \quad 3 \quad 0 \quad 0 \quad 3 \quad -2 \quad 0 \quad 1 \quad 0\,]$, and $[\,0 \quad 0 \quad -2 \quad 3 \quad -2 \quad 3 \quad 0 \quad 0 \quad 1\,]$, then the diagram is accurate, and the distinction ceases to be a concern.

In general, this graph will be a *convex polyhedral cone*, that is the intersection of finitely many linear half-spaces. For every lattice point on this graph, there is a corresponding nonnegative solution to $\mathbf{\Phi}$ and vice-versa. It is now easy to create elements that are not the sum of other essentially different elements. Define a *facet* to be the (nontrivial) intersection of a hyperplane with the cone, such that the cone

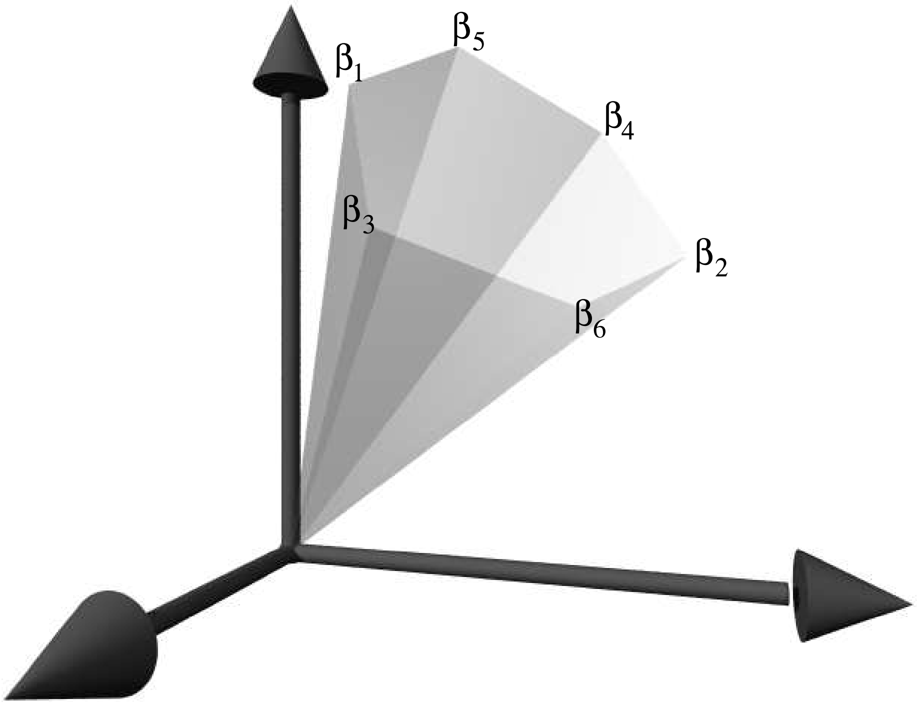


Figure 2. *Graph of Solutions*

lies completely in a half-space created by the hyperplane. Define the *dimension* of a set to be the dimension of the subspace spanned by that set. An *i-face* is an i-dimensional intersection of facets (which are each $(n-1)$-dimensional). The entire cone and the set $\{\mathbf{0}\}$ are called *improper* faces. A 1-face is called an *extreme ray*. The vectors β_1 through β_4 correspond to extreme rays of the solution set. Since the cone is convex, only vectors along the ray could possibly be part of a decomposition and those listed were the smallest nonzero vectors in the ray. In case you are curious, the smallest nonzero vectors in the other two rays are:

$$\beta_5 = [\,2 \quad 2 \quad 0 \quad 5 \quad 0 \quad 5 \quad 2 \quad 2 \quad 3\,]$$

$$\beta_6 = [\,3 \quad 3 \quad 5 \quad 0 \quad 5 \quad 0 \quad 3 \quad 3 \quad 2\,].$$

A monoid is said to be *positive* if it contains a vector with only strictly positive components. If a monoid is not positive, then there is an i such that the i^{th} element is always zero. We may freely ignore this element since it never varies, so we may consider only positive monoids without loss of generality. From the analysis above, it is clear that our monoids correspond to convex polyhedral cones. The cone is formed by considering the nonnegative *real* solutions to the same (in)equalities. We are only concerned with the lattice points, and the distinction between monoids and cones from now on will be blurred. Cones are said to be *pointed* if for every nonzero $\mathbf{v}$ that is an element of the cone, $-\mathbf{v}$ is not an element of the cone. It is clear that our cones will be pointed since our monoids only contain nonnegative vectors.

1.3. Supports and Lattices

In the example, there were six extreme rays and six facets. Each of the facets was spanned (and bordered) by two extreme rays. By including the improper faces, the empty set ($\emptyset$) and E, we may form a *partially ordered set* (actually a lattice, a specific type of partially ordered set) out of the faces (see figure 3). A partially ordered set, or poset, is simply a set where some of the elements have an order relation. Here we say a face σ is less than τ just if $\tau \subset \sigma$. This lattice may be constructed for the faces of any monoid. It is hard to visualize in higher dimensions, but the pattern continues as expected. Pairs of 1-faces (extreme rays) will span the 2-faces, pairs of 2-faces will span the 3-faces, up to the pairs of $(n-2)$-faces spanning the facets.

There is an easy way of characterizing the elements of the monoid via this lattice structure. Define the *support* of a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, denoted $\operatorname{supp} \boldsymbol{\alpha}$, to be $\{i \mid \alpha_i \neq 0\}$. The support is a subset of the integers from 1 to m, and contains i precisely when the i^{th} component of $\boldsymbol{\alpha}$ is non-zero. The vectors in E on an extreme ray are scalar multiples so they all have the same support. The vectors in a 2-face are a linear combination of two extreme rays, so the support of the vectors in the 2-face is contained in the union of the supports of the spanning extreme rays. If in fact the extreme ray has *minimal support*, i.e., no vector in E has a support that is a proper subset of the support of the extreme ray then the

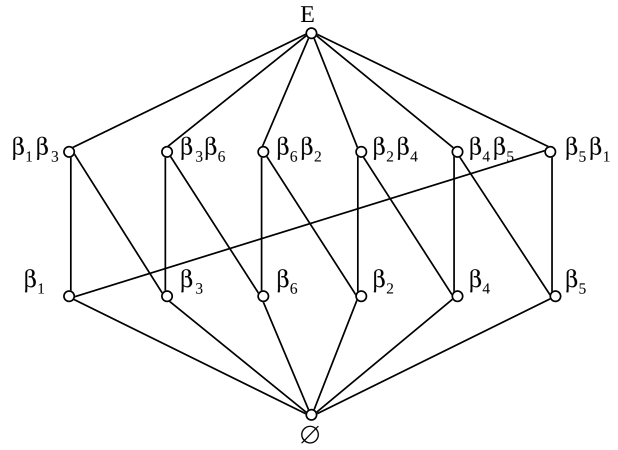


Figure 3. *Lattice of Faces of Convex Polyhedral Cone*

containment may be extended into equality. We will show later that the extreme rays will always have minimal support. Examining the possible supports for all the elements of E, and drawing in the containment hierarchy, we again get a lattice. As can be seen in figure 4, the structure is the same as the lattice of faces. In fact, the support function is in general an isomorphism from the lattice of faces of the cone onto the lattice of supports of the vectors in the monoid.

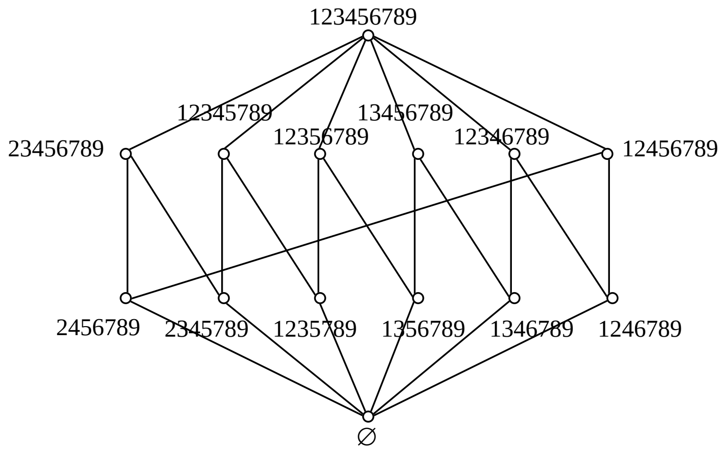


Figure 4. *Lattice of Supports of the Vectors in a Monoid*

Chapter 2.

Completely Fundamental Elements

We call a $\boldsymbol{\beta} \in E$ *completely fundamental* if for all $n \in \mathbf{N}$ and $\boldsymbol{\alpha}, \boldsymbol{\alpha}' \in E$ for which $n\beta = \boldsymbol{\alpha} + \boldsymbol{\alpha}'$, we have $\boldsymbol{\alpha} = i\beta$ and $\boldsymbol{\alpha}' = (n-i)\beta$ for some $i \in \mathbf{N}$, $0 \leq i \leq n$. The set of completely fundamental elements of E is denoted $CF(E)$. No completely fundamental element is the sum of two nonzero nonparallel elements. Given a ray of elements of E with a completely fundamental element, the completely fundamental element will be the smallest in the sense that it has the smallest norm. Indeed, every vector in E on a ray with a completely fundamental element will be a positive integer multiple of it.

In fact, $\boldsymbol{\beta}_1$ through $\boldsymbol{\beta}_6$ above are the completely fundamental elements for the above system. It is not a coincidence that the extreme rays correspond to completely fundamental elements, as can be seen in the following theorem. A slightly stronger version appears in [5], Proposition 4.6.10.

Theorem. *Let E be a monoid and $\mathcal{C}$ be the corresponding pointed convex polyhedral cone. The following sets are identical:*

i. $CF(E)$,

ii. $\{\beta \in E \mid \boldsymbol{\beta}$ lies on an extreme ray of $\mathcal{C}$, and $\beta \neq n\beta'$ for some $n > 1,\ \beta' \in E\}$,

iii. the non-zero elements β of E of minimal support that are not of the form $n\boldsymbol{\beta}'$ for some $n > 1$, $\boldsymbol{\beta}' \in E$.

Proof. The proof breaks down into a few components.

(i) $\Longrightarrow$ (iii) If the support of some $\boldsymbol{\beta} \in CF(E)$ is not minimal then there is another vector $\boldsymbol{\alpha}$ with a smaller support which we may subtract off. For n sufficiently large, the result of $n\boldsymbol{\beta} - \boldsymbol{\alpha}$ will be positive and therefore in E. Since $\operatorname{supp} \boldsymbol{\alpha} \neq \operatorname{supp} \boldsymbol{\beta}$, $\boldsymbol{\alpha}$ could never be a multiple of β, contradicting the definition of completely fundamental element. So every completely fundamental element must have minimal support.

(iii) $\Longrightarrow$ (i) Assume there existed a $\boldsymbol{\beta}$ which has minimal support and otherwise satisfies the conditions of (iii), but was not a completely fundamental element of E. So there exists some $n \in \mathbf{N}$ and nonzero $\boldsymbol{\alpha}, \boldsymbol{\alpha}' \in E$ such that $n\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\alpha}'$. The support of either side must be the $\operatorname{supp} \beta$ which we know to be minimal, so $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ must have the same support. Now if α were not a multiple of β then we could find some positive integer variables q and p such that $q\beta - p\alpha$ would still be positive and therefore in E but would have a smaller support. We would just have to find the component of $\boldsymbol{\alpha}$ which was smallest in comparison to the corresponding component of $\boldsymbol{\beta}$ and then set p and q to cancel this term. The resulting vector would have support a proper subset of the support of $\boldsymbol{\beta}$ violating our assumption, so the conditions of (iii) are sufficient to guarantee an element is completely fundamental.

(i) $\Longleftrightarrow$ (ii) Given an element $\boldsymbol{\beta}$ of an extreme ray satisfying (ii) above, it is clear that any $\boldsymbol{\alpha}, \alpha' \in E$ such that $n\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\alpha}'$ for some $n \in \mathbf{N}$, must also lie on the same extreme ray. Otherwise, one of $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ would have to be in the interior of the half space created by some facet containing β. In order for the sum $\boldsymbol{\alpha} + \boldsymbol{\alpha}'$ to be on the extreme ray, the other would have to be on the exterior of the same half space, putting it outside E. All that remains is to make sure $\boldsymbol{\beta}$ is cannot be written as the sum of two smaller vectors on the same extreme ray, and insuring that it is not a multiple of any other element in the monoid is sufficient. It is also clear that any completely fundamental element $\boldsymbol{\beta}$ must lie on an extreme ray, since otherwise some multiple of it could be decomposed into vectors in E lying on either side of β. Finally, since $\boldsymbol{\beta}$ cannot be a scalar multiple of other elements of E, the equality of sets is established. ∎

2.1. Generating the set CF(E)

The above proposition allows us to actually calculate the completely fundamental elements of the nonnegative solutions to an arbitrary system $\mathbf{\Phi}$. Recall, if $\mathbf{\Phi}$ has r equations and m unknowns, then the monoid E lives in an $(n = m - r)$-dimensional subspace. Each of the m unknowns must be nonnegative, so the monoid must satisfy m inequalities. Each of the m inequalities defines a linear half-space. The linear half-space is split by the hyperplane defined by when the inequalities are replaced by equalities. In our example above, $3\alpha_7 - 2\alpha_8 \geq 0$ was an inequality that defined a linear half-space. We think of this as defining three regions: the exterior which is all points satisfying $3\alpha_7 - 2\alpha_8 < 0$, the interior which has all those satisfying $3\alpha_7 - 2\alpha_8 > 0$, and the border (which is included in the linear half-space) which has the remaining points which must satisfy $3\alpha_7 - 2\alpha_8 = 0$. Every facet of the convex polyhedral cone $\mathcal{C}$ of the monoid will satisfy one of these hyperplane equations. Furthermore, every extreme ray will be in the intersection of two neighboring 2-faces, and the intersection of three 3-faces, and in the intersection of $n - 1$ facets. Remember, as a one-dimensional subspace of a n-dimensional space, it must satisfy $n - 1$ equations.

To clarify this, let us derive the extreme rays of our example system. There were $r = 6$ equations, $m = 9$ unknowns, so the solutions space has $n = 3$ dimensions. Looking back we find the nine inequalities for the facets, in terms of α_7

through α_9:

$$
\begin{aligned}
\alpha_1 &= 3\alpha_7 - 2\alpha_8 \geq 0 \\
\alpha_2 &= -2\alpha_7 + 3\alpha_8 \geq 0 \\
\alpha_3 &= 3\alpha_7 - 2\alpha_9 \geq 0 \\
\alpha_4 &= -2\alpha_7 + 3\alpha_9 \geq 0 \\
\alpha_5 &= 3\alpha_8 - 2\alpha_9 \geq 0 \\
\alpha_6 &= -2\alpha_8 + 3\alpha_9 \geq 0 \\
&\qquad\qquad \alpha_7 \geq 0 \\
&\qquad\qquad \alpha_8 \geq 0 \\
&\qquad\qquad \alpha_9 \geq 0
\end{aligned}
$$

Now, every $\beta \in CF(E)$ satisfies two of these equations with strict equalities. For instance, $\beta_1 = [\,0 \quad 5 \quad 0 \quad 5 \quad 3 \quad 3 \quad 2 \quad 3 \quad 3\,]$ satisfies the first and third equation precisely. Its elements have no common factor, so there is no smaller element on that extreme ray, so it is completely fundamental. Conversely, forcing the first and third equations to be strict equalities yields

$$
\begin{aligned}
3\alpha_7 - 2\alpha_8 &= 0 \\
3\alpha_7 - 2\alpha_9 &= 0
\end{aligned}
$$

which may only be satisfied by vectors on the ray from the origin containing β_1.

Now, to generate $CF(E)$ for this set, take all possible ways of choosing two $(n-1)$ facets, and examine the intersection. If the two facets are parallel the result is a plane, or in general the hyperplane equations of the facets are not independent, disregard that combination. Otherwise the result is a one-dimensional subspace. If there is a ray with nonnegative components in that subspace, then there is a smallest one with no common factor among its components. That is a completely fundamental element. This process will give all the elements of $CF(E)$ since every completely fundamental element satisfies these conditions for some set of $n-1$ facets. The algorithm is quite laborious, involving repeated matrix reduction, but, it is simple to implement on a computer. A 'C++' program that uses this algorithm to exhaustively compute the set $CF(E)$ (tested with Borland C++ 3.0 and GNU's g++ v2.4) is included in Appendix B.

2.2. Results

To execute this program, all that is necessary is a description of a matrix representing a linear homogeneous system of equations. I generated these with a small 'C' program which, given the size and type of magic square, would generate the matrix in a suitable format for the first program. The description consists of the equations which equate the sums of every row, column, and (in the case of pandiagonal magic squares) the broken diagonals to the sum of the first row. Since every equation is just the sum and difference of the various elements of the magic square, the system is linear. Furthermore the system is homogeneous, since every equation is of the form

$$\text{(terms from the first row)} - \text{(terms from a row, column, or diagonal)} = 0\,.$$

Finally, all the elements of magic squares must be nonnegative according to our definition, so magic squares exactly fit the criteria for applying the theory of linear homogeneous diophantine equations, and of course the program for finding the completely fundamental elements.

Without further ado, here are the results of running the program on several input data sets. To compare with existing information, I started with 3×3 simple magic squares (which have equal main diagonal sums but have unrestricted broken diagonals) which are more traditional, though less symmetric than pandiagonal magic squares. The completely fundamental 3×3 simple magic squares, all of which have row sum 3, are:

$$\boxed{\begin{array}{ccc} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{array}} \qquad \boxed{\begin{array}{ccc} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{array}} \qquad \boxed{\begin{array}{ccc} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{array}} \qquad \boxed{\begin{array}{ccc} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 0 \end{array}}$$

By examining these completely fundamental elements, we may derive properties possessed by the entire set of magic squares. We know these magic squares, which we may view as vectors in $\mathbf{Z}^9$, correspond to the quasi-generators of the simplicial monoids of a triangulation Γ of the monoid E of magic squares. Furthermore, every vector in the monoid is in some simplicial monoid in Γ. So every magic square is a nonnegative linear combination of (at most three, the dimension of the largest simplicial monoids) of the completely fundamental magic squares. Taking one of the magic squares listed at the beginning we see it decomposes like so:

$$\boxed{\begin{array}{ccc} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{array}} = \frac{1}{2}\boxed{\begin{array}{ccc} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{array}} + \frac{3}{2}\boxed{\begin{array}{ccc} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{array}} + 3\boxed{\begin{array}{ccc} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 0 \end{array}}.$$

Notice that the center value in each completely fundamental square is 1, and therefore one third the row sum (or magic number) of the magic square. Since multiplying and adding these magic squares preserves this property, the center value for every $\boldsymbol{\rho} \in E$ is one third the row sum (this explains the 5 that was in the center of each of the normal magic squares listed earlier). We know the center value is an integer, so the row sum for every 3×3 simple magic square is a multiple of 3.

Moving onto pandiagonal magic squares, the only completely fundamental magic squares with size less than four are matrices of all ones. This means that, for instance, the only 3×3 pandiagonal magic squares are of the form

$$\boxed{\begin{array}{ccc} k & k & k \\ k & k & k \\ k & k & k \end{array}}$$

for some nonnegative integer k. The components of a 3×3 matrix must satisfy eight equations to be a pandiagonal magic square. These reduce down to $a_{11} = a_{12} = \cdots = a_{33}$, giving the above result. The program quickly computed the completely fundamental 4×4 pandiagonal magic squares. They all have row sum 2, and are listed here:

$$\boxed{\begin{matrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{matrix}}$$

$$\boxed{\begin{matrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{matrix}} \quad \boxed{\begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{matrix}}$$

Calculating the completely fundamental 5×5 pandiagonal magic squares took about two hours. They all have row sum 1:

$$\boxed{\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}}$$

$$\boxed{\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{matrix}}$$

$$\boxed{\begin{matrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}} \quad \boxed{\begin{matrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{matrix}}$$

All of these have the same basic structure. There is a 1 in the top row, and then every row after that has the 1 a fixed amount to the right of the 1 in the previous row with wrap around (a more formal definition is given below). Here the offset, called the *cyclic offset* is either 2 or 3. I will call any element of $CF(E)$ that follows this pattern *cyclic*. Furthermore, I will call any $n \times n$ matrix (with nonnegative elements) that follows this pattern *cyclic* as long as the offset is greater than 1 and smaller than $n - 1$.

Proposition. *The $p \times p$ cyclic elements are magic squares for p a prime. Furthermore, they are members of $CF(E)$.*

Proof. For a given size, cyclic squares are determined by two parameters, the position k of the 1 in the first row, and the cyclic offset l, the amount each 1 is offset from the 1 in the previous row. Fix k and l to some generic values with $1 \leq k \leq p$ and $2 \leq l \leq p-2$. Let a_{ij} be the element of this cyclic square in the i^{th} row and j^{th} column. Clearly $a_{1k} = 1$ by the definition of k, and $a_{1i} = 0$ for all $i \neq k$.

In order to deal with the cases when the offset wraps around, define $a \equiv b \pmod{p}$ (read "a is equivalent to b modulo p") for a and b integers to mean $b-a$ is an integer multiple of p (sometimes written $p \mid b-a$). Notice that $a \equiv a \pmod{p}$ for all integers a (which is clear because $a - a = 0 = 0p$). If $a \equiv b \pmod{p}$ then $a-b = kp$ for some integer k, so $b-a = (-k)p$ and $b \equiv a \pmod{p}$. Also, if $a \equiv b \pmod{p}$, $b \equiv c \pmod{p}$, then $a-b = k_1 p$ and $b-c = k_2 p$ for some integers k_1 and k_2, so

$$a - c = (a-b) + (b-c) = k_1 p + k_2 p = (k_1 + k_2)p$$

and $a \equiv c \pmod{p}$. These three conditions (the relation is reflexive, symmetric, and transitive respectively) mean that equivalence modulus p is actually an equivalence relation, and the suggestive naming is justified. This allows us to write the value of a_{ij} for all $1 \leq i, j \leq p$:

$$a_{ij} = \begin{cases} 1, & \text{if } j \equiv k + (i-1)l \pmod{p} \\ 0, & \text{otherwise.} \end{cases}$$

This agrees with the earlier expression for $i = 1$. Furthermore, if $a_{ij} = 1$ and $j + l \leq p$ then $a_{i+1,j+l} = 1$ as we expect. Lastly, if $a_{ij} = 1$ and $j + l > p$ then $a_{i+1,j+l-p} = 1$ which accounts for wraparound.

We will need a result from number theory, that there exists a unique x in $\{1, \ldots, p\}$ satisfying $ax \equiv b \pmod{p}$ for a not a multiple of p. First of all, notice that any integer is equivalent to some number in $\{1, \ldots, p\}$. In particular, there is some value equivalent to b in that range. Now, assume there are two values x_1 and

x_2 in $\{1, \ldots, p\}$ such that $ax_1 \equiv ax_2 \pmod{p}$. By the definition of equivalence, this means $ax_2 - ax_1 = a(x_2 - x_1)$ is divisible by p. But p is prime and a is not divisible by p so p divides $x_2 - x_1$. Since the maximum difference between x_1 and x_2 is $p - 1$, $x_2 - x_1$ must be zero and $x_1 = x_2$. So the values $a(1)$ through $a(p)$ are all distinct under equivalence, so they must include values equivalent to each of the numbers in $\{1, \ldots, p\}$. We conclude that there must exist a value x such that $ax \equiv b \pmod{p}$.

To check if this matrix is a pandiagonal magic square, look at the row, column, and diagonal sums. Only one 1 is placed in any row, so the row sums are all 1. Column j has a 1 in the i^{th} row precisely when $il \equiv j + l - k \pmod{p}$. The above result from number theory tells us that for p prime, there will exist a unique i between 1 and p satisfying that equation. This means all the columns have row sum 1. On any given diagonal going down and to the right, i and j satisfy $j - i \equiv c \pmod{p}$ for some integer constant c. Substituting to find when there is a 1 in this diagonal gives $i + c \equiv k + (i-1)l \pmod{p}$ which by the same result, happens exactly once each diagonal. The other diagonals each also have a 1 by a similar token (or by symmetry if you prefer). So these cyclic squares are pandiagonal magic. Furthermore, since they have row sum 1, they have minimal support and are not a multiple of any other elements, so they are completely fundamental magic squares. ■

Define a *right cyclic rotation* of a magic square to be the taking the rightmost column and moving it to the left of the square. Similarly define *down cyclic rotation* to be taking the bottommost row of a magic square to the top. Define *left cyclic rotations* and *up cyclic rotations* to be the inverse operations of right and down cyclic rotations respectively. Two magic squares are *cyclic rotations* of each other if you may get one from a series of left, right, up, and down cyclic rotations. Note that there are only two distinct types of completely fundamental pandiagonal 5×5 magic squares after cyclic rotations, characterized by the cyclic offset.

De la Hire (see [1] and [6]) used *latin squares* to construct normal magic squares. An $n \times n$ latin square is a magic square with row sum $\frac{n(n-1)}{2}$ such that each of the first n natural numbers appears n times. So

3	2	0	1	4
1	4	3	2	0
2	0	1	4	3
4	3	2	0	1
0	1	4	3	2

is a 5×5 latin square. If we have two latin squares U and V such that for every k and l in $\{0, \ldots, n-1\}$ there exists a position in the magic square where U has a k and V has an l, then we may combine them $nU + V$ and get a magic square using the numbers $\{0, \ldots, n^2 - 1\}$. Adding one to each element gives a normal magic square. Given a cyclic completely fundamental magic square $\boldsymbol{\rho}$, we may construct a *cyclic latin square*, U, by taking the sum of $\boldsymbol{\rho}$, twice a cyclic rotation of $\boldsymbol{\rho}$, three times a different cyclic rotation of $\boldsymbol{\rho}$, up to a $n-1$ times a cyclic rotation of $\boldsymbol{\rho}$ different from all the previous cyclic rotations. Here two cyclic rotations are different if the results after applying them to $\boldsymbol{\rho}$ are different. If another cyclic latin square, V, is generated from another cyclic completely fundamental element with a different cyclic offset, then U and V satisfy the conditions to generate a normal magic square. Any magic square that may be constructed this way is called a *cyclic magic square*. The magic square in Figure 1 was generated in precisely this fashion. In order to be able to determine this, you would have to subtract one from every element, and then examine the elements as numbers in base 5 (the order of the magic square). If the numbers in the ones and fives places are arranged cyclically, then the magic square is cyclic. In practice though, all of the completely fundamental 5×5 magic squares are cyclic, so we expect all the 5×5 magic squares to be inherently cyclic too.

In order to apply its algorithm, the program in Appendix B must reduce the system of equations into a simplified form. It needs to express the first r variables as linear functions of the last $m - r$ variables. This is equivalent to row reducing the matrix $\mathbf{\Phi}$ until the leftmost $r \times r$ section is a diagonal matrix. So it is finding the matrix, equivalent to $\mathbf{\Phi}$ that is in the following form:

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 & a_{1,r+1} & a_{1,r+2} & \cdots & a_{1,m} \\ 0 & a_{22} & \ddots & \vdots & a_{2,r+1} & a_{2,r+2} & \cdots & a_{2,m} \\ \vdots & \ddots & \ddots & 0 & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{rr} & a_{r,r+1} & a_{r,r+2} & \cdots & a_{r,m} \end{bmatrix}.$$

Without swapping columns, reducing the 6×6 case to this form is impossible. However, it is always possible to relabel the variables (and perform the corresponding rearrangement of the columns) so that the matrix may be put in this form. In practice, the program would reduce the matrix to an upper triangular form until it got to a problem column. When the matrix is in this form, it is easy to determine which column swaps, and therefore what variable relabeling, are necessary by inspection.

After overcoming that hurdle, I ran the program for 6×6 pandiagonal magic squares. I was surprised to discover the program found no completely fundamental elements. I checked the literature (see [2]) and found that indeed, there are no nontrivial pandiagonal magic squares of size $4k + 2$ for integer k. The trivial pandiagonal magic squares have the same number in every position in the square.

The exponential increase in processing time needed to process larger systems prevented the program from generating the completely fundamental elements for 7×7 pandiagonal magic squares. There are 49 unknowns and 24 independent equations, so the program must analyze $\binom{49}{49-24-1}$ subsystems, a number in excess of 63.2 trillion, as opposed to $\binom{36}{16}$ for 6×6 pandiagonal magic squares, which is about 7.3 billion. We of course know that at least the cyclic matrices are in $CF(E)$, but there may be others.

2.3. Symmetric Magic Squares

Defining *symmetric magic squares* to be those magic squares $\boldsymbol{\rho}$ such that $\boldsymbol{\rho} = \boldsymbol{\rho}^t$ we may easily determine the completely fundamental elements of the symmetric subset of a given type of magic square.

Theorem. *Let E_p be the monoid of $n \times n$ pandiagonal magic squares, and E_s be the submonoid of symmetric pandiagonal magic squares. Then $CF(E_s)$ is contained in the set of matrices*

$$\{\, \beta \mid \beta \in CF(E_p),\ \beta = \beta^t \,\} \cup \{\, \beta + \beta^t \mid \beta \in CF(E_p),\ \beta \neq \beta^t \,\} \ .$$

Proof. Take some $\boldsymbol{\alpha} \in E_s$. Since $E_s \subset E_p$, $\boldsymbol{\alpha}$ must be in E_p, so there exists a $\boldsymbol{\beta} \in CF(E_p)$ such that the support of $\boldsymbol{\alpha}$ contains the support of $\boldsymbol{\beta}$. Thus for some $k \in \mathbf{P}$, $\boldsymbol{\rho} = k\boldsymbol{\alpha} - \boldsymbol{\beta}$ will have nonnegative components, and $\boldsymbol{\rho} \in E_p$. Since $\boldsymbol{\alpha}$ is symmetric, $2k\boldsymbol{\alpha} = k(\boldsymbol{\alpha} + \boldsymbol{\alpha}^t) = \boldsymbol{\beta} + \boldsymbol{\beta}^t + \boldsymbol{\rho} + \boldsymbol{\rho}^t$. Hence, $\operatorname{supp}(\boldsymbol{\beta} + \boldsymbol{\beta}^t) \subseteq \operatorname{supp} \boldsymbol{\alpha}$, so if $\boldsymbol{\alpha}$ is to have minimum support it must satisfy $j\boldsymbol{\alpha} = \boldsymbol{\beta} + \boldsymbol{\beta}^t$ for some $j \in \mathbf{P}$ and some $\boldsymbol{\beta} \in CF(E_p)$. So for our pandiagonal magic squares which are matrices of 0's and 1's, $j = 1$ if $\boldsymbol{\beta} \neq \boldsymbol{\beta}^t$ and $j = 2$ if $\boldsymbol{\beta} = \boldsymbol{\beta}^t$. ■

This theorem is easy to generalize to other cases. For instance, any fixed permutation π of elements will work in place of transposition. In addition, the result holds for types of magic squares other than pandiagonal. The only point of variation is that less can be said about j. Given E a monoid of a system of linear homogeneous Diophantine equations, and E_s the submonoid of elements of E that satisfy $\boldsymbol{\rho} = \boldsymbol{\rho}^\pi$, then $CF(E_s) \subset \left\{ \frac{\boldsymbol{\beta} + \boldsymbol{\beta}^\pi}{j} \mid \boldsymbol{\beta} \in CF(E),\ j \in \mathbf{P} \right\}$. To illustrate the need to include j, examine this element $\boldsymbol{\beta}$ of $CF(E)$ for E the monoid of simple magic squares:

$$\boxed{\begin{matrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{matrix}}\,.$$

Here $\boldsymbol{\beta} \neq \boldsymbol{\beta}^t$, but $\boldsymbol{\beta} + \boldsymbol{\beta}^t$ has a factor of 2.

Another point of interest is that the converse seems to be true in the pandiagonal case, and not in the simple magic case. Using the $\boldsymbol{\beta}$ defined above, note $\frac{\boldsymbol{\beta} + \boldsymbol{\beta}^t}{2} \notin CF(E)$ for E the monoid of simple symmetric 3×3 magic squares.

Chapter 3.

Counting Magic Squares

For much of the remainder of this thesis, there will be a development of a reciprocity relationship for the monoid of solutions to linear homogeneous diophantine equations. With this and some additional machinery, the final problem of creating polynomial descriptions of the number of pandiagonal magic squares for given sizes will be attacked.

3.1. Triangulations

In order to prove facts about monoids, it is often desirable to partition a given monoid E into simpler pieces. To do this, first examine the corresponding convex polyhedral cone $\mathcal{C}$. The simplest type of cones are called *simplicial*. Simplicial cones are e-dimensional pointed convex polyhedral cones (we will write the dimension of a simplicial cone σ as $\dim \sigma$) with e extreme rays. This is the minimum number a cone may have. We would like to have a *triangulation* of $\mathcal{C}$ into simplicial cones. A triangulation is a finite decomposition of a convex polyhedral cone into simplicial cones $\{\sigma_i\}$ such that $\bigcup \sigma_i = \mathcal{C}$, every face of σ_i is also a member of the triangulation, and $\sigma_i \cap \sigma_j$ is a common face of σ_i and σ_j. So a triangulation Γ must contain the faces (which are also simplicial cones) of every cone inside Γ. It is not immediately clear that a finite collection of simplicial cones may satisfy the last condition. This problem is solved with the realization that the faces of a cone span a space that is smaller than the space of the original cone. So the two-dimensional cones have one-dimensional faces, the one-dimensional faces have the zero vector as the only face, and the zero vector has no faces. The definition of triangulation is restricted to avoid too much redundancy; if σ_i an σ_j are simplicial cones in Γ then they overlap precisely in a common face of σ_i and σ_j.

We are only interested in triangulations which do not create any additional extreme rays. In fact, given a convex polyhedral cone $\mathcal{C}$, there will always exist a triangulation into simplicial cones with the same extreme rays as $\mathcal{C}$ (see Stanley [5], Lemma 4.6.1).

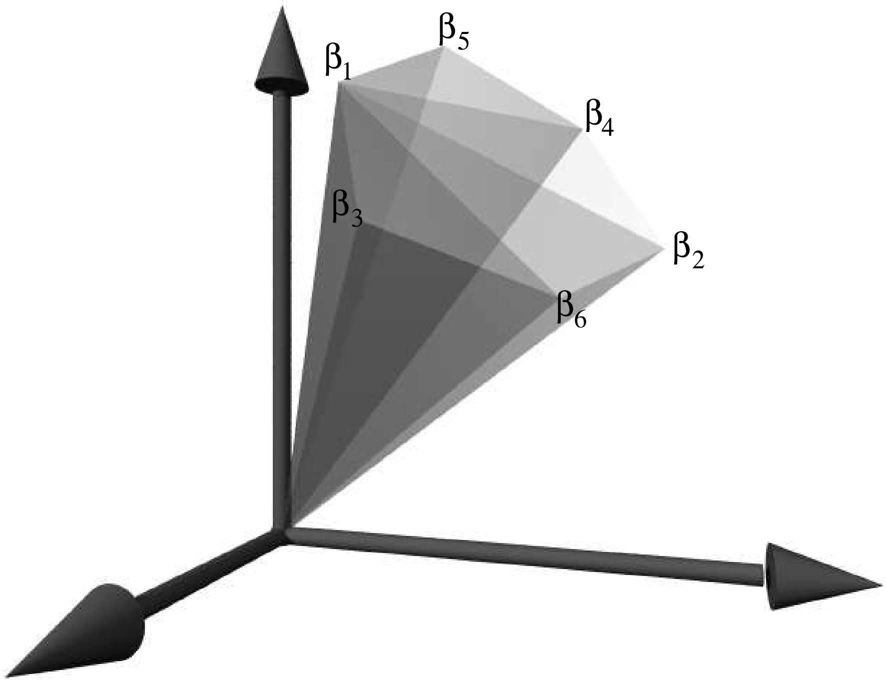


Figure 5. *Triangulation of a Convex Polyhedral Cone*

To illustrate this, the example polyhedral cone is shown in figure 5 with one of its triangulations. Of course, other triangulations are possible. The four three-dimensional simplicial cones shown here contain the following sets of extreme rays:

$$\{\beta_1, \beta_3, \beta_6\}$$
$$\{\beta_1, \beta_6, \beta_2\}$$
$$\{\beta_1, \beta_2, \beta_4\}$$
$$\{\beta_1, \beta_4, \beta_5\}$$

Also included in the triangulation are the two-dimensional faces (facets of the three-dimensional cones) spanned by

$$\{\{\beta_1, \beta_3\}, \{\beta_1, \beta_6\}, \{\beta_3, \beta_6\}, \{\beta_1, \beta_2\},$$
$$\{\beta_2, \beta_6\}, \{\beta_1, \beta_4\}, \{\beta_2, \beta_4\}, \{\beta_1, \beta_5\}, \{\beta_4, \beta_5\}\},$$

the six extreme rays β_1 through β_6, and the zero vector.

In the world of monoids, there is a object corresponding to σ called a *simplicial monoid*. Given a simplicial monoid F, it will have t linearly independent vectors (called the *quasi-generators*) such that every $\lambda \in F$ can be written as a nonnegative integer sum of quasi-generators, divided by a positive integer. The correspondence between cones and monoids carries over to simplicial cones and monoids; if you take the lattice elements of a simplicial cone, you get the corresponding simplicial monoid. Furthermore, the extreme rays of a simplicial cone contain the quasi-generators of the corresponding monoid.

The quasi-generators of a simplicial monoid may be scaled by rational numbers as long as the result has nonnegative integer components. However, there exists a unique set of "smallest" quasi-generators such that every other quasi-generator is a positive integer multiple of one of the smallest quasi-generators. Existence is guaranteed since by the well-ordering principle there exist smallest (in magnitude) integer vectors parallel to the quasi-generators. Furthermore, if there exists a quasi-generator parallel to a smallest quasi-generator that is not an integer multiple of the smallest quasi-generator, then you could subtract off the smallest quasi-generator several times and get another quasi-generator that was smaller, a contradiction. This set of small quasi-generators is in fact the same as the completely fundamental elements of the simplicial monoid. And since the completely fundamental elements are the extreme rays, and the decomposition of E into simplicial monoids has the same extreme rays as E, the union of the completely fundamental elements of the simplicial monoids of a triangulation of E is the same as $CF(E)$. This is independent of the triangulation of E used.

On many of these objects we have been examining, there is a notion of boundary and interior. We now have the tools to define them so that we may speak meaningfully of them later. First off, the usual topological notion of the *boundary* of $\mathcal{C}$ corresponds to the union of the facets of $\mathcal{C}$, which we will call $\partial\mathcal{C}$. Notice that the boundary of $\mathcal{C}$ will always be contained in any triangulation Γ of $\mathcal{C}$, so define the *boundary of the triangulation*, $\partial\Gamma$, as all simplicial cones in Γ completely contained in $\partial\mathcal{C}$, the boundary of $\mathcal{C}$. Defining the *interior* as everything in the set but the boundary then follows naturally for both cones and triangulations. The interior of a cone $\mathcal{C}$ will be denoted $\bar{\mathcal{C}}$, similarly the interior of a triangulation will be represented as $\bar{\Gamma}$.

In particular, the boundary of the above $\mathcal{C}$ is simply the union of the facets with the following extreme rays:

$$\{\{\beta_1, \beta_3\}, \{\beta_3, \beta_6\}, \{\beta_6, \beta_2\}, \{\beta_2, \beta_4\}, \{\beta_4, \beta_5\}, \{\beta_5, \beta_1\}\} \ .$$

In the corresponding triangulation, the boundary, $\partial\Gamma$ has all the facets in $\partial\mathcal{C}$ listed above, plus all the extreme rays, β_1 through β_6, and the zero vector. The interior of $\mathcal{C}$ is simply all of $\mathcal{C}$ not in the boundary. The interior of the triangulation, $\bar{\Gamma}$ contains all the three-dimensional cones of Γ plus the internal faces of Γ, those spanned by

$$\{\{\beta_1, \beta_6\}, \{\beta_1, \beta_2\}, \{\beta_1, \beta_4\}\} \ .$$

We will also be interested later in the *interior of the monoid E*. The interior of a monoid is the set of elements of E that have strictly positive elements. Since every facet of $\mathcal{C}$ corresponds to vectors in E with a particular support that is not maximal, every vector on a facet has a zero component and is therefore not in the interior of E. Conversely, every vector not in a facet has a support which is the union of the supports of all the facets, which is a full support since we are only concerned with positive monoids (see the definition of positive monoids above). Therefore, those vectors not in facets have no nonpositive components and are in the interior of E. So $\bar{E}$, the interior of E, corresponds directly to the interior of $\mathcal{C}$, i.e., $\bar{\mathcal{C}}$. Finally, we will be examining the interior of a simplicial monoid, $\bar{E}_\sigma$. This inherits its definition from the definition of the interior of a monoid. In a triangulation, it is equivalent to define

$$\bar{E}_\sigma = \{\boldsymbol{\alpha} \in E_\sigma \mid \boldsymbol{\alpha} \notin E_\tau \text{ for any } \tau \text{ a proper subset of } \sigma\}$$

and with this definition it is easy to see that $\bar{E}$ is the disjoint union of $\bar{E}_\sigma$ over all $\sigma \in \bar{\Gamma}$.

Unfortunately, E is not in general a disjoint union of simplicial monoids. In order to work with monoids, we will want to manipulate the simplicial monoids in a triangulation of E. In order to translate between the two, it is necessary to take care of the overlapping parts. To this end, we define the *Möbius function* on the simplicial cones of a triangulation Γ of a convex cone $\mathcal{C}$ that has dimension d as

$$\mu(\sigma) = \begin{cases} (-1)^{d-\dim\sigma+1}, & \text{if } \sigma \in \bar{\Gamma} \\ 0, & \text{if } \sigma \in \partial\Gamma. \end{cases} \tag{1}$$

Actually the Möbius function is defined on the elements of the lattice (ordered by inclusion) of elements of Γ, with the element $\hat{1}$ defined as containing all the elements of Γ. Given this μ which is derived from the theory of partially ordered sets (see Stanley [5], Chapter 3), there is a rule for compensating for the overlap between the simplicial monoids in a triangulation. If you take all the $\sigma \in \Gamma$ for which $\mu(\sigma) = -1$, the result will be the entire cone $\mathcal{C}$ overlapping precisely in the set of $\sigma \in \Gamma$ for which $\mu(\sigma) = 1$. The elements of $\mathcal{C}$ in the simplicial cones of the boundary of Γ are included in the d-dimensional simplicial cones of the interior;, they do not figure into the covering of $\mathcal{C}$. In some informal sense, we may think of $\mathcal{C}$ being the sum of the simplicial cones of Γ each times the negative of the Möbius function. We will formalize this later in the context of generating functions. The Möbius function is a generalization of the principle of inclusion-exclusion from combinatorics. This is a technique to count the number of objects having either property A or B by adding the number with A to the number with B and subtracting off those that were counted twice, the number that are both A and B.

Using the triangulation created above for the example monoid, we may write E as the union

$$E = E_{136} \cup E_{162} \cup E_{124} \cup E_{145}$$

with the elements in $E_{16} \cup E_{12} \cup E_{14}$ included twice. Here E_{ij} is the simplicial monoid with quasi-generators β_i and β_j. So in this case, the four three-dimensional simplicial monoids cover E but overlap in the three internal two-dimensional faces.

3.2. Generating Functions

A *generating function* is a compact way of representing a counting function $f(i)$. For instance, we will be looking at the number of magic squares of size n as a function of the row sum. The generating function for $f(i)$ is a *formal power series*, with the standard form being $\sum_{i \geq 0} f(i)x^i$.[4] In a formal power series, we treat x as an arbitrary symbol in the sense that no actual numbers are expected to be substituted. This is usually done if there is not an explicit formula for $f(i)$, but often the generating function form is easier to work with and can more readily be given combinatorial meaning.

Freely ignoring issues of convergence in this formal setting, we may perform several operations upon generating functions. Addition and multiplication of two

---

[4] There is another common form called the *exponential generating function* of $f(i)$ which is the series $\sum_{i \geq 0} \frac{f(i)x^i}{i!}$.

generating functions $A(x)$ and $B(x)$ is evaluated with:

$$A(x) + B(x) = \left(\sum_{i\geq 0} a_i x^i\right) + \left(\sum_{i\geq 0} b_i x^i\right) = \sum_{i\geq 0} (a_i + b_i)\, x^i$$

$$A(x)B(x) = \left(\sum_{i\geq 0} a_i x^i\right)\left(\sum_{i\geq 0} b_i x^i\right) = \sum_{i\geq 0}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i\,.$$

These definitions are motivated by how we would expect to manipulate polynomials. In fact, if we consider $a_i \in \mathbf{C}$, then the set of all formal power series $\sum_{i\geq 0} a_i x^i$ forms an integral domain called $\mathbf{C}[[x]]$ with several nice properties.[5] In particular, if $a_0 \neq 0$ (often written $A(0) \neq 0$) then there exists a $B(x)$ such that $A(x)B(x) = 1$ (here 1 is the series $1 + 0x + 0x^2 + \cdots$). We call this $B(x)$ the inverse of $A(x)$ and denote it $A(x)^{-1}$. This inverse function satisfies all the algebraic relations we expect, so $\left(A(x)^{-1}\right)^{-1} = A(x)$ and the inverse of a product is the product of the inverses.

To invert a polynomial, the only formula we will actually use is

$$(1 - \alpha x)(\sum_{i\geq 0} \alpha^i x^i) = 1$$

which is easily verified using the definition of power series multiplication. A more aesthetic way of looking at this is to define division as multiplication by the inverse, and then writing

$$\sum_{i\geq 0} \alpha^i x^i = \frac{1}{1 - \alpha x}$$

which is the formula for summing a geometric series.

For counting magic squares the use of multi-variable generating functions will be necessary. These are functions of m variables $x_1, \ldots, x_m$, so let $\mathbf{x}$ be the vector $[\, x_1 \quad \cdots \quad x_m \,]$. Given $\boldsymbol{\alpha} = [\, \alpha_1 \quad \cdots \quad \alpha_m \,] \in \mathbf{N}^m$ define $\mathbf{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdots x_m^{\alpha_m}$. Now

5 In fact $\mathbf{C}[[x]]$ is a principal ideal domain, (therefore) a unique factorization domain, every ideal has the form x^i for some $i \geq 0$, as well as other algebraic properties listed in [5], section 1.1.

the general power series of m variable is

$$\sum_{\boldsymbol{\alpha}\in\mathbf{N}^m} b_{\boldsymbol{\alpha}}\mathbf{x}^{\boldsymbol{\alpha}}\,.$$

Again, multiplication and addition of functions is defined as it would be in ordinary algebra.[6] Given a subset of vectors $S \in \mathbf{N}^m$, we will be looking at the generating function

$$S(\mathbf{x}) = \sum_{\boldsymbol{\alpha}\in S} \mathbf{x}^{\boldsymbol{\alpha}}\,.$$

(Notice the lack of coefficient in front of the variable term.) In order to invert generating functions of m-dimensional vectors, we will need the analog to the sum of a geometric series,

$$\sum_{i\geq 0} \mathbf{x}^{\alpha i} = \frac{1}{1-\mathbf{x}^{\alpha}}$$

which follows from the definition of power series multiplication in much the same way as the single variable case. What makes generating functions useful is that certain algebraic operations have combinatorial meaning. The product of two generating functions F and G is the function that counts all possible sums of vectors counted by F added to vectors counted by G.

Before examining the generating function for magic squares of a particular sum, we must first look at the generating function of a monoid. These are generating functions of m-dimensional vectors $\mathbf{x}$. Following the definitions above, for a given monoid E we write

$$E(\mathbf{x}) = E(x_1,\ldots,x_m) = \sum_{\boldsymbol{\alpha}\in E} \mathbf{x}^{\boldsymbol{\alpha}}\,.$$

Define the generating functions $E_\sigma(\mathbf{x})$ and $\bar{E}(\mathbf{x})$ for the sets E_σ and $\bar{E}$ similarly. In particular, $E_\sigma(\mathbf{x})$ and $\bar{E}_\sigma(\mathbf{x})$ are well behaved, and we will be able to write the generating functions for E and $\bar{E}$ in terms of these. In fact, given a triangulation

6 $\mathbf{C}[[x_1,\ldots,x_m]]$ is a unique factorization domain, but not a principal ideal domain for $m > 2$, see Stanley [5], section 1.1.

Γ of $\mathcal{C}$, since $\bar{E}$ is the disjoint union of the interiors of the simplicial monoids in $\bar{\Gamma}$, it follows immediately that

$$\bar{E}(\mathbf{x}) = \sum_{\sigma \in \bar{\Gamma}} \bar{E}_\sigma(\mathbf{x})\,. \tag{2}$$

The corresponding formula for $E(\mathbf{x})$ uses the Möbius function μ defined earlier,

$$E(\mathbf{x}) = -\sum_{\sigma \in \Gamma} \mu(\sigma) E_\sigma(\mathbf{x})\,. \tag{3}$$

The formal justification of the Möbius inversion used to prove this formula is in [5], Lemma 4.6.4. Intuitively, we are adding in the elements of the n-dimensional simplicial monoids, but then subtracting off the internal $(n-1)$-dimensional faces that are the intersection of two n-dimensional simplicial monoids. The intersections of the internal $(n-1)$-dimensional simplicial monoids must then be added back in, and so on.

3.3. Rational Generating Functions

Consider a generic simplicial cone σ, and the corresponding simplicial monoid E_σ, with quasi-generators $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_d\}$. Later we will take σ to be a member of a triangulation, but for the moment, the results hold for all simplicial cones. Let D_σ and $\bar{D}_\sigma$ be the subsets of E_σ, dependent upon the choice of quasi-generators, defined by

$$D_\sigma = \{\boldsymbol{\alpha} \in E_\sigma \mid \boldsymbol{\alpha} = a_1\boldsymbol{\beta}_1 + \cdots + a_d\boldsymbol{\beta}_d,\ 0 \le a_i < 1, a_i \in \mathbf{Q}\}$$
$$\bar{D}_\sigma = \{\boldsymbol{\alpha} \in E_\sigma \mid \boldsymbol{\alpha} = a_1\boldsymbol{\beta}_1 + \cdots + a_d\boldsymbol{\beta}_d,\ 0 < a_i \le 1, a_i \in \mathbf{Q}\}\,.$$

Since E_σ is a discrete set containing only lattice points of which both D_σ and $\bar{D}_\sigma$ are bounded subsets, they are both finite sets. Also note that letting the a_i include members of $\mathbf{R}$ would not increase the size of D_σ or $\bar{D}_\sigma$, because every component of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_i$ is an integer, and therefore every a_i must satisfy some $p = a_i q$. Examining (with a short computer program) the simplicial monoids E_{136}, E_{145}, E_{162}, E_{124}, and E_{12} of the example triangulation, we may find the corresponding D_σ (D_σ is the empty set for any σ corresponding to all the other members of Γ)

$$
\begin{aligned}
D_{136} = \{\omega_1, \omega_2\} &= \{[1 \quad 6 \quad 3 \quad 3 \quad 6 \quad 1 \quad 3 \quad 4 \quad 3],\\
&\qquad [2 \quad 7 \quad 4 \quad 4 \quad 7 \quad 2 \quad 4 \quad 5 \quad 4]\}\\
D_{145} = \{\omega_3, \omega_4\} &= \{[3 \quad 3 \quad 1 \quad 6 \quad 1 \quad 6 \quad 3 \quad 3 \quad 4],\\
&\qquad [4 \quad 4 \quad 2 \quad 7 \quad 2 \quad 7 \quad 4 \quad 4 \quad 5]\}\\
D_{162} = D_{124} = D_{12} &= \{\omega_5, 2\omega_5, 3\omega_5, 4\omega_5\} =\\
&\quad \{[1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1],\\
&\qquad [2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2],\\
&\qquad [3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3],\\
&\qquad [4 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4]\}.
\end{aligned}
$$

Think of D_σ as the set of lattice points at the very tip of the cone σ, so that no quasi-generator may be subtracted off while staying inside the cone. If σ is three dimensional D_σ looks like a parallelopiped, with one corner at the origin, one corner each at $\boldsymbol{\beta}_1$ through $\boldsymbol{\beta}_3$, one at each of $\boldsymbol{\beta}_1 + \boldsymbol{\beta}_2$, $\boldsymbol{\beta}_2 + \boldsymbol{\beta}_3$, $\boldsymbol{\beta}_3 + \boldsymbol{\beta}_1$, and one at $\boldsymbol{\beta}_1 + \boldsymbol{\beta}_2 + \boldsymbol{\beta}_3$ (see figure 6). In general, it will be a convex polyhedron with corners at each of the possible sums of subsets of the quasi-generators.

Proposition. *Every element $\boldsymbol{\alpha} \in E_\sigma$ may be uniquely decomposed into an element of D_σ plus a nonnegative integer sum of quasi-generators, and any such vector is in E_σ.*

Proof. Given a vector $\boldsymbol{\alpha} \in E_\sigma$, it may be written uniquely in terms of the (linearly independent) quasi-generators, i.e., $\boldsymbol{\alpha} = b_1\boldsymbol{\beta}_1 + \cdots + b_d\boldsymbol{\beta}_d$, for $b_i \in \mathbf{Q}$ and $b_i \geq 0$. Let a_i be the smallest integer less than or equal to b_i for each i, $1 \leq i \leq d$. Notice $\boldsymbol{\gamma} = \boldsymbol{\alpha} - a_1\boldsymbol{\beta}_1 - \cdots - a_d\boldsymbol{\beta}_d$ is a member of E_σ but does not contain a positive integer multiple of any quasi-generator $\boldsymbol{\beta}_i$, so $0 \leq b_i - a_i < 1$ and $\boldsymbol{\gamma} \in D_\sigma$. To prove that this representation is unique, assume there is another representation $\boldsymbol{\alpha} = \gamma' + a_1'\boldsymbol{\beta}_1 + \cdots + a_d'\boldsymbol{\beta}_d$. Equating the two expressions for $\boldsymbol{\alpha}$ and subtracting

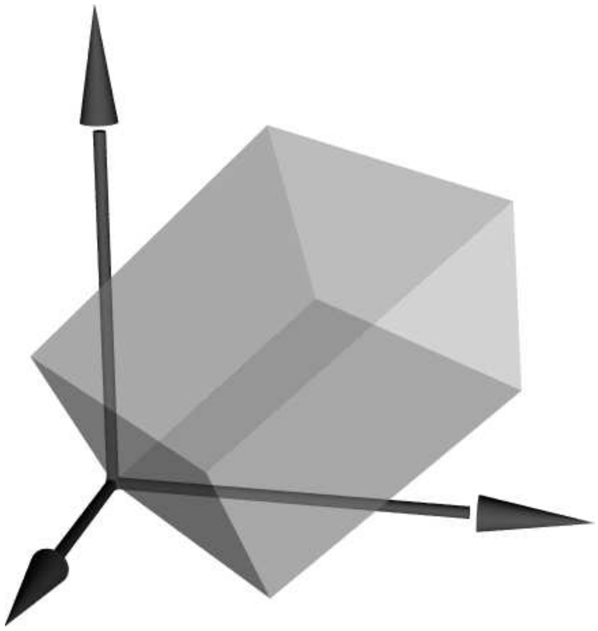


Figure 6. *A three dimensional D_σ*

yields $\boldsymbol{\gamma} - \boldsymbol{\gamma}' = (a_1' - a_1)\boldsymbol{\beta}_1 + \cdots + (a_d' - a_d)\boldsymbol{\beta}_d$. The coefficients on the left are all strictly between -1 and 1. On the other hand, the coefficients on the right are all integers, so all the coefficients must be 0, and $\boldsymbol{\gamma}$ and $\boldsymbol{\gamma}'$ are the same decomposition of $\boldsymbol{\alpha}$. Finally every vector written as a member of D_σ plus a nonnegative integer sum of quasi-generators is clearly in E_σ since it is a sum of vectors in E_σ. ■

Similarly, $\bar{E}_\sigma$ is exactly the set of vectors of the form $\bar{\boldsymbol{\gamma}} + a_1\boldsymbol{\beta}_1 + \cdots + a_d\boldsymbol{\beta}_d$, where $\bar{\boldsymbol{\gamma}} \in \bar{D}_\sigma$ and $a_i \in \mathbf{N}$. The only difference in the proof is that a_i is the least integer strictly less than b_i.[7] This proof suggests the intuitive notion that the elements of D_σ represent the fractional parts of elements of E_σ.

Now that we have this decomposition of E_σ and $\bar{E}_\sigma$, we may write their generating functions in terms of a finite number of vectors. Recall that $(1 - \mathbf{x}^{\boldsymbol{\beta}})^{-1}$

---

[7] Stanley [5] seems to get this wrong in Lemma 4.6.7, letting a_i be the least integer $\geq b_i$.

is the geometric series representing all nonnegative factors of β, or

$$\sum_{i\geq 0} \mathbf{x}^{i\beta}\,.$$

If we multiply two of these, we get the generating function for all possible nonnegative integer sums of the two vectors. Continuing this, we arrive at the expression for all nonnegative integer sums of the quasi-generators of E_σ:

$$\prod_{i=1}^{d}(1-\mathbf{x}^{\beta_i})^{-1}\,.$$

By the previous proposition, we may get the generating function for a simplicial monoid by multiplying the generating function for D_σ into the above product, or:

$$E_\sigma(\mathbf{x}) = \left(\sum_{\gamma\in D_\sigma} \mathbf{x}^{\gamma}\right)\prod_{i=1}^{d}(1-\mathbf{x}^{\beta_i})^{-1}\,. \tag{4}$$

For example, from the above expression of D_{136}, we may write

$$E_{136}(\mathbf{x}) = \frac{1+\mathbf{x}^{\omega_1}+\mathbf{x}^{\omega_2}}{(1-\mathbf{x}^{\beta_1})(1-\mathbf{x}^{\beta_3})(1-\mathbf{x}^{\beta_6})}\,.$$

Similarly, for the interior of the monoid, we have:

$$\bar{E}_\sigma(\mathbf{x}) = \left(\sum_{\bar{\gamma}\in \bar{D}_\sigma} \mathbf{x}^{\bar{\gamma}}\right)\prod_{i=1}^{d}(1-\mathbf{x}^{\beta_i})^{-1}\,. \tag{5}$$

We have already expressed $E(\mathbf{x})$ and $\bar{E}(\mathbf{x})$ in terms of the generating functions for the simplicial monoids in a triangulation of E. Since each simplicial monoid's generating function may be represented as a finite polynomial in $\mathbf{x}$ (formally) divided by another finite polynomial in $\mathbf{x}$, we may conclude that the same is true for $E(\mathbf{x})$ and $\bar{E}(\mathbf{x})$.[8] Such generating functions are called *rational.* Define the *degree*, written $\deg R(\mathbf{x})$ of a rational function to be the degree of the numerator

[8] Stanley goes on to prove the exact form of the denominator, namely $D(\mathbf{x}) = \prod_{\beta\in CF(E)}(1-\mathbf{x}^{\beta})$ in Theorem 4.6.11, but this is not necessary for our purposes.

minus the degree of the denominator. The generating function for the example monoid may be written using the above process as:

$$
\begin{aligned}
E(\mathbf{x}) = & \frac{1+\mathbf{x}^{\boldsymbol{\omega}_1}+\mathbf{x}^{\boldsymbol{\omega}_2}}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_3})(1-\mathbf{x}^{\boldsymbol{\beta}_6})} + \frac{1+\mathbf{x}^{\boldsymbol{\omega}_3}+\mathbf{x}^{\boldsymbol{\omega}_4}}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_4})(1-\mathbf{x}^{\boldsymbol{\beta}_5})} + \\
& \frac{\mathbf{x}^{\boldsymbol{\omega}_5}+\mathbf{x}^{2\boldsymbol{\omega}_5}+\mathbf{x}^{3\boldsymbol{\omega}_5}+\mathbf{x}^{4\boldsymbol{\omega}_5}}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_6})(1-\mathbf{x}^{\boldsymbol{\beta}_2})} + \frac{\mathbf{x}^{\boldsymbol{\omega}_5}+\mathbf{x}^{2\boldsymbol{\omega}_5}+\mathbf{x}^{3\boldsymbol{\omega}_5}+\mathbf{x}^{4\boldsymbol{\omega}_5}}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_2})(1-\mathbf{x}^{\boldsymbol{\beta}_4})} - \\
& \frac{1}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_6})} - \frac{1}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_4})} - \\
& \frac{\mathbf{x}^{\boldsymbol{\omega}_5}+\mathbf{x}^{2\boldsymbol{\omega}_5}+\mathbf{x}^{3\boldsymbol{\omega}_5}+\mathbf{x}^{4\boldsymbol{\omega}_5}}{(1-\mathbf{x}^{\boldsymbol{\beta}_1})(1-\mathbf{x}^{\boldsymbol{\beta}_2})}\,.
\end{aligned}
$$

3.4. Reciprocity

Given a rational function $R(\mathbf{x}) = R(x_1, \ldots, x_m)$, define $R(1/\mathbf{x})$ to be the rational function $R(1/x_1, \ldots, 1/x_m)$. We are now in a position to prove a reciprocity relationship for simplicial monoids, an important stepping stone on the way to establishing one for monoids in general.

First of all, given the simplicial monoid E_σ with quasi-generators $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_d$, and $D_\sigma = \{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s\}$, then if $\boldsymbol{\beta} = \boldsymbol{\beta}_1 + \cdots + \boldsymbol{\beta}_d$ we may write

$$\bar{D}_\sigma = \{\beta - \boldsymbol{\alpha}_1, \ldots, \beta - \boldsymbol{\alpha}_s\}\,.$$

This follows from the fact that $\boldsymbol{\beta} - \boldsymbol{\gamma}$ takes $\boldsymbol{\gamma} \in D_\sigma$ bijectively to elements of $\bar{D}_\sigma$. This is clear since $0 \le a_i < 1$ and $\boldsymbol{\gamma} = \boldsymbol{\alpha}_i = a_1\boldsymbol{\beta}_1 + \cdots + a_d\boldsymbol{\beta}_d \in D_\sigma$ if and only if $0 < 1 - a_i \le 1$ and $\boldsymbol{\beta} - \boldsymbol{\gamma} = \boldsymbol{\beta} - \boldsymbol{\alpha}_i = (1-a_1)\boldsymbol{\beta}_1 + \cdots + (1-a_d)\boldsymbol{\beta}_d \in \bar{D}_\sigma$.

Lemma. *(Reciprocity of simplicial monoids) Given a simplicial monoid $E_\sigma \subseteq \mathbf{N}^m$ of dimension d, then*

$$\bar{E}_\sigma(\mathbf{x}) = (-1)^d E_\sigma(1/\mathbf{x})\,.$$

Proof. Substituting $1/\mathbf{x}$ into equation (4) yields,

$$E_\sigma(1/\mathbf{x}) = \left(\sum_{\boldsymbol{\alpha}\in D_\sigma} \mathbf{x}^{-\boldsymbol{\alpha}}\right)\prod_{i=1}^{d}\left(1-\mathbf{x}^{-\boldsymbol{\beta}_i}\right)^{-1}$$

Now take the $\boldsymbol{\beta} = \boldsymbol{\beta}_1 + \cdots + \boldsymbol{\beta}_d$ as was defined above, and multiply the right side of the previous equation by $\mathbf{x}^{\boldsymbol{\beta}}\mathbf{x}^{-\boldsymbol{\beta}} = 1$. Distributing the $\mathbf{x}^{\boldsymbol{\beta}}$ into the numerator gives the relation

$$\mathbf{x}^{\boldsymbol{\beta}}\sum_{\boldsymbol{\alpha}\in D_\sigma}\mathbf{x}^{-\boldsymbol{\alpha}} = \sum_{\boldsymbol{\alpha}\in D_\sigma}\mathbf{x}^{\boldsymbol{\beta}-\boldsymbol{\alpha}} = \sum_{\boldsymbol{\alpha}\in \bar{D}_\sigma}\mathbf{x}^{\boldsymbol{\alpha}}$$

by the relation from the previous paragraph. Now take the $\mathbf{x}^{-\boldsymbol{\beta}} = (\mathbf{x}^{\boldsymbol{\beta}})^{-1}$ and write it as $\prod_{i=1}^{d}(\mathbf{x}^{\boldsymbol{\beta}_i})^{-1}$. Merging that with the product in the denominator yields

$$\begin{aligned}\prod_{i=1}^{d}(\mathbf{x}^{\boldsymbol{\beta}_i})^{-1}\prod_{i=1}^{d}\left(1-\mathbf{x}^{-\boldsymbol{\beta}_i}\right)^{-1} &= \prod_{i=1}^{d}\left[(\mathbf{x}^{\boldsymbol{\beta}_i})\left(1-\mathbf{x}^{-\boldsymbol{\beta}_i}\right)\right]^{-1} \\ &= \prod_{i=1}^{d}\left(\mathbf{x}^{\boldsymbol{\beta}_i}-1\right)^{-1} \\ &= (-1)^d\prod_{i=1}^{d}\left(1-\mathbf{x}^{\boldsymbol{\beta}_i}\right)^{-1}.\end{aligned}$$

Writing this all together,

$$\begin{aligned}E_\sigma(1/\mathbf{x}) &= \left(\mathbf{x}^{\boldsymbol{\beta}}\sum_{\boldsymbol{\alpha}\in D_\sigma}\mathbf{x}^{-\boldsymbol{\alpha}}\right)\mathbf{x}^{-\boldsymbol{\beta}}\prod_{i=1}^{d}\left(1-\mathbf{x}^{-\boldsymbol{\beta}_i}\right)^{-1} \\ &= \left(\sum_{\boldsymbol{\alpha}\in \bar{D}_\sigma}\mathbf{x}^{\boldsymbol{\alpha}}\right)(-1)^d\prod_{i=1}^{d}\left(1-\mathbf{x}^{\boldsymbol{\beta}_i}\right)^{-1} \\ &= (-1)^d\bar{E}_\sigma(\mathbf{x})\end{aligned}$$

by equation (5). Multiplying both sides by $(-1)^d$ gives the desired relation. ∎

With this, we may prove the reciprocity theorem for linear homogeneous diophantine equations. We will actually use a specific corollary of this.

Theorem. *Let E be a positive monoid of $\mathbf{N}$-solutions to $\boldsymbol{\Phi\alpha} = \mathbf{0}$, and let $d = \dim\mathcal{C}$, where $\mathcal{C}$ is the corresponding convex polyhedral cone to E. Then*

$$\bar{E}(\mathbf{x}) = (-1)^d E(1/\mathbf{x}).$$

Proof. Substitute the definition of the Möbius function from (1) into the formula for the decomposition of $E(\mathbf{x})$, the generating function for E, into simplicial monoids, equation (3),

$$\begin{aligned} E(\mathbf{x}) &= -\left(\sum_{\sigma\in\bar{\Gamma}}(-1)^{d-\dim\sigma+1}E_\sigma(\mathbf{x}) + \sum_{\sigma\in\partial\Gamma} 0E_\sigma(\mathbf{x})\right) \\ &= -\sum_{\sigma\in\bar{\Gamma}}(-1)^{d-\dim\sigma+1}E_\sigma(\mathbf{x})\,. \end{aligned}$$

Substituting in $1/\mathbf{x}$ for $\mathbf{x}$ yields (after simplifying the powers of -1):

$$\begin{aligned} E(1/\mathbf{x}) &= (-1)^d\sum_{\sigma\in\bar{\Gamma}}(-1)^{\dim\sigma}E_\sigma(1/\mathbf{x}) \\ &= (-1)^d\sum_{\sigma\in\bar{\Gamma}}\bar{E}_\sigma(\mathbf{x}) \end{aligned}$$

by the reciprocity of simplicial monoids. Substituting equation (2) completes the theorem. ∎

In our example system, notice the vector

$$\boldsymbol{\omega}_5 = [1\quad 1\quad 1\quad 1\quad 1\quad 1\quad 1\quad 1\quad 1] \in D_{12}$$

is in E (since $D_{12} \subset E$). Now if we take any vector $\boldsymbol{\alpha}$ in E, and add $\boldsymbol{\omega}_5$ to it, it will still be in the monoid, but since the result can not have a nonpositive component, $\alpha + \boldsymbol{\omega}_5$ will be in $\bar{E}$. Conversely, every component of a vector $\bar{\alpha}$ in $\bar{E}$ is at least 1, so we may safely subtract $\boldsymbol{\omega}_5$ from $\bar{\boldsymbol{\alpha}}$ and get an element of E. So $\boldsymbol{\alpha} \in E$ if and only if $\boldsymbol{\alpha} + \boldsymbol{\omega}_5 \in \bar{E}$, which we will write $\bar{E} = \boldsymbol{\alpha} + E$. Also, the square matrix containing only ones is a pandiagonal magic square, and by the same argument the monoid of pandiagonal magic squares satisfies a similar relation. Motivated by this, let us rephrase the reciprocity relationship in terms of E alone.

Corollary. *Under the conditions of the reciprocity theorem for linear homogeneous diophantine equations, given $\boldsymbol{\gamma} \in \mathbf{Z}^m$, $E(1/\mathbf{x}) = (-1)^d\mathbf{x}^{\boldsymbol{\gamma}}E(\mathbf{x})$ if and only if $\bar{E} = \boldsymbol{\gamma} + E$.*

Proof. The condition $\bar{E} = \boldsymbol{\gamma} + E$, meaning $\boldsymbol{\alpha} \in E$ if and only if $\boldsymbol{\alpha} + \boldsymbol{\gamma} \in \bar{E}$, is equivalent to $\bar{E}(\mathbf{x}) = \mathbf{x}^{\boldsymbol{\gamma}} E(\mathbf{x})$. Both are equivalent to increasing each vector in E by $\boldsymbol{\gamma}$ to get $\bar{E}$. Substituting $\mathbf{x}^{\boldsymbol{\gamma}} E$ for $\bar{E}$ in the reciprocity theorem gives

$$(-1)^d E(1/\mathbf{x}) = \mathbf{x}^{\boldsymbol{\gamma}} E(\mathbf{x})$$

which, after multiplying both sides by $(-1)^d$, gives the result. ∎

It is generally impractical to compute $E(\mathbf{x})$ explicitly as was done for the sample monoid. However, we do not need to know every solution to count the number of them, so all is not lost. In particular, all we really want is a function which generates the row sum of a given set of magic squares. For every magic square $\boldsymbol{\rho}$ in E, there is exactly one term of $E(\mathbf{x})$ of the form $x_1^{\rho_{11}} x_2^{\rho_{12}} \cdots x_n^{\rho_{1n}} \cdots x_{n^2}^{\rho_{nn}}$. If you replace $x_1, \ldots, x_n$ with a variable λ and replace the remaining $x_{n+1}, \ldots, x_{n^2}$ with ones, you get $\lambda^{\rho_{11}} \cdots \lambda^{\rho_{1n}} = \lambda^{\rho_{11} + \cdots + \rho_{1n}}$. However, $\rho_{11} + \cdots + \rho_{1n}$ is the sum of the elements in the first row of the magic square $\boldsymbol{\rho}$, which is the row sum for $\boldsymbol{\rho}$. So substituting λ for x_1 through x_n, and 1 for x_{n+1} through x_{n^2}, into $E(x_1, \ldots, x_n, x_{n+1} \ldots, x_{n^2})$, gives you a function of one variable, $G(\lambda) = E(\lambda, \ldots, \lambda, 1, \ldots, 1)$, which generates the number of magic squares as a function of the row sum. So if $f(r)$ is the number of magic squares of a particular type with row sum r, then $G(\lambda) = \sum_{r \geq 0} g(r) \lambda^r$.

Motivated by this, we prove our last theorem from the theory of linear homogeneous diophantine equations.

Proposition. *Let E be as usual the positive monoid of* $\mathbf{N}$*-solutions to the system* $\boldsymbol{\Phi\alpha} = \mathbf{0}$. *Choose $a_1, \ldots, a_m \in \mathbf{Z}$. Let $L(\boldsymbol{\alpha})$ be $a_1\alpha_1 + \cdots + a_m\alpha_m$. Let $g(r)$ be the number of elements $\boldsymbol{\alpha} \in E$ satisfying $L(\boldsymbol{\alpha}) = r$. If $g(r)$ is finite, then let $G(\lambda) = \sum_{r \geq 0} g(r)\lambda^r$, and:*

- **i.** *$G(\lambda)$ is exactly $E(\lambda^{a_1}, \ldots, \lambda^{a_m})$ and is a polynomial in λ over* $\mathbf{C}$, *and*
- **ii.** *Since $E \neq \{\mathbf{0}\}$ being a positive monoid,* $\deg G(\lambda) < 0$.

Proof. The discussion above makes **i.** clear, given the fact that $E(\mathbf{x})$ a polynomial in $\mathbf{x}$ over $\mathbf{C}$ implies $G(\lambda)$ is a polynomial in λ over $\mathbf{C}$. To prove **ii.**, notice that given a triangulation Γ of $\mathcal{C}$, which will always exist, $E(\lambda^{a_1}, \ldots, \lambda^{a_m})$ is a sum of $E_\sigma(\lambda^{a_1}, \ldots, \lambda^{a_m})$ with $\sigma \in \bar{\Gamma}$. It is therefore sufficient to show that $\deg E_\sigma(\lambda^{a_1}, \ldots, \lambda^{a_m}) < 0$ for $\sigma \in \bar{\Gamma}$. Referring to equation (4), we see this occurs when

$$\deg \sum_{\boldsymbol{\gamma} \in D_\sigma} (\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\gamma}} < \deg \prod_{i=1}^{d} (1 - (\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\beta}_i})$$

Where $(\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\alpha}}$ is $\lambda^{a_1\alpha_1 + \cdots + a_m\alpha_m}$, which is in fact precisely $\lambda^{L(\boldsymbol{\alpha})}$. Now, for every $\boldsymbol{\gamma} \in D_\sigma$ we know $\boldsymbol{\gamma} = b_1\boldsymbol{\beta}_1 + \cdots + b_d\boldsymbol{\beta}_d$, $0 \leq b_i < 1$ by the definition of D_σ. Now the γ term in the numerator is $\lambda^{L(\boldsymbol{\gamma})}$, which has degree $L(\gamma)$ which is certainly less than $L(\boldsymbol{\beta}_1) + \cdots + L(\boldsymbol{\beta}_d)$ as long as $d > 0$. However, $d = 0$ only when $\sigma = \{\mathbf{0}\}$ but $\sigma \in \bar{\Gamma}$ and $\{\mathbf{0}\} \notin \bar{\Gamma}$. So at most the degree of the numerator is the maximum degree of each of the terms of the numerator, each of which has degree less than $L(\boldsymbol{\beta}_1) + \cdots + L(\boldsymbol{\beta}_d)$. So the degree of the numerator is less than $L(\boldsymbol{\beta}_1) + \cdots + L(\boldsymbol{\beta}_d)$. Now the degree of the denominator is

$$\begin{aligned}
\deg \prod_{i=1}^{d} (1 - (\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\beta}_i}) &= \sum_{i=1}^{d} \deg(1 - (\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\beta}_i}) \\
&= \sum_{i=1}^{d} \deg(\lambda^{a_1}, \ldots, \lambda^{a_m})^{\boldsymbol{\beta}_i} \\
&= \sum_{i=1}^{d} L(\boldsymbol{\beta}_i) \\
&= L(\boldsymbol{\beta}_1) + \cdots + L(\boldsymbol{\beta}_d)
\end{aligned}$$

which we know to be greater than the degree of the numerator, which proves the proposition. ■

3.5. The Counting Function

If E is the monoid of $n \times n$ magic squares, we will let $a_i = 1$ for $i \leq n$ and $a_i = 0$ for $i > n$. Then $G(\lambda)$ is the generating function for $g(r)$, the number of magic squares with row sum r. Furthermore, if all of the completely fundamental elements of E have the same row sum, then a theorem of rational generating functions gives us that $g(r)$ is a polynomial of finite degree) in r. This is wonderful since we may bound the degree of g and then g will be completely determined by a fixed number of points. Furthermore, the reciprocity theorem gives us an expression for $G(1/\lambda)$ which is the generating function for $g(-r)$, which we may use to get additional values for $g(r)$. This is where all the work developing the machinery pays off.

In order to bound the degree of the polynomial $g(r)$, one may simply notice how many independent variables there are in $\mathbf{\Phi}$. If we let the k independent variables range over $0, \ldots, r$ then we certainly will get all of the magic squares of row sum r. However, there are only $(r+1)^k$ possible ways of assigning the variables, a polynomial of degree k in r, so the degree of $g(r)$ is at most k. In practice, we will always have to row reduce the system $\mathbf{\Phi}\boldsymbol{\alpha} = \mathbf{0}$, so we may easily compute the number of independent variables (the number of columns of $\mathbf{\Phi}$ minus the rank, or number of row-reduced rows, of $\mathbf{\Phi}$). A priori, we may note that for $n \times n$ pandiagonal magic squares, the row reduced $\mathbf{\Phi}$ will have at most $4n-4$ rows. This is immediate from the following expression of $\mathbf{\Phi}$:

$n-1$ equations of the form (the first row)$-$(the k^{th} row)=0

$n-1$ equations of the form (the first column)$-$(the k^{th} column)=0

$n-1$ equations of the form (the main $\searrow$ diagonal)$-$(the k^{th} $\searrow$ diagonal)=0

$n-1$ equations of the form (the main $\swarrow$ diagonal)$-$(the k^{th} $\swarrow$ diagonal)=0 .

This insures that all the rows are equal to each other, all the columns are equal, etc. To see that the rows must equal the columns (and the diagonals of course), look at the sum of all the elements of the magic square. Clearly this must be nr, where r is the sum of any row. But each of the n columns are equal, and together they sum to the entire square, so each must sum to $rn/n = r$. Similarly for the diagonals. In practice, there is one additional equation: the restriction of the magic square to a particular row sum. This lowers the upper bound by one degree. This bound was exact for some of the systems, but for 4×4 and 6×6 there was one fewer dependent variable. I expect in order to get a lower bound (and possibly a stricter upper bound for the cases noted), an argument like that in [3] could be used (the argument in [3] is for semi-magic squares).

3.6. Accelerating the Counting of Magic Squares

All that is left is to count the number of pandiagonal 4×4 and 5×5 magic squares for particular row sums. To start off, there is only one magic square with row sum 0, the magic square filled with zeros. Rather than attempting brute force, let us take advantage of all the structure that is present in E.

I have fashioned the following theorem to count the pandiagonal magic squares since I knew for 4×4 and 5×5, the completely fundamental elements satisfy the conditions of the theorem.

Theorem. *Let $\mathbf{\Phi}$ be a linear system, E be the monoid of $\mathbf{N}$-solutions of $\mathbf{\Phi}$, and $CF(E)$ be the completely fundamental elements of E. If no component of any of the vectors in $CF(E)$ is greater than 1 (the vectors are made up of zeros and ones), then every element of E is the nonnegative integer sum of vectors in $CF(E)$.*

Proof. The zero vector is the empty sum of vectors from $CF(E)$, so consider the nonzero vectors. Note that every nonzero vector in E is a a nonnegative real sum of vectors in $CF(E)$. We want to show there exists an equal nonnegative integer sum.

Choose a generic vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_t]$ with $\boldsymbol{\alpha} \in E$, or $\alpha_i \in \mathbf{N}$ $\forall i \in \{1, 2, \ldots, t\}$. Now examine the support of $\boldsymbol{\alpha}$. This is clearly a superset of the support of some vector $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_t]$ in $CF(E)$, since every vector with minimum support is in $CF(E)$ (recall from chapter 2). For every i in the support of $\boldsymbol{\alpha}$ (and therefore for every i in the support of $\boldsymbol{\beta}$), $\alpha_i > 0$, and since $\alpha_i \in \mathbf{N}$ we know $\alpha_i \geq 1$. Since for every i in the support of $\boldsymbol{\beta}$, $\beta_i = 1$ and $\alpha_i \geq 1$, we know $\boldsymbol{\alpha} - \boldsymbol{\beta}$ is in E.

At this point either $\boldsymbol{\alpha} - \boldsymbol{\beta}$ is the zero vector and we are done, or we may continue the above process with $\boldsymbol{\alpha} - \boldsymbol{\beta}$ until we have decomposed $\boldsymbol{\alpha}$ into a nonnegative integer sum of vectors in $CF(E)$. This process is guaranteed to terminate since at each step $\sum \alpha_i$ decreases. ∎

An immediate result of this is that there are no 4×4 pandiagonal magic squares with an odd row sum, since no element of any CF(E) is greater than 1, and all of the elements of the CF(E) have row sum 2. This also gives us a reliable method of finding all magic squares with a given row sum. We just have to examine all of the possible ways of choosing k elements of CF(E), and then throwing away the different combinations with the same sum. The number of 4×4 pandiagonal magic squares with row sum 2 is 8, exactly the elements of CF(E). A computer using this technique may rapidly produce and count magic squares with a given row sum (see program in Appendix C). For example, the thirty-three 4×4 pandiagonal magic squares with row sum 4 are:

$$
\begin{array}{|cccc|}\hline 2&2&0&0\\0&0&2&2\\2&2&0&0\\0&0&2&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 2&1&1&0\\1&0&2&1\\1&2&0&1\\0&1&1&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 2&1&1&0\\0&1&1&2\\1&2&0&1\\1&0&2&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 2&1&0&1\\0&1&2&1\\2&1&0&1\\0&1&2&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&2&1&0\\1&0&1&2\\1&2&1&0\\1&0&1&2\\\hline\end{array}
$$

$$
\begin{array}{|cccc|}\hline 1&1&1&1\\0&2&0&2\\1&1&1&1\\2&0&2&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&0&2&1\\1&2&0&1\\0&1&1&2\\2&1&1&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 2&0&0&2\\0&2&2&0\\2&0&0&2\\0&2&2&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&1&0&2\\1&1&2&0\\2&0&1&1\\0&2&1&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&0&1&2\\1&2&1&0\\1&0&1&2\\1&2&1&0\\\hline\end{array}
$$

$$
\begin{array}{|cccc|}\hline 0&2&2&0\\2&0&0&2\\0&2&2&0\\2&0&0&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&2&1&1\\2&0&1&1\\1&1&2&0\\1&1&0&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&2&1&1\\1&1&0&2\\1&1&2&0\\2&0&1&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&1&2&0\\1&1&0&2\\0&2&1&1\\2&0&1&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&2&0&1\\1&0&2&1\\2&1&1&0\\0&1&1&2\\\hline\end{array}
$$

$$
\begin{array}{|cccc|}\hline 2&0&1&1\\1&1&2&0\\1&1&0&2\\0&2&1&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 2&0&1&1\\0&2&1&1\\1&1&0&2\\1&1&2&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&1&2&0\\2&0&1&1\\0&2&1&1\\1&1&0&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&1&1&1\\2&0&2&0\\1&1&1&1\\0&2&0&2\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 1&0&2&1\\2&1&1&0\\0&1&1&2\\1&2&0&1\\\hline\end{array}
$$

$$
\begin{array}{|cccc|}\hline 2&0&2&0\\0&2&0&2\\0&2&0&2\\2&0&2&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&2&0&2\\1&1&1&1\\2&0&2&0\\1&1&1&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&1&1&2\\2&1&1&0\\1&0&2&1\\1&2&0&1\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&2&0&2\\0&2&0&2\\2&0&2&0\\2&0&2&0\\\hline\end{array}\quad
\begin{array}{|cccc|}\hline 0&1&1&2\\1&2&0&1\\1&0&2&1\\2&1&1&0\\\hline\end{array}
$$

$$\begin{array}{|cccc|}\hline 1&2&0&1\\0&1&1&2\\2&1&1&0\\1&0&2&1\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 2&0&2&0\\2&0&2&0\\0&2&0&2\\0&2&0&2\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 2&0&2&0\\1&1&1&1\\0&2&0&2\\1&1&1&1\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 0&2&0&2\\2&0&2&0\\2&0&2&0\\0&2&0&2\\\hline\end{array}$$

$$\begin{array}{|cccc|}\hline 0&0&2&2\\2&2&0&0\\0&0&2&2\\2&2&0&0\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 1&1&0&2\\0&2&1&1\\2&0&1&1\\1&1&2&0\\\hline\end{array}\quad\begin{array}{|cccc|}\hline 0&1&2&1\\2&1&0&1\\0&1&2&1\\2&1&0&1\\\hline\end{array}$$

Since the row sum of the completely fundamental 4×4 pandiagonal magic squares are all 2, the number of magic squares g_4 is a function of k, one-half the row sum r. Furthermore, we expect from the reciprocity relationship, $g_4(-k) = \bar{g}_4(k)$. Since there are no 4×4 magic squares with only strictly positive entries with row sum less than 4, we expect $g_4(-1) = 0$. Finally, using the corollary to the reciprocity theorem, $g_4(-2-k) = \pm g_4(k)$ for $k \geq 0$. Actually computing the number of magic squares with particular row sums using the above algorithm produces the following data:

Row Sum	Number
0	1
2	8
4	33
6	96
8	225
10	456
12	833
14	1408
16	2241
18	3400

Not nearly this many values were needed to calculate the polynomial $g_4(k)$ since the system of equations only had five independent variables. In fact, it turned out that the polynomial was only degree 4, which agrees with the upper bound after taking into account the restriction to a particular row sum as described earlier. Without further ado,

$$g_4(k) = \frac{1}{3}k^4 + \frac{4}{3}k^3 + \frac{8}{3}k^2 + \frac{8}{3}k + 1 = \frac{1}{3}(k+1)^2(k^2+2k+3)$$

is the number of 4×4 pandiagonal magic squares with row sum $2k$. Note that it indeed has a zero at -1 and satisfies the reciprocity relationship, $g_4(-2-k) = g_4(k)$.

The 5×5 case exercised more of the theory. On the one hand, no special handling was needed for the row sum since the row sum of the completely fundamental elements are 1. On the other hand, many more values are necessary to determine the polynomial. The system of equations has 16 dependent and 9 independent variables, giving a polynomial with maximum degree 8. This time however, the theory gives us the fact that $g_5(r)$ should have zeros at -1, -2, -3, and -4, in addition to the reciprocity relationship, $g_5(-5-k) = g_5(k)$. So only a few data points were needed, but the counting algorithm was efficient enough to give a lot of points to check with:

Row Sum	Number
0	1
1	10
2	55
3	220
4	715
5	2001
6	4995
7	11385
8	24090
9	47905
10	90376

This gives the formula for $g_5(r)$ which agrees with all of our predictions:

$$g_5(r) = \frac{1}{8064}(r+4)(r+3)(r+2)(r+1)(r^2+5r+8)(r^2+5r+42)$$

is the number of pandiagonal 5×5 magic squares with row sum r. Since this function takes on integer values for all integer inputs, $(r+4)(r+3)(r+2)(r+1)(r^2+5r+8)(r^2+5r+42)$ must always have 8064 as a factor. Without knowing that $g_5(r)$ is a counting function, this would be quite laborious to show.

When you have completely fundamental elements with components larger than one, you may not apply the above algorithm for counting. Having worked through some examples, I came up with the following:

Conjecture. *If no component of any completely fundamental element is greater than k, then every magic square is the nonnegative integer sum of elements of $CF(E)$ times factors of either* 1, $1/2$, $\ldots$, *or* $1/k$.

The program in Appendix C uses this algorithm when a completely fundamental element has a component larger than 1, though it gives a warning. Using this modified algorithm, I generated 3×3 simple magic squares. Using those results I generated $g_3(k) = 2k^2 + 2k + 1$ as the number of simple 3×3 magic squares with row sum $3k$. The reciprocity relation here is $g_3(-1-k) = g_3(k)$. I later used a more brute force algorithm (which is not dependent upon the conjecture) and verified the result. I further checked the conjecture when generating D_σ for the example monoid, providing a slightly more complicated case. I have so far been unable to find a proof.

3.7. Larger Magic Squares

When I first started finding the completely fundamental elements of pandiagonal magic squares, I made conjectures about what patterns existed between the different sized magic squares. After finishing four and five, I guessed the main issue to be whether the size of the square was prime. After running the computations for the 6×6 case, remainder modulus 4 seemed also to be an important factor. Examining the 7×7 has revealed that there is quite a lot of complexity to pandiagonal magic squares.

The 28 completely fundamental 7×7 pandiagonal magic squares which have row sum 1 are cyclic rotations of the following four completely fundamental elements:

$$\begin{array}{|ccccccc|}\hline 1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1\\ 0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0\\ \hline\end{array} \qquad \begin{array}{|ccccccc|}\hline 1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0\\ \hline\end{array}$$

1	0	0	0	0	0	0
0	0	0	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	1	0	0	0	0
0	0	0	0	0	0	1
0	0	0	1	0	0	0

1	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0

Without a definitive call from the program which generates the completely fundamental elements, these, the cyclic completely fundamental elements, were at first the only candidates for $CF(E)$. I did exhaustively show that these were the only 7×7 pandiagonal magic squares with row sum 1. I then used the program to combine them to see how many magic squares could be made from these. I quickly ran into memory limitations. After several revisions to the program, I generated values for row sums up to 5.

As a point of interest, not all row sum 1 pandiagonal prime order magic squares are cyclic, though exceptions are rare. Stern describes the conditions necessary to be able to find such a magic square in [6]. For instance, here is an 'irregular' 13×13 pandiagonal magic square with row sum 1:

1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0

I have exhaustively checked, and there are no pandiagonal 8×8 magic squares with row sum 1. I did however find one with row sum 2 (which is incidentally a completely fundamental element):

$$\begin{array}{|cccccccc|}\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \hline
\end{array}$$

To better experiment with magic squares, I wrote the last program of this thesis (see Appendix D) which uses brute force to directly find solutions to $\mathbf{\Phi}\boldsymbol{\alpha} = \mathbf{0}$ with a particular row sum.[9] This new program was used to verify $g_3(k)$ without using the conjecture. It also verified that the only 7×7 magic squares with row sum 1 are the cyclics shown above. It did however disagree with the number of 7×7 pandiagonal magic squares with row sum 2. Upon further inspection, I separated out 854 completely fundamental 7×7 pandiagonal magic squares which have row sum 2. They may all be formed by taking cyclic rotations of the following subset of the completely fundamental elements:

$$\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline
\end{array}
\quad
\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline
\end{array}
\quad
\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline
\end{array}$$

$$\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline
\end{array}
\quad
\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline
\end{array}
\quad
\begin{array}{|ccccccc|}\hline
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline
\end{array}$$

[9] For purposes of the program, a row sum is just the sum of some specified elements of the vector $\boldsymbol{\alpha}$.

$$\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&0&1&0&0\\0&1&0&0&1&0&0\\0&1&0&0&0&1&0\\1&0&0&0&0&1&0\\0&0&1&1&0&0&0\\0&0&0&1&0&0&1\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&0&1&0&0\\0&0&1&0&0&1&0\\0&1&0&0&0&1&0\\0&1&0&0&0&0&1\\0&0&0&1&1&0&0\\1&0&0&1&0&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&1&0&0&0\\0&0&0&0&1&1&0\\1&1&0&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&1&1&0&0\\0&0&1&0&0&0&1\end{matrix}}$$

$$\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&0&1&1&0&0\\0&1&1&0&0&0&0\\0&0&0&0&0&1&1\\0&1&0&0&0&1&0\\0&0&1&1&0&0&0\\1&0&0&0&1&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&0&1&1&0&0\\1&1&0&0&0&0&0\\0&0&0&1&0&1&0\\0&1&0&0&0&1&0\\0&0&1&0&0&0&1\\0&0&1&0&1&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&1&0&0&0\\0&0&0&0&0&1&1\\0&1&0&1&0&0&0\\0&1&0&0&0&1&0\\1&0&0&0&1&0&0\\0&0&1&0&1&0&0\end{matrix}}$$

$$\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&0&1&0&1&0\\0&1&1&0&0&0&0\\0&0&0&0&1&0&1\\0&1&0&0&0&1&0\\1&0&0&1&0&0&0\\0&0&1&0&1&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&1&0&1&0&0&0\\0&0&0&0&1&1&0\\1&0&1&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&1&0&0&1\\0&0&1&0&1&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&0&1&0&0\\0&0&0&0&1&0&1\\0&1&1&0&0&0&0\\1&0&0&0&0&1&0\\0&0&0&1&0&1&0\\0&1&0&1&0&0&0\end{matrix}}$$

$$\boxed{\begin{matrix}1&0&0&0&0&0&1\\0&0&1&0&1&0&0\\1&0&1&0&0&0&0\\0&0&0&0&1&1&0\\0&1&0&0&0&0&1\\0&1&0&1&0&0&0\\0&0&0&1&0&1&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&1&0\\0&0&1&0&1&0&0\\0&0&1&0&0&0&1\\0&1&0&0&0&1&0\\0&0&0&1&0&0&1\\0&1&0&0&1&0&0\\1&0&0&1&0&0&0\end{matrix}}\quad\boxed{\begin{matrix}1&0&0&0&0&1&0\\0&1&0&1&0&0&0\\0&0&0&1&0&0&1\\1&0&0&0&1&0&0\\0&0&1&0&0&0&1\\0&1&0&0&1&0&0\\0&0&1&0&0&1&0\end{matrix}}$$

Among other things, this means that the function $g_7(r)$ which counts them as a function of row sum is not a polynomial. Instead, it is a quasi-polynomial (see Stanley [5], section 4.4). I do not know if any of these completely fundamental elements have been found before, or if they may be used as the basis for constructing a noncyclic normal 7×7 pandiagonal magic square.

3.8. Concluding Remarks

In this thesis, I have computed the completely fundamental elements of pandiagonal magic squares of order up to six. A technique was presented to use this to compute the number of magic squares of given row sum. The function describing the number of pandiagonal magic squares given the row sum was determined for those same cases. The completely fundamental elements of 7×7 magic squares of row sum up to two have been given. The question of whether there are any 7×7 completely fundamental elements with row sum greater than two is still open. Included are several programs for analyzing systems of linear homogeneous diophantine equations. Best of all was the road getting here.

Bibliography

[1] Andrews, W. S. *Magic Squares and Cubes*. New York: Dover Publications, 1960.

[2] Benson, W. H. and O. Jacoby. *New Recreations with Magic Squares*. New York: Dover Publications, 1976.

[3] Spencer, J. "Counting Magic Squares." *American Mathematical Monthly* 87 (1980), pp. 397–399.

[4] Stafford, J. K. *Applications of Counting the Number of Solutions to Linear Equations*, Reed College Thesis, 1993.

[5] Stanley, R. P. *Enumerative Combinatorics*, Vol. I. Monterey, Calif.: Wadsworth & Brooks/Cole, 1986.

[6] Stern, E. "Number of Magic Squares Belonging to Certain Classes" (translated from the German by W. R. Transue) *American Mathematical Monthly* 46 (1939), pp. 555–581.

Appendix A: A small matrix programming library

```
/*
      mtrxlib.h
  This has all the declarations for the functions defined in mtrxlib.C.
  It also includes various other useful macros and inline functions
  common to findcfes.C and brutefrc.C.
*/
#define ptr(mn, xn, yn) ((mn)+(xn)+(yn)*wid)
#define elt(mn, xn, yn) (*((mn)+(xn)+(yn)*wid))

/* Auxiliary min/maximizing functions */
inline int min(int a, int b)
  {
    if (a<b)
      return a;
    return b;
  }

inline int max(int a, int b)
  {
    if (a>b)
      return a;
    return b;
  }


extern int wid;
extern int high;
extern int *m;

void swaprow(int y0, int y1);
void divrow(int r, int v);
void mulrow(int r, int v);
void subrow(int rt, int rf, int v);
void print(void);
int gcd(int a, int b);
void normalrow(int r);
void selectmatrix(int _wid, int _high, int *_m);
int reducematrix(int output);
int reducematrix(int output, int starting);
```

```
/*
      mtrxlib.C
  This file contains the definitions of the general purpose matrix
  functions.  Any speed improvements made here would speed up all of
  these programs.  Another such optimization would be to utilize an
  optimized linear algebra package, though this is complicated by the
  fact that integer results are required by the programs.  At one
  point, reducematrix was overloaded to include an alternate version
  if a second parameter was specified.  This new version assumed that
  the upper left n by n section of the matrix was already reduced to a
  diagonal matrix, avoiding some calculations.  This was used in a
  version of findcfes that maintained a cache of reduced matrices to
  improve reuse of calculations.
*/

#if defined(__TURBOC__)             /* Turbo/Borland C */
#include <mem.h>
#else
#ifdef __WATCOMC__                    /* Watcom C++ */
#include <mem.h>
#else                               /* Assume Unix */
#include <memory.h>
#endif
#endif
#include <iostream.h>
#include "mtrxlib.h"

int wid;
int high;
int *m;

/* swaprow
 *  Swaps rows <y0> and <y1> in the currently selected matrix
 *  (selected with selectmatrix).
 */
void swaprow(int y0, int y1)
  {
    int sz=wid;
    int *tmp=new int[sz];
    sz*=sizeof(int);
    memcpy(tmp, ptr(m, 0, y0), sz);
    memcpy(ptr(m, 0, y0), ptr(m, 0, y1), sz);
    memcpy(ptr(m, 0, y1), tmp, sz);
    delete tmp;
  }

/* divrow
 *  Divides every element of row <r> by <v>.
 */
```

```
void divrow(int r, int v)
  {
    int c;
    for (c=0; c<wid; ++c)
      elt(m, c, r)/=v;
  }

/* mulrow
 *  Multiplies every element of a row <r> by <v>.
 */
void mulrow(int r, int v)
  {
    int c;
    for (c=0; c<wid; ++c)
      elt(m, c, r)*=v;
  }

/* subrow
 *  Subtracts <v> times the elements of row <rf> from the elements of
 *  row <rt>.  I.e. row rt -= v * row rf.
 */
void subrow(int rt, int rf, int v)
  {
    int c;
    for (c=0; c<wid; ++c)
      elt(m, c, rt)-=v*elt(m, c, rf);
  }

/* print
 *  Displays the currently selected matrix to <cout>, the standard
 *  output.
 */
void print(void)
  {
    int x,y;

    for (y=0; y<high; ++y)
      {
        for (x=0; x<wid; ++x)
          cout << elt(m, x, y) << ' ';
        cout << '\n';
      }
  }

/* gcd
 *  Finds the greatest common divisor of two integers using the
 *  Euclidean Algorithm.
 */
int gcd(int a, int b)
```

```
  {
    int q, r;

    if (b==0)
      return a;
    else if (b<0)
      b=0-b;
    if (a<0)
      a=0-a;

    do
      {
        q=a/b;
        r=a-b*q;
        a=b;
        b=r;
      } while (r);

    return a;
  }

/* normalrow
 *  Takes the row <r>, factors out the greatest common divisor, and
 *  insures that the leading coefficient (the leftmost nonzero value)
 *  is positive
 */
void normalrow(int r)
  {
    int sgn=1, curelt, gcdv;
    for (curelt=0; curelt<wid; ++curelt)
      if (elt(m, curelt, r))
        break;
    if (curelt<wid)
      {
        gcdv=elt(m, curelt, r);
        if (gcdv<0)
          {
            sgn= -1;
            gcdv=0-gcdv;
          }
        for (++curelt; curelt<wid; ++curelt)
          gcdv=gcd(gcdv, elt(m, curelt, r));
        divrow(r, gcdv*sgn);
      }
  }

/* selectmatrix
 *  Selects a matrix to be manipulated by the functions in this
 *  module.
```

```
 */
void selectmatrix(int _wid, int _high, int *_m)
  {
    wid=_wid;
    high=_high;
    m=_m;
  }

/* reducematrix
 *  If (output) then error messages will be sent to the screen.
 *  Returns 0 if all went well, 1 if there were linearly dependent
 *  rows, 2 if a column is all zeros (stops process), 3 if both.
 *  If condition is 2 or 3, then the high bits store the
 *  column number.
 */
int reducematrix(int output)
  {
    int x,y;
    int result=0;
    int mn=high;
    if (wid<mn)
      mn=wid;
    for (x=0; x<mn; ++x)
      { /* Put first row of {x, ..., mn-1} nonzero elt in x */
        /* column, in row x                                  */
        for (y=x; y<high; ++y)
          if (elt(m, x, y))
            break;
        if (y<high)
          {
            if (x!=y)
              swaprow(x,y);
            normalrow(x);
            int leadval=elt(m, x, x);
            for (++y; y<high; ++y)
              { /* Combine row x with row y to eliminate xth */
                /* column from row y                         */
                int yleadval=elt(m, x, y);
                if (yleadval)
                  {
                    mulrow(y, leadval);
                    subrow(y, x, yleadval);
                  }
              }
          }
        else
          {
            int t;
            for (t=0; t<wid; ++t)
```

```
          if (elt(m, t, high-1))
            break;
        if (t==wid)
          {
            if (output)
              {
                cout << "One of the equations is linearly "
                     << "dependent upon the others in the system."
                     << endl
                     << "Eliminating and continuing..."
                     << endl;
              }
            result=1;
            --high;
            if (high<mn)
              mn=high;
            --x;
          }
        else
          {
            if (output)
              cout << "The system of equations is independent "
                   << "of column " << x+1 << "." << endl;
            return 2+result+(x<<2);
          }
      }
    }
  /* Lower left triangle is now filled with 0s */
  for (x=1; x<high; ++x)
    {
      int leadval=elt(m, x, x);
      for (y=0; y<x; ++y)
        {
          int yleadval=elt(m, x, y);
          if (yleadval)
            {
              mulrow(y, leadval);
              subrow(y, x, yleadval);
            }
        }
    }
  for (y=0; y<high; ++y)
    normalrow(y);
  return result;
}
```

Appendix B: Finding the set CF(E)

```
/*
      findcfes.C
  This program finds the completely fundamental elements of a system
  of equations using the technique described in the text.  One version
  of this program used a matrix cache, which kept track of
  reduction of matrices already performed, allowing the results to be
  reused.  There was also a version that printed out its progress and
  could be started at any point.  Special note: if the system only has
  a trivial completely fundamental element (i.e., one with no
  components equal zero), then this program will give no results.
  This trivial vector (usually the vector of all ones) has full
  support so it cannot satisfy alpha sub i equal zero for any i.
*/
#include <stdio.h>
#if defined(__TURBOC__)             /* Turbo/Borland C */
#include <mem.h>
#else
#ifdef __WATCOMC__                  /* Watcom C++ */
#include <mem.h>
#else                               /* Assume Unix */
#include <memory.h>
#endif
#endif
#include <iostream.h>
#include <stdlib.h>
#include "mtrxlib.h"

#define swap(a1, a2)  ((a1)^=(a2), (a2)^=(a1), (a1)^=(a2))

struct solnlist
  {
    int *soln;
    solnlist *next;
  };

int main(void)
  {
    int x,y;
    int w,h;
    int *m;

    cout << "Enter width of matrix: ";
    cin >> w;
    cout << "Enter height of matrix: ";
    cin >> h;
    cout << "Enter matrix values:\n";
```

```
m=new int[w*h];
selectmatrix(w, h, m);

for (y=0; y<h; ++y)
  {
    for (x=0; x<w; ++x)
      cin >> elt(m, x, y);
  }

if (reducematrix(1)&2)
  return 2;
w=wid;
h=high;

cout << "Reduced matrix:\n";
print();

/* Now we have <h> =0 restrictions, and <w> >=0 restrictions
 * so form the <w-h> dimensional vector space that satisfies
 * the <h> =0 restrictions.  In this space, any subset of <w-h-1>
 * elements of the set of >=0 restrictions define an extremal ray.  Of
 * course, only the first <h> of the >=0 restrictions are really
 * interesting.
 * Example
 * a  -c+d  =0      a>=0   b>=0   c>=0   d>=0
 *   b+c-2d =0
 * Equivalent to ignoring a and b with:
 * c-d>=0            (a>=0)
 * -c+2d>=0          (b>=0)
 * c>=0
 * d>=0
 * Now each 1 (=4-2-1) of these defines an extremal ray:
 * c=d    c=2d    c=0   d=0
 * For each of which there is at most one CFE:
 * <1,1>  <2,1>   <0,1> <1,0>
 * which translate back to (including a and b):
 * <0,1,1,1>  <1,0,2,1>  <-1,2,0,1>  <1,-1,1,0>
 * So the last two are not in the set (we could have seen this by
 * plugging back into our >=0 conditions with just c and d), so CFE =
 * { <0,1,1,1>, <1,0,2,1> }
 */

int numsimul=w-h-1;
if (numsimul<1)
  {
    cout << "Trivial system, reduced matrix describes only CFE.\n";
    return 1;
  }
int *indx=new int[numsimul];
```

```
/* The following is equivalent to
 *  for (indx[0]=0; indx[0]<h+2; ++indx[0])
 *    for (indx[1]=indx[0]+1; indx[1]<h+3; ++indx[1])
 *      for (indx[2]=indx[1]+1; indx[2]<h+4; ++indx[2])
 *         ...
 *        for (indx[numsimul-1]=indx[numsimul-2]+1;
 *             indx[numsimul-1]<w; ++indx[numsimul-1])
 *          {  ...
 *          }
 * In effect getting every numsimul subset of the numbers 0, ..., w-1
 * in order in indx[0], ..., indx[numsimul-1]
 */

int i, j;
for (i=0; i<numsimul; ++i)    /* Initialize each indx */
  indx[i]=i;
int ovrflw=0;
int *nm=new int[numsimul*(numsimul+1)];
solnlist *sl=0, *cur;
while (!ovrflw)               /* Until done counting indx's ... */
  {
    int res, swapped, last;
    int *soln=0;
    /* For each subset of 0, ..., w-1                        */
    /* Form a matrix with numsimul rows, numsimul+1 columns */

    selectmatrix(numsimul+1, numsimul, nm);
    for (i=0; i<numsimul && indx[i]<h; ++i)
      {   /* For the interesting equations... */
        int *tmp=m+h+indx[i]*w;
        int *tmp2=ptr(nm, 0, i);
        for (j=0; j<w-h; ++j, ++tmp, ++tmp2)
          *tmp2=*tmp;
      }

    for (; i<numsimul; ++i)
      {   /* For the rest, put an equations of the form var=0 */
        for (j=0; j<=numsimul; ++j)
          elt(nm, j, i)=0;
        elt(nm, indx[i]-h, i)=1;
      }

    res=reducematrix(0);
    swapped=numsimul;
    if (res)
      {
        if ((res&3)==2)
          {         /* Case like d=0 above where the equations are */
```

```
                /*  independent of one of the first numsimul   */
                /*  variables */
        swapped=res>>2;
        /* Swap columns so independent is lastvar */
        for (i=0; i<numsimul; ++i)
          swap(elt(nm, swapped, i), elt(nm, numsimul, i));
        res=reducematrix(0);
        if (res)
                /* Swapping didn't work, i.e., the current  */
                /*  system does not describe an extreme ray */
          goto nosoln;
      }
    else
      goto nosoln;
  }

/* Form solution with least nonnegative integer components */
soln=new int[w];
last=elt(nm,0,0);
for (i=1; i<numsimul; ++i)
  last=last*elt(nm,i,i)/gcd(last, elt(nm,i,i));
last=abs(last);
soln[w-1]=last;
for (i=h; i<w-1; ++i)
  {
    int denom=elt(nm, i-h, i-h);
    if (denom==0)
      {
        soln[i]=1;  /* Independent variable, shouldn't happen */
        cerr << "Unreachable code error 1" << endl;
      }
    else
      {
        soln[i]=(int) (((long) -elt(nm, numsimul,
                 i-h)*last)/denom);
        if (soln[i]<0)
          goto nosoln;
      }
  }
if (swapped<numsimul)
  /* Undo swap, if any, so labeling for variables consistent */
  /* within program and output                               */
  swap(soln[swapped+h], soln[w-1]);

selectmatrix(w, h, m);
for (i=0; i<h; ++i)
  { /* First h variables are linearly dependent upon last w-h */
    /* vars */
    soln[i]=0;
```

```
            for (j=h; j<w; ++j)
              soln[i]-=elt(m, j, i)*soln[j];
            int e=elt(m, i, i);
            int g=gcd(soln[i], e);
            if (g!=abs(e))
              {
                int multfact=abs(e)/g;
                for (j=0; j<w; ++j)
                  soln[j]*=multfact;
              }
            soln[i]/=e;
            if (soln[i]<0)
              goto nosoln;
          }
        /* Check if solution is nonzero */
        for (i=0; i<w; ++i)
          if (soln[i]!=0)
            break;
        if (i<w)
          {  /* Yes! Solution is nonzero, is this a new solution? */
            cur=sl;
            while (cur)
              {
                if (memcmp(cur->soln, soln, w*sizeof(int))==0)
                  break;
                cur=cur->next;
              }
            if (!cur)
              { /* Solution is different than previous solutions */
                cur=new solnlist;      /* Add it to the list of found */
                                       /* solutions                   */
                cur->next=sl;
                cur->soln=soln;
                soln=0;
                sl=cur;
                cout << "Found CFE: ";  /* And report it to the user */
                for (i=0; i<w; ++i)
                  {
                    cout << cur->soln[i];
                    if (i<w-1)
                      cout << ' ';
                  }
                cout << endl;
              }
          }
nosoln: /* Clean up.  Jumped here if no solution. */
        if (soln)
          delete soln;
        int ovrnum=w-1;         /* Increment indx's */
```

```
      ovrflw=1;
      for (i=numsimul-1; i>=0 && ovrflw; --i)
        {
          ++indx[i];
          if (indx[i]<=ovrnum)
            {
              ovrflw=0;
              ovrnum=indx[i]+1;
              for (j=i+1; j<numsimul; ++j, ++ovrnum)
                indx[j]=ovrnum;
            }
          --ovrnum;
        }
    }
  /* Free list of solutions */
  cur=sl;
  while (cur)
    {
      solnlist *Next=cur->next;
      delete cur;
      cur=Next;
    }
  /* Free matrices */
  delete nm;
  delete m;

  return 0;
}
```

Appendix C: Combining magic squares

```
/*
      combcfes.C
  This program finds all possible nonnegative linear combinations of k
  members of a set of vectors.  This algorithm's major constraint is
  memory usage.  In various versions of this program, different
  strategies were used to minimize memory usage (not included here
  since they make the program less general and less readable).  If you
  assume that no component will every exceed 15, then the saved magic
  squares may be packed into four bits for each component.  Another
  option which I did not use would be to not save the dependent
  variables.  This could give as much as a factor of two reduction in
  memory requirements, but would require additional input from the
  user.  An earlier version of the program used linked lists, but they
  used more memory and tended to scatter the data.  This prevented
  memory cache hits and could cause thrashing of the program.  Thanks
  to Mark Jefferys for pointing out that realloc was often reasonably
  fast on systems where logical memory may be freely remapped to allow
  arrays to grow dynamically.
*/

#include <stdio.h>
#if defined(__TURBOC__)            /* Turbo/Borland C */
#include <mem.h>
#else
#ifdef __WATCOMC__                   /* Watcom C++ */
#include <mem.h>
#else
#include <memory.h>
#endif
#endif
#include <iostream.h>
#include <stdlib.h>

int wid;

#define ptr(mn, xn, yn) ((mn)+(xn)+(yn)*wid)
#define elt(mn, xn, yn) (*((mn)+(xn)+(yn)*wid))

/* ------------------- List management ------------------- */
/* These functions manipulate the list of solutions found  */
/* by the program.  The list is stored as a hash table     */
/* of sublists of items with the same hash value.  The     */
/* hash is used to speed up searching.  In general, at     */
/* most (number of solutions)/(HashSize) items will have   */
/* to be checked to determine that a particular item is    */
/* in the list.  Other methods which are O(log n) might be */
/* used, but this uses a minimum of additional memory and  */
```

```
/* reduces search time to a small constant for all but     */
/* very large datasets (where you have the option of       */
/* increasing the hash size).                              */

#define HashSize 65536
#define HashMask 65535

char *HashTable[HashSize];

/* InitList
 *  Initializes the list to empty.  The list is actually composed of
 *  <HashSize> sublists, one for each possible hash value.
 */
void InitList()
  {
    memset(HashTable, 0, HashSize*sizeof(char *));
  }

/* AddToList
 *  Scans the list of solutions; if <item> is in it, the function
 *  returns 0.  Otherwise <item> is added to the list and the function
 *  returns 1.
 */
int AddToList(char *item)
  {
    /* Calculate hash value */
    int hashv=0;  /* Arbitrary */
    int i;
    int shl=0;
    int notmask=!HashMask;
    for (i=0; i<wid; ++i, ++shl)
      {
        int vl=item[i];
        if ((vl<<shl)&notmask && shl)
          shl=0;
        hashv|=(vl<<shl)&HashMask;
      }
    /* Hash value calculated, look at corresponding sublist */
    int *Arr=(int *) HashTable[hashv];
    int size=0, allocd=0;
    if (Arr)
      { /* If there is already a list of items with this hash value,... */
        /* See how big it is, */
        size=*Arr;
        ++Arr;
        allocd=*Arr;
        ++Arr;
        int cnt;
        /* And search it */
```

```
        char *ArrChr=(char *) Arr;
        for (cnt=0; cnt<size; ++cnt, ArrChr+=wid)
          { /* Found equal item, return 0 */
            if (memcmp(ArrChr, item, wid*sizeof(char))==0)
              return 0;
          }
        /* New item to be added to the list, see if there is room
           already allocated */
        if (size>=allocd)
          { /* Allocate some more space so we do not have to keep */
            /* realloc-ing                                        */
            allocd+=17;
            if ((HashTable[hashv]=(char *) realloc(HashTable[hashv],
                allocd*wid*sizeof(char)+2*sizeof(int)))==0)
             {
               cerr << "Memory allocation error\n";
               exit(1);
             }
            ((int *) HashTable[hashv])[1]=allocd;
          }
      }
    else
      { /* This is the first item with this hash value, allocate and */
        /* initialize a list for this item.                          */
        if ((HashTable[hashv]=(char *) malloc(wid*sizeof(char)
                +2*sizeof(int))) == 0)
          {
            cerr << "Memory allocation error\n";
            exit(1);
          }
        ((int *) HashTable[hashv])[1]=allocd=1;
      }
    /* Write the item to the list for this hash value */
    memcpy(HashTable[hashv]+size*wid+2*sizeof(int), item,
           wid*sizeof(char));
    ++size;
    ((int *)HashTable[hashv])[0]=size;
    /* Return that this was a new item */
    return 1;
  }

/* ListCount
 *  Returns the number of items in the list.
 */
unsigned long ListCount(void)
  {
    unsigned long totnum=0;
    int i;
    for (i=0; i<HashSize; ++i)
```

```
      {
        int *Curr=(int *)HashTable[i];
        if (Curr)
          totnum+=(long) *Curr;
      }
    return totnum;
  }

/* FreeList
 *  Deallocates space used by the list structure
 */
void FreeList(void)
  {
    int i;
    for (i=0; i<HashSize; ++i)
      {
        char *Curr=HashTable[i];
        if (Curr)
           free(Curr);
      }
  }

/* ------------------ The actual program ------------------ */
int main(void)
  {
    cout << "Enter width of CFEs: ";
    cin >> wid;

    int nCFE;
    char *CFEs;

    cout << "Enter number of CFEs: ";
    cin >> nCFE;

    cout << "Enter each CFE:\n";

    // Each CFE has wid values
    CFEs=new char[nCFE*wid];

    int mx=0;
    int x,y;

    for (y=0; y<nCFE; ++y)
      {
        for (x=0; x<wid; ++x)
          {
            int tmp;
            cin >> tmp;
            elt(CFEs, x, y)=(char) tmp;
```

```
        if (tmp>mx)
          mx=tmp;
      }
  }

int total;

cout << "Enter total number of CFEs to combine: ";
cin >> total;
if (total<1)
  {
    cout << "Don't be ridiculous.";
    return 1;
  }
cout << "Note, denominators of up to " << mx <<
  " may appear in the combination.\n";

InitList();
int totmul;
/* The following loop could be optimized to take into account that
 * if <totmul>= j times k is calculated, then you do not have to
 * compute <totmul>=j
 */
for (totmul=1; totmul<=mx; ++totmul)
  {
    int eftot=totmul*total;
    /* The following is equivalent to
     *  for (indx[0]=0; indx[0]<nCFE; ++indx[0])
     *    for (indx[1]=indx[0]; indx[1]<nCFE; ++indx[1])
     *      for (indx[2]=indx[1]; indx[2]<nCFE; ++indx[2])
     *         ...
     *        for (indx[eftot-1]=indx[eftot-2];
     *             indx[eftot-1]<nCFE; ++indx[eftot-1])
     *          {  ...
     *          }
     * In effect getting every <eftot> multichoose of the numbers
     * 0, ..., nCFE-1, in order in indx[0], ..., indx[eftot-1]
     */
    int *indx=new int[eftot];
    int i, j;
    for (i=0; i<eftot; ++i)    /* Initialize each indx */
      indx[i]=0;
    int ovrflw=0;
    while (!ovrflw)            /* Until done counting indx's ... */
      {
        char *result=new char[wid];
        memset(result, 0, wid*sizeof(char));
        for (i=0; i<eftot; ++i)
          {
```

```
              char *p1=result;
              char *p2=ptr(CFEs, 0, indx[i]);
              for (j=0; j<wid; ++j, ++p1, ++p2)
                *p1+=*p2;
            }

          if (totmul!=1)
            { /* Check if all elements are multiples of <eftot> */
              for (i=0; i<wid; ++i)
                if (result[i]%totmul!=0)
                  goto nosoln;
              for (i=0; i<wid; ++i)
                result[i]/=totmul;
            }

          AddToList(result);

nosoln:
          if (result)
            delete result;
          /* Increment indx's */
          ovrflw=1;
          for (i=eftot-1; i>=0 && ovrflw; --i)
            {
              ++indx[i];
              if (indx[i]<nCFE)
                {
                  ovrflw=0;
                  int tmp=indx[i];
                  for (j=i+1; j<eftot; ++j)
                    indx[j]=tmp;
                }
            }
        }
      delete indx;
    }

    cout << "Total number of solutions: " << ListCount() << "\n";
    FreeList();

    delete CFEs;

    return 0;
  }
```

Appendix D: Counting magic squares, brute force

```
/*
      brutefrc.C
  This employs a brute force algorithm of stepping through all
  possible values of the independent variables that could give a magic
  square of particular row sum.  A human trying to perform this same
  task rules out large classes of magic squares quickly, but there is
  no straightforward algorithm for determining good constraints on
  the independent variables given a system of equations and the
  definition of row sum.  In order to capitalize on the potential gain
  from human intelligence, the input data was expanded to include
  several rules for determining upper bounds on the free variables.
  Rules are of the form:
      Let variable k go from 0 to the magic number minus the maximum of:
          the sum of the elements x sub a through x sub c
          the sum of the elements x sub d through x sub f
              ...
          the sum of the elements x sub v through x sub z.
      In the input data, this is represented:
          k>xa+xb+xc,xd+xe+xf, ... ,xv+xw+xx+xy+xz
      where k, xa, xb, ..., xz are element indices.
  This added bit of intelligence improved the performance
  dramatically, though it complicated the process of creating an input
  file.  An additional speed gain would be to perform a preprocessing
  step to store the delta change in the matrix for increasing each
  individual free variable.  Combined with a faster matrix addition
  routine, the matrix could be kept current by adding matrix deltas as
  the free components are increments.  Some additional complication
  would be necessary to only perform this step after the increment on
  the free variables is known, but this could dramatically improve the
  performance of the algorithm for large magic squares.
*/
#include <stdio.h>
#if defined(__TURBOC__)             /* Turbo/Borland C */
#include <mem.h>
#else
#ifdef __WATCOMC__                  /* Watcom C++ */
#include <mem.h>
#else                               /* Assume Unix */
#include <memory.h>
#endif
#endif
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>
#include "mtrxlib.h"

/* Represents the sum of a set of components */
```

```
struct param
  {
    int numsummands;
    int *summands;
  };

/* Represents a rule for an iterator; take the magic number and
 * subtract the max of the <pars> items.
 */
struct iterator
  {
    int varnum;
    int numparams;
    param *pars;
  };

/* magicsum
 *  Calculates the magic sum of a vector, given the list of element
 *  indices of one row.
 */
int magicsum(int mnum, int *indx, int numelts, int *elts)
  {
    int tot=0;
    int i;
    for (i=0; i<numelts; ++i)
      tot+=indx[elts[i]];
    return (tot==mnum);
  }

/* increment
 *  Increments the iterators once, obeying the iterator rules.  The
 *  result does not necessarily have the right magic sum as evaluated
 *  above.
 */
int increment(int mnum, int numindx, int *indx, iterator *itrs)
  {
    int i, j, k, l;
    int overflow=1;
    for (i=numindx-1; i>=0 && overflow; --i)
      {
        overflow=0;
        j=itrs[i].varnum;
        ++indx[j];
        int mx=0;
        for (k=0; k<itrs[i].numparams; ++k)
          {
            int cur=0;
            for (l=0; l<itrs[i].pars[k].numsummands; ++l)
              cur+=indx[itrs[i].pars[k].summands[l]];
```

```
          if (cur>mx)
            mx=cur;
        }
      if (indx[j]>mnum-mx)
        {
          indx[j]=0;
          overflow=1;
        }
    }
  return overflow;
}

/* ------------------- The actual program  ------------------- */
int main(void)
  {
    int x,y;
    int w,h;
    int *m;

    /* Get the matrix representing the system of equations */
    cout << "Enter width of matrix: ";
    cin >> w;
    cout << "Enter height of matrix: ";
    cin >> h;
    cout << "Enter matrix values:\n";

    m=new int[w*h];
    selectmatrix(w, h, m);

    for (y=0; y<h; ++y)
      {
        for (x=0; x<w; ++x)
          cin >> elt(m, x, y);
      }

    /* Reduce it and print it out */
    if (reducematrix(1)&2)
      return 2;
    w=wid;
    h=high;

    cout << "Reduced matrix:\n";
    print();

    /* Get the magic number/row sum */
    int magic;
    cout << "Input magic number: ";
    cin >> magic;
```

```
/*
 * Now we have <h> dependent variables, and <w>-<h> independent
 *  variables.  So input the rules describing how we are going to
 *  step through the possibilities for each independent variable.
 */

int indp=w-h;
iterator *its=new iterator[indp];
char inputstring[256];
cout << "Input iterator expressions: ";
int i, j, k;
for (i=0; i<indp; ++i)
  {
    cin.width(sizeof(inputstring));
    cin >> inputstring;
    if ((its[i].varnum=atoi(inputstring))>=h+1)
      its[i].varnum-=h+1;
    else
      {
        cerr << "Error in variable number" << endl;
        exit(1);
      }
    char *curpos=inputstring;
    while (isdigit(*curpos) && *curpos)
      ++curpos;
    if (*curpos)
      {
        if (*curpos!='>')
          cerr << "Warning, variable number should be followed by '>'"
            << endl;
        ++curpos;
        int numcommas=0;
        char *scan; // Count number of commas
        for (scan=curpos; *scan; ++scan)
          if (*scan==',')
            ++numcommas;
        its[i].numparams=numcommas+1;
        its[i].pars=new param[numcommas+1];
        for (j=0; j<numcommas+1; ++j)
          {
            int numplusses=0;
            for (scan=curpos; *scan && *scan!=','; ++scan)
              if (*scan=='+')
                ++numplusses;
            its[i].pars[j].numsummands=numplusses+1;
            its[i].pars[j].summands=new int[numplusses+1];
            for (k=0; k<numplusses+1; ++k)
              {
                if ((its[i].pars[j].summands[k]=atoi(curpos))>=h+1)
```

```
                  its[i].pars[j].summands[k]-=h+1;
                else
                  {
                    cerr << "Invalid summand." << endl;
                    exit(1);
                  }
                while (*curpos && *curpos!=',' && *curpos!='+')
                  ++curpos;
                if (*curpos)
                  ++curpos;
              }
          }
      }
    else
      {
        its[i].numparams=0;
        its[i].pars=0;
      }
  }

/*
 * Now <its> structure filled out.  Get information on a row
 * of the magic square from the independent variables
 */
int rowlen, *magicrow;
cout << "Enter the number of elements in a row of the square: ";
cin >> rowlen;
if (rowlen<1)
  {
    cerr << "Invalid row length value" << endl;
    exit(1);
  }
magicrow=new int[rowlen];
cout << "Enter the indices of independent variables in one row: ";
for (i=0; i<rowlen; ++i)
  {
    cin >> magicrow[i];
    magicrow[i]-=h+1;
    if (magicrow[i]<0)
      {
        cerr << "Invalid magic row value" << endl;
        exit(1);
      }
  }

/*
 * <indx> contains values of
 * of the <w>-<h> independent variables
 */
```

```
    int *indx=new int[indp];
    memset(indx, 0, sizeof(int)*indp);  /* Start off at all 0s */

    int overflow=0;
    while (!overflow && !magicsum(magic, indx, rowlen, magicrow))
      overflow=increment(magic, indp, indx, its);

    unsigned long numsolns=0;

    cout << endl;

    while (!overflow)
      { // check if values in <indx> are valid
        int *soln=new int[w];
        memcpy(soln+h, indx, (w-h)*sizeof(int)); /* Initialize soln */

        selectmatrix(w, h, m);
        for (i=0; i<h; ++i)
          { /* First h variables are linearly dependent upon last */
            /* w-h vars                                          */
            soln[i]=0;
            for (j=h; j<w; ++j)
              soln[i]-=elt(m, j, i)*soln[j];
            int e=elt(m, i, i);
            j=soln[i]/e;
            if (e*j!=soln[i] || j<0)
              goto nosoln;
            soln[i]=j;
          }
        for (i=0; i<w; ++i)
          if (soln[i]!=0)
            break;
        if (i<w)
          {  /* Solution is nonzero */
            cout << "Found solution: (";
            for (i=0; i<w; ++i)
              {
                cout << soln[i];
                if (i<w-1)
                  cout << ", ";
              }
            cout << ")\n";
            ++numsolns;
          }

nosoln:
        if (soln)
          {
```

```
          delete soln;
          soln=0;
        }
      do
        overflow=increment(magic, indp, indx, its);
      while (!overflow && !magicsum(magic, indx, rowlen, magicrow));
    }
  cout << "Number of solutions: " << numsolns << endl;

  /* Free <its> structure */
  for (i=0; i<indp; ++i)
    { // free its[i]
      for (j=0; j<its[i].numparams; ++j)
        { // free its[i].pars[j]
          delete its[i].pars[j].summands;
        }
      delete its[i].pars;
    }
  delete its;

  /* Free other structures */
  delete m;
  delete magicrow;
  delete indx;

  return 0;
}
```