This is to certify that the

dissertation entitled

THE RING OF CYCLOTOMIC INTEGERS OF MODULUS THIRTEEN IS NORM-EUCLIDEAN

presented by

Robert George McKenzie

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Mathematics

Major professor

Date July 29, 1988

0-12771

THE RING OF CYCLOTOMIC INTEGERS OF MODULUS THIRTEEN IS NORM-EUCLIDEAN

By

Robert George McKenzie

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mathematics

1988

ABSTRACT

THE RING OF CYCLOTOMIC INTEGERS OF MODULUS THIRTEEN IS NORM-EUCLIDEAN

By

Robert George McKenzie

In this dissertation the ring of integers of the thirteenth cyclotomic field is shown to be euclidean with respect to the field norm. The basic method, in outline, is to cover the fundamental cell of the quotient field by numerous small subregions. Then, for each subregion, an integral point q is found such that $\text{Norm}(x-q) < 1$ for all points x in the subregion. The generation of subregions, the finding of integral points, and the bounding of the Norm were all done by a FORTRAN program written for an electronic computer.

This dissertation is dedicated to

Janice Marie Szur,

who is both necessary and sufficient.

ACKNOWLEDGMENTS

I would like to acknowledge the contributions of the following people to this dissertation. Firstly, my dissertation advisor, Charles MacCluer, who taught me number theory. Secondly, my principal instructors in the many branches of Algebra: Jonathan Hall, Richard Phillips, William Brown, and Andrew Bailey. Thirdly, my principal instructors in other branches of Mathematics: Sheldon Axler, Bruce Sagan, Peter Lappan, Steve Dragosh, Charles Seebeck, and Donna Carr. Next, those colleagues whose presence and efforts in seminars and committee have aided me: John Wolfskill, Capi Corrales, William Sledd, Lee Sonneborn, Edward Ingraham, Susan Schuur, Wei Kuan, Joseph Brennan, and Timothy Sauer.

I owe a particular debt of gratitude to H. W. Lenstra, Jr., whose work has been an inspiration, and whose prompt and courteous advice is deeply appreciated. Also, T. Ojala, M. Ram Murty, and Don Lewis have been helpful both in their work and elsewhere.

The faculty and staff of the Mathematics Department at Michigan State have been invariably kind and helpful. Worthy of special thanks are the Chairmen, past and present, Joseph Adney and Kyung Whan Kwun, and the Associate Chairman, Douglas Hall. Also, Judith Miller, Barbara Miller, Berle Reiter, Velma DeMyers, and Sterling Tryon-Hartwig have exerted themselves on my behalf above and beyond the call of duty.

Finally, I would like to thank all of my fellow graduate students. Of particular note are: Richard Hensh, Paul Hewitt, Michael Vernon, José Geraldo, Kathy McKeon, John Long, Martin Bredeck, Kathy Dempsy, Richard Reynolds, and Joseph Spencer.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1

A Historical Perspective on the Euclidean Problem

§1 Euclidean Rings and the First Question.

Over two thousand years ago Euclid gave an algorithmic procedure (still used today) to compute the greatest common divisor of two positive integers [Eu, Book VII, Propositions 1-3]. This procedure is called **Euclid's algorithm** and can be applied to rings other than the integers. A ring for which Euclid's algorithm will give a greatest common divisor of two elements is thus known as a euclidean ring. The precise conditions needed to enable Euclid's algorithm to work are given in the following definition.

Definition 1.1 A ring R is said to be a **euclidean** ring, or **euclidean with respect to the algorithm** ψ, if there exists a well ordered set W and a mapping $\psi: R \to W$, called a **euclidean algorithm**, such that for each pair of elements $a, b \in R$, with $b \neq 0$, there exists a pair of elements $q, r \in R$ satisfying the following two properties:

$$a = bq + r \tag{1.2}$$

$$\psi(r) < \psi(b). \tag{1.3}$$

Thus there are three interrelated concepts. (i) A euclidean ring is a ring satisfying Definition 1.1. (ii) Euclid's algorithm is a process that can be used on a euclidean ring to find greatest common divisors. (iii) A euclidean algorithm is a mapping satisfying (1.2) and (1.3) above. A ring is euclidean when a euclidean algorithm exists for it, and in this case, Euclid's algorithm can be used (in conjunction with its euclidean algorithm) to find

greatest common divisors. Note that the ring Z of integers is euclidean with respect to the usual absolute value $\psi(x) = |x|$.

The following natural question can be asked about any ring.

Question 1.4. Is R euclidean? This will be called the **euclidean** question.

Note that this is a question about the existence of an algorithm and does not require the exhibition of any specific algorithm satisfying (1.2) and (1.3).

§2 Number Fields and Two Further Questions.

Because Z is the ring of integers in the algebraic number field Q it is natural to pose the euclidean question for the ring of integers of an arbitrary number field. Because a number field is the quotient field of its ring of integers and because it is often difficult to describe the ring of integers simply, number theorists speak of the number field when they mean its ring of integers. We will use this abuse of language and speak of a number field as being or not being euclidean when we mean that its ring of integers is or is not euclidean.

Henceforth we will let K be an arbitrary number field and R be its ring of integers -- that is, R consists of those elements of K that satisfy a monic polynomial with coefficients from Z. A natural generalization to R of the absolute value in Z is the absolute value of the field norm from K to Q. Thus we have the following definition and question.

Definition 1.5. Let K be a number field with ring of integers R. If R is euclidean with respect to $\psi(x) = |\mathrm{Norm}_{K/Q}(x)|$ we say that K is **norm-euclidean.**

Question 1.6. Is K norm-euclidean? This will be called the **norm-euclidean** question.

Note that the norm-euclidean question is much more detailed than the euclidean question, for we are asking that the ring be euclidean with respect to a very specific algorithm. Of course, if a field is norm-euclidean it is necessarily euclidean. The converse is an interesting open question: are there fields that are euclidean but not norm-euclidean?

It is immediate that every euclidean ring is principal. Thus a euclidean number field necessarily has class number one. The following definition will make this more precise.

Definition 1.7. Let K be a number field and I be the group of non-zero fractional ideals of K. Let P be the group of non-zero principal fractional ideals of K. The quotient group I/P is known as the **class group**. The order of I/P is known as the **class number** of K.

For more about fractional ideals and the class number see [Cu, Chapter III]. The class number of a number field is finite [Cu, Theorem 20.6], and the field has class number one if and only if the ring of integers is principal. That is, in a number field K, it is the class number that measures whether or not R is principal. This leads us to the final question we will ask about a number field.

Question 1.8. Does K have class number one? This will be called the **class number** question.

This question is the weakest of the three, for a norm-euclidean field is necessarily euclidean, and a euclidean field necessarily has class number one. There are class number one fields that are not euclidean, for example $Q(\sqrt{-19})$ [Mo].

For a given number field K, the comparative level of difficulty of these questions is as follows. The class number of K is reasonably easy to compute. When K has class number one, whether K is norm-euclidean is a more difficult matter to settle. Lastly, if K has class number one and is not norm-euclidean, it is extremely difficult to determine whether K is euclidean.

§3 Quadratic Fields

The above questions have been considered for many families of number fields. The family of fields quadratic over Q has been investigated successfully. For this section d will be a square free integer, K will be the quadratic field $Q(\sqrt{d})$, and R will be the ring of integers of K. The cases with $d > 0$ and $d < 0$ are different and are dealt with separately.

Let $K = Q(\sqrt{d})$, where d is negative. K is called an **imaginary quadratic** field. All three questions: the euclidean question; the norm-euclidean question; and the class number question, have been settled for all of the imaginary quadratic fields.

It is a classical result that for imaginary quadratic fields K, the class number is one if d is one of the following nine integers: $d = -1, -2, -3, -7, -11, -19, -43, -67, -163$. K. F. Gauss conjectured that these are the only examples [Co, page 151]. This conjecture was finally shown to be correct in 1967 by H. Stark [S]. Thus these are the only possible imaginary quadratic candidates for euclidean and norm-euclidean fields.

Historically, the norm-euclidean question and the euclidean question were answered for the imaginary quadratic fields before the class number question was. It is a classical result that R is norm-euclidean if and only if d is one of the following five numbers: $d = -1, -2, -3, -7, -11$. Finally, it is known that R is not euclidean under any algorithm for

the remaining four values: d = -19, -43, -67, -163. An excellent proof of all this can be found in [Le1].

Let $K = Q(\sqrt{d})$, where d is positive. K is called a **real quadratic** field. The class number question and the euclidean question are both open in this case. The norm-euclidean question is completely settled for all real quadratic fields.

The class number has been computed for all reasonable values of d. Gauss conjectured that there are infinitely many real quadratic fields that have class number one. This difficult question is still open. A good reference is [Co, chapter 9].

The norm-euclidean question is answered for all real quadratic fields. In particular, K is norm-euclidean if and only if d is one of the following sixteen numbers: d= 2, 3, 5, 6, 7, 11, 13, 17, 19, 21, 29, 33, 37, 41, 57, 73. For references that these fields are all norm-euclidean, see [Ch1]. The difficult part of the previous result is that these are the only real quadratic norm-euclidean fields. That was settled in 1950 by H. Chatland and H. Davenport [Ch2]. Both of the above papers falsely list $Q(\sqrt{97})$ as norm-euclidean. See [B, theorem 12, pages 318-322] for the truth in this matter.

There are, however, real quadratic fields that do have class number one and that are not norm-euclidean. Indeed, if Gauss is correct, there are infinitely many of these. For example, there are twenty three values of d less than 100: d = 14, 22, 23, 31, 38, 41, 43, 46, 47, 53, 59, 61, 62, 67, 69, 71, 77, 83, 86, 89, 93, 94, 97. For each of these, the euclidean question is open. That is, it is unknown whether or not the field is euclidean with respect to some algorithm other than the norm (see [Le1]).

Definition 1.9. Let K be a number field and R be its ring of integers. The **Dedekind zeta function**, ζ_K, is the meromorphic extension of

$$\zeta_K(s) = \sum_{I \subset R} \frac{1}{(\text{Norm}_{K/Q}(I))^s}, \tag{1.13}$$

where the sum runs over the non-zero ideals $I \subset R$ and s is a complex number with $\text{Re}(s) > 1$.

Definition 1.10. The **generalized Riemann hypothesis** is: for each number field K, all non-trivial (non real) zeros of ζ_K satisfy $\text{Re}(s) = 1/2$.

P. J. Weinberger [We] showed that if (i) the generalized Riemann hypothesis is true, (ii) R has infinitely many units, and (iii) K has class number one, then K is euclidean. As the only number fields whose ring of integers has finitely many units are the imaginary quadratic fields, Weinberger's result says, subject to the generalized Riemann hypothesis, that all the class number one real quadratic fields are euclidean with respect to some algorithm.

§4 Cyclotomic Fields

The other major family of fields for which these questions have been posed is the family of cyclotomic fields. These fields were studied systematically in the middle of the nineteenth century in conjunction with Fermat's "last" theorem [Le4]. Thus the three above questions are classical in tone.

Let m be a positive integer not congruent to 2 modulo 4, let ζ_m be a primitive m^{th} root of unity, and let $K = Q(\zeta_m)$. K is known as a **cyclotomic** field with **modulus** m.

Here $R = Z[\zeta_m]$ is the ring of integers in K. It is known that K has class number one if and only if m is one of the following thirty numbers: $m = 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 21, 24, 25, 27, 28, 32, 33, 35, 36, 40, 44, 45, 48, 60, 84$. Of course, for a particular value of m the class number is a relatively straightforward computation. The difficult portion of the above result is that these are the only values of m for which K has class number one. This was done by J. M. Masley and H. M. Montgomery in 1976 [Ma]. Thus the class number question has been settled for all cyclotomic fields.

Of these thirty fields, the norm-euclidean question has been answered if m is one of the following fifteen numbers: $m = 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 20, 24, 32$. It is known that $Q[\zeta_{32}]$ is not norm-euclidean and that the other fourteen are. Whether $Q[\zeta_{32}]$ is euclidean with respect to some algorithm other than the norm is not known. For the remaining fifteen values of m, both the norm-euclidean question and the euclidean question are open.

Of course, Weinberger's aforementioned result says that all thirty of the above fields are euclidean if the generalized Riemann hypothesis holds.

The case with $m = 1$ dates back to Euclid. The case $m = 4$ can be found in Gauss [Ga, pages 117-118]. The first published proof for $m = 3$ was given by P. L. Wantzel [Wa] in 1847. The case $m = 8$ was done by G. Eisenstein [Ei, pages 585-587] in 1850. The case $m = 5$ was published by J. Ouspensky [Ou] in 1909. It is likely that $m = 12$ was done before, but the earliest published proof known is due to R. B. Lakien [La] in 1972.

The cases $m = 7,9,11,15,20$ were done in a landmark paper by H. W. Lenstra, Jr. [Le2] in 1975. Lenstra's techniques are fundamental to what follows in this work. T. Ojala [Oj] did the case $m = 16$ in 1977. We will also use some of Ojala's

techniques. Lenstra [Le3] also did the case $m = 24$ in 1978 using the lattice Γ_8. Finally we have the case $m = 13$ which will be shown to be norm-euclidean in this dissertation.

§5 $Z[\zeta_{13}]$

In what follows in this work, the ring $Z[\zeta_{13}]$ is shown to be norm-euclidean. To my knowledge, this is the highest degree (12) number field whose ring of integers has been shown to be norm-euclidean. Consequentially, the method is computationaly laborious.

The basic method, in outline, is to cover F, a fundamental cell of the quotient field K, by numerous small subregions F_x. Next, for each subregion F_x, an integral point $q \in R$ is found. Finally, $M_x(q) = \max\{ \psi(z - q) : z \in F_x \}$ is bounded above, where $\psi(y) = |\,\mathrm{Norm}_{K/Q}(y)\,|$. Because for each subregion F_x there exists an integral point $q \in R$ with $M_x(q) < 1$, R is norm-euclidean.

Chapter 2

The Theoretical Background

§0 Introduction

The objective of this work is the following theorem.

Theorem 2.1. Let ζ_{13} be a primitive 13^{th} root of unity. Then $Z[\zeta_{13}]$ is norm-euclidean.

The method which we will use to prove Theorem 2.1 is reduction to a large, but finite, number of cases. A computer program is used to generate and do each case. This chapter will describe the theoretical background of both the reduction to cases and of what is done with each case. The next chapter will describe the computer program which implements the work of this chapter.

Sections 1 through 4 of this chapter are in direct correspondence with sections 1 through 4 of the following chapter.

§1 Passable Points and Sufficient Sets

To prove Theorem 2.1 we will need to employ methods from linear algebra, combinatorics and analysis. We start by using linear algebra to translate the problem into the setting of an inner product space.

The methods employed in this work will apply to any cyclotomic field of prime modulus, so we will work in that generality. From now on, m will be an odd prime number, $\zeta = \zeta_m$ will be a primitive m^{th} root of unity, R will denote $Z[\zeta]$, and K will denote

its quotient field $Q(\zeta)$. K is a vector space of dimension $2s$ over $\mathbf{Q}$, where $2s = m\text{-}1$. We will let d be a fixed positive integer. This number d will determine the size of the subdivision to be carried out.

Let G be the Galois group of K/Q. G is isomorphic with the multiplicative group of Z/mZ, where we will write $\sigma = \sigma_j$ when $\sigma(\zeta) = \zeta^j$. Note that $\sigma_j \sigma_k = \sigma_{jk}$ and that σ_{-1} is complex conjugation, so denoting the complex conjugate of x by $\overline{x}$, we have that $\sigma_{-j}(x) = \overline{\sigma_j}(x) = \sigma_j(\overline{x})$.

Definition 2.2. For each $x \in K$ define the norm map ψ by

$$\psi(x) = \prod_{j=1}^{2s} \sigma_j(x) = \prod_{j=1}^{s} \sigma_j(x)\sigma_{-j}(x) = \prod_{j=1}^{s} |\sigma_j(x)|^2.$$

Note that $\psi(x)$ is positive definite: that is, $\psi(x) \geq 0$ with $\psi(x) = 0$ only if $x = 0$. Further, if $x \in R$, then $\psi(x) \in Z$. Thus $\psi(R)$ is contained in the set of non-negative integers, a well ordered set. We now define an inner product on K.

Definition 2.3. For $x, y \in K$ we define the **inner product of x and y**, also called the **trace form**, by

$$(x, y) = \mathrm{Trace}_{K/Q}(x \cdot \overline{y}) = \sum_{j=1}^{2s} \sigma_j(x \cdot \overline{y}).$$

Note that the trace form is a rational bilinear symmetric positive definite form on K. Note also that for the elements $\zeta^1, \ldots, \zeta^m$, which span K as a vector space over Q,

$$(\zeta^i, \zeta^j) = m\,\delta_{ij} - 1\,, \tag{2.4}$$

where δ_{ij} is the Kronecker delta.

Definition 2.5. Let the **fundamental cell, F**, consist of those vectors in K which are closer to the origin than to any other point of R. That is:

$$F = \{ x \in K: \text{ for all } q \in R, \| x \| \leq \| x - q \| \},$$

where $\|\cdot\|$ is the distance metric induced by the trace form.

Note that

$$\| x \| \leq \| x-q \| \text{ if and only if } (x , q) \leq \frac{(q , q)}{2}. \tag{2.6}$$

We also have the following elementary lemma which explains why the fundamental cell is called "fundamental".

Lemma 2.7. For each $x \in K$ there exists $p \in R$ such that $x-p \in F$.

Proof. In the topology induced by the trace form on K, R is a discrete subset. Thus for $x \in K$ the set $\{ \| x - r \|: r \in R \}$ has a minimal element, say $\| x-p \|$. We will show that for this p, $x-p \in F$. To do this, let $q \in R$ be arbitrary and set $r = q+p$. Then

$$\| x - p \| \leq \| x - r \| = \| (x-p) - (r-p) \| = \| (x-p) - q \|,$$

so that the conditions of Definition 2.5 are satisfied and $x-p \in F$.

Definition 2.8. Let S be a subset of K, $\alpha \in Q$ and $x \in K$. Define the sets αS and $S + x$ by

$$\alpha S = \{ \alpha s\colon\ s \in S \} \text{ and } S + x = \{ s + x\colon\ s \in S \}.$$

For our fixed positive integer d, we define translates of a shrunken fundamental cell. We are interested in bounding translates of the norm function ψ on each of these regions.

Definition 2.9. For each $x \in K$, define the set F_x by

$$F_x = (d^{-1})F + x.$$

The following definition is one of the two central concepts necessary to prove Theorem 2.1.

Definition 2.10. We will say, for $x \in K$, that x is **passable** when there exists a $q \in R$ such that for all $z \in F_x$ we have $\psi(z - q) < 1$.

An immediate consequence of the above definitions is the following lemma which says that the property of passability is well behaved under translation by elements of R.

Lemma 2.11. For each $x \in K$ and $p \in R$, if x is passable then $x + p$ is passable.

Proof. Assume that x is passable and let $r \in R$ be chosen so that for each $w \in F_x$ we have $\psi(w-r) < 1$. Consider $z \in F_{x+p}$. As $F_{(x+p)} = (F_x)+p$, we have that $z-p \in F_x$. Hence $\psi((z-p) - r) < 1$. Thus letting $q = p+r$, we have that there exists $q \in R$ such that for each $z \in F_{x+p}$, $\psi(z-q) < 1$.

We develop the concept of a sufficient set S. This is the second of the two central concepts necessary to prove Theorem 2.1. Given such a set S we will reduce the proof of Theorem 2.1 to the proof that all elements of S are passable.

Definition 2.12. Let S be a subset of K. We will say that S is **sufficient** when the following implication holds -- for R to be euclidean with respect to the algorithm ψ, it suffices that every element $x \in S$ is passable.

For computational reasons, we are interested in finding a small sufficient set S. To start with we have:

Proposition 2.13. $S_1 = (d^{-1})R$ is sufficient.

Proof. Assume that each $x \in (d^{-1})R$ is passable. We need to show that R is euclidean. Let $a \in R$ and $b \in R$ be given with $b \neq 0$. Let $z = a/b$. Noting that Lemma 2.7 says $K = \bigcup\{F+p: p \in R\}$, we can choose $p \in R$ such that $dz \in F+p$. Then letting $x = p/d$ we have $x \in (d^{-1})R$ and $z \in F_x$. As x is assumed to be passable, there exists a $q \in R$ such that $\psi(z-q) < 1$. Setting $r = a - bq$, then $a = bq + r$ and $z-q = (a/b) - q = (r/b)$ so that $\psi(r/b) < 1$. Since ψ is multiplicative, we have that $\psi(r) < \psi(b)$ and R is euclidean with respect to the algorithm ψ. Thus S_1 is a sufficient set and Proposition 2.13 is proven.

We can get a smaller sufficient set S_2 in the following manner.

Proposition 2.14. The following set, S_2, is sufficient.

$$S_2 = (d^{-1})R \cap F.$$

Proof. Suppose that each element of S_2 is passable. We need to show that R is euclidean with respect to the algorithm ψ. Let S_1 be the set of Proposition 2.13 and let $x \in S_1$. Using Lemma 2.7 select $p \in R$ such that $x\text{-}p \in F$. As $x\text{-}p \in S_2$ then by hypothesis $x\text{-}p$ is passable. Thus Lemma 2.11 says that x is passable. Hence all elements of S_1 are passable. Since S_1 is sufficient, we conclude that R is euclidean with respect to the algorithm ψ and that S_2 is sufficient.

The following lemma, due to H. W. Lenstra, Jr., gives an upper bound for the radius of the fundamental cell for any prime modulus m. The bound given is the exact radius, although we will not need that fact.

Lemma 2.15. [Le2, Proposition 3.1] For every $x \in F$ we have

$$\| x \| \leq \sqrt{\frac{m^2 - 1}{12}} \, . \tag{2.16}$$

Note that Lemma 2.15 shows that F is a bounded set. Thus since S_2 is a discrete subset of F, S_2 is a finite set. Using (2.16) and the arithmetic-geometric mean inequality we can refine S_2 to the next set S_3 which is somewhat smaller. The following definition and lemma will be used to define S_3.

Definition 2.17. Define the real number L by

$$L = \sqrt{m - 1} - \frac{1}{d}\sqrt{\frac{m^2 - 1}{12}} \, . \tag{2.18}$$

Lemma 2.19. Let $x \in K$. If $\| x \| < L$, then x is passable.

Proof. Let $x \in K$ be such that $\| x \| < L$ and let $z \in F_x$. Then by Definition 2.17 and Lemma 2.15 we have that $\| z \| < \sqrt{m-1}$. Note that definition 2.2 gives

$$\psi(z) = \prod_{j=1}^{s} | \sigma_j(z) |^2 = \left(\left(\prod_{j=1}^{s} | \sigma_j(z) |^2 \right)^{1/s} \right)^s.$$

Then by using the arithmetic-geometric mean inequality we have

$$\psi(z) \leq \left(\frac{1}{s} \sum_{j=1}^{s} | \sigma_j(z) |^2 \right)^s. \tag{2.20}$$

But $| \sigma_{m-j}(z) | = | \sigma_{-j}(z) | = | \sigma_j(z) |$ and $\sigma_j(z)\,\overline{\sigma_j}(z) = \sigma_j(z \cdot \overline{z})$, so that

$$\left(\frac{1}{s} \sum_{j=1}^{s} | \sigma_j(z) |^2 \right)^s = \left(\frac{1}{2s} \sum_{j=1}^{2s} | \sigma_j(z) |^2 \right)^s = \left(\frac{1}{m-1} \sum_{j=1}^{2s} \sigma_j(z \cdot \overline{z}) \right)^s. \tag{2.21}$$

Then noting that by definition 2.3

$$\sum_{j=1}^{2s} \sigma_j(z \cdot \overline{z}) = (z, z) < m - 1, \tag{2.22}$$

we can combine (2.20), (2.21) and (2.22) to get

$$\psi(z) < 1.$$

This leads to our third sufficient set, S_3, which will be refined to the final sufficient set, S, in section 4. The following proposition is a simple consequence of Proposition 2.14 and Lemma 2.19.

Proposition 2.23. The following set, S_3, is sufficient.

$$S_3 = (d^{-1})R \cap F \cap \{x \in K: \|x\| \geq L\}. \tag{2.24}$$

§2 Standard Form, Reduced Points and Patterns

It is not clear how to generate the elements in S_3 directly. However, combinatorial methods can generate a slightly larger set of points, the set of reduced points.

To begin with, we need the following characterization of elements of K. This characterization exploits the fact that while $\zeta^1, \ldots, \zeta^m$ span K as a vector space, they are not a basis for K.

Lemma 2.25. for every $x \in K$, there exist unique $x_1, \ldots, x_m \in Q$ satisfying the following three conditions:

$$x = \sum_{i=1}^{m} x_i \zeta^i, \tag{2.26}$$

$$x_i \geq 0 \quad \text{for all } i, \tag{2.27}$$

$$x_j = 0 \quad \text{for some } j. \tag{2.28}$$

Proof. First we prove existence. K is generated as a Q vector space by $\zeta^1, \ldots, \zeta^m$. Thus there exist $y_1, \ldots, y_m \in Q$ such that

$$x = \sum_{i=1}^{m} y_i \zeta^i .$$

Since $\zeta^m - 1 = 0$ and $\zeta \neq 1$, the zetas are subject to the relation

$$0 = \sum_{i=1}^{m} \zeta^i. \tag{2.29}$$

Let j be chosen so that $y_j = \min\{y_i\}$. Define $x_i = y_i - y_j$ Then $x_j = 0$, and for all i, $x_i \geq 0$. Further, using (2.29),

$$x = x - 0 = \sum_{i=1}^{m} y_i \zeta^i - y_j \sum_{i=1}^{m} \zeta^i = \sum_{i=1}^{m} x_i \zeta^i.$$

Next we prove uniqueness. Suppose that $x_1, \ldots, x_m \in Q$ and $y_1, \ldots, y_m \in Q$ both satisfy (2.26), (2.27), and (2.28). Then by (2.26),

$$0 = \sum_{i=1}^{m} (x_i - y_i) \zeta^i.$$

But since m is a prime number, (2.29) is the only relation which the zetas satisfy. Thus we have that

$$(x_1 - y_1) = \ldots = (x_m - y_m). \tag{2.30}$$

Since by (2.28) there exist j and k with $x_j = 0$ and $y_k = 0$, and since by (2.27) we have $x_i \geq 0$ and $y_i \geq 0$ for each i, we can conclude from (2.30) that

$$(x_i - y_i) = (x_j - y_j) = -y_j \leq 0$$

and

$$(x_i - y_i) = (x_k - y_k) = x_k \geq 0.$$

Thus $x_i = y_i$, and we have shown uniqueness. This concludes the proof of Lemma 2.25.

Note that for $x \in K$ with $x_1, \ldots, x_m \in Q$ satisfying (2.26), (2.27), and (2.28), $x \in R$ if and only if $x_i \in Z$ for each i.

Definition 2.31. Let $x \in K$ and let $x_1, \ldots, x_m \in Q$ satisfy (2.26), (2.27), and (2.28). When this holds, we say that $x_1, \ldots, x_m$ is the **standard form** for x.

The following definition will be of use in generating elements of S_3.

Definition 2.32. Let $x \in K$ and let $x_1, \ldots, x_m$ be the standard form for x. We will call x **reduced** when $x_i < 1$ for all i.

Note that by the uniqueness of the standard form, the concept of reduced depends only on x and not on its standard form.

We have the following Lemma concerning reduced points and the fundamental cell F.

Lemma 2.33. Let $x \in K$. If $x \in F$, then x is reduced.

Proof. Let $x_1, \ldots, x_m$ be the standard form for x where $x_j = 0$. Let $p = -\zeta^j$. Then $p \in R$, so that, applying Definition 2.5 and equation (2.6) for this p, we can conclude that

$$\sum_{i=1}^{m} x_i \leq \frac{m-1}{2}. \tag{2.34}$$

Similarly, let $p = \zeta^k$ where k is chosen such that $x_k = \max\{ x_i \}$. Then, applying Definition 2.5 and equation (2.6) for this p, we have

$$m x_k - \sum_{i=1}^{m} x_i \leq \frac{m-1}{2}. \tag{2.35}$$

Combining (2.34) and (2.35) we have

$$m x_k \leq \left(\sum_{i=1}^{m} x_i \right) + \frac{m-1}{2} \leq m - 1,$$

and thus x is reduced since

$$x_i \leq \max\{ x_i \} = x_k \leq 1 - \frac{1}{m}.$$

This concludes the proof of Lemma 2.33.

The above results show that the sufficient set S_3 is a subset of the set of reduced points belonging to $(d^{-1})R$. Thus to generate all points of S_3 it certainly suffices to generate all the reduced points belonging to $(d^{-1})R$. This is the technique which will be employed.

Next we introduce the concept of the pattern of a reduced point of $(d^{-1})R$.

Definition 2.36. Let $x \in (d^{-1})R$ be reduced. Let $x_1, \ldots, x_m$ be the standard form for x. For j with $0 \le j < d$, define a_j to be the number of indices i such that $x_i = j/d$. We will write $\boldsymbol{a} = (a_0, \ldots, a_{d-1})$ and say that $\boldsymbol{a}$ is the **pattern** for x.

Note that by the definition of standard form there exists some index j with $x_j = 0$, hence $a_0 \ge 1$. It is also immediate that

$$\sum_{j=0}^{d-1} a_j = m. \tag{2.37}$$

A further consequence of the definition is that

$$\sum_{i=1}^{m} x_i = \frac{1}{d} \sum_{j=0}^{d-1} j\, a_j \tag{2.38}$$

and

$$\sum_{i=1}^{m} x_i^2 = \frac{1}{d^2} \sum_{j=0}^{d-1} j^2\, a_j. \tag{2.39}$$

The length restriction from Definition 2.17 for membership in S_3 can now be computed from the pattern of a point.

Lemma 2.40. Let $x \in (d^{-1})R$ have pattern a. Then

$$\| x \|^2 = \frac{1}{d^2} \left(m \sum_{j=0}^{d-1} j^2 a_j - \left(\sum_{j=0}^{d-1} j a_j \right)^2 \right). \tag{2.41}$$

In particular, if $x \in S_3$ then

$$(dL)^2 \le m \sum_{j=0}^{d-1} j^2 a_j - \left(\sum_{j=0}^{d-1} j a_j \right)^2 \le d^2 \frac{m^2 - 1}{12}. \tag{2.42}$$

Proof. Since (2.4) gives

$$\| x \|^2 = m \sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i \right)^2,$$

(2.38) and (2.39) yield (2.41). Lemma 2.15 and the definition of S_3 in (2.24) yield (2.42).

§3 The Fundamental Cell and Patterns

This section will establish that whether a reduced point, $x \in (d^{-1})R$, belongs to the fundamental cell is determinable solely from the pattern, a, of x.

Basic to working with patterns is the symmetric group Γ. We will define Γ and give important examples and elementary properties of these permutations next.

Definition 2.43. Let Γ be the m^{th} symmetric group, i.e., the set of permutations of $\{1, \ldots, m\}$. The group Γ acts on the vector space K in the following manner: For $x \in K$ and $P \in \Gamma$ with

$$x = \sum_{i=1}^{m} x_i \zeta^i,$$

we define $P(x)$ to be

$$P(x) = \sum_{i=1}^{m} x_i \zeta^{P(i)}. \tag{2.44}$$

Each $P \in \Gamma$ acts linearly on K. This action is well defined, for it respects the relation (2.29). That is,

$$P(0) = P\Big(\sum_{i=1}^{m} \zeta^i\Big) = \sum_{i=1}^{m} \zeta^{P(i)} = 0. \tag{2.45}$$

Finally, if $Q \in \Gamma$ is such that $Q = P^{-1}$, then

$$P(x) = \sum_{i=1}^{m} x_{Q(i)} \zeta^i. \tag{2.46}$$

This is to say that each $P \in \Gamma$ acts on K by permuting the spanning set $\zeta^1, \ldots, \zeta^m$, hence the coordinates of each $x \in K$ with respect to this spanning set are permuted by P^{-1}.

Note that the set of reduced $x \in (d^{-1})R$ with given pattern a is precisely an orbit under the action of Γ on K.

Patterns are invariant under permutations. In general, any property which is invariant under all permutations is a property which depends only on pattern.

Note also that the galois group G acts on K as a group of permutations. In particular, recalling that $\sigma_j(\zeta) = \zeta^j$, then for $x \in K$ with

$$x = \sum_{i=1}^{m} x_i \zeta^i,$$

we have that

$$\sigma_j(x) = \sum_{i=1}^{m} x_i \zeta^{ij}.$$

Thus as a permutation, $\sigma_j(i) \equiv ij$ modulo m. Hence we may think of G as embedded in Γ.

The other important example of a permutation on K is multiplication by ζ. That is, for $x \in K$ with

$$x = \sum_{i=1}^{m} x_i \zeta^i,$$

we have that

$$\zeta x = \sum_{i=1}^{m} x_i \zeta^{i+1}.$$

Thus multiplication by ζ can be realized as the permutation $P \in \Gamma$ where $P(i) \equiv i + 1$ modulo m. This can also be thought of as a circular shift of the coordinates x_i.

The above permutations generate the affine group modulo m. This group will play a significant role in the next section.

Definition 2.47. We will define Λ, **the affine group modulo** m, to be the group of all $P \in \Gamma$ satisfying, for $\alpha, \beta \in Z$, with $(\alpha,m) = 1$,

$$P(i) \equiv i\,\alpha + \beta \text{ (modulo } m) \tag{2.48}$$

It is immediate that the affine group Λ is generated by the galois group G and multiplication by ζ.

The following lemma concerning the symmetric group Γ is of note.

Lemma 2.49. Γ is a group of isometries for the trace form. That is, for each $x, y \in K$ and $P \in \Gamma$, $(x, y) = (P(x), P(y))$.

Proof. Since $\zeta^1, \ldots, \zeta^m$ span the vector space K, it suffices to show that $(\zeta^i, \zeta^j) = (P(\zeta^i), P(\zeta^j))$. But

$$(P(\zeta^i), P(\zeta^j)) = (\zeta^{P(i)}, \zeta^{P(j)}) = m\delta_{P(i)P(j)} - 1 = m\delta_{ij} - 1 = (\zeta^i, \zeta^j).$$

We will uncover some relationships between the fundamental cell F and the symmetric group Γ. First we show that F is Γ-invariant.

Lemma 2.50. If $x \in F$ and $P \in \Gamma$ then $P(x) \in F$.

Proof. Let x and P be as above and note that by the previous lemma we have $\| x \| = \| P(x) \|$. Note also that R is invariant under P: that is, if $q \in R$ then $P(q) \in R$ and $P^{-1}(q) \in R$ as well. Thus for an arbitrary $q \in R$ we have

$$\| P(x) \| = \| x \| \leq \| x - P^{-1}(q) \| = \| P(x-P^{-1}(q)) \| = \| P(x) - q \|.$$

Thus from definition 2.5, $P(x)$ must be an element of F, and the proof of Lemma 2.50 is complete.

Next we shall work on a series of computational results which will enable us to determine membership in the fundamental cell entirely from a point's pattern.

To start this process, we will need a better characterization of the fundamental cell. The following collection of elements of R, defined by H. W. Lenstra, Jr. in [Le2, p461], will be of use.

Definition 2.51. For every subset A of $\{1, \ldots, m\}$, define e_A to be

$$e_A = \sum_{i \in A} \zeta^i.$$

The relationship between the points e_A and the symmetric group Γ is given in the following elementary lemma.

Lemma 2.52. For every subset A of $\{1, \ldots, m\}$ and every $P \in \Gamma$, $P(e_A) = e_{P(A)}$.

Next we have the exact description of the fundamental cell. The following lemma states that the fundamental cell is the intersection of half spaces determined by the e_A.

Lemma 2.53. [Le2, Lemma 3.3] Let $x \in K$. Then $x \in F$ if and only if for every subset A of $\{1, \ldots, m\}$,

$$(x, e_A) \leq \frac{(e_A, e_A)}{2}. \tag{2.54}$$

The following two definitions and lemma give a preliminary computational method of determining membership in the fundamental cell.

Definition 2.55. Let $x \in K$, $P \in \Gamma$, and $y = P(x)$. Let $y_1, \ldots, y_m$ be the standard form for y. If $y_1 \geq \ldots \geq y_m$, we say that P **orders** x and that $y_1 \geq \ldots \geq y_m$ is an **ordering** of x.

Note that by the uniqueness of standard form guaranteed by Lemma 2.25 and by the order relationship $y_1 \geq \ldots \geq y_m$, an ordering $y_1 \geq \ldots \geq y_m$ is uniquely determined by the point x although the permutation P may not be.

Definition 2.56. Let $x \in K$ and let $y_1 \geq \ldots \geq y_m$ order x. For k with $0 \leq k \leq m$, define the function $g_x(k)$ by

$$g_x(k) = k \sum_{i=1}^{m} y_i - m \sum_{i=1}^{k} y_i + \frac{k(m-k)}{2}. \tag{2.57}$$

Note that since the ordering $y_1 \geq \ldots \geq y_m$ is unique, the function g_x is dependent only on x and not on its ordering.

Lemma 2.58. Let $x \in K$ and let $y_1 \geq \ldots \geq y_m$ order x. Then $x \in F$ if and only if for each k with $0 \leq k \leq m$, $g_x(k) \geq 0$.

Proof. We will use Lemma 2.53. To this end, let A be a subset of $\{1, \ldots, m\}$ containing k elements. Then

$$(e_A, e_A) = k(m-k),$$

and

$$(x, e_A) = m \sum_{j \in A} x_j - k \sum_{i=1}^{m} x_i.$$

Thus, applying (2.54), we have that $x \in F$ if and only if for all subsets A of $\{1, \ldots, m\}$ we have that

$$k \sum_{i=1}^{m} x_i + \frac{k(m-k)}{2} - m \sum_{j \in A} x_j \geq 0.$$

But the y_i's are a permutation of the x_i's such that $y_1, \ldots, y_k$ are the k largest of the x_i's. Further, the sum of all the x_i's is the same as the sum of all the y_i's. Thus $x \in F$ if and only if (2.57) holds and the proof of Lemma 2.58 is complete.

In order to apply Lemma 2.58 to patterns, we will need the finite differences of the function $g_x(k)$. An elementary computation gives the following lemma whose proof is omitted.

Lemma 2.59. For i with $0 \le i \le m - 1$, we have

$$\Delta g_x(i) = -m\, y_{i+1} + \frac{m-1}{2} - i + \sum_{j=1}^{m} y_j. \tag{2.60}$$

For i with $0 \le i \le m - 2$, we have

$$\Delta^2 g_x(i) = m(y_{i+1} - y_{i+2}) - 1. \tag{2.61}$$

Finally, we can translate the concept of belonging to the fundamental cell F into terms totally determinable from the point's pattern.

Proposition 2.62. For $x \in (d^{-1})R$ reduced with pattern a, $x \in F$ if and only if $g_x(k) \ge 0$ for the following $d + 1$ values of $k = k_j$

$$k_j = \sum_{i=j}^{d-1} a_i, \qquad 0 \le j \le d. \tag{2.63}$$

Further, $g_x(k_d) = g_x(0) = 0$, $g_x(k_0) = g_x(m) = 0$, and for j with $d\text{-}1 \le j \le 1$,

$$g_x(k_j) = g_x(k_{j+1}) + a_j \left(\frac{m - a_j}{2} - \frac{m\, j}{d} - k_{j+1} + \frac{1}{d} \sum_{i=0}^{d-1} i\, a_i \right). \tag{2.64}$$

Proof. Note that $k_j \geq k_{j+1}$, so that

$$g_x(k_j) = g_x(k_{j+1}) + \sum_{i=k_{j+1}}^{k_j - 1} \Delta g_x(i).$$

Thus by using (2.60) we have

$$g_x(k_j) = g_x(k_{j+1}) + \sum_{i=k_{j+1}}^{k_j-1} \left(-m\, y_{i+1} + \frac{m-1}{2} - i + \sum_{j=1}^{m} y_j \right) \tag{2.65}$$

Also, for i with $k_{j+1} + 1 \leq i+1 \leq k_j$, we have $y_{i+1} = j/d$. Then, since $k_j - k_{j+1} = a_j$, (2.65) yields

$$g_x(k_j) = g_x(k_{j+1}) + a_j \left(-\frac{m\, j}{d} + \frac{m-1}{2} + \sum_{j=1}^{m} y_j \right) - \sum_{i=k_{j+1}}^{k_j - 1} i\ , \tag{2.66}$$

and also

$$\sum_{i=k_{j+1}}^{k_j - 1} i. = a_j \left(k_{j+1} + \frac{a_j - 1}{2} \right) \tag{2.67}$$

Combining (2.66) and (2.67) yields (2.64).

Lastly, using Lemma 2.58, we need only show that $g_x(k) \geq 0$ for those k with $k \neq k_j$ for any j. Thus we will suppose that k is such that $k \neq k_j$ and that $1 \leq k \leq m$. But then there is a j with $k_{j+1} < k < k_j$ and $a_j > 1$. Also for those i with $k_{j+1} < i \leq k_j$ we have $y_i = j/d$. Thus Lemma 2.59 says that $\Delta^2 g_x(i) = -1$ for i with $k_{j+1} \leq i \leq k_j - 2$. But this says

that $g_x(i)$ is concave downward for i with $k_{j+1} \le i \le k_j$. Hence the minimum value of $g_x(i)$ must occur at the end points: $i = k_{j+1}$ or $i = k_j$. By hypothesis we have that $g_x(k_{j+1}) \ge 0$ and $g_x(k_j) \ge 0$. Thus $g_x(k) \ge 0$ and this concludes the proof of Proposition 2.62.

§4 The Final Sufficient Set -- Points from Patterns

The preceding sections show that, given a reduced point $x \in (d^{-1})R$, x is an element of the sufficient set S_3 if and only if its pattern a satisfies (2.42), (2.63) and (2.64). These are the last of the checks for membership in the final sufficient set, S, which can be done at the pattern level.

It is time to consider all points belonging to a specific pattern. Here the action of the affine group Λ of Definition 2.47 needs to be taken into account.

We have the following lemma which explains our interest in the affine group Λ.

Lemma 2.68. The norm map ψ is Λ-invariant. That is, for $x \in K$ and $P \in \Lambda$ we have

$$\psi(P(x)) = \psi(x). \tag{2.69}$$

Proof. Let $\alpha, \beta \in Z$ be such that $(\alpha, m) = 1$. Let $P \in \Lambda$ be given by $P(i) = i\alpha + \beta$. Then if $x \in K$ is

$$x = \sum_{i=1}^{m} x_i \zeta^i,$$

we have that

$$P(x) = \sum_{i=1}^{m} x_i \, \zeta^{(i\alpha+\beta)}.$$

But if $\sigma_j \in G$, we have that

$$\sigma_j(P(x)) = \sum_{i=1}^{m} x_i \, \zeta^{j(i\alpha+\beta)} = \zeta^{j\beta} \sum_{i=1}^{m} x_i \, \zeta^{ij\alpha} = \zeta^{j\beta} \, \sigma_{j\alpha}(x) \tag{2.70}$$

So that, combining (2.70) with Definition 2.2, we have that

$$\psi(P(x)) = \prod_{j=1}^{2s} \sigma_j(P(x)) = \left(\prod_{j=1}^{2s} \zeta^{j\beta} \right) \left(\prod_{j=1}^{2s} \sigma_{j\alpha}(x) \right) \tag{2.71}$$

But as $2s = m\text{-}1$ and m is odd,

$$\sum_{j=1}^{2s} j\beta = \beta \, \frac{m(m\text{-}1)}{2} \equiv 0 \quad \text{modulo } m. \tag{2.72}$$

And, as $(\alpha,m) = 1$,

$$\prod_{j=1}^{2s} \sigma_{j\alpha}(x) = \prod_{j=1}^{2s} \sigma_j(x) = \psi(x). \tag{2.73}$$

Then, combining (2.71), (2.72) and (2.73) we have $\psi(P(x)) = \psi(x)$ and the proof of Lemma 2.68 is concluded.

An immediate consequence of Lemma 2.68 is the following proposition.

Proposition 2.74. Let $x \in K$ and $P \in \Lambda$. Then x is passable if and only if $P(x)$ is passable.

Proposition 2.74 says that the property of passability needs to be determined only modulo the action of the group Λ. In particular, if S is a Λ-invariant sufficient set, and S' is a subset of S consisting of one representative of each orbit in S under the action of Λ, then S' is sufficient.

It is not clear how to pick a single representative of each Λ-orbit. However, since Λ is 2-transitive on $\{1, \ldots, m\}$, each orbit contains a representative with selected first and last coordinates. These ideas will be made more precise in what follows.

The following two definitions will be used to single out one representative of each Λ-orbit.

Definition 2.75. Let $a = (a_0, \ldots, a_{d-1})$ be a pattern and j' and j'' be integers, not necessarily distinct, satisfying the following three conditions:

$$0 \leq j', j'' < d, \tag{2.76}$$

$$a_{j'} \geq 1 \quad \& \quad a_{j''} \geq 1 \tag{2.77}$$

$$a_{j'} \geq 2, \quad \text{if } j' = j''. \tag{2.78}$$

Then we say that the pair $(j',j'')_a$ is a **selection for the pattern** a.

We will drop the subscript for the selection when it is clear which pattern is intended.

Definition 2.79. Let (j',j'') be a selection for the pattern a. Suppose that $y \in (d^{-1})R$ is reduced with pattern a. If $y = (y_1, \ldots, y_m)$ in standard form with $y_1 = j'/d$ and $y_m = j''/d$, we say that y is **(j',j'') selected.**

The $(j',j'')_a$ selected points are simply all those points, with a given pattern, which have first and last coordinates equal to j'/d and j''/d respectively.

Given a pattern, a, and a selection, $(j',j'')_a$, the collection of $(j',j'')_a$ selected points will be the representatives of the Λ-orbits. The following lemma says that for each Λ-orbit, there is a representative which is $(j',j'')_a$ selected.

Lemma 2.80. Let $(j',j'')_a$ be a selection for the pattern a. Then for each $x \in (d^{-1})R$, reduced with pattern a, there exists a $Q \in \Lambda$ such that $Q(x)$ is $(j',j'')_a$ selected.

Definition 2.81. For each pattern a, let a selection $(j',j'')_a$ be fixed. Let

$$\Sigma = \{ y \in (d^{-1})R: \ y \text{ has pattern } a \ \& \ \text{is } (j',j'')_a \text{ selected} \}. \tag{2.82}$$

This gives us our final sufficient set.

Proposition 2.83. The following set S is sufficient:

$$S = (d^{-1})R \cap F \cap \{ x \in K: \| x \| \geq L \} \cap \Sigma. \tag{2.84}$$

For any choices of selections, we get a sufficient set, S, in this manner. However, different selections will result in a larger or smaller set.

For a fixed pattern, $a = (a_0, \ldots, a_{d-1})$, whose points lie in S_3, the number of such points is given by the multinomial coefficient:

$$\binom{m}{a_0 \ldots a_{d-1}} = m! \left(\prod_{j=0}^{d-1} a_j! \right)^{-1}. \tag{2.85}$$

Let $(j'j'')_a$ be a selection for the pattern a. Let b_j be defined by:

$$b_j = \begin{cases} a_j & j \neq j' \text{ and } j \neq j'' \\ a_j - 1 & j = j' \text{ or } j = j'' \text{ when } j' \neq j'' \\ a_j - 2 & j = j' = j'' \end{cases} \tag{2.86}$$

The number of $(j'j'')_a$ selected points with pattern a is given by the following multinomial coefficient:

$$\binom{m-2}{b_0 \ldots b_{d-1}} = (m-2)! \left(\prod_{j=0}^{d-1} b_j! \right)^{-1}. \tag{2.87}$$

Thus the ratio of number of these elements of S_3 to number of elements of S is

$$\frac{m\,(m-1)}{a_{j'}\, a_{j''}} \quad \text{if } j' \neq j'', \tag{2.88}$$

or

$$\frac{m\ (m-1)}{a_{j'}\ (a_{j'}-1)} \quad \text{if } j' = j'' \ . \tag{2.89}$$

Thus, in both cases, we want to choose the selection $(j',j'')_a$ such that $a_{j'}$ and $a_{j''}$ are minimal subject to (2.76), (2.77) and (2.78).

§5 Passing Points -- Finding a Nearby Integer

To prove Theorem 2.1, we need to show that all the points in our sufficient set S of (2.84) are passable. Recalling Definition 2.10, we have the following definition.

Definition 2.90. Let $x \in K$ and $q \in R$. We will say that x **is passed by** q when for all $z \in F_x$ we have that $\psi(z-q) < 1$.

Given an $x \in S$, we can then split the problem of showing that x is passable into the following two parts:

Find a $q \in R$ which **might** pass x.. (2.91)

Given a $q \in R$, **does** q pass x? (2.92)

Combinatorial methods will be used to produce q's for (2.91). Analytic methods will be used to decide (2.92). We will discuss (2.91) in this section.

To intelligently select q's which might pass a given point x, we will first note that the norm function ψ is continuous with respect to the trace form. The value of ψ at 0 is

$\psi(0) = 0$. Thus ψ is small in a neighborhood of 0. In particular, the proof of Lemma 2.19 shows that if $\| z \| < \sqrt{m-1}$, then $\psi(z) < 1$.

Thus if $q \in R$ is such that $\| x-q \|$ is small, we can expect that $\psi(z-q)$ will be small for all $z \in F_x$. Of course, by Definition 2.5 we have that $\| x \| \leq \| x-q \|$ for all $q \in R$. Thus $q = 0$ is the place to start.

By Lemma 2.53, F is the intersection of the half spaces determined by the e_A, where A is any subset of $\{1, \ldots, m\}$. Thus $q = e_A$ are reasonable second choices. However, there are 2^m subsets of $\{1, \ldots, m\}$, and there are $2^m - 1$ distinct such e_A, so a certain degree of selectivity is indicated.

Lemma 2.58 says that to determine if a given $x \in K$ lies in F, we need only check $g_x(k) \geq 0$ for $k = 0, \ldots, m$. This is equivalent to checking that

$$(x, e_k) \leq \frac{(e_k, e_k)}{2},$$

where $e_k = e_{A(k)}$ and where $A(k)$ is a subset of $\{1, \ldots, m\}$ which consists of the indices of the k largest coordinates of x in standard form. This gives the first set of points $q \in R$ for which (2.92) will be checked.

Definition 2.93. Let $x \in S$ and suppose that $P \in \Gamma$ is such that $y = P(x)$ orders x. Define the set $N_1(x,P)$ by

$$N_1(x,P) = \left\{ e_{k,P} = \sum_{i=k+1}^{m} \zeta^{P^{-1}(i)}: \ 1 \leq k \leq m \right\}.$$

Note that $e_{m,P} = 0$, as the sum is empty.

While the actual ordering of any $x \in K$ is unique, the choice of permutation $P \in \Gamma$ such that $P(x)$ orders x is not necessarily unique. This leads to the second collection of "nearby" $q \in R$.

Definition 2.94. Let $x \in S$. Define the set $N_2(x)$ by

$$N_2(x) = \bigcup \{ N_1(x,P):\ P(x) \text{ orders } x \}.$$

The final collection of $q \in R$ which might pass x will be selected more exhaustively. Recalling that the collection of e_A is the set of points which in standard form have coordinates chosen from the set $\{0,1\}$, we have the following definition.

Definition 2.95. Let f be a positive integer. Define the set $N_3(f)$ by:

$$N_3(f) = \{ q \in R:\ q = (q_1, \ldots, q_m) \text{ in standard form } \& \ q_i \le f \}.$$

The set of all e_A is thus $N_3(1)$.

§6 Bounding the Norm ψ on a Sphere

We turn finally to the analytic portion of this work. Given a $x \in K$ and $q \in R$, we wish to bound $\psi(z\text{-}q)$ for all $z \in F_x$. We will translate the question into one concerning a function from R^s to R. Our work is a generalization of the work of T. Ojala in [Oj].

First some preliminary definitions. Recall that the Galois group G consists of the isomorphisms σ_j, where $\sigma_j(\zeta) = \zeta^j$ and $(j,m) = 1$, and that $s = (m\text{-}1)/2$.

Definition 2.96. Let $x \in S$ and $q \in R$. Define the real numbers c_j by $c_j = | \sigma_j(q-x) |$ for j where $1 \le j \le s$. Define the function $f_{x,q}$ from $R^s \to R$ by

$$f_{x,q}(z_1, \ldots, z_s) = \prod_{j=1}^{s} (c_j + z_j).$$

Definition 2.97. Let the real number r be given by

$$r = \frac{1}{d}\sqrt{\frac{m^2 - 1}{24}}.$$

Definition 2.98. Let the real number $M_{x,q}$ be given by

$$M_{x,q} = \sup\Big\{ f_{x,q}(z_1,\ldots,z_s): \sum_{j=1}^{s} z_j^2 \le r^2 \Big\}.$$

The following lemma phrases passability in terms of the behavior of the real function $f_{x,q}$.

Lemma 2.99. Let $x \in S$ and $q \in R$. If $M_{x,q} < 1$, then x is passed by q.

Proof. Suppose that x is not passed by q. Thus there exists a $z \in F_x$ such that $\psi(z-q) \ge 1$. Since $z \in F_x$ then $z-x \in F_x - x = (d^{-1})F$. Hence Lemma 2.15 says that

$$\| z-x \| \le \frac{1}{d}\sqrt{\frac{m^2 - 1}{12}}.$$

Then, noting that

$$(z\text{-}x,\ z\text{-}x) = \sum_{j=1}^{2s} | \sigma_j(z\text{-}x) |^2 = 2 \sum_{j=1}^{s} | \sigma_j(z\text{-}x) |^2 ,$$

we can set $z_j = | \sigma_j(z\text{-}x) |$. Then

$$\sum_{j=1}^{s} z_j^2 \leq r^2. \tag{2.100}$$

Since by Definition 2.2

$$1 \leq \psi(z\text{-}q) = \prod_{j=1}^{s} | \sigma_j(z\text{-}q) |^2,$$

then, by taking square roots,

$$1 \leq \prod_{j=1}^{s} | \sigma_j(z\text{-}q) |. \tag{2.101}$$

But

$$\prod_{j=1}^{s} | \sigma_j(z\text{-}q) | = \prod_{j=1}^{s} | \sigma_j(z\text{-}x) - \sigma_j(q\text{-}x) |, \tag{2.102}$$

and Definition 2.96 yields

$$| \sigma_j(z\text{-}x) - \sigma_j(q\text{-}x) | \leq | \sigma_j(z\text{-}x) | + | \sigma_j(q\text{-}x) | = c_j + z_j. \tag{2.103}$$

Thus, combining (2.101), (2.102), and (2.103) results in

$$1 \leq \prod_{j=1}^{s} (c_j + z_j) = f_{x,q}(z_1, \ldots, z_s). \tag{2.104}$$

But (2.100) and our hypothesis yield that

$$f_{x,q}(z_1, \ldots, z_s) \leq M_{x,q} < 1. \tag{2.105}$$

As (2.104) and (2.105) are in direct contradiction, Lemma 2.99 is proven.

Next we will apply a simple Lagrange multiplier argument to help bound $M_{x,q}$.

Lemma 2.106. Let $x \in S$ and $q \in R$ be such that $x\text{-}q \neq 0$. Let $z_1, \ldots, z_s$ be real numbers such that $z_1^2 + \ldots + z_s^2 \leq r^2$ and $f_{x,q}(z_1, \ldots, z_s) = M_{x,q}$. Then $z_1, \ldots, z_s$ satisfy the following three conditions:

$$z_j \geq 0 \quad (\text{for } 1 \leq j \leq s), \tag{2.107}$$

$$\sum_{j=1}^{s} z_j^2 = r^2, \tag{2.108}$$

$$z_i (z_i + c_i) = z_j (z_j + c_j) \quad (\text{for } 1 \leq i,j \leq s). \tag{2.109}$$

Proof. Since $x - q \neq 0$, then we have that $c_j = |\sigma_j(x-q)| > 0$ for all j. Thus, as $z_1, \ldots, z_s$ maximizes $f_{x,q}$, we have that $z_j \geq 0$ and that

$$M_{x,q} = f_{x,q}(z_1, \ldots, z_s) = \prod_{j=1}^{s} (c_j+z_j) > 0.$$

The gradient of $f_{x,q}$ is

$$\nabla f_{x,q}(y_1, \ldots, y_s) = \left(\prod_{j \neq 1} (c_j+y_j), \ldots, \prod_{j \neq s} (c_j+y_j) \right). \tag{2.110}$$

Thus $\nabla f_{x,q}(z_1, \ldots, z_s) \neq \mathbf{0}$, and the maximum of $f_{x,q}$ must occur on the boundary of the sphere of radius r. Thus

$$\sum_{j=1}^{s} z_j^2 = r^2.$$

Let

$$g(y_1, \ldots, y_s) = \left(\sum_{j=1}^{s} y_j^2 \right) - r^2. \tag{2.111}$$

Then $z_1, \ldots, z_s$ is the maximum of $f_{x,q}$ subject to the constraint $g = 0$. Thus there exists a Lagrange multiplier $\lambda \neq 0$ with

$$\nabla f_{x,q}(z_1, \ldots, z_s) = \lambda \nabla g(z_1, \ldots, z_s).$$

But this yields, for $1 \le i \le s$, that

$$\prod_{j \neq i} (c_j + z_j)) = 2\lambda z_i.$$

So, for $1 \le i \le s$,

$$M_{x,q} = f_{x,q}(z_1, \ldots, z_s) = \prod_{j=1}^{s} (c_j + z_j) = 2\lambda\, z_i\, (\, c_i + z_i\,). \tag{2.112}$$

As the left hand side of (2.112) is independent of i, (2.109) follows and Lemma 2.106 is proven.

Next we have a proposition which will give an algorithm which is used in practice to compute an upper bound on $M_{x,q}$.

Proposition 2.113. Let $x \in S$ and $q \in R$ be such that $x\text{-}q \neq 0$. Let $z_1, \ldots, z_s$ be real numbers such that $z_1^2 + \ldots + z_s^2 \le r^2$ and $f_{x,q}(z_1, \ldots, z_s) = M_{x,q}$. Let the index k be chosen such that $c = c_k = \min\{c_j\}$ and let β be a positive real number. Let the positive real numbers β_j be defined by, for $1 \le j \le s$,

$$\beta_j\, (\, \beta_j + c_j\,) = \beta\, (\, \beta + c\,). \tag{2.114}$$

Let the positive real number γ be defined by

$$\gamma = r\,\beta \left(\sum_{j=1}^{s} \beta_j^2 \right)^{-1/2}. \tag{2.115}$$

Then $z_k \le \beta$ implies that $z_k \ge \gamma$, and $z_k \ge \beta$ implies that $z_k \le \gamma$.

Proof. Using Lemma 2.106 we have that $y = z_j$ and $x = z_k$ satisfy the hyperbolic relation

$$y\,(\,y + c_j\,) = x\,(\,x + c\,). \tag{2.116}$$

An elementary computation yields that

$$\frac{dy}{dx} = \frac{2x + c}{2y + c_j} \tag{2.117}$$

$$\frac{d^2y}{dx^2} = 2\,\frac{c_j^2 - c^2}{(2y + c_j)^3}. \tag{2.118}$$

Since $c_j \ge c$, the graph of (2.116) is increasing and concave upwards in the first quadrant. Further, (2.114) says that the point $(x,y) = (\beta,\beta_j)$ also lies in the first quadrant on the graph of (2.116).

If $z_k \le \beta$, the above remarks show that the secant line from the point (0,0) to the point (β,β_j) passes above the point (z_k,z_j) on the curve. Thus

$$\frac{\beta_j}{\beta} \ge \frac{z_j}{z_k}$$

and hence

$$z_j \le \frac{z_k}{\beta}\beta_j.$$

But then

$$r^2 = \sum_{j=1}^{s} z_j^2 \leq \frac{z_k^2}{\beta^2} \sum_{j=1}^{s} \beta_j^2,$$

so that

$$z_k^2 \geq r^2 \beta^2 \left(\sum_{j=1}^{s} \beta_j^2 \right)^{-1} = \gamma^2.$$

The remaining case is done by reversing the inequalities in the above argument. This concludes the proof of Proposition 2.113.

Proposition 2.113 gives a method for translating upper bounds on z_k into lower bounds, and translating lower bounds into upper bounds. As each z_j is an increasing function of z_k, this gives upper and lower bounds on all of the z_j's. Thus we have both bounds on $M_{x,q}$.

Lastly, we have a proposition which will enable us, working from an arbitrary positive real number, to get bounds on $M_{x,q}$.

Proposition 2.119. Let x, q, c, β, and β_j be as in Proposition 2.113. Let the positive real number λ be defined by

$$\lambda = \left(\sum_{j=1}^{s} \beta_j^2 \right)^{1/2}. \tag{2.120}$$

If $r \leq \lambda$ then $M_{x,q} \leq f_{x,q}(\beta_1, \ldots, \beta_s)$. Conversely, if $r \geq \lambda$ then $M_{x,q} \geq f_{x,q}(\beta_1, \ldots, \beta_s)$.

Proof. Equation (2.114) defines each β_j as a continuous, strictly increasing function of β. Thus equation (2.120) defines λ as a continuous, strictly increasing function of β. Since $\lambda \to 0$ as $\beta \to 0$ and $\lambda \to \infty$ as $\beta \to \infty$, there exists a unique value of β, call it β_0, such that the corresponding value of λ is equal to r.

Conversely, let $z_1, \ldots, z_s$ be real numbers such that $z_1^2 + \ldots + z_s^2 \leq r^2$ and $f_{x,q}(z_1, \ldots, z_s) = M_{x,q}$. Let $z = \max\{z_j\}$. Then by equations (2.107) and (2.109), if k is an index such that $z = z_k$, then $c = c_k$. Thus from equation (2.108), $\beta_0 = z$. Further, as λ is an increasing function of β, $\beta < z$ if and only if $\lambda < r$.

But equation (2.127) defines $f_{x,q}(\beta_1, \ldots, \beta_s)$ as a function of β, and it is clear that this is also a continuous, strictly increasing function of β. Thus $\beta < z$ if and only if $f_{x,q}(\beta_1, \ldots, \beta_s) < f_{x,q}(z_1, \ldots, z_s) = M_{x,q}$. Thus Proposition 2.119 is proven.

Proposition 2.119 gives us a method which avoids accumulation of numerical error while we iterate Proposition 2.113. That is, our β's will be selected via Proposition 2.113 and then the λ's calculated via Proposition 2.119. Then if λ is larger than r, $f_{x,q}(\beta_1, \ldots, \beta_s)$ is an upper bound on $M_{x,q}$, and if λ is smaller than r, $f_{x,q}(\beta_1, \ldots, \beta_s)$ is a lower bound on $M_{x,q}$. Finally, if λ is close to r, then $f_{x,q}(\beta_1, \ldots, \beta_s)$ is close to $M_{x,q}$.

Chapter 3

A Description of the Program and Its Algorithms

§0 Introduction

In the previous chapter we developed a plan of attack to prove that $K = Q(\zeta_m)$ is norm-euclidean for prime m. Due to the large number of computations involved, a computer program, written in 1977 ANSI standard FORTRAN, was written to carry out this plan. This chapter consists of an informal description of the program and a verification that it does carry out the plan of attack outlined in the previous chapter. Throughout this chapter we will use bold type for the program's names for variables and routines. The program listing can be found in Appendix A and flowcharts for each routine in Appendix B. We will proceed on a procedure by procedure basis.

§1 The Control Routine

The control routine handles bookkeeping and directs the generation and checking of the points of the sufficient set.

First three files are opened for the program to write onto: (i) **buginfo**, onto which all debugging information is written; (ii) **flunk**, onto which all points which are not passed are written; and (iii) **runinfo**, onto which output is written periodically to reassure the operator that the program is still running.

The next task performed is to input the following three quantities: (i) ***m***, the value of the modulus; (ii) ***d***, the size of the partition; and (iii) ***above***, a bound for the final checker.

Next, all of the counts are initialized, and six flags are inputted to control the printing of periodic information. Finally, the three procedures which need initialization are given their initialization call. This completes the initialization section of the control routine.

After initialization is the pattern generation loop. Immediately before the loop the pattern generator is initialized and the first pattern is returned. The top of the loop checks to see if the pattern generator has returned a pattern or if all patterns have been generated. If all patterns have been generated, the program proceeds to the finalization section. Else a count of the number of returned patterns is incremented, and the cell checking routine is called to determine if the pattern lies in the fundamental cell or not. If the pattern lies within the fundamental cell, the pattern is written onto the **runinfo** file, and the point generation loop is executed. Finally, the bottom of the pattern generation loop generates a new pattern and loops back to the top.

The point generation loop is invoked from within the pattern generation loop. A count is kept of the number of points generated and each point is checked after generation. There are three point checkers which are invoked successively until the point either passes one of them or is flunked by all three. If the point passes a check the program proceeds to the bottom of the point generation loop. Else, after each flunk a counter of the number of points which have flunked that checker is incremented and the next checker is called. If the point flunks all three checks, its coordinates are written onto the **flunk** and **runinfo** files.

The end of the control routine writes all counts onto the **runinfo** file. If no point flunked the third and final checker, all points have passed and a congratulatory message is written, else a consolation message is written. Finally all files are closed and the program terminates.

§2 The Pattern Generator

The pattern generator generates and returns patterns for all points in the sufficient set S defined in (2.83). To do this, it first generates all reduced patterns. Then it subjects them to a norm check. Only those patterns whose norm is larger than the L of Definition 2.17 and smaller than the bound on the radius of F of Lemma 2.15 are returned.

The ordering of Definition 3.06 and the algorithm implicit in Proposition 3.07 are standard and may be found in [N, ch. 5, pp 40-46]. The proofs given are the creations of this author. They are elementary and are included here for the sake of completeness. Additional norm calculations have been intertwined with the combinatorial algorithm in order to minimize the running time of the program.

The pattern generator is called with three input variables and four output variables. The input variables are ***m***, ***d***, and ***pflag***. The meanings of ***m*** and ***d*** are the same as that in the control loop. The logical variable ***pflag*** controls the printing of debugging information. The output variables are ***a***, ***flag***, ***sum***, and ***count***. The pattern is returned in the integer array ***a***. The logical variable ***flag*** indicates if all patterns have been returned or not. The meaning of the integer variable ***sum*** will be given below. Finally the integer variable ***count*** counts the total number of patterns generated -- including those which flunk the norm checks and are not returned.

There are two internal integer constants: ***lbound*** and ***ubound***. These are the constants on the left and right sides of the inequalities (2.42) and are used to check the norm of a pattern. The two important internal variables are ***ssum*** and ***mnorm***. If the point $x \in (d^{-1})R$ is given by $x_1, \ldots, x_m$ in standard form with pattern $a = (a_0, \ldots, a_{d-1})$, then the values of ***sum***, ***ssum***, and ***mnorm*** are given by:

$$\boldsymbol{sum} = \sum_{i=1}^{m} dx_i = \sum_{j=0}^{d-1} ja_j, \tag{3.01}$$

$$\boldsymbol{ssum} = \sum_{i=1}^{m} (dx_i)^2 = \sum_{j=0}^{d-1} j^2 a_j, \tag{3.02}$$

and

$$\boldsymbol{mnorm} = (d\|x\|)^2 = \boldsymbol{m} * \boldsymbol{ssum} - \boldsymbol{sum} * \boldsymbol{sum}. \tag{3.03}$$

Thus ***sum*** is the sum of the coordinates of a point x with pattern a, ***ssum*** is the sum of the squares of the coordinates, and ***mnorm*** is the square of the norm with respect to the trace form. All three quantities have been multiplied by an appropriate power of d so that they are integers. The variables ***sum*** and ***ssum*** are used to compute ***mnorm***, and ***mnorm*** is the expression which appears in the middle of the inequalities (2.42).

The pattern generator has two entries: (i) **spgen** to initialize and create the first pattern, and (ii) **pgen** to create the next pattern.

The initialization section **spgen** computes the two constants for later use: ***lbound*** and ***ubound***. It then computes the first pattern ***a*** and the values of ***sum*** and ***ssum*** for this pattern. There are two methods to compute the first pattern ***a***: (i) the user may input an

arbitrary starting pattern, or (ii) the user may allow the machine to select its own starting pattern. The machine starting pattern is

$$(a_0, \ldots, a_{d-1}) = (m, 0, \ldots, 0). \tag{3.04}$$

After selection of starting pattern, ***flag*** is set equal to **.true.** and the pattern generator returns to the calling program.

The next pattern section **pgen** generates the next pattern a and the new values of ***sum***, ***ssum***, and ***mnorm*** for this pattern. If ***lbound*** is less than ***mnorm*** and ***mnorm*** is less than or equal to ***ubound***, then the inequalities (2.42) are satisfied. Thus ***flag*** is set equal to **.true.** and the routine returns to the calling program. Else by Lemma 2.40 the pattern cannot belong to the sufficient set S and the routine loops to compute another pattern. If there are no more patterns, the value of ***flag*** is set equal to **.false.** and the routine returns to the calling program.

In order to verify that all patterns for the sufficient set S are considered, we will need the following ideas.

First recall from Definition 2.36 that a pattern $a = (a_0, \ldots, a_{d-1})$ is a d-tuple of non-negative integers whose sum is m under the additional restriction that a_0 is positive. To eliminate this last restriction, we will subtract one from the first coordinate as follows:

Definition 3.05. Given a point $x \in (d^{-1})R$ with pattern $a = (a_0, \ldots, a_{d-1})$, the **essential pattern** of x (or of a) is the d-tuple $b = (b_0, \ldots, b_{d-1})$ where $b_0 = a_0 - 1$ and for $j > 0$, $b_j = a_j$.

Every d-tuple of non-negative integers $b = (b_0, \ldots, b_{d-1})$ which sum to $m-1$ is the essential pattern of the pattern $a = (1+b_0, b_1, \ldots, b_{d-1})$. Thus to generate all patterns

a, it suffices to generate all essential patterns b. To do this we totally order the set of essential patterns.

Definition 3.06. Let $b = (b_0, \ldots, b_{d-1})$ and $c = (c_0, \ldots, c_{d-1})$ be essential patterns with $b \neq c$. Define $k = \max\{ j: b_j \neq c_j \}$. Then if $b_k < c_k$, we say that $b < c$. This ordering is called the **(reverse) lexicographic ordering**.

It is clear that the machine starting pattern (3.04) is minimal with respect to this ordering.

The essential patterns are generated in lexicographic order. To do this, we need to characterize the immediate successor of an essential pattern.

Proposition 3.07. Let $b = (b_0, \ldots, b_{d-1})$ be an essential pattern. Define i by $i = \min\{ j: b_j > 0 \}$. If $i = d-1$ then b is maximal. Else, when $i < d-1$, define the d-tuple $c = (c_0, \ldots, c_{d-1})$ by

$$c_j = \begin{cases} b_j & j > i+1 \\ b_j + 1 & j = i+1 \\ 0 & 0 < j \leq i \\ b_i - 1 & j = 0 \end{cases} \tag{3.08}$$

Then c is the essential pattern that is the immediate successor of b.

Proof: If $i = d-1$, then $b = (0, \ldots, 0, m-1)$ which is clearly maximal.

For the other case, note that by the definition of i, b_i is positive, so that all of the c_j's are non-negative integers. Further, from (3.08) and the definition of i, $c_j = 0$ for $0 < j \leq i$ and $b_j = 0$ for $0 \leq j < i$. Clearly we also have

$$\sum_{j=0}^{d-1} c_j = \sum_{j=0}^{d-1} b_j = m\text{-}1.$$

Thus c is an essential pattern.

Now, as $b_j = c_j$ for $j > i + 1$ and $b_{i+1} < c_{i+1}$, we have $b < c$. Thus c is a successor to b. We need show that it is the immediate successor. Thus suppose that $e = (e_0, \ldots, e_{d-1})$ is an essential pattern with $b \leq e \leq c$.

Since $b_j = c_j$ for $j > i + 1$, we must have $b_j = e_j = c_j$ by Definition 3.06. Similarly, since $b_{i+1} + 1 = c_{i+1}$, we have two cases.

Case 1. $b_{i+1} = e_{i+1}$. In this case we will show that $b = e$. To see this note that $b_j = e_j$ for $j \geq i+1$. Hence since $b \leq e$, then $b_i \leq e_i$. But for $j < i$, we have that $b_j = 0$. Thus

$$m\text{-}1 = \sum_{j=0}^{d-1} b_j = \sum_{j=\mathrm{i}}^{d-1} b_j \leq \sum_{j=\mathrm{i}}^{d-1} e_j \leq \sum_{j=0}^{d-1} e_j = m\text{-}1.$$

Hence equality holds throughout. In particular,

$$\sum_{j=0}^{i-1} e_j = 0.$$

But as $e_j \geq 0$, this implies that for $j < i$ we have $e_j = 0 = b_j$. Thus we have that $e_j = b_j$ for $j \neq i$ and hence $e_i = b_i$, and thus $e = b$.

Case 2. $e_{i+1} = c_{i+1}$. In this case we will show that $e = c$. We will argue by contradiction and thus suppose that $e < c$. Define $k = \max\{ j: e_j \neq c_j \}$, so that $e_k < c_k$. Note that since both the e_j's and c_j's add up to m-1, we have that $k > 0$. But by the case 2 assumption, $k \leq i$. Thus using (3.08) we have that $0 \leq e_k < c_k = 0$ which is impossible.

This completes the proof of Proposition 3.07.

Proposition 3.07 gives the method used for computing the next pattern. First $i = \min\{ j: b_j > 0 \}$ is computed. Then the next essential pattern is generated by incrementing b_{i+1}, setting b_0 equal to b_i-1, and, if $i > 0$, setting b_i equal to 0.

We also have the relationship between the old and new values of ***sum*** and ***ssum***, whose proof, being immediate, is omitted.

Lemma 3.09. Let a be a pattern and let ***sum*** and ***ssum*** be defined by (3.01) and (3.02) respectively. Let b be the essential pattern for a, define $i = \min\{ j: bj > 0 \}$, and suppose $i < d$-1. Let the values of ***sum*** and ***ssum*** for the immediate successor to a be denoted by ***newsum*** and ***newssum*** respectively. Then

$$\textbf{\textit{newsum}} = \textbf{\textit{sum}} + (i+1) - ib_i \tag{3.10}$$

and

$$\textbf{\textit{newssum}} = \textbf{\textit{ssum}} + (i+1)^2 - i^2 b_i. \tag{3.11}$$

Lemma 2.40, Definition 3.04, and Proposition 3.07 show that the pattern generator, when started with the machine starting pattern, returns the pattern of all points in our sufficient set.

§3 Checking The Fundamental Cell

The cell checker determines whether a pattern lies within the fundamental cell or not.

Recall that the sufficient set S consists of points $x \in K$ which satisfy the four restrictions: (i) $x \in (d^{-1})R$, (ii) $\| x \| \geq L$, (iii) $x \in F$, and (iv) $x \in \Sigma$. As the pattern generator only returns patterns satisfying (i) and (ii), this cell restriction is the last one computable in terms of the pattern of a point.

Further note that the pattern generator only returns patterns of reduced points. As Lemma 2.33 says that all members of the fundamental cell F are reduced, no points of our sufficient set S will be skipped by considering only reduced points.

The cell checker is called with five input variables and one output variable. The input variables are: ***m***, ***d***, ***a***, ***sum***, and ***pflag***. The meanings of ***m*** and ***d*** are as usual. The variables ***a*** and ***sum*** are defined in the pattern generator and have the same meanings here. The variable ***pflag*** is a logical variable to control the printing of debugging information. The output variable is ***flag***. This is a logical variable whose value indicates whether the pattern lies in the fundamental cell or not.

The cell routine has two entries: **scell** for initialization purposes and **cell** to perform the actual checking.

The initialization section **scell** simply stores the values of ***m***-1 and ***d***-1 in ***m1*** and ***d1*** respectively.

The cell checking section **cell** is based on Proposition 2.62. The d-1 values of k, defined in (2.63), are computed starting with k_{d-1} and ending with k_1. Translating (2.63) into a recursive statement, we get

$$k_j = a_j + k_{j+1}. \tag{3.12}$$

In order to use only integer arithmetic, the routine calculates $h = d^* g_x$ instead of g_x (the division by 2 in the definition of g_x will be taken care of momentarily). As we need to check if g_x is non-negative, this will not affect things.

Writing $h(j) = d^* g_x(k_j)$, equation (2.64) translates into

$$h(j) = h(j+1) + a_j\left(\mathbf{sum} + d\,\frac{m-a_j}{2} - m\ j - d\ k_{j+1}\right).$$

Now using (3.12) we can recursively define $h(j)$ by

$$h(j) = h(j+1) + \frac{d\,a_j\,(m + a_j)}{2} - a_j\,(\,m\,j + d\,k_j - \mathbf{sum}\,)\,. \tag{3.13}$$

Note that since m is odd, $a_j\,(\,m + a_j\,)$ is even, so this is indeed a calculation which can be performed using integer arithmetic.

Thus the cell checking routine initializes ***k*** and ***h*** equal to 0 and ***j*** equal to ***dl***. Then it calculates the next values of ***k*** and ***h*** using (3.12) and (3.13). If ***h*** is negative, then the pattern ***a*** cannot lie in the fundamental cell and the routine returns with ***flag*** set equal to **.false.**. Else ***j*** is decremented. If ***j*** is less than 1, all checks have been successfully passed and the pattern ***a*** does lie in the fundamental cell. Thus ***flag*** is set equal to **.true.** upon return.

§4 The Point Generator

The point generator generates all points of the sufficient set S which have a given pattern. To do this, it first makes a selection for the pattern which is designed to minimize the number of points to be generated. Then the routine generates all points in lexicographic order with this pattern and selection. The final restriction on membership in the sufficient set is thus accounted for by this routine.

Definition 3.15 and Proposition 3.21 are from [R, sect. 2.17, pp 65-67]. They have been slightly modified to handle the repetitions of coordinates in points with a given pattern. The proof given for Proposition 3.21 was suggested to us by B. Sagan. Additional sorting calculations have been intertwined with the combinatorial algorithm in order to minimize the running time of the program.

The point generator is called with four input variables and three output variables. The four input variables are ***m***, ***d***, ***a***, and ***pflag*** with their usual meanings. The pattern ***a*** has already been passed by the cell checker. The three output variables are ***x***, ***sort***, and ***flag***. The integer array ***x*** holds the coordinates of the point returned. These coordinates are multiplied by ***d*** so that they are integers. The integer array ***sort*** is used to order the coordinates of the point in ***x***. The logical variable ***flag*** indicates if all points have been generated or not.

The significant internal variable is ***mort***. This is an integer array and is the inverse of ***sort***. It is used to compute the new value of ***sort*** for the next point generated.

The point generating routine has two entries: (i) **spoint** for initialization purposes, and (ii) **point** to generate the next point.

The initialization section **spoint** is called once for each pattern. It makes the selection for the pattern and generates the starting point. Recall that a selection $(j', j'')_a$ for a pattern $a = (a_0, \ldots, a_{d-1})$ is a pair of indices, not necessarily distinct, such that:

$$a_{j'} \geq 1 \;\&\; a_{j''} \geq 1 \quad j' \neq j''$$
$$a_{j'} \geq 2 \qquad\qquad\quad j' = j''$$

As remarked in the previous chapter, the number of points generated is minimized by choosing a selection (j', j'') which minimize $a_{j'}$ and $a_{j''}$ subject to the above restriction.

Thus the initialization section first searches for indices s and ss such that a_s is the smallest positive a_j and a_{ss} is the second smallest positive a_j. If $a_s > 1$, then the selection chosen is $(j', j'') = (s, s)$. Else $a_s = 1$ and the selection chosen is $(j', j'') = (s, ss)$. In both cases, x_1 is set equal to j' and x_m is set equal to j''. These coordinates remain fixed throughout the generation of all (j', j'') selected points by the pattern generator. Next the middle coordinates $x_2, \ldots, x_{m-1}$ are filled in in weakly increasing order: that is with $x_2 \leq \ldots \leq x_{m-1}$.

Then the array ***sort*** is computed. This array orders the coordinates of the point x, such that

$$x_{sort(1)} \leq x_{sort(2)} \leq \ldots \leq x_{sort(m)}.$$

Next the inverse array ***mort*** is computed. The final action of the initialization section ***spoint*** is to set ***flag*** equal to **.true.** and return to the calling program.

The next point section ***point*** generates the next point x and the new values of ***mort*** and ***sort*** for this point. The points are generated in lexicographic order. If there are no more points for this pattern, the variable ***flag*** is set equal to **.false.** and the program

returns. Else the next point is created by successively transposing coordinates. During the creation of the next point, the arrays ***mort*** and ***sort*** are updated.

In order to verify that all points for the pattern a are considered, we will need the following ideas.

Definition 3.14. Let a be a pattern and (j', j'') be a selection for the pattern a. The set of all $x \in (d^{-1})R$ which have pattern a and are (j', j'') selected will be denoted Ω.

Throughout the remainder of this section, for each $x \in \Omega$ we will let $x_1, \ldots, x_m$ be the standard form for x. Note that for each $x \in \Omega$, since x is (j', j'') selected we have $x_1 = j'/d$ and $x_m = j''/d$. In particular, each $x \in \Omega$ is uniquely determined by the middle coordinates x_k for $2 \le k \le m-1$.

The following definition parallels that of Definition 3.06 and totally orders Ω. Note that this ordering is, for no good reason, from the opposite end of the m-tuple.

Definition 3.15. Let $x \in \Omega$ and $y \in \Omega$. If $x \neq y$ we define $k = \min\{ i: x_i \neq y_i \}$. Then if $x_k < y_k$, we will say that $x < y$. This ordering is called the **lexicographic ordering**.

In order to create the immediate successor of a point, we will concentrate our attention on the places where the coordinates increase. Thus we have the following definition.

Definition 3.16. Let $x \in \Omega$. We define the set I_x, called the **ascent** set, by

$$I_x = \{ k \in Z: 2 \le k \le m-2 \text{ and } x_k < x_{k+1} \}.$$

The following lemma identifies the maximal element of Ω. Its proof, being immediate, is omitted.

Lemma 3.17. Let $x \in \Omega$. I_x is the empty set if and only if x is the largest element of Ω.

When $x \in \Omega$ is not maximal, the last ascent in the coordinates x_i is of special significance. Thus we have the following definition.

Definition 3.18. Let $x \in \Omega$ be not maximal. Define the index i_x by

$$i_x = \max\{\, k: k \in I_x \,\}$$

Note that Lemma 3.17 shows that i_x is well defined.

Definition 3.19. Let $x \in \Omega$ be not maximal. Let $i = i_x$. Then $x_{i+1}, \ldots, x_{m-1}$ is called the **tail** of x. Note that

$$x_{i+1} \geq x_{i+2} \geq \ldots \geq x_{m-1}.$$

To create the immediate successor of $x \in \Omega$, the rightmost coordinate of the tail which is strictly larger than x_i is also of significance.

Definition 3.20. Let $x \in \Omega$ be not maximal. Let $i = i_x$. We define the index j_x by

$$j_x = \max\{\, j:\ i < j < m \text{ and } x_j > x_i \,\}.$$

Note that since $x_i < x_{i+1}$, j_x is a maximum over a non-empty set and hence well defined. Also we have $i_x < j_x < m$.

Proposition 3.21. Let $x \in \Omega$ be not maximal. Let $i = i_x$ and $j = j_x$. Let $y \in K$ be defined by first interchanging the coordinates x_i and x_j, and then by inverting the order of the $(i+1)^{st}$ through $(m-1)^{st}$ coordinates. That is, $y = (y_1, \ldots, y_m)$ where

$$y_k = \begin{cases} x_k & 1 \le k < i \text{ or } k = m \\ x_j & k = i \\ x_i & k = i + m - j \\ x_{i+m-k} & k \ne i + m - j \text{ and } i < k < m \end{cases} \tag{3.22}$$

Then $y \in \Omega$ and y is the immediate successor to x.

Proof. Since y is a permutation of x, y has the same pattern. As the first and last coordinates were left unchanged, y has the same selection as x and hence $y \in \Omega$.

For $k < i$, by (3.22) we have $y_k = x_k$. Also $x_i < x_j = y_i$, so that $x < y$. Thus we need to show that y is the immediate successor to x.

Suppose that $z \in \Omega$ is the immediate successor to x. Thus $x < z \le y$. We will show $z = y$. As $x_k = y_k$ for $1 \le k < i$, we have $x_k = z_k = y_k$ in this range.

To show that $z_i = y_i$, note that $x_i \le z_i \le y_i = x_j$. But if $x_i = z_i$, then the last coordinates of z are a rearrangement of the tail of x. Since the tail is weakly decreasing, any rearrangement would make $x > z$. Hence z_i is an element of the tail of x larger than x_i. It must be the smallest such, since any larger choice makes $z > y$. Thus $z_i = x_j = y_i$ as claimed.

Finally, in z we must arrange the remaining elements of x in weakly increasing order (if not, $z > y$ as before). But the remaining elements of y are in weakly increasing order, since swapping x_i and x_j leaves the tail in weakly decreasing order, and then y is

obtained by inverting the order of these elements. Hence $z_k = y_k$ for $k > i$, and Proposition 3.21 is proved.

Note also that inverting the order of the $(i+1)^{st}$ through $(m-1)^{st}$ coordinates can be accomplished by interchanging the $(i+1)^{st}$ and $(m-1)^{st}$ coordinates, then interchanging the $(i+2)^{nd}$ and $(m-2)^{nd}$, and so on.

Proposition 3.21 gives the method used to compute the next pattern. First $i = i_x$ and $j = j_x$ are computed. Then x_i and x_j are swapped. Then the loop to invert the tail is initialized by setting i equal to $i+1$ and j equal to $m-1$. Then whilst $i < j$, x_i and x_j are swapped, i is incremented, and j is decremented.

Lastly, we handle the updating of the arrays ***sort*** and ***mort***. First note that ***sort*** is a permutation of the indices $\{ 1, \ldots, m \}$, and recall that the permutation group Γ acts on K by permuting the ζ's -- not the coordinates x_i. Thus, as ***mort*** is the inverse of ***sort***, we have that ***mort*** is an ordering of x in the sense of Definition 2.55. We need to keep track of orderings under the action of the permutation group Γ. The following lemma is immediate.

Lemma 3.23. Let $x \in K$ and $Q \in \Gamma$ be a permutation such that Q orders x. Let $P \in \Gamma$ be any permutation. Then QP^{-1} orders $P(x)$.

A quick consequence of this is the following lemma which tells how to update ***sort*** and ***mort*** when two coordinates of the point are interchanged.

Lemma 3.24. Let $x \in \Omega$. Let ***mort*** be a permutation which orders x. Let i and j be distinct integers with $1 < i, j < m$. Let $y \in \Omega$ be obtained from x by interchanging x_i and x_j. Let ***newmort*** be the permutation which orders y. Then

$$\mathbf{newmort}(k) = \begin{cases} \mathbf{mort}(j) & k = i \\ \mathbf{mort}(i) & k = j \\ \mathbf{mort}(k) & \text{else} \end{cases} \tag{3.25}$$

Lemma 3.24 shows how to update ***mort*** and hence ***sort*** as x is changed, since x is modified by repeated transpositions. Then the inverse relationship ***sort*** is updated by setting $\mathbf{sort}(\mathbf{mort}(i)) = i$ and $\mathbf{sort}(\mathbf{mort}(j)) = j$.

§5 The First Checker

The first checker attempts to pass the point x. It uses the ordering computed by the point generator to create the set of m "nearby" integers N_1 of Definition 2.93. It uses single precision complex arithmetic and a single iteration of the algorithm implicit in Proposition 2.113 to bound the $M_{x,q}$ of Definition 2.98 for each $q \in N_1$. As a starting value for Proposition 2.113 it uses $\beta = r/\sqrt{s}$ where r is defined in Definition 2.97. To account for floating point error, it will pass x only if it computes that $M_{x,q} < 0.99$.

The code for the first checker has been written with a view towards minimizing execution time. Thus many invariant calculations are performed outside of loops and stored in temporary locations.

The first checker is called with five input variables and two output variables. The five input variables are ***m***, ***d***, ***x***, ***sort***, and ***pflag***. The meanings of ***m***, ***d***, and ***pflag*** are as usual. The integer arrays ***x*** and ***sort*** contain the point to be passed and sorting

information respectively. They were created by the point generator. The two output variables are ***flag*** and ***count***. The logical variable ***flag*** indicates if the point x is passed. The integer variable ***count*** counts the number of times $M_{x,q}$ is bounded.

The first checker can be entered at **schek1** for initialization purposes, or at **check1** to try to pass a particular point.

The initialization section **schek1** computes internal constants for use with all points and then returns. The eight significant internal constants are ***radius***, ***lower***, ***lower4***, ***lowrad***, ***bound***, ***bnd2s***, ***e***, and ***ed***.

The double precision constant ***radius*** is set equal to the r of Definition 2.97. The real constant ***lower*** is set equal to $r/\sqrt{s}$, the starting lower bound for Proposition 2.113. The real constants ***lower4*** and ***lowrad*** are set equal to 4****lower*** and 2****lower**radius** respectively. These are computed here as they are used inside loops later. The real constant ***bound*** is set equal to 0.99. This is the upper bound used to pass points. The real constant ***bnd2s*** is set equal to ***bound****2^s, as, to speed execution, what is actually bounded is $(2^s)M_{x,q}$.

The complex array ***e*** is set equal to the j^{th} conjugate of the i^{th} power of ζ:

$$e(i,j) = \sigma_j(\zeta^i) = \zeta^{ij}.$$

The complex array ***ed*** is set equal to ***e/d***.

The point passing section **check1** is called to try to pass the point x.

The first task is to convert the integer array ***x*** into a complex array ***xq***, where $xq(j) = \sigma_j(x\text{-}q)$, and $q = 0$.

Recall that the integer array x consists of the coordinates of x multiplied times d. That is, $x(i) = d \cdot x_i$ where $x_1, \ldots, x_m$ is the standard form for x. Thus for j with $1 \le j \le s$,

$$xq(j) = \sigma_j(x\text{-}0) = \sum_{i=1}^{m} x_i \zeta^{ij} = \sum_{i=1}^{m} x(i) * e\, d(i,j).$$

Next comes the main loop. This computes an upper bound on $(2^s)M_{x,q}$ for $q \in N_1$. At the head of the loop ***count*** is incremented. Next ***xq*** is updated to handle the next $q \in N_1$.

To understand how to get the next $q \in N_1$, recall that ***mort*** orders x in the sense of Definition 2.55 and that ***sort*** is the inverse of ***mort***. Recall also that,

$$N_1 = \{ e_{i,mort} : 1 \le i \le m \}$$

where

$$e_{i,mort} = \sum_{k=i+1}^{m} \zeta^{sort(k)}.$$

Thus we have the recursive relationship, for $i < m$, of

$$e_{i,mort} = e_{i+1,mort} + \zeta^{sort(i+1)}. \qquad (3.26)$$

Thus if it is not the first time through the loop, the new value for ***xq(j)*** is computed from the old by subtracting $\sigma_j(\zeta^{sort(i+1)})$. That is, if $i < m$, the first checker redefines ***xq(j)*** by

$$xq(j) := xq(j) - e(sort(i+1),j).$$

Next ***c(j)***, the magnitude of ***xq(j)***, is computed. Concurrently with this computation, the minimal ***c(j)*** is determined and stored in ***cmin***.

Next comes the calculation applying Proposition 2.113. This computes an upper bound for z_k.

Recall that k is an index such that $z_k = \max\{z_j\}$. Thus $z_k \geq lower$. Solving (2.114) for β_j yields that

$$\beta_j = \frac{-c_j + \sqrt{c_j^2 + 4*lower*(cmin+lower)}}{2}.$$

We store in ***temp1*** the value independent of j:

$$temp1 := 4*lower*(cmin+lower) = lower4*(cmin+lower).$$

What is computed is not β_j but $temp2 = 2\beta_j$. Thus we set

$$temp2 := -c(j) + \sqrt{c(j)*c(j) + temp1}.$$

Define ***length*** by

$$\textbf{\textit{length}} = \sum_{j=1}^{s} (2\beta_j)^2 = \sum_{j=1}^{s} (\textbf{\textit{temp2}})^2 ,$$

then Proposition 2.113 says that

$$z_k \le \frac{2*\textbf{\textit{lower}}*r}{\sqrt{\textbf{\textit{length}}}} = \frac{\textbf{\textit{lowrad}}}{\sqrt{\textbf{\textit{length}}}} .$$

This upper bound for z_k is called ***upper***.

Next upper bounds for the z_j's are calculated along with an upper bound, ***ubound***, for the value of $(2^s)M_{x,q}$. This can be done as (2.109) defines z_j as an increasing function of z_k:

$$z_j = \frac{-c_j + \sqrt{c_j^2 + 4*z_k*(\textbf{\textit{cmin}}+z_k)}}{2} .$$

Using ***temp1*** as before,

$$\textbf{\textit{temp1}} = 4*\textbf{\textit{upper}}*(\textbf{\textit{cmin}}+\textbf{\textit{upper}}),$$

we conclude that

$$2(z_j + c_j) \le c_j + \sqrt{c_j^2 + \textbf{\textit{temp1}}} .$$

Thus, defining ***ubound*** by

$$\textbf{\textit{ubound}} = \prod_{j=1}^{s} \left(c(j) + \sqrt{c(j)*c(j) + \textit{tempI}} \right),$$

we have that

$$(2^s)M_{x,q} = (2^s)f_{x,q}(z_1, \ldots, z_s) = \prod_{j=1}^{s} 2(z_j+c_j) \leq \textbf{\textit{ubound}}.$$

Finally, if ***ubound*** is less than ***bnd2s***, the point passes. That is, ***flag*** is set equal to **.true.** and the first checker returns to the calling program. Else the first checker loops for the next $q \in N_1$. If N_1 has been exhausted the point has not passed. Thus ***flag*** is set equal to **.false.** before the first checker returns.

§6 The Second Checker

The second checker attempts to pass the point x. It computes the set N_2 of Definition 2.94. Then, for each $q \in N_2$, the actual bounding of $M_{x,q}$ is done in the separate routine named **slick**.

The strategy of using separate checkers minimizes the running time of the program. The earlier, less exhaustive, checkers are also faster. Only when it is clear, by flunking the previous checker, that more attention is needed to pass a given point, is that point turned over to the next checker for a more detailed examination.

The second checker is called with four input variables and two output variables. The four input variables are ***m***, ***d***, ***x***, and ***pflag***, and the two output variables are ***flag*** and ***count***; all with the usual meaning.

The initialization section **schek2** computes two internal constants for later use and then returns.

To understand the operation of the point checking section **check2** we will need the following ideas. Throughout the rest of this section, the point $x \in S$ will be fixed, with $x_1, \ldots, x_m$ being its standard form. We start with a definition.

Definition 3.27. Let j be such that $0 \le j < d$. Define the set A_j by

$$A_j = \left\{ i:\ x_i = \frac{j}{d} \right\}$$

Note that if x has pattern $a = (a_0, \ldots, a_{d-1})$, then a_j is the number of elements in A_j.

The following pair of definitions will result in the set N_2 of Definition 2.94.

Definition 3.28. Let j be such that $0 \le j < d$. Define $p_j \in R$ and, for each subset $A \subseteq A_j$, $p_{j,A} \in R$ by

$$p_j = \sum_{i \in A_j} \zeta^{i)} \quad \text{and} \quad p_{j,A} = \sum_{i \in A} \zeta^i \ .$$

Definition 3.29. Let j be such that $0 \le j < d$. Define $q_j \in R$ and, for each subset $A \subseteq A_j$, $q_{j,A} \in R$ by

$$q_j = \sum_{i \ge j} p_i \quad \text{and} \quad q_{j,A} = p_{j,A} + \sum_{i > j} p_i \ .$$

Note that for any two subsets $A \subseteq A_j$ and $B \subseteq A_j$, we have $q_{j,A} - q_{j,B} = p_{j,A} - p_{j,B}$. It is also evident that $q_0 = 0$. It is this collection of q's which forms the set N_2.

Proposition 3.30.

$$N_2 = \{\, q_{j,A} \colon\ 0 \le j < d \text{ and } A \subseteq A_j \,\}. \tag{3.31}$$

Proof. Immediate from the definitions.

Now we can describe the second checker. The first task of the second checker is to determine the sets A_j. These are stored in the integer array b where

$$A_j = \{\, b(j,i) \colon\ 1 \le i \le a_j \,\}.$$

Next comes the main loop. This loop is indexed with the variable j which runs from d-1 down to 0. For each value of j, all non-empty subsets $A \subseteq A_j$ are generated. This is done by generating all non-empty subsets $I \subseteq \{1, \ldots, a_j\}$. These are stored in the integer array ii, which is the characteristic function of the set I.

The subsets I are generated in lexicographical order by treating ii as a binary number with each array element, $ii(k)$, as one binary digit and simply adding and carrying. If a carry occurs past the a_j^{th} digit, then all non-empty subsets $I \subseteq \{1, \ldots, a_j\}$ have been

found and the routine loops for the next j. Else the set I is given by $I = \{ k: ii(k) = 1 \}$, and the corresponding set A is given by $A = \{ b(j,k): ii(k) = 1 \}$.

Next ***count*** is incremented. Then the next $q \in N_2$ is found by modifying the previous one to account for this subset $A \subseteq A_j$. In particular, the second checker defines $q = q_{j,A}$ by setting

$$q(b(j,i)) := ii(i),$$

for i with $1 \le i \le a_j$. Here we have that $e_1, \ldots, e_m$ is the standard form for q, and the array entry $q(i)$ is the value of e_i, which is either 0 or 1. Note that as i runs from 1 to a_j, then $b(j,i)$ runs over all elements of A_j. Also for $A = A_j$, which is the last A generated for each j, $q_{j,A} = q_{j+1,\Phi} = q_{j+1}$. Thus empty subsets need not be considered.

Finally, once the $q \in N_2$ has been found, the routine **slick** is called to see if x passes with this q. The routine **slick** returns this information in the logical variable ***flag*** which is returned equal to **.true.** when x has passed and returned equal to **.false.** when x has not passed. Thus ***flag*** is tested, and if the point has passed, the second checker returns to the calling program. Else the second checker loops to calculate the next subset $A \subseteq A_j$.

Finally, if all the digits j have been exhausted, the second checker has considered all $q \in N_2$ and x has not passed. As ***flag*** must be **.false.** at this stage, the second checker simply returns.

§7 The Third Checker

The third checker attempts to pass the point x. It computes the set N_3 of Definition 2.95. Then, for each $q \in N_3$, the actual bounding of $M_{x,q}$ is done in the separate routine named **slick**.

As the set N_3 is essentially an exhaustive listing of R, the method used to generate N_3 is essentially the method used to generate the sufficient set S by the control routine. That is, the third checker first generates patterns and then points from patterns. The third checker consists of a control routine **check3**, a pattern generator **pgen3**, and a point generator **point3**. We will describe the control routine **check3** in some detail and only indicate how the pattern and point generators differ from their counterparts described in sections 2 and 4 respectively.

The third checker is called with five input variables and two output variables. The five input variables are ***m***, ***d***, ***f***, ***x***, and ***pflag***, and the two output variables are ***flag*** and ***count***. The only one needing explanation is the integer variable ***f***, which controls the size of the set N_3.

The third checker **check3** has but a single entry named **check3** and does not need to be initialized. The routine **slick**, which is used by the third checker, must have been initialized (using the entry **sslick**) before the third checker was called.

The first task of the third checker is to initialize the point checking routine **slick** (using the entry **xslick**) for this point.

Next is the pattern generation loop. Patterns are generated with the bound $1+f$ substituted for the ***d*** of the control routine. Immediately before the loop the pattern generator is initialized and the first pattern is returned. The top of the loop checks to see if the pattern generator has returned a pattern or if all patterns have been generated. If all

patterns have been generated, the point has not passed and the third checker returns to the calling program. Next the point generation loop is executed. Finally, the bottom of the pattern generation loop generates a new pattern and loops back to the top.

The point generation loop is invoked from within the pattern generation loop. The points generated are stored in the integer array q. Immediately before the loop the point generator is initialized and the first point is returned. The top of the loop checks to see if the point generator has returned a point or if all points have been generated. If all points have been generated, the program exits the point generation loop and returns to the pattern generation loop. Else a count of the number of points generated is incremented and the point checker **slick** is called. If the point passes the check, the third checker returns to the calling program. Else, the bottom of the point generation loop generates the next point for that pattern and loops back to the top.

The pattern generator **pgen3** is essentially the pattern generator of section 2. It has been modified by the removal of all norm calculations and returns all patterns that it generates. It is called with $f+1$ in place of d. Thus it returns the pattern of all integers $q \in R$, where $q_1, \ldots, q_m$ is the standard form for q and $0 \le q \le f$.

The point generator **point3** is essentially the point generator of section 4. It has been modified by the removal of all selection calculations. It thus permutes all coordinates q_1 through q_m, not just q_2 through q_{m-1} as the point generator of section 4 does. It is also called with $f+1$ in place of d. Recall that the point generator of section 4 returns integers x stored in the array x where $x(i)$ is the i^{th} coordinate of x multiplied times d. Thus the point generator of section 4 returned x where $0 \le x(i) < d$. Thus the change from $x \in (d^{-1})R$ to $q \in R$ is accomplished by assuming that the q returned by the point generator **point3** contains the exact coordinate of q.

§8 Slick, the Point Passer

The point passer bounds the value of $M_{x,q}$. Using double precision arithmetic, it uses Proposition 2.119 to compute bounds on the value of $M_{x,q}$. To start it uses the value $\beta = r$, where r is defined in Definition 2.97. For further values of β, it applies Proposition 2.113, yielding successive upper and lower bounds for $M_{x,q}$. To account for floating point error, it will pass x only if it computes that $M_{x,q} < 1 - 10^{-6}$. The point passer will flunk x if one of three events occur: (1) it computes that $M_{x,q} > 1 + 10^{-6}$, (ii) the algorithm of Proposition 2.113 fails to converge, or (iii) if the algorithm of Proposition 2.113 converges to a value in between $1 - 10^{-6}$ and $1 + 10^{-6}$.

The point passer is called with five input variables, ***m***, ***d***, ***x***, ***q***, and ***pflag***; and one output variable, ***flag***. The integer array ***q*** contains the integer $q \in R$ with respect to which x is to be passed.

The point passer has three entries: (i) **sslick** for global initialization purposes, (ii) **xslick** for initialization purposes for a particular x, and (iii) **slick** to try to pass a particular point x with respect to a particular integer q.

The global initialization section **sslick** computes internal constants for use with all points x and integers q and then returns. The eight significant internal constants are ***radius***, ***radsqd***, ***difbnd***, ***cntbnd***, ***okbnd***, ***nokbnd***, ***e***, and ***ed***.

The double precision constant ***radius*** is set equal to the r of Definition 2.97. The double precision constant ***radsqd*** is set equal to r^2. The double precision constant ***difbnd*** is set equal to 10^{-10}, a bound to indicate that convergence has occurred. The integer constant ***cntbnd*** is set equal to 100, a bound to indicate that convergence has not occurred. The double precision constants ***okbnd*** and ***nokbnd*** are set equal to 0.999999 and

1.000001 respectively. These are the upper and lower bounds used to respectively pass or flunk points.

The double precision complex array ***e*** is set equal to the j^{th} conjugate of the i^{th} power of ζ and the double precision complex array ***ed*** is set equal to ***e/d*** as in the first checker. Each array is stored as two double precision arrays containing the real and imaginary parts separately.

The initialization section **xslick** is called once for each point x. Its task is to convert the integer array ***x*** into double precision arrays ***realx*** and ***imagx***. Here the parts of the j^{th} conjugate of the point x are given by

$$realx(j) := real(\sigma_j(x)) \;\; \& \;\; imagx(j) := imag(\sigma_j(x)).$$

After the initialization section **xslick** has been called, the point passing section **slick** can be called repeatedly for differing q's to try to pass the point $x \in S$.

First $c(j)$, the magnitude of $\sigma_j(x\text{-}q)$, is computed. Concurrently with this computation, the minimal $c(j)$ is determined and stored in ***cmin***. Finally, the starting value of $\beta = r$ is stored in the variable ***approx*** and the number of iterations in ***count*** is initialized at 0.

Next comes the calculation applying Proposition 2.119. The values of β_j are computed and kept in the variable ***temp2***. Simultaneously, the values for ***length*** and ***bound*** are computed. Here

$$length = \sum_{j=1}^{s} \beta_j{}^2$$

and

$$\textbf{\textit{bound}} = \prod_{j=1}^{s} (c_j + \beta_j) = f_{x,q}(\beta_1 , \ldots , \beta_s).$$

Then the variable ***diff*** is set equal to ***length- radsqd***. This is tested in accordance to Proposition 2.119 to see if we have an upper bound on $M_{x,q}$ or a lower bound. To allow simultaneously for convergence to the true value, we check for an upper bound when ***diff*** is greater than or equal to -***difbnd***, and for a lower bound when ***diff*** is less than or equal to +***difbnd***.

If ***bound*** is less than ***okbnd***, then the point passes, ***flag*** is set equal to **.true.**, and the routine returns to the calling program. If ***bound*** is greater than ***nokbnd***, then the point flunks and ***flag*** is set equal to **.false.** before the routine returns.

Lastly, the absolute value of ***diff*** is checked. If this is greater than ***difbnd***, then convergence has not yet occurred. Thus a new value of ***approx*** is computed, in accordance with Proposition 2.113, by

$$\textbf{\textit{approx}} := \frac{\textbf{\textit{approx}} * \textbf{\textit{radius}}}{\sqrt{\textbf{\textit{length}}}} .$$

Then the routine loops. Of course, there is a fail-safe limit on the number of iterations so that the routine cannot loop forever. This is stored in ***cntbnd***.

On the other hand, if the absolute value of ***diff*** is less than or equal to ***difbnd***, then the routine has converged and the value of ***bound*** is approximately 1 -- that is, is greater than ***okbnd*** and less than ***nokbnd***. As these cases are not decidable, the point is flunked if this occurs. Thus ***flag*** is set equal to **.false.** and the routine returns.

Chapter 4

Bounding the Growth of Floating Point Error

§0 Introduction

The program written to pass all the points of the sufficient set S involves arithmetic of two kinds. The generation of points and patterns uses only integer arithmetic. Thus it is not subject to floating point error. The estimation of the maximum $M_{x,q}$ is done in an s-dimensional real vector space and may induce floating point error. The algorithms used in the routines **check1** and **slick** make allowance for floating point error. The purpose of this chapter is to prove that those allowances are sufficient.

First some notation to be used throughout this chapter. Real numbers will be denoted by Latin letters (with the exception of our m^{th} root of unity, which will still be denoted by ζ). Their machine floating point approximations will be denoted by bold Latin letters. In general, the program's names for quantities will be used. For example, the r of Definition 2.97 will be denoted *radius*, with its machine representation denoted ***radius***.

The errors in the machine approximations will be denoted by subscripted ε's. The symbol δ will stand for machine error -- for the machine on which the program ran, the single precision value of δ is 2^{-24} while the double precision value of δ is 2^{-53}. Subscripted δ's will stand for arbitrary errors smaller, in absolute value, than the machine error δ.

It is assumed that all operations are carried out with extra precision, and that the error occurs when quantities are rounded back to the machine word length. Quadratic error

terms will be neglected throughout. For the sake of brevity of exposition, equalities will be used when approximations are (technically) meant.

A certain amount of preliminary processing of invariant quantities was removed from within loops in order to minimize the running time of the program. For the purpose of analyzing the growth of floating point error, we will assume that these calculations are performed within the loops. Also, the variable names in the following discussion have been slightly modified from the actual names in the program for the sake of clarity of exposition.

This chapter will start with two preliminary sections. The first bounds the size of various quantities. The second gives a general error estimation used extensively later. The last four sections deal with the actual numerical calculations.

§1 Bounding size

We assume throughout this chapter that $x \in S$ (see Proposition 2.83) and that $x_1, \ldots, x_m$ is its standard form. We will let $x(i)$ be the machine representative for $d \cdot x_i$. Recall that this means that each $x(i)$ is an integer. We will assume throughout this section that $q \in N_3(f)$ (see Definition 2.95). We first wish to bound $|\sigma_j(x)|$, where σ_j is an element of the Galois group G.

Lemma 4.01.

$$|\sigma_j(x)| \leq \sqrt{\frac{m^2 - 1}{24}} \quad . \tag{4.02}$$

Proof. Since $x \in F$, Lemma 2.15 says

$$\| x \| \leq \sqrt{\frac{m^2 - 1}{12}} . \tag{4.03}$$

But, from the definition of the Trace form,

$$\| x \|^2 = \sum_{j=1}^{2s} | \sigma_j(x) |^2 = 2 \sum_{j=1}^{s} | \sigma_j(x) |^2 .$$

Hence, for each j,

$$| \sigma_j(x) | \leq \frac{1}{\sqrt{2}} \| x \| ,$$

and equation (4.02) follows.

Next we wish to bound $c_j = | \sigma_j(x - q) |$.

Lemma 4.04.

$$c_j \leq \sqrt{\frac{m^2 - 1}{24}} + \frac{m \cdot f}{\sqrt{8}} . \tag{4.05}$$

Proof. By Definition 2.93,

$$\sigma_j(q) = \sum_{i=1}^{m} q_i \zeta^{ij} ,$$

where $q_i \in \mathbf{Z}$ with $0 \le q_i \le f$. Setting $y_i = q_i - f/2$, we have

$$\sigma_j(q) = \sum_{i=1}^{m} y_i \zeta^{ij},$$

where $|y_i| \le f/2$. Thus,

$$\| \sigma_j(q) \|^2 = m \sum_{i=1}^{m} y_i^2 - \left(\sum_{i=1}^{m} y_i \right)^2 \le \frac{m^2 f^2}{4} . \tag{4.06}$$

Thus, combining (4.03) and (4.06), we have that

$$\| x-q \| \le \sqrt{\frac{m^2 - 1}{12}} + \frac{m \cdot f}{2} .$$

Hence, in a similar manner as in Lemma 4.01, equation (4.05) follows and Lemma 4.04 is proven.

A slight modification of the proof of Lemma 4.04 yields the following lemma, whose proof will be omitted.

Lemma 4.07. Let $c_{min} = \min\{ c_j : 1 \le j \le s \}$. Then

$$c_{min} \le \sqrt{\frac{m^2 - 1}{24s}} + \frac{m \cdot f}{\sqrt{8s}} \tag{4.08}$$

Lastly, we need to bound the output of Proposition 2.113.

Lemma 4.09. Let *approx* be a positive real number. Define

$$temp_j = \frac{-c_j + \sqrt{c_j^2 + 4 \cdot approx \cdot (approx + c_{min})}}{2} , \tag{4.10}$$

$$length = \sum_{j=1}^{s} temp_j^2 , \tag{4.11}$$

and

$$newapprox = \frac{approx \cdot radius}{\sqrt{length}} . \tag{4.12}$$

Then

$$\frac{c_{min}}{c_j} approx \leq temp_j \leq approx , \tag{4.13}$$

$$(approx)^2 \leq length \leq s \cdot (approx)^2 , \tag{4.14}$$

and

$$\frac{radius}{\sqrt{s}} \leq newapprox \leq radius . \tag{4.15}$$

Proof. Letting $y = temp_j$, we have that y is the positive solution to equation (2.116) where $x = approx$ and $c = c_{min}$. Using equations (2.117) and (2.118), we have that the secant line from (0,0) to (x,y) has greater slope than the tangent line at (0,0). Thus

the left-most inequality of (4.13) follows. The right-most inequality of (4.13) is clear from (4.10).

Noting that, for some j, $c_j = c_{min}$, (4.14) follows from (4.13). Equation (4.15) is a direct consequence of (4.14). Thus Lemma 4.09 is proven.

§2 A preliminary error estimation

Lemma 4.16. Let $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ be real numbers. Let $\boldsymbol{a}(1), \ldots, \boldsymbol{a}(n)$ and $\boldsymbol{b}(1), \ldots, \boldsymbol{b}(n)$ be their machine representations with errors given by $\boldsymbol{a}(i) - a_i = \alpha_i$ and $\boldsymbol{b}(i) - b_i = \beta_i$. Let the partial sums be recursively defined by $s_i = s_{i-1} + a_i b_i$, $\boldsymbol{s}(i) = \boldsymbol{s}(i-1) + \boldsymbol{a}(i) * \boldsymbol{b}(i)$, $s_1 = a_1 b_1$, and $\boldsymbol{s}(1) = \boldsymbol{a}(1) * \boldsymbol{b}(1)$. Then the error, $\boldsymbol{s}(n) - s_n$, on the summation can be approximated by:

$$\sum_{i=1}^{n} (b_i\alpha_i + a_i\beta_i + a_i b_i \delta_{1,i}) + \sum_{k=2}^{n} s_k \delta_{2,k} \tag{4.17}$$

Proof: Let $t_i = a_i b_i$. There exist real numbers $\delta_{1,i}$, with $|\delta_{1,i}| \leq \delta$, such that:

$$\boldsymbol{t}(i) = \boldsymbol{a}(i) * \boldsymbol{b}(i) = (a_i + \alpha_i)(b_i + \beta_i)(1 + \delta_{1,i}).$$

Thus, if we define τ_i by $\tau_i = \boldsymbol{t}(i) - t_i$, we have

$$\tau_i = a_i\beta_i + b_i\alpha_i + t_i\delta_{1,i}. \tag{4.18}$$

Define σ_i by $\sigma_i = \boldsymbol{s}(i) - s_i$. Then, for some $\delta_{2,i}$ with $|\delta_{2,i}| \leq \delta$

$$\boldsymbol{s}(i) = \boldsymbol{s}(i-1) + \boldsymbol{t}(i) = (s_{i-1} + \sigma_{i-1} + t_i + \tau_i)(1 + \delta_{2,i}).$$

Thus,

$$\sigma_i = \sigma_{i-1} + \tau_i + s_i\delta_{2,i} .$$

A simple induction gives

$$\sigma_n = \sum_{i=1}^{n} \tau_i + \sum_{k=2}^{n} s_k\delta_{2,k} . \tag{4.19}$$

Combining (4.18) and (4.19) yields (4.17), and Lemma 4.16 is proven.

§3 Initializing the single precision routine check1

We will assume throughout the next two sections that $q \in N_1(x,\textbf{\textit{mort}})$ (see Definition 2.93). Note that $N_1(x,\textbf{\textit{mort}}) \subseteq N_3(f)$ with $f = 1$.

The subroutine **check1** does two different things. It runs through those q which belong to $N_1(x,\textbf{\textit{mort}})$. Secondly, it puts an upper bound on the value of $M_{x,q}$. It is this process which will be analyzed in the next two sections.

A summary of the initializing stages of the algorithm follows. We refer the reader to §5 of Chapter 3 for a more precise description and to Appendix A for the program listing.

This section will analyze the following steps. First is an initialization, which is done in double precision with the results rounded back into single precision variables. Next the conjugates of x and of $x - q$ are computed as complex numbers. Then the magnitudes, c_j, are computed.

We will follow each result with the values for $m = 13$, when $d = 6$ and $d = 7$. These are the values of m and d for which the program was run. These values will be expressed, except as otherwise stated, in multiples of machine error, δ.

We will start with the error on x as a complex number. In order to make the estimate as sharp as possible, we will use the following definitions.

Definition 4.20. Let the real number θ be defined by $\theta = 2\pi/m$.

Definition 4.21. Let k and j be integers with $1 \le k \le m$ and $1 \le j \le s$. We define the real numbers $P_{k,j}$, $N_{k,j}$, and $M_{k,j}$ by

$$P_{k,j} = \sum_{\substack{i \\ 1 \le i \le k \\ \cos(ij\theta) > 0}} \cos(ij\theta) , \tag{4.22}$$

$$N_{k,j} = \sum_{\substack{i \\ 1 \le i \le k \\ \cos(ij\theta) < 0}} \cos(ij\theta) , \tag{4.23}$$

and

$$M_{k,j} = \max\{ P_{k,j}; -N_{k,j} \} . \tag{4.24}$$

Lemma 4.25. Let ***realx(j)*** be the machine representation of $\mathrm{Re}(\sigma_j(x))$. Let $\varepsilon_{1,j} =$ ***realx(j)***$- \mathrm{Re}(\sigma_j(x))$. Then

$$|\varepsilon_{1,j}| \le \frac{d-1}{d}\left(2\sum_{i=1}^{m}|\cos(i\theta)| + \sum_{k=2}^{m} M_{k,j}\right)\delta \tag{4.26}$$

Table 1. ε_1: the error on ***realx(j)***.

j	$d=6$	$d=7$
1	43. 6844 1168	44. 9325 3773
2	40. 6735 2500	41. 8356 2572
3	40. 1307 0182	41. 2772 9330
4	39. 5999 1409	40. 7313 4020
5	39. 1547 1817	40. 2734 2440
6	38. 9436 4966	40. 0563 2537

Proof. We use Lemma 4.16. If the real part of ***ed(i,j)*** is given by ***reald(i,j)***, then the error on this is given by $(d^{-1})\cos(ij\theta)\,\delta_{1,i,j}$, as ***ed*** is calculated with double precision and the result rounded back to a single precision quantity. Thus, as

$$\textbf{\textit{realx}}(j) = \sum_{i=1}^{m} x(i) * \textbf{\textit{reald}}(i,j)\ ,$$

the error on the summation can be approximated by

$$\sum_{i=1}^{m} x_i \cos(ij\theta)\,(\delta_{1,i,j} + \delta_{2,i,j}) + \sum_{k=2}^{m}\left(\sum_{i=1}^{k} x_i \cos(ij\theta)\right)\delta_{3,k,j}\ . \tag{4.27}$$

Now, the first term of equation (4.27) is bounded by

$$2\frac{d\text{-}1}{d}\left(\sum_{i=1}^{m} |\cos(i\theta)|\right)\delta \tag{4.28}$$

To bound the second term of equation (4.27), we need to bound the quantity

$$\left|\sum_{i=1}^{k} x_i \cos(ij\theta)\right| \tag{4.29}$$

But (4.29) is maximized when there is no additive cancellation within the summation. That is, for some choice (+ or -) of sign, x_i is $(d\text{-}1)/d$ for those values of i when $\cos(ij\theta)$ has that sign, and x_i is 0 for those values of i when $\cos(ij\theta)$ has the opposite sign. Then Definition 4.21 yields that the second term is bounded by

$$\frac{d\text{-}1}{d}\left(\sum_{k=2}^{m} M_{k,j}\right)\delta\,. \tag{4.30}$$

Combining the bounds (4.28) and (4.30) with the error (4.27) yields (4.26). Thus Lemma 4.25 is proven.

Definition 4.31. Let k and j be integers with $1 \le k \le m$ and $1 \le j \le s$. We define the real numbers $P'_{k,j}$, $N'_{k,j}$, and $M'_{k,j}$ as in Definition 4.21 by replacing cosine with sine throughout.

Now the result analogous to Lemma 4.25 can be given.

Lemma 4.32. Let ***imagx(j)*** be the machine representation of $\mathrm{Im}(\sigma_j(x))$. Let $\varepsilon_{2,j} = \boldsymbol{imagx(j)} - \mathrm{Im}(\sigma_j(x))$. Then

$$|\varepsilon_{2,j}| \le \frac{d-1}{d}\Big(2\sum_{i=1}^{m} |\sin(i\theta)| + \sum_{k=2}^{m} M'_{k,j}\Big)\,\delta \tag{4.33}$$

Table 2. ε_2: the error on ***imagx(j)***.

j	$d=6$	$d=7$
1	50. 0637 9637	51. 4941 9055
2	44. 0401 0265	45. 2983 9131
3	41. 9722 6918	43. 1714 7687
4	41. 0932 3891	42. 2673 3145
5	40. 8378 3755	42. 0046 3290
6	40. 9792 7505	42. 1501 1148

The next task of the routine **check1** is to compute x- q. Again, we will do the real part and simply give the result for the imaginary part. Also, to sharpen the estimate, we will use the following definitions.

Definition 4.34. Let k be an integer with $1 \le k \le m$. We define the real numbers $\boldsymbol{P_k}$, $\boldsymbol{N_k}$, and $\boldsymbol{M_k}$ by

$$\boldsymbol{P_k} = \max\Big\{ \sum_{i \in K} \cos(i\theta) : K \subseteq \{1, \ldots, m\} \text{ and } \#K = k \Big\}, \tag{4.35}$$

$$\boldsymbol{N_k} = \min\Big\{ \sum_{i \in K} \cos(i\theta) : K \subseteq \{1, \ldots, m\} \text{ and } \#K = k \Big\}, \tag{4.36}$$

and

$$M_k = \max\{ P_k, -N_k \}. \tag{4.37}$$

Lemma 4.38. Let ***realxq***(i) be the machine representation of $\mathrm{Re}(\sigma_j(x\text{-}q))$. Let $\varepsilon_{3,j} =$ ***realxq***(j) - $\mathrm{Re}(\sigma_j(x\text{-}q))$. Then

$$|\varepsilon_{3,j}| \le |\varepsilon_{1,j}| + \Big((m\text{-}1)\sqrt{\tfrac{m^2\text{-}1}{24}} + \sum_{i=1}^{m} |\cos(i\theta)| + \sum_{k=1}^{m-1} M_k \Big)\cdot\delta \tag{4.39}$$

Table 3. ε_3: the error on ***realxq***(j).

j	$d=6$	$d=7$
1	118. 1433 718	119. 3914 978
2	115. 1324 851	116. 2945 858
3	114. 5896 619	115. 7362 534
4	114. 0588 742	115. 1903 003
5	113. 6136 782	114. 7323 845
6	113. 4026 097	114. 5152 854

Proof. We use a modification of Lemma 4.16, as ***realxq***(j) is obtained from ***realx***(j) by repeated subtraction of the terms which comprise the conjugates of q. In particular, there exists a permutation $P \in \Gamma$ such that for each q, there is an integer k with $1 \le k < m$ and

$$realxq(j) = realx(j) - \sum_{i=1}^{k} reale(P(i),j).$$

Thus we define the partial sums by

$$s(n,j) = realx(j) - \sum_{i=1}^{n} reale(P(i),j)$$

and

$$s_{n,j} = \mathrm{Re}(\sigma_j(x)) - \sum_{i=1}^{n} \cos(\, P(i)j\theta \,) \, .$$

Then $s_{n,j} = s_{n-1,j} - \cos(P(n)j\theta)$ and $s(n,j) = s(n-1,j) - e(P(n),j)$. Thus if the error $\sigma_{n,j}$ is defined by $\sigma_{n,j} = s(n,j) - s_{n,j}$, we have

$$s(n,j) = \left(s_{n-1,j} + \sigma_{n-1,j} - (1+\delta_{1,n,j})\cos(P(n)j\theta) \right) (1+\delta_{2,n,j})$$

Thus we have the following recursive relation on the σ's:

$$\sigma_{n,j} = \sigma_{n-1,j} - \cos(P(n)j\theta)\delta_{1,n,j} + s_{n,j}\delta_{2,n,j} \, ,$$

where $\sigma_{0,j} = \varepsilon_{1,j}$. An induction yields that

$$\sigma_{n,j} = \varepsilon_{1,j} - \sum_{i=1}^{n} \cos(P(i)j\theta)\delta_{1,i,j} + \sum_{k=1}^{n} s_{k,j}\delta_{2,k,j} \, .$$

Hence we can conclude that

$$|\varepsilon_{3,j}| \le |\varepsilon_{i,j}| + \delta \left(\sum_{i=1}^{m-1} |\cos(P(i)j\theta)| + \sum_{k=1}^{m-1} |s_{k,j}| \right). \tag{4.40}$$

Now, the first term inside the parenthesis in (4.40) is bounded by

$$\sum_{i=1}^{m} |\cos(i\theta)| \tag{4.41}$$

To bound the second term, we need to bound the quantity

$$|s_{k,j}| = \left| \mathrm{Re}(\sigma_j(x)) - \sum_{i=1}^{k} \cos(P(i)j\theta) \right| \tag{4.42}$$

By Lemma 4.01, the first term of (4.42) is bounded by equation (4.02). The second term of (4.42), in absolute value, is maximized when there is as little additive cancellation as possible within the summation. Thus by Definition 4.19, the second term of (4.42) is bounded by M_k. Hence the second term inside the parenthesis in (4.40) is bounded by

$$\sum_{k=1}^{m-1} \left(M_k + \sqrt{\frac{m^2 - 1}{24}} \right). \tag{4.43}$$

Using the bounds (4.41) and (4.43) on the error in (4.40) yields (4.39). Thus Lemma 4.38 is proven.

Definition 4.44. Let k be an integer with $1 \le k \le m$. We define the real numbers P'_k, N'_k, and M'_k as in Definition 4.19 by replacing cosine with sine throughout.

Now the result analogous to Lemma 4.38 can be given.

Lemma 4.45. Let ***imagxq(j)*** be the machine representation of $\mathrm{Im}(\sigma_j(x\text{-}q))$. Let $\varepsilon_{4,j} =$ ***imagxq(j)*** $- \mathrm{Im}(\sigma_j(x\text{-}q))$. Then

$$|\varepsilon_{4,j}| \le |\varepsilon_{2,j}| + \left((m\text{-}1)\sqrt{\frac{m^2\text{-}1}{24}} + \sum_{i=1}^{m} |\sin(i\theta)| + \sum_{k=1}^{m-1} M'_k \right) \cdot \delta \qquad (4.46)$$

Table 4. ε_4: the error on ***imagxq(j)***.

j	$d=6$	$d=7$
1	124. 2113 529	125. 6417 470
2	118. 1876 592	119. 4459 478
3	116. 1198 257	117. 3190 334
4	115. 2407 954	116. 4148 879
5	114. 9853 940	116. 1521 894
6	115. 1268 315	116. 2976 680

Next, the bound on the error in computing the c_j's.

Lemma 4.47. Let $\varepsilon_{5,j} = c(j) - c_j$. Then

$$|\varepsilon_{5,j}| \le \sqrt{(\varepsilon_{3,j})^2 + (\varepsilon_{4,j})^2} + 2\left(\sqrt{\frac{m^2\text{-}1}{24}} + \frac{m \cdot f}{\sqrt{8}}\right)\delta \qquad (4.48)$$

Table 5. ε_5: the error on $c(j)$.

j	$d=6$	$d=7$
1	185. 9082 662	187. 8049 176
2	179. 4802 904	181. 1925 142
3	177. 6237 220	179. 2826 649
4	176. 6253 960	178. 2557 791
5	176. 1307 550	177. 7470 255
6	176. 0832 077	177. 6981 615

Proof.

$$c(j) = (1+\delta_{1,j}) \cdot$$

$$\sqrt{(1+\delta_{2,j})\left[(1+\delta_{3,j})\left(\mathrm{Re}(\sigma_j(x\text{-}q))+\varepsilon_{3,j}\right)^2+(1+\delta_{4,j})\left(\mathrm{Im}(\sigma_j(x\text{-}q))+\varepsilon_{4,j}\right)^2\right]}\,.$$

As the squared quantities are positive, there exists a $\delta_{5,j}$ such that

$$c(j) = (1+\delta_{1,j})(1+\delta_{5,j})\sqrt{\left(\mathrm{Re}(\sigma_j(x\text{-}q))+\varepsilon_{3,j}\right)^2+\left(\mathrm{Im}(\sigma_j(x\text{-}q))+\varepsilon_{4,j}\right)^2}\,.$$

To estimate the square root, define the function

$$f(u,v) = \sqrt{u^2 + v^2}\,.$$

Then the increment in f can be bounded by

$$|\Delta f| \le \sqrt{\Delta u^2 + \Delta v^2}\,. \qquad (4.49)$$

Letting $u = \mathrm{Re}(\sigma_j(x\text{-}q))$, $\Delta u = \varepsilon_{3,j}$, $v = \mathrm{Im}(\sigma_j(x\text{-}q))$, and $\Delta v = \varepsilon_{4,j}$, we have that $f(u,v) = c_j$, and

$$c(j) = (1+\delta_{1,j})(1+\delta_{5,j})\,(\,c_j + \Delta f\,).$$

Thus

$$\varepsilon_{5,j} = \Delta f + (\,\delta_{1,j} + \delta_{5,j}\,)\,c_j \tag{4.50}$$

Combining (4.49) and (4.50) with Lemma 4.04 yields (4.48) and Lemma 4.47 is proven.

Lemmas 4.07 and 4.47 yield the following lemma.

Lemma 4.51. Let $c_{min} = \min\{c_j : 1 \le j \le s\}$. Let ***cmin*** be the machine representation of c_{min} and let $\varepsilon_{5,min} =$ ***cmin*** $- c_{min}$. Then

$$|\,\varepsilon_{5,min}\,| \le \max_j \left\{ \sqrt{(\varepsilon_{3,j})^2 + (\varepsilon_{4,j})^2} \right\} + \frac{2}{\sqrt{s}}\left(\sqrt{\frac{m^2-1}{24}} + \frac{m}{\sqrt{8}} \right)\delta \tag{4.52}$$

Table 6. $\varepsilon_{5,min}$: the error on ***cmin***.

$d = 6$	$d = 7$
177. 3373 990	179. 2340 505

This concludes the initialization of the algorithm of **check1**. Our bounds on the error in the $c(j)$'s determine how far the algorithm of **check1** has been perturbed from the theoretical algorithm of Proposition 2.113.

§4 Point passing by the single precision routine check1

A summary of the remaining stages of the routine **check1** follows.

Once the values of the c_j's are computed and stored in the $c(j)$'s, the routine **check1** applies Proposition 2.113 to compute an upper bound for $M_{x,q}$. The routine starts with a lower bound, ***lower***, for the β of Proposition 2.113. This is transformed to an upper bound, ***upper***, for β. Lastly, the upper bound, ***ubound***, for $M_{x,q}$ is computed from ***upper***. We will show that the relative error on ***ubound*** is less than 10^{-6}.

The next lemma gives the bound on the error in the value of ***lower***.

Lemma 4.53. Let $\varepsilon_6 = lower - radius/\sqrt{s}$. Then

$$|\varepsilon_6| \le \frac{radius}{\sqrt{s}}\,\delta\,. \tag{4.54}$$

Table 7. ε_6: the error on ***lower***.

$d=6$	$d=7$
0. 1800 2057 5	0. 1543 0335 0

Proof. The proof is immediate as the calculation is performed in double precision and rounded back to the single precision quantity ***lower***.

Next the routine **check1** applies Proposition 2.113. The error in the β_j's of Proposition 2.113 is bounded in the following lemma.

Lemma 4.55. Define $templ_j$ by

$$templ_j = -c_j + \sqrt{c_j^2 + 4 \cdot lower \cdot (lower + c_{min})} \quad . \tag{4.56}$$

Let ***templ(j)*** be the machine representation of $templ_j$ and define the error $\varepsilon_{7,j}$ by $\varepsilon_{7,j} = ***templ(j)*** - templ_j$. Then

$$| \varepsilon_{7,j} | \leq 2| \varepsilon_6 | + 2| \varepsilon_{5,j} | + | \varepsilon_{5,min} | + \left(8 \cdot \frac{radius}{\sqrt{s}} + 2c_j + c_{min} \right)\delta \tag{4.57}$$

Table 8. ε_7: the error on ***templ(j)***.

j	$d = 6$	$d = 7$
1	568. 3945 397	573. 8273 219
2	555. 5385 882	560. 6025 151
3	551. 8254 514	556. 7828 163
4	549. 8287 993	554. 7290 449
5	548. 8395 175	553. 7115 376
6	548. 7444 228	553. 6138 097

Proof.

$$***templ(j)*** = (1+\delta_{1,j})\Big[-(c_j+\varepsilon_{5,j})+(1+\delta_{2,j})\Big\{(1+\delta_{3,j})\Big((1+\delta_{4,j})(c_j+\varepsilon_{5,j})^2 +$$

$$(1+\delta_{5,j})(1+\delta_{6,j})4(lower+\varepsilon_6)(1+\delta_{7,j})(lower+\varepsilon_6+c_{min}+\varepsilon_{5,min})\Big)\Big\}^{1/2}\Big].$$

As the quantities inside the square root are positive, this can be regrouped, pulling two δ's outside the root and absorbing two in with the ($lower+\varepsilon_7$) and ($c_{min}+\varepsilon_{5,min}$) terms, so that

$$\boldsymbol{temp1}(j) = (1 + \delta_{1,j}) \left[-(c_j + \varepsilon_{5,j}) + (1 + \alpha_1) \cdot \right.$$

$$\left. \sqrt{(c_j + \varepsilon_{5,j})^2 + 4(lower + \alpha_2)\big((lower + \alpha_2) + (c_{min} + \alpha_3)\big)} \right]; \qquad (4.58)$$

where

$$| \alpha_1 | \leq 2\delta , \qquad (4.59)$$

$$| \alpha_2 | \leq | \varepsilon_6 | + lower \cdot \delta , \qquad (4.60)$$

and

$$| \alpha_3 | \leq | \varepsilon_{5,min} | + c_{min} \cdot \delta . \qquad (4.61)$$

Defining the function

$$f(u,v,w) = \sqrt{u^2 + 4v(v+w)} = \sqrt{u^2 - w^2 + (2v+w)^2} ,$$

we have that

$$df = \frac{udu}{f(u,v,w)} + 2\frac{(2v+w)dv}{f(u,v,w)} + \frac{2vdw}{f(u,v,w)} .$$

Thus,

$$| \Delta f | \leq | \Delta u | + 2| \Delta v | + | \Delta w | \tag{4.62}$$

Applying (4.62) to (4.58) with $u = c_j$, $\Delta u = \varepsilon_{5,j}$, $v = lower$, $\Delta v = \alpha_2$, $w = c_{min}$, and $\Delta w = \alpha_3$, we have that

$$\boldsymbol{templ(j)} = (1 + \delta_{1,j}) \left[-(c_j + \varepsilon_{5,j}) + (1 + \alpha_1) (f + \Delta f) \right] .$$

so that

$$\boldsymbol{templ(j)} = templ_j + \Delta f - \varepsilon_{5,j} + f \alpha_1 + templ_j \delta_{1,j} . \tag{4.63}$$

Combining (4.59), (4.60), (4.61), (4.62), and (4.63) we get

$$| \varepsilon_{7,j} | \leq 2| \varepsilon_6 | + 2| \varepsilon_{5,j} | + | \varepsilon_{5,min} | +$$

$$\left(2 \cdot lower + c_{min} + 2f + templ_j \right) \delta$$

Finally, we need to bound f and $templ_j$. As $f = c_j + templ_j$, it will suffice to bound $templ_j$. We use Lemma 4.09 with $approx = lower$ and $temp_j = templ_j/2$. Equation (4.13) says

$$templ_j \leq 2 \cdot lower = 2 \frac{radius}{\sqrt{s}} . \tag{4.64}$$

Combining the last two equations yields (4.57), and the proof of Lemma 4.55 is complete.

The next task of the routine **check1** is to sum the squares of the ***templ(j)***'s. This quantity is stored in ***length***, whose error will be bounded next.

Lemma 4.65. Let $\varepsilon_8 =$ ***length*** $-$ *length*. Then

$$|\varepsilon_8| \le 4 \cdot \frac{radius}{\sqrt{s}} \sum_{j=1}^{s} |\varepsilon_{7j}| + (2s^2 + 6s - 4) \cdot \frac{(radius)^2}{s} \delta \ . \tag{4.66}$$

Table 9. ε_8: the error on ***length***.

$d = 6$	$d = 7$
2396. 3272 16	2072. 1575 44

Proof. We will use Lemma 4.16. Equation (4.17) translates as

$$\varepsilon_8 = \sum_{j=1}^{s} templ_j \left(2\,\varepsilon_{7j} + templ_j\,\delta_{1j}\right) + \sum_{k=2}^{s} \sum_{j=1}^{k} (templ_j)^2\,\delta_{2j}$$

Using the bound of (4.64) results in (4.66), and Lemma 4.65 is proven.

The next quantity computed by **check1** is ***upper***.

Lemma 4.67. Let $\varepsilon_9 =$ ***upper*** $-$ *upper*. Then

$$|\varepsilon_9| \le \frac{s}{8 \cdot radius} |\varepsilon_8| + 3 \cdot radius \cdot \delta \tag{4.68}$$

Table 10. ε_9: the error on ***upper***.

$d=6$	$d=7$
4077. 0923 66	4112. 9440 48

Proof. As ***lowrad*** is computed in double precision and rounded back to a single precision quantity, $\textbf{\textit{lowrad}} = 2\cdot lower\cdot radius\cdot(1+\delta_1)$. Thus

$$\textbf{\textit{upper}} = (1+\delta_1)(1+\delta_2)(1+\delta_3)\frac{2\cdot lower\cdot radius}{\sqrt{length + \varepsilon_8}}. \tag{4.69}$$

Letting

$$f(u) = \frac{2\cdot lower\cdot radius}{\sqrt{u}},$$

we have that

$$df = -\frac{lower\cdot radius}{u\cdot\sqrt{u}}du. \tag{4.70}$$

and

$$\textbf{\textit{upper}} = f + \Delta f + (\delta_1 + \delta_2 + \delta_3)\cdot f \tag{4.71}$$

Letting $u = length$ and $\Delta u = \varepsilon_8$, we need a lower bound on length. We use Lemma 4.09 with $approx = lower$ and $temp_j = temp1_j/2$. Equation (4.14) says

$$4 \cdot lower^2 \le length \ . \tag{4.72}$$

Thus, applying this to (4.70), we have

$$|\Delta f| \le \frac{radius}{8 \cdot lower^2} |\varepsilon_8| = \frac{s}{8 \cdot radius} |\varepsilon_8| .$$

Directly from Lemma 4.09, we have that $upper \le radius$. Applying this and (4.72) to (4.71) yields (4.68) and Lemma 4.67 is proven.

Next, the routine **check1** calculates values for ***temp2(j)***. These are only slightly modified from the calculations for ***temp1(j)***, so we will simply give the bound on the error and omit the proof.

Lemma 4.73. Define $temp2_j$ by

$$temp2_j = c_j + \sqrt{c_j^2 + 4 \cdot lower \cdot (lower + c_{min})} \ . \tag{4.74}$$

Let ***temp2(j)*** be the machine representation of $temp2_j$ and define the error $\varepsilon_{10,j}$ by $\varepsilon_{10,j} =$ ***temp2(j)*** $- temp2_j$. Then

$$|\varepsilon_{10,j}| \le 2|\varepsilon_9| + 2|\varepsilon_{5,j}| + |\varepsilon_{5,min}| + (8 \cdot radius + 4c_j + c_{min}) \tag{4.75}$$

Table 11. ε_{10}: the error on ***temp2(j)***.

j	$d=6$	$d=7$
1	8738. 7906 26	8815. 6799 90
2	8725. 9346 74	8802. 4551 84
3	8722. 2215 37	8798. 6354 85
4	8720. 2248 85	8796. 5817 13
5	8719. 2356 04	8795. 5642 06
6	8719. 1405 09	8795. 4664 78

Lastly, the routine **check1** computes the value of ***ubound***. The relative error in this calculation is bounded in the following proposition. We use relative error here as we are looking for an error of less than 0.01 on $M_{x,q}$ when it is close to 1, and we actually compute $2^s M_{x,q}$.

Proposition 4.76. Let $\varepsilon_{11} = (\textit{ubound} - \textit{ubound})/\textit{ubound}$. Then

$$|\varepsilon_{11}| \le \left(\frac{\sqrt{s}}{2 \cdot radius} \sum_{j=1}^{s} |\varepsilon_{10,j}| + (s-1)\,\delta \right) \tag{4.77}$$

Table 12. ε_{11}: the relative error on ***ubound***.

	$d=6$	$d=7$
ε_{11}	14 5392. 6810	17 1110. 7572
$\delta \cdot \varepsilon_{11}$	0. 0086 6608	0. 0101 9900

Proof. Define $\boldsymbol{ubound}(0) = 1$ and

$$\boldsymbol{ubound}(n) = \boldsymbol{ubound}(n\text{-}1) * \boldsymbol{temp2}(n)$$

Let $\rho_n = \boldsymbol{ubound}(n) - ubound_n$. Then

$$\boldsymbol{ubound}(n) = (ubound_{n-1} + \rho_{n-1})(temp2_n + \varepsilon_{10,n})(1 + \delta_n),$$

so that $\rho_1 = \varepsilon_{10,1}$ and

$$\rho_n = (temp2_n)\rho_{n-1} + (ubound_{n-1})\varepsilon_{10,n} + (ubound_n)\delta_n .$$

Inductively,

$$\rho_n = ubound_n \Big(\sum_{j=1}^{n} \frac{\varepsilon_{10,j}}{temp2_j} + \sum_{j=2}^{n} \delta_j \Big) . \tag{4.78}$$

Thus we need a lower bound on $temp2_j$. From (4.74), it is clear that $temp2_j \geq 2 \cdot upper$. But from Lemma 4.09, $upper \geq radius/\sqrt{s}$. Combining this with (4.78) yields (4.77), and Proposition 4.76 is proven.

Proposition 4.76 shows that when the routine **check1** computes that ***ubound*** is less than $0.99 \cdot 2^s$, then *ubound* is less than 2^s. As Proposition 2.113 shows that $2^s M_{x,q}$ is less than or equal to *ubound*, we can conclude in this case that x is passed by q.

§5 Initializing the double precision routine slick

Throughout the next two sections, we assume that $q \in N_3(f)$ (see Definition 2.95).

The subroutine **slick** is concerned solely with determining $M_{x,q}$. It is called from the two latter checkers: **check2** and **check3**. It uses Proposition 2.119 to compute either an upper or lower bound on $M_{x,q}$, and it uses Proposition 2.113 to obtain the next approximation.

A summary of the preliminary stages of the algorithm follows. We refer the reader to §8 of Chapter 3 for a more precise description and to Appendix A for the program listing. First the constants ***angle***, ***radius***, and ***radsqd*** are computed. Next the arrays ***e*** and ***ed*** are computed. After this initialization, the conjugates of x and x- q and the magnitudes, c_j, are computed.

As before, we will follow each result with two sets of values: (i) $m = 13$, $d = 6$, $f = 2$; and (ii) $m = 13$, $d = 7$, $f = 1$. These are the values of m, d, and f for which the program was run.

We start with three elementary preliminaries, whose proofs will be omitted.

Lemma 4.79. Let $\varepsilon_{12} =$ ***angle*** $- \theta$. Then

$$| \varepsilon_{12} | \leq 2 \cdot \theta \cdot \delta .$$

Lemma 4.80. Let $\varepsilon_{13} =$ ***radius*** $-$ *radius*. Then

$$| \varepsilon_{13} | \leq 2 \cdot radius \cdot \delta .$$

Lemma 4.81. Let $\varepsilon_{14} =$ ***radsqd*** $-$ $(radius)^2$. Then

$$| \varepsilon_{14} | \leq 2 \cdot radius \cdot | \varepsilon_{13} | + (radius)^2 \cdot \delta .$$

Table 13. the errors on ***angle***, ***radius***, and ***radsqd***.

	$d=6$	$d=7$
ε_{12}	0. 9666 4389	0. 9666 4389
ε_{13}	0. 8819 1710	0. 7559 2894
ε_{14}	0. 9722 2222	0. 7142 8571

The errors for ζ^{ij} and $d^{-1}\zeta^{ij}$ will be bounded next. As the calculations for the imaginary parts are entirely analogous to those for the real parts, those results will be given without proofs. First, the error in representing ζ^{ij}. Note that this is independent of d and f.

Lemma 4.82. Let $\varepsilon_{15,i,j} = \textbf{\textit{reale}}(i,j) - \mathrm{Re}(\zeta^{ij})$. Let h be the integer satisfying: (i) $0 \le h < m$, (ii) $h \equiv ij$ (modulo m). Then

$$| \varepsilon_{15,i,j} | \le h \cdot | \sin(h\theta) | \cdot \left(| \varepsilon_{12} | + \theta \cdot \delta \right) + | \cos(h\theta) | \cdot \delta \ . \tag{4.83}$$

Table 14. the error on ***reale***(i,j) and ***image***(i,j).

h	ε_{15}	ε_{16}
1	1. 5592 8875	1. 7486 0416
2	2. 9546 6173	2. 4703 3282
3	4. 4387 1855	1. 5170 3108
4	5. 7775 7133	2. 9916 7613
5	5. 5560 3676	6. 0896 9774
6	3. 0529 3905	8. 6863 1047
7	3. 3999 3858	10. 0941 4294
8	8. 4405 5237	9. 3456 4278
9	12. 5562 7939	5. 5625 0100
10	14. 5144 7625	2. 7404 4956
11	13. 6943 4816	9. 8834 0312
12	8. 9714 4872	15. 8712 9505
13	1. 0000 0000	0. 0000 0000

Proof.

$$\textbf{\textit{reale}}(i,j) = (1+\delta_{1,i,j})\cos\big((1+\delta_{2,i,j})h(\theta+\varepsilon_{12})\big).$$

Letting $f(u) = \cos(u)$, $u = h\theta$, and $\Delta u = h\varepsilon_{12} + h\theta\delta_{2,i,j}$, we have that

$$\textbf{\textit{reale}}(i,j) = (1+\delta_{1,i,j})\big(\cos(h\theta) - (h\varepsilon_{12} + h\theta\delta_{2,i,j})\sin(h\theta)\big).$$

Equation (4.83) follows immediately.

Lemma 4.84. Let $\varepsilon_{16,i,j} = \textbf{\textit{image}}(i,j) - \mathrm{Im}(\zeta^{ij})$. Let h be as above. Then

$$|\varepsilon_{16,i,j}| \le h\cdot|\cos(h\theta)|\cdot\big(|\varepsilon_{12}| + \theta\cdot\delta\big) + |\sin(h\theta)|\cdot\delta \quad . \tag{4.85}$$

Next, the error in representing $d^{-1}\zeta^{ij}$.

Lemma 4.86. Let $\varepsilon_{17,i,j} = \textbf{\textit{reald}}(i,j) - \mathrm{Re}(d^{-1}\zeta^{ij})$. Let h be as above. Then

$$|\varepsilon_{17,i,j}| \le \frac{1}{d}\big(|\varepsilon_{15,i,j}| + |\cos(h\theta)|\cdot\delta\big) \quad . \tag{4.87}$$

Table 15. ε_{17}: the error on ***realed(i,j)***.

h	$d=6, f=2$	$d=7, f=1$
1	0. 4074 5746 3	0.3492 4925 4
2	0. 5871 2108 0	0.5032 4664 0
3	0. 7598 7587 2	0.6513 2217 6
4	1. 0220 2937 0	0.8760 2517 4
5	1. 0507 5791 8	0.9006 4964 4
6	0. 6706 4681 1	0.5748 4012 3
7	0. 7284 8006 7	0.6244 1148 6
8	1. 5315 1051 9	1.3127 2330 2
9	2. 1518 1404 7	1.8444 1204 0
10	2. 4391 6882 1	2.0907 1613 2
11	2. 3770 6881 9	2.0374 8755 9
12	1. 6428 1745 8	1.4081 2924 9
13	0. 3333 3333 3	0.2857 1428 6

Lemma 4.88. Let $\varepsilon_{18,ij} = \boldsymbol{imaged(i,j)} - \mathrm{Im}(d^{-1}\zeta^{ij})$. Let h be as before. Then

$$|\varepsilon_{18,ij}| \le \frac{1}{d}\left(|\varepsilon_{16,ij}| + |\sin(h\theta)| \cdot \delta \right) . \tag{4.89}$$

Table 16. ε_{18}: the error on ***imaged(i,j)***.

h	$d=6, f=2$	$d=7, f=1$
1	0.3688 8788 9	0. 3161 8961 9
2	0.5488 8611 5	0. 4704 7381 2
3	0.4182 8999 2	0. 3585 3427 9
4	0.6544 4873 0	0. 5609 5605 4
5	1.1254 7006 6	0. 9646 8862 8
6	1.4876 0435 6	1. 2750 8944 8
7	1.7222 4310 1	1. 4762 0837 2
8	1.6681 2757 4	1. 4298 2363 5
9	1.0829 1954 0	0. 9282 1674 9
10	0.6221 9307 3	0. 5333 0834 8
11	1.7843 9783 1	1. 5294 8385 5
12	2.7226 6970 4	2. 3337 1688 9
13	0. 0000 0000 0	0. 0000 0000 0

The following computations on the errors of ***realx(j)***, ***imagx(j)***, ***realxq(j)***, ***imagxq(j)***, and ***c(j)*** exactly parallels the work in the third section of this chapter. Thus we will give only the results and omit the proofs. We start with a bound on the machine representations of $\sigma_j(x)$.

Lemma 4.90. Let ***realx(j)*** be the machine representation of $\mathrm{Re}(\sigma_j(x))$. Let $\varepsilon_{19,j} =$ ***realx(j)***$- \mathrm{Re}(\sigma_j(x))$. Then

$$|\varepsilon_{19,j}| \le (d\text{-}1) \sum_{i=1}^{m} |\varepsilon_{17,i,j}| + \frac{d\text{-}1}{d}\Big(\sum_{i=1}^{m} |\cos(i\theta)| + \sum_{k=2}^{m} M_{k,j} \Big)\, \delta \qquad (4.91)$$

Table 17. ε_{19}: the error on ***realx(j)***.

j	$d=6, f=2$	$d=7, f=1$
1	115. 2812 947	118. 5750 460
2	112. 2704 080	115. 4781 340
3	111. 7275 848	114. 9198 016
4	111. 1967 971	114. 3738 485
5	110. 7516 012	113. 9159 327
6	110. 5405 327	113. 6988 336

Lemma 4.92. Let ***imagx(j)*** be the machine representation of $\mathrm{Im}(\sigma_j(x))$. Let $\varepsilon_{20,j} =$ ***imagx(j)*** $- \mathrm{Im}(\sigma_j(x))$. Then

$$|\varepsilon_{20,j}| \le (d\text{-}1) \sum_{i=1}^{m} |\varepsilon_{18,i,j}| + \frac{d\text{-}1}{d}\Big(\sum_{i=1}^{m} |\sin(i\theta)| + \sum_{k=2}^{m} M'_{k,j} \Big)\, \delta \qquad (4.93)$$

Table 18. ε_{20}: the error on ***imagx(j)***.

j	$d=6, f=2$	$d=7, f=1$
1	114. 2313 688	117. 4951 222
2	108. 2076 751	111. 2993 229
3	106. 1398 416	109. 1724 085
4	105. 2608 113	108. 2682 631
5	105. 0054 099	108. 0055 645
6	105. 1468 474	108. 1510 431

Next we bound the error in the machine representation of $\sigma_j(x\text{-}q)$.

Lemma 4.94. Let ***realxq(j)*** be the machine representation of $\mathrm{Re}(\sigma_j(x\text{-}q))$. Let $\varepsilon_{21,j} =$ ***realxq(j)*** $- \mathrm{Re}(\sigma_j(x\text{-}q))$. Then

$$|\varepsilon_{21,j}| \le |\varepsilon_{19,j}| + f\sum_{i=1}^{m}\left(|\varepsilon_{15,i,j}| + \delta|\cos(i\theta)| + \delta M_{i,j}\right) + \delta\cdot m\cdot\sqrt{\frac{m^2\text{-}1}{24}} \quad (4.95)$$

Table 19. ε_{21}: the error on ***realxq(j)***.

j	$d=6, f=2$	$d=7, f=1$
1	411. 5296 215	283. 8965 929
2	400. 6578 243	276. 8692 256
3	397. 9171 693	275. 2119 773
4	396. 5806 274	274. 2631 472
5	395. 8547 730	273. 6649 021
6	395. 5820 022	273. 4169 519

Lemma 4.96. Let ***imagxq*(j)** be the machine representation of $\mathrm{Im}(\sigma_j(x\text{-}q))$. Let $\varepsilon_{22,j} = \boldsymbol{imagxq(j)} - \mathrm{Im}(\sigma_j(x\text{-}q))$. Then

$$|\varepsilon_{22,j}| \le |\varepsilon_{20,j}| + f\sum_{i=1}^{m}\left(|\varepsilon_{16,i,j}| + \delta|\sin(i\theta)| + \delta M'_{i,j}\right) + \delta\cdot m\cdot\sqrt{\frac{m^2-1}{24}} \quad (4.97)$$

Table 20. ε_{22}: the error on ***imagxq*(j)**.

j	$d=6, f=2$	$d=7, f=1$
1	407. 2393 853	281. 1965 139
2	387. 4753 480	268. 1305 429
3	380. 7841 642	263. 6919 533
4	377. 6800 760	261. 6752 789
5	376. 2679 242	260. 8342 052
6	375. 9011 978	260. 7256 018

Lastly, bounds on the error in the machine representation of the c_j's.

Lemma 4.98. Let $c_j = |\sigma_j(x\text{-}q)|$. Let ***c*(j)** be the machine representation of c_j and let $\varepsilon_{23,j} = \boldsymbol{c(j)} - c_j$. Then

$$|\varepsilon_{23,j}| \le \sqrt{(\varepsilon_{21,j})^2 + (\varepsilon_{22,j})^2} + 2\left(\sqrt{\frac{m^2-1}{24}} + \frac{m\cdot f}{\sqrt{8}}\right)\delta \quad (4.99)$$

Table 21. ε_{23}: the error on $c(j)$.

j	$d=6, f=2$	$d=7, f=1$
1	602. 6413 438	414. 0696 199
2	581. 0485 401	399. 9064 591
3	574. 4345 320	395. 6335 710
4	571. 3243 721	393. 5541 026
5	569. 8251 159	392. 5407 135
6	569. 3747 589	392. 2863 114

Lemma 4.100. Let $\varepsilon_{23,min} = \boldsymbol{cmin} - c_{min}$. Then

$$|\varepsilon_{23,min}| \leq \max_j \left\{ \sqrt{(\varepsilon_{21,j})^2+(\varepsilon_{22,j})^2} \right\} + \frac{2\delta}{\sqrt{s}}\left(\sqrt{\frac{m^2-1}{24}}+\frac{m\cdot f}{\sqrt{8}}\right) \qquad (4.101)$$

Table 22. $\varepsilon_{23,min}$: the error on ***cmin***.

$d=6, f=2$	$d=7, f=1$
588. 6308 652	405. 4987 528

This completes the beginning error analysis on the algorithm of the routine **slick**. It determines, in effect, how far the algorithm implemented by **slick** has been perturbed from the theoretical algorithms of Proposition 2.113 and Proposition 2.119 that **slick** is designed to implement.

§6 Point passing by the double precision routine slick

A summary of the concluding stages of the algorithm of the routine **slick** follows. We refer the reader to §8 of Chapter 3 for a more precise description and to Appendix A for the program listing. Given a value for ***approx***, the routine **slick** computes the corresponding values of ***temp2(j)***. Simultaneously, the values of ***bound*** and ***length*** are computed. Then, ***length*** and ***bound*** are tested. If the point has neither passed nor flunked, a new value of ***approx*** is computed and the routine loops.

Next will follow a sequence of conditional results, based on the quantity

$$\varepsilon_{24} = \textbf{\textit{approx}} - approx\,. \tag{4.102}$$

We first bound the error on ***temp2(j)***, as in Lemma 4.55. As the proof is entirely analogous, it is omitted.

Lemma 4.103. Let $\varepsilon_{25,j} = \textbf{\textit{temp2(j)}} - temp2_j$. Then

$$|\varepsilon_{25,j}| \le |\varepsilon_{24}| + |\varepsilon_{23,j}| + \frac{1}{2}|\varepsilon_{23,min}| + \left(5 \cdot radius + c_j + \frac{1}{2}c_{min}\right)\delta \tag{4.104}$$

Next we bound the error on ***length***, as in Lemma 4.65. Again, as the proof is entirely analogous, it is omitted.

Lemma 4.105. Let $\varepsilon_{26} = \textbf{\textit{length}} - length$. Then

$$|\varepsilon_{26}| \le 2 \cdot radius \cdot \sum_{j=1}^{s} |\varepsilon_{25,j}| + \frac{s^2 + 3s - 2}{2}(radius)^2 \cdot \delta \tag{4.106}$$

Next we bound the relative error on ***bound***, as in Lemma 4.76, omitting the proof.

Lemma 4.107. Let $\varepsilon_{27} = (\textbf{\textit{bound}} - bound)/bound$. Then

$$|\varepsilon_{27}| \le \left(\frac{\sqrt{s}}{radius} \sum_{j=1}^{s} (|\varepsilon_{23,j}| + |\varepsilon_{25,j}| + c_j\delta) + (s-1)\delta \right) \tag{4.108}$$

Finally, we bound the error on ***diff***. The proof, being elementary, is omitted.

Lemma 4.109. Let $\varepsilon_{28} = \textbf{\textit{diff}} - diff$. Then

$$|\varepsilon_{28}| \le |\varepsilon_{14}| + |\varepsilon_{26}| + 2s\cdot(radius)^2\cdot\delta \tag{4.110}$$

Finally, we can show that the algorithm of the routine **slick** is correct. The first step is to show that if $|\textbf{\textit{diff}}|$ is large, then the sign of ***diff*** accurately reflects the sign of *diff*, and hence the appropriate conclusions on the order relationship between *bound* and $M_{x,q}$ can be drawn.

Proposition 4.111. If $\textbf{\textit{diff}} \ge 10^{-10}$ and $\textbf{\textit{bound}} < 1-10^{-6}$, then $M_{x,q} < 1$. In particular, x is passed by q.

Proof. We will first show that if $\textbf{\textit{diff}} \ge 10^{-10}$, then $diff > 0$, so that $length > (radius)^2$. To do this, it suffices to show that if $\varepsilon_{24} = 0$, then $|\varepsilon_{28}| < 10^{-10}$. Applying Lemmas 4.103, 4.105, and 4.109 with $\varepsilon_{24} = 0$, we get the following values of $\varepsilon_{25,j}$, ε_{26}, and ε_{28}.

Table 23. ε_{25}: the error on ***temp2(j)*** when $\varepsilon_{24} = 0$.

j	$d = 6, f = 2$	$d = 7, f = 1$
1	913. 4161 587	627. 4290 200
2	891. 8233 550	613. 2658 591
3	885. 2093 470	608. 9929 710
4	882. 0991 870	606. 9135 027
5	880. 5999 309	605. 9001 136
6	880. 1495 738	605. 6457 114

Table 24. ε_{26} & ε_{28}: the errors on ***length*** and ***diff*** when $\varepsilon_{24} = 0$.

	$d = 6, f = 2$	$d = 7, f = 1$
ε_{26}	4708. 5818 86	2776. 5729 16
ε_{28}	4711. 8874 72	2779. 0014 87
$\delta \cdot \varepsilon_{28}$	$5.\ 2312 \cdot 10^{-13}$	$3.\ 0853 \cdot 10^{-13}$

As this shows that, in both cases, $|\varepsilon_{28}| \leq 10^{-10}$, we can conclude that *length* > $(radius)^2$. Thus we have, by Proposition 2.119, that $M_{x,q} <$ *bound*. Thus, to prove Proposition 4.111, it suffices to show that $|\varepsilon_{27}| \leq 10^{-6}$ when $\varepsilon_{24} = 0$. Using Lemma 4.107, the following table gives the relative error on ***bound*** -- first as multiples of machine error, δ, secondly as a real number.

Table 25. ε_{27}: the relative error on ***bound*** when $\varepsilon_{24} = 0$.

	$d = 6, f = 2$	$d = 7, f = 1$
ε_{27}	4 9293. 6718 9	3 9534. 8587 4
$\delta \cdot \varepsilon_{27}$	$5.\ 4727 \cdot 10^{-12}$	$4.\ 3893 \cdot 10^{-12}$

Thus Proposition 4.111 is proven.

The following result, dual to Proposition 4.111, is proven in the same way. Thus its proof is omitted.

Proposition 4.112. If $\boldsymbol{diff} \leq -10^{-10}$ and $\boldsymbol{bound} > 1 + 10^{-6}$, then $M_{x,q} > 1$. In particular, x is not passed by q.

Lastly, we show that when $|\boldsymbol{diff}|$ is small, the $\boldsymbol{temp2}(j)$'s are good approximations to the solution of (2.107), (2.108), and (2.109), and hence ***bound*** is a good approximation to $M_{x,q}$. More precisely, we have the following proposition.

Proposition 4.113. Suppose that $|\boldsymbol{diff}| \leq 10^{-10}$. If $\boldsymbol{bound} < 1 - 10^{-6}$, then $M_{x,q} < 1$. In particular, x is passed by q. Conversely, if $\boldsymbol{bound} > 1 + 10^{-6}$, then $M_{x,q} > 1$. In particular, x is not passed by q

Proof. Each $temp2_j$ is an increasing function of ***approx***. Thus ***length*** is an increasing function of ***approx***. Setting $x = \boldsymbol{approx}$, $y_j = \boldsymbol{temp2}_j$, and $l = \boldsymbol{length}$ we have that

$$\frac{dl}{dx} = \sum_{j=1}^{s} 2y_j \frac{dy_j}{dx} .$$

Further, as $y_j = x$ for some j, and all of the y_j's are positive, we have that

$$\frac{dl}{dx} \geq 2x .$$

Let x_0 be the solution to $l(x) = (radius)^2$. Then we have

$$\frac{dl}{dx}(x_0) \geq 2 \cdot x_0 \geq 2 \frac{radius}{\sqrt{s}} \ . \tag{4.114}$$

In particular, suppose that $|\ \boldsymbol{diff}\ | \leq 10^{-10}$. Then, taking ***approx*** to be exactly equal to *approx*, Table 24 shows that $|\ \boldsymbol{diff} - diff\ | \leq 10^{-10}$, so that $|\ diff\ | < 2 \cdot 10^{-10}$. But

$$diff = length - (radius)^2 = l(x) - l(x_0) \ . \tag{4.115}$$

Thus we have a bound on the change in the function value of l. Using the lower bound (4.114) on the derivative, we can conclude that

$$|\ x - x_0\ | \leq \frac{\sqrt{s}}{2 \cdot radius} 2 \cdot 10^{-10} \ . \tag{4.116}$$

Now let us analyze the calculations which the routine **slick** performs from a slightly different viewpoint. Rather than assuming that ***approx*** is an exact value, let us view it as an approximation to x_0. That is, we suppose that we have the same value in ***approx***, but $approx = x_0$ and $\varepsilon_{24} = \boldsymbol{approx} - x_0$. Thus equation (4.116) gives an upper bound on ε_{24} which is summarized in the following table.

Table 26. ε_{24}: the error on ***approx***.

	$d = 6, f = 2$	$d = 7, f = 1$
$\delta \cdot \varepsilon_{24}$	$1.1110 \cdot 10^{-9}$	$1.2961 \cdot 10^{-9}$
ε_{24}	1000 6855. 34	1167 4664. 56

As $l(x_0) = (radius)^2$, the $temp2_j$'s are the exact solutions to equations (2.107), (2.108), and (2.109). Thus using the bounds on ε_{24} of Table 26 and Lemma 4.103, the differences between the exact $temp2_j$'s and the computed ***temp2(j)***'s are given in the following table.

Table 27. ε_{25}: the error on ***temp2(j)***.

j	$d=6, f=2$	$d=7, f=1$
1	1000 7768. 75	1167 5291. 99
2	1000 7747. 16	1167 5277. 82
3	1000 7740. 54	1167 5273. 55
4	1000 7737. 43	1167 5271. 47
5	1000 7735. 94	1167 5270. 46
6	1000 7735. 48	1167 5270. 20

Finally, as the $temp2_j$'s solve equations (2.107), (2.108), and (2.109), $bound = M_{x,q}$. Thus to prove our Proposition, we need only show that the relative error on ***bound*** is less than 10^{-6}. But, for the values of ε_{24} given in Table 26, Lemma 4.107 gives the following values for ε_{27}.

Table 28. ε_{27}: the relative error on ***bound***.

	$d=6, f=2$	$d=7, f=1$
ε_{27}	3 3357 3014. 6	4 5400 2377. 3
$\delta \cdot \varepsilon_{27}$	$3.7034 \cdot 10^{-8}$	$5.0404 \cdot 10^{-8}$

As both values are less than 10^{-6}, Proposition 4.113 is proven.

Chapter 5

Conclusions and Open Questions

§1 A summary of runs

The program written to implement the work of the previous three chapters has been run three times. All three runs were on a Sun 3/160 minicomputer of the Mathematics Department at Michigan State University.

From January 16, 1988 to January 25, 1988, a preliminary version of this program ran with $m = 13$ and $d = 7$. There were three principal differences between the preliminary version and the final version described in this dissertation: (i) The preliminary **check1** did all of its calculations, including initialization, in single precision arithmetic. (ii) In the preliminary version, the routine **check3** was run separately from the main program. (iii) The preliminary **slick** only applied Proposition 2.113 and not 2.119, and thus had somewhat different logic.

During this run, the pattern generator considered 18,110 patterns and reported back 7,887 as potentially lying within the fundamental cell. The cell checker determined that 1,486 of these patterns did indeed lie within the fundamental cell. The disposition of the points generated is summarized in the following table.

Table 29. Run summary ($d = 7$).

checker	points	estimations	passed
1	90,980,978	154,100,673	90,709,723
2	271,255	1,301,787	269,414
3	1,841	212,136	1,841

In February 1988, another preliminary version of the program ran with $m = 13$, $d = 6$, and $f = 1$. There were two main differences between this version and the previous version: (i) For this run, the routine **check3** was incorporated within the main program. (ii) The sufficient set used was the set of all reduced points with a given selection whose norms were larger than the L of Definition 2.17. This second modification was undertaken due to our temporary lack of a proof of Lemma 2.33, as it is relatively easy to show that the set of reduced points is sufficient. This resulted in a larger than necessary sufficient set.

During this second run, the pattern generator considered 6,098 patterns and returned 4,931 as having norm larger than L. The disposition of the points generated is summarized in the following table. Note that the third checker was run twice, with differing values of f.

Table 30. Run summary ($d = 6$).

checker	points	estimations	passed
1	110,782,378	469,070,258	108,554,018
2	2,228,360	18,013,137	2,176,783
3 ($f = 1$)	51,577	24,619,110	51,570
3 ($f = 2$)	7	323,802	7

Finally, the exact program described in this dissertation and whose listing appears in Appendix A, was run with $m = 13$, $d = 6$, and $f = 2$. The run started on July 20, 1988 and finished on July 21, 1988.

During this final run, the pattern generator considered 6,098 patterns and reported back 2,806 as potentially lying within the fundamental cell. The cell checker determined that 661 patterns did belong to the fundamental cell. The disposition of the points generated is summarized in the following table.

Table 31. Run summary ($d = 6, f = 2$).

checker	points	estimations	passed
1	20,450,056	86,045,968	15,200,578
2	5,249,478	22,093,825	5,237,325
3	12,153	5,338,621	12,153

In addition, a run was made with $m = 11$ on $\mathbf{Z}[\zeta_{11}]$. With $d = 5$ and $f = 2$, all 105,918 points were passed, with only 671 being reported to the second checker and 4 to the third. Of course this is no surprise, as Lenstra [Le2] proved in 1975 that this ring was norm-euclidean.

§2 Conclusions

The basic method of proof described in this dissertation of subdividing the quotient field into regions and working analytically on each region is not ours. However, the ideas of using shrunken copies of the fundamental cell of the trace form and a single subdivision are. These techniques have two principal advantages: (i) Only a single subdivision of the quotient field is necessary, so that the number of cases is computable beforehand. (ii) The

fundamental cell of the trace form is quite closely approximated by its circumscribed sphere, so that the bound on the norm computed on the sphere is quite likely to be very close to the true bound on the shrunken copy of the cell in question.

Of course, the greatest difficulty in any attack along this line lies in its computational complexity. That is, considerable effort must be expended in order to both reduce the total number of cases and to handle each case as efficiently as possible.

For the method of proof employed in this dissertation, it is the size of the variable d which governs how well the algorithm runs. In particular, extreme values of d are ill advised.

As the value of d becomes smaller, it becomes more difficult to pass points. In addition, as d decreases, the hunt for an appropriate "nearby" q becomes more extensive and the number of unsuccessful norm estimations grows, and hence the running time increases. This occurs because the regions on which the norm must be bounded become too large.

In particular, for different portions of one region, different "nearby" integers q must be chosen. This is seen most easily in the extreme case, when $d = 1$. For this case, the sufficient set consists of only one point, $x = 0$, and thus the only region is the fundamental cell itself. Now for each $q \in R$, $\psi(0\text{-}q) = \psi(p) \geq 1$ unless $q = 0$. Thus $q = 0$ is the only hope of passing the fundamental cell. However, there are elements of the fundamental cell whose norm is strictly larger than 1.

However, as d tends towards infinity, the size of the sufficient set S tends towards infinity. Thus running time also will increase without bound. The following table gives the size of the sufficient set S as a function of the size of d, for the case $m = 13$.

Table 32. The size of S when $m = 13$.

d	$\#S$
3	22,407
4	384,802
5	3,109,997
6	20,450,056
7	90,980,978
8	384,695,212
9	1,246,250,676

Thus we need to choose d minimal, subject to the restriction that points still pass. Trial runs have shown that a value of $d = 6$ is optimal when $m = 13$.

Lastly, some comments concerning the bounds on numerical error obtained in Chapter 4. As shown in Table 12, the relative error on **ubound** is somewhat high in the $d = 7$ case. This, however, should not be grounds for concern. On one hand, such error analysis as was performed always assumed the worst case. In practice, error almost never reinforces itself throughout a large calculation, and it is the fundamental stability (or lack thereof) of the underlying algorithm which determines the size of the error. On the other hand, the estimations used in the proofs were not as absolutely sharp as they might have been. Thus we feel confidant that a considerably more exhaustive analysis would bring the critical figure down from 0.0102 to below 0.01.

§3 Other attacks on $Z[\zeta_{13}]$

The method of attack described in this dissertation is not the only method which we used to try to prove that $Z[\zeta_{13}]$ is norm-euclidean. It is instead the cumulation of many different attempts which were made.

The first attempt made was very similar to the method presented. In essence, it was the method which T. Ojala [Oj] used on $Z[\zeta_{16}]$. It is outlined below.

The field $Q(\zeta_m)$ is represented as a $n = \varphi(m)$ dimensional vector space over Q. A fundamental cube, C, is given by

$$C = \{ (x_1, \ldots, x_n): \ 0 \le x_i \le 1 \}.$$

Then the norm function is maximized on the circumscribed sphere containing this cube. If the maximum is found to be too large, the parent cube is subdivided by bisecting each of its bounding hyper-surfaces, and the norm bounding process is repeated on the 2^n resulting offspring cubes. When all of the offspring cubes of a given parent pass the norm bounding, then the parent cell also passes.

There are two principal benefits of this process: (i) As the number of subdivisions increase, the size of the cell shrinks. Thus the norm bounding process becomes more accurate. (ii) For a given cube C_0, what is maximized is not $|\,\mathrm{Norm}(z)\,|$ for $z \in C_0$, but rather $|\,\mathrm{Norm}(z-q)\,|$ for $z \in C_0$ and some integer $q \in R$. Thus for different cubes, one can choose different q's.

There are, however, drawbacks to this method which rendered it unworkable on $Z[\zeta_{13}]$. The first and most obvious difficulty is the potential combinatorial explosion inherent in repeated subdivisions. From each parent cube, 4096 (2^{12}) offspring are generated. Ojala [Oj] reported that in his simpler ring, up to five levels of subdivisions were carried out. It seems likely that the situation would be much worse here.

The second drawback to this method is geometric. The analytic techniques used here to bound the norm function work on a sphere. One would expect that the supremum of the norm function on a cube would be significantly less than the supremum of the same function on the circumscribed sphere. A rough measure of the difference might be the ratio

of the volumes of the cube and its circumscribed sphere. It is well known that, as dimension increases, this ratio tends to zero. In particular, the fit in a twelve dimensional space is sure to be quite poor.

We also made a second, quite different, attempt on $Z[\zeta_{13}]$.

As previously mentioned, P. J. Weinberger [We] showed, subject to the generalized Riemann hypothesis, that all class-one fields with infinitely many units are euclidean. The basic method of proof is an induction using the minimal algorithm (for reference on the minimal algorithm, see [Le1], as [Mo] is unreadable). The generalized Riemann hypothesis is used in a starting stage of the induction.

Several attempts have been made to remove the dependence of this argument on the generalized Riemann hypothesis. Lenstra [Le5] has isolated the difficulty, and the work of Gupta, Murty, and Murty [Gu] showed promise of proving Weinberger's result independent of any unproven hypothesis. However, technical difficulties arose and the result which Gupta, Murty, and Murty arrived at does not apply to the ring of integers of a number field.

It seemed to us that the difficulties which impeded Gupta, Murty, and Murty came about, in part, from the generality of their approach. Thus, working on a very specific and very well behaved ring such as $Z[\zeta_{13}]$ might be easier. Unfortunately, this proved not to be the case. We believe, however, that the real truth in these matters lies with an attack along these lines

Thirdly, we tried an attack based on a bootstrapping method. The basic idea was, in outline, to select an intermediate field Φ, with $Q \subset \Phi \subset K$. Then, treating K as an extension of degree n over Φ, we attempt to prove that K is euclidean with respect to the map ψ, where

$$\psi(x) = \| \mathrm{Norm}_{K/\Phi}(x) \| ,$$

and $\|\cdot\|$ is the trace form from Φ to Q.

For example, if Φ is chosen so that $n = 2$, we could hope that methods devised to work on quadratic fields would work in this case, as K is quadratic over Φ. That is, we are pretending that Φ is Q, and that $\|\cdot\|$ is the usual absolute value.

We were unable to carry this attack through to conclusion. In particular, for a cyclotomic field of prime modulus, the maximal real quadratic subfield, K^+, is the unique subfield with $(K:\Phi) = 2$. Here, $| \mathrm{Norm}_{K/K^+}(x) | = | x |^2$, so that, in some sense, the "absolute value" properties of $\|\cdot\|$ have already been exhausted in dropping from K to K^+.

In general, if we choose Φ to be too close to K, then Φ is too complicated to behave like Q. Conversely, if we choose Φ to be too close to Q, then the degree of K over Φ is too high for simple methods to apply. However, some modification of this line of approach might work.

§4 Open cyclotomic questions

There are, as was mentioned in Chapter 1, fifteen class-one cyclotomic fields for which the norm-euclidean question is still open. Of these, only two have prime modulus -- $m = 17$ and $m = 19$. The techniques used here can be applied directly to these fields. A reasonable choice of d for these fields would seem to be approximately half of the

modulus. However, the size of the sufficient sets in these cases is significantly larger than the $m = 13$ case. The following tables give the size of the sufficient set for various values of d for the two open prime moduli.

Table 33. The size of S when $m = 17$.

d	$\#S$
3	2,280,738
4	95,996,878
5	160,374,512
6	22,821,184,489
7	201,443,970,380
8	1,487,745,013,098
9	8,248,634,581,516

Table 34. The size of S when $m = 19$.

d	$\#S$
3	19,974,286
4	1,563,544,538
5	39,334,715,702
6	742,913,975,156
7	8,870,097,807,556
8	83,105,562,335,066
9	578,304,732,608,758

Thus we could expect the running time in the $m = 17$ case to be about 70,000 times that in the $m = 13$ case, and the $m = 19$ case to be perhaps 400 times beyond that. Further, as m increases so does s, so that the time consuming floating point operations will be correspondingly more complex. All in all, we suspect that simply stepping up to a super computer would prove insufficient to overcome the added complexity.

Attacking the composite moduli with our methods presents different problems. For the prime modulus m, we made but little use of any relations on the roots of unity, as there was only one.

For composite moduli there are more relations, as the dimension of $Q(\zeta_m)$ as a vector space over Q is $\varphi(m)$. If for no other reasons than that of computational complexity, it seems necessary to exploit these additional relations. It does not seem necessary to abandon the ideas of patterns and generating selected points from them. Instead it seems advisable to modify the pattern and point generation algorithms to exploit the additional relations -- perhaps while continuing to work with points expressed in terms of m-tuples.

We hope to be able to successfully handle some, if not all, of the remaining cyclotomic fields at some future date.

APPENDICIES

Appendix A

The Program Listing

```
      PROGRAM MAC9

C
C     ************************************************
C     *                                              *
C     *       VERSION 9.2                            *
C     *                                              *
C     *                                              *
C     ************************************************

C     TUESDAY, 26 JULY, 1988

C23456789012345678901234567890123456789012345678901234567890123456
C      1         2         3         4         5         6

      IMPLICIT INTEGER (A-Z)
      DIMENSION A(0:20), X(20), SORT(20)
      DIMENSION PFLAG(10), CFLAG(10), XFLAG(10)
      LOGICAL FLAG1, FLAG2, FLAG3, FLAG4, FLAG5, FLAG6
      LOGICAL PFLAG

C     INITIALIZE

      OPEN (1, FILE = 'FLUNK' )
      OPEN (9, FILE = 'RUNINFO' )
      OPEN (3, FILE = 'BUGINFO' )

      WRITE (*,2500)
2500  FORMAT (' INPUT THE MODULUS NOW')
      READ (*,2600) M
2600  FORMAT (I8)
      WRITE (*,2700)
2700  FORMAT (' INPUT THE TILING MESH NOW')
      READ (*,2800) D
2800  FORMAT (I8)
      WRITE (*,2900)
2900  FORMAT (' INPUT CHECK3 BOUND NOW')
      READ (*,2950) F
2950  FORMAT (I8)


C     MEANINGS OF THE COUNTS:

C             COUNT1  NUMBER OF PATTERNS PGEN LOOKS AT
C             COUNT2  NUMBER OF PATTERNS PGEN REPORTS
C             COUNT3  NUMBER OF PATTERNS PASSING ALL CHECKS
C             COUNT4  NUMBER OF POINTS GENERATED
C             COUNT5  NUMBER OF FIRST CHECKS MADE
C             COUNT6  NUMBER OF POINTS FLUNKING FIRST CHECKER
C             COUNT7  NUMBER OF SECOND CHECKS MADE
C             COUNT8  NUMBER OF POINTS FLUNKING SECOND CHECKER
```

```
C               COUNT9  NUMBER OF THIRD CHECKS MADE
C               COUN10  NUMBER OF POINTS FLUNKING THIRD CHECKER

        COUNT1 = 0
        COUNT2 = 0
        COUNT3 = 0
        COUNT4 = 0
        COUNT5 = 0
        COUNT6 = 0
        COUNT7 = 0
        COUNT8 = 0
        COUNT9 = 0
        COUN10 = 0

C       MEANING OF THE PRINT FLAGS
C       PFLAG = PRINT ON/OF
C       CFLAG = NUMBER OF PRINTS
C       XFLAG = NUMBER OF PRINTS (STORED FOR RUNINFO PRINT)
C               1  PATTERN GENERATOR
C               2  CELL BOUND CHECKER
C               3  POINT GENERATOR
C               4  FIRST CHECKER
C               5  SECOND CHECKER
C               6  THIRD CHECKER
C               7  SLICK

        DO 2, I=1, 7
            WRITE (*,3000) I
3000        FORMAT (' INPUT COUNT NUMBER',I4)
            READ (*,3100) XFLAG(I)
3100        FORMAT (I8)
            CFLAG(I) = XFLAG(I)
            PFLAG(I) = (XFLAG(I).GT.0)
2         CONTINUE


C       INITIALIZE SUBROUTINES

        CALL SCELL (M,D,A,SUM,FLAG2,PFLAG(2))
        CALL SCHEK1 (M,D,X,SORT,FLAG4,COUNT5,PFLAG(4))
        CALL SCHEK2 (M,D,X,FLAG5,COUNT7,PFLAG(5),CFLAG(7))
        CALL SSLICK (M,D,X,X,FLAG5,PFLAG(7))

C       NOTE:   CHECK3 DOESN'T NEED TO BE INITIALIZED

C       TOP OF PATTERN GENERATION LOOP
        CALL SPGEN (M,D,A,SUM,FLAG1,COUNT1,PFLAG(1))
10      CONTINUE
        IF (FLAG1) THEN

C           INCREMENT PATTERN COUNTER
            COUNT2 = COUNT2 + 1

C           CHECK TO SEE IF PATTERN IS WITHIN FUNDAMENTAL CELL
            CALL CELL (M,D,A,SUM,FLAG2,PFLAG(2))

            IF (PFLAG(2)) THEN
                CFLAG(2) = CFLAG(2)-1
```

```
                PFLAG(2) = (CFLAG(2).GT.0)
              ENDIF

            IF (FLAG2) THEN

C               INCREMENT WITHIN CELL COUNTER
                COUNT3 = COUNT3 + 1
                WRITE (9,1000) COUNT3, COUNT4, COUNT6,
     *          COUNT8, COUN10, ( A(I), I=0, D-1 )
1000            FORMAT (5I10,3X,9I3)

C       TOP OF POINT GENERATION LOOP
                CALL SPOINT (M,D,A,X,SORT,FLAG3,PFLAG(3))
20              CONTINUE
                IF (FLAG3) THEN

C                   INCREMENT POINT COUNTER
                    COUNT4 = COUNT4 + 1

C                   INVOKE FIRST CHECKER
                    CALL CHECK1 (M,D,X,SORT,FLAG4,COUNT5,PFLAG(4))

                    IF (PFLAG(4)) THEN
                        CFLAG(4) = CFLAG(4)-1
                        PFLAG(4) = (CFLAG(4).GT.0)
                      ENDIF

                    IF (.NOT.FLAG4) THEN

C                       WE HAVE FLUNKED FIRST CHECK
C                       INCREMENT COUNTER
                        COUNT6 = COUNT6 + 1

C                       INVOKE SECOND CHECKER
                        CALL CHECK2 (M,D,X,
     *                    FLAG5,COUNT7,PFLAG(5),CFLAG(7))

                        IF (PFLAG(5)) THEN
                            CFLAG(5) = CFLAG(5)-1
                            PFLAG(5) = (CFLAG(5).GT.0)
                          ENDIF

                        IF (.NOT.FLAG5) THEN

C                           WE HAVE FLUNKED SECOND CHECK
C                           INCREMENT COUNTER
                            COUNT8 = COUNT8 + 1

C                           INVOKE THIRD CHECKER
                            CALL CHECK3 (M,D,F,X,
     *                        FLAG6,COUNT9,PFLAG(6),CFLAG(7))

                            IF (PFLAG(6)) THEN
                                CFLAG(6) = CFLAG(6)-1
                                PFLAG(6) = (CFLAG(6).GT.0)
                              ENDIF

                            IF (.NOT.FLAG6) THEN
```

```
C                               WE HAVE FLUNKED EVERYTHING!
C                               INCREMENT COUNTER & WRITE POINT
                                COUN10 = COUN10 + 1
                                WRITE (1,1100) (X(I), I=1,M)
1100                            FORMAT (20I3)

                              ENDIF
C                             END THIRD CHECK FAIL

                            ENDIF
C                           END SECOND CHECK FAIL

                          ENDIF
C                         END FIRST CHECK FAIL

C                       GET NEXT POINT
                        CALL POINT (M,D,A,X,SORT,FLAG3,PFLAG(3))

                        IF (PFLAG(3)) THEN
                            CFLAG(3) = CFLAG(3)-1
                            PFLAG(3) = (CFLAG(3).GT.0)
                          ENDIF

                      GOTO 20
                      ENDIF
C                     END POINT LOOP

                    ENDIF
C                   END PASS CELL CHECK -- GO TO NEXT PATTERN

                  CALL PGEN (M,D,A,SUM,FLAG1,COUNT1,PFLAG(1))

                  IF (PFLAG(1)) THEN
                      CFLAG(1) = CFLAG(1)-1
                      PFLAG(1) = (CFLAG(1).GT.0)
                    ENDIF

                GOTO 10
                ENDIF
C               END PATTERN GENERATION LOOP

              WRITE (9,1300) M,D,F
1300          FORMAT (' MODULUS / DIVISION / CHECK3 BOUND ',5X,3I5)
              WRITE (9,1400) (XFLAG(I), I=1,7)
1400          FORMAT (' PRINT COUNTS                      ',7I5)
              WRITE (9,1500) COUNT1, COUNT2
1500          FORMAT (' PATTERNS LOOKED AT / REPORTED     ',2I10)
              WRITE (9,1600) COUNT3, COUNT4
1600          FORMAT (' PATTERNS PASSED / POINTS GENERATED',2I10)
              WRITE (9,1700) COUNT5, COUNT6
1700          FORMAT (' CHECK #1 -- CHECKS / FLUNKS       ',2I10)
              WRITE (9,1800) COUNT7, COUNT8
1800          FORMAT (' CHECK #2 -- CHECKS / FLUNKS       ',2I10)
              WRITE (9,1900) COUN9, COUN10
1900          FORMAT (' CHECK #3 -- CHECKS / FLUNKS       ',2I10)

              IF (COUN10.EQ.0) THEN
```

```
            WRITE (9,2000)
2000        FORMAT (' \CONGRATULATIONS, \DR. \BOB!  \ALL POINTS PASS.')
          ELSE
            WRITE (9,2100)
2100        FORMAT (' \CURSES, FOILED AGAIN!  \SOME POINTS FLUNK.')
          ENDIF

        CLOSE (1)
        CLOSE (9)
        CLOSE (3)

        END


        SUBROUTINE SPGEN (M,D,A,SUM,FLAG,COUNT,PFLAG)

C       PATTERN GENERATOR.  RETURNS THE RESULT IN THE ARRAY A(I).
C       THE MULTISET \(\( 0**A(0), 1**A(1), ... \)\) CORRESPONDS TO A
C       AND THE POINT X = (1/D) SUM \( X(I)*E(I) \), WHERE
C       A(I) = #\( J: X(J) = I \).

C       SINCE THE PATTERNS ARE FOR POINTS IN STANDARD FORM,
C       WE ALWAYS HAVE A(0)>0
C       \THUS WE INTERNALLY GENERATE PATTERNS B(J) ON M-1 DIGITS
C       AND RETURN A(J)=B(J) WHERE A(0)=B(0)+1

C       THE METHOD USED TO GENERATE PATTERNS IS TO CARRY A 1 FROM THE
C       FIRST NON ZERO B(I) TO B(I+1) AND IF I>0, DROP THE REST
C       OF B(I) TO B(0).

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        IMPLICIT INTEGER (A-Z)
        DIMENSION A(0:20), B(0:20)
        LOGICAL FLAG, TFLAG, PFLAG
        REAL RM, RD, TEMP
        SAVE

C       LBOUND AND UBOUND DETERMINE A SHELL OUTSIDE OF WHICH THE
C       PATTERNS DO NOT NEED TO BE CONSIDERED.
C       IF X IS WITHIN THE SPHERE DETERMINED BY LBOUND, THEN
C       THE (1/M) CELL HAS NORM .LT. 1 BY ARITH-GEOMETRIC MEAN.
C       THE SPHERE DETERMINED BY UBOUND IS THE CIRCUMSCRIBED SPHERE
C       CONTAINING THE FUNDAMENTAL CELL.

C       INITIALIZE

        M1 = M-1
        D1 = D-1
        RM = REAL(M)
        RD = REAL(D)
        TEMP = SQRT(RM-1.) * ( RD - SQRT( (RM+1.)/12. ) )
        LBOUND = TEMP * TEMP
        UBOUND = D*D * ( (M*M-1)/12 )

        IF (PFLAG) THEN
            WRITE (3,5000) TEMP,LBOUND,UBOUND
```

```
5000        FORMAT (' INIT PGEN: TEMP,LBOUND,UBOUND',F12.8,2I8)
            WRITE (3,5010)
5010        FORMAT ('  ')
          ENDIF

C       IF TFLAG THEN INPUT A STARTING PATTERN
        WRITE (*,1000)
1000    FORMAT (' I WISH TO INPUT A STARTING VALUE T/F')
        READ (*,1100) TFLAG
1100    FORMAT (L8)

C       BYPASS STARTING PATTERN INPUT
C       TFLAG = .FALSE.

        IF (TFLAG) THEN

C           GET PATTERN FROM HUMAN
            DO 2, I=0,D1
                WRITE (*,1200) I
1200            FORMAT (' INPUT THE MULTIPLICITY OF #',I3)
                READ (*,1300) A(I)
1300            FORMAT (I8)
2             CONTINUE

C           TOP OF VERIFICATION LOOP
100         CONTINUE

C               DISPLAY AND CHECK THE PATTERN
                WRITE (*,1400) (A(I), I=0,D1)
1400            FORMAT (20I3)

                SUM = 0
                SSUM = 0
                WSUM = 0

                DO 4, I=0,D1

                    IF (A(I).LT.0 .OR. A(I).GT.M) THEN
C                       FAULTY PATTERN
                        GOTO 150
                      ENDIF

                    WSUM = A(I)+WSUM
                    SUM = I*A(I)+SUM
                    SSUM = I*I*A(I)+SSUM

4                 CONTINUE

                IF (WSUM.EQ.M .AND. A(0).GT.0) THEN

C                   PATTERN IS OKAY -- DOUBLE CHECK WITH HUMAN
                    WRITE (*,1500)
1500                FORMAT (' THIS IS CORRECT T/F')
                    READ (*,1600) TFLAG
1600                FORMAT (L8)

                    IF (TFLAG) THEN
C                       EVERYTHING COOL -- RETURN
```

```
                        B(0) = A(0) - 1
                        DO 5, J=1, D1
                            B(J) = A(J)
5                         CONTINUE

                        FLAG = .TRUE.
                        RETURN
                      ENDIF

                    ENDIF

C                 FIX A USER SELECTED NUMBER IN THE PATTERN
150               CONTINUE
                  WRITE (*,1700)
1700              FORMAT (' INPUT THE INDEX OF THE ONE TO FIX NOW.')
                  READ (*,1800) I
1800              FORMAT (I8)
                  WRITE (*,1900)
1900              FORMAT (' INPUT THE CORRECT VALUE NOW.')
                  READ (*,2000) A(I)
2000              FORMAT (I8)

                GOTO 100
C               END OF PATTERN VERIFICATION LOOP

            ENDIF
C           END OF HUMAN INPUTED PATTERN SECTION


C         COMPUTER GENERATED FIRST PATTERN
          DO 6, I=0,D1
              A(I) = 0
              B(I) = 0
6           CONTINUE


          SUM = 0
          SSUM = 0
          A(0) = M
          B(0) = M1
          FLAG = .TRUE.


          ENTRY PGEN (M,D,A,SUM,FLAG,COUNT,PFLAG)

          IF (PFLAG) THEN
              WRITE (3,5100) (A(I), I=0,D1)
5100          FORMAT (' PGEN: OLD PATTERN ',20I3)
              WRITE (3,5200) (B(I), I=0,D1)
5200          FORMAT (' PGEN: OLD B PATTERN ',20I3)
            ENDIF

200       COUNT = COUNT+1
          I=0

C         FIND A NON ZERO B(I)
```

```
C       DO WHILE (B(I).EQ.0)
12      CONTINUE
        IF (B(I).EQ.0) THEN

            I = I+1

          GOTO 12
          ENDIF
C         REPEAT

        IF (PFLAG) THEN
            WRITE (3,5300) I
5300        FORMAT (' PGEN: NON ZERO SLOT',I3)
          ENDIF

        IF (I.EQ.D1) THEN
C           OUT OF PATTERNS (DON'T NEED NO X(I) .GE. D!)
            FLAG = .FALSE.

            IF (PFLAG) THEN
                WRITE (3,5400)
                WRITE (3,5400)
5400            FORMAT ('  ')
              ENDIF

            RETURN
          ENDIF

C       CARRY THE 1 UPWARDS AND DROP THE REST DOWN.
C       COMPUTE NEW SUM & SSUM

        SUM = -I*B(I) + I + 1 + SUM
        SSUM = -I*I*B(I) + (I+1)*(I+1) + SSUM
        B(I+1) = B(I+1)+1
        B(0) = B(I)-1

C       IF I .GT. 0 THEN ZERO A(I)

        IF (I.GT.0) THEN
            B(I) = 0
          ENDIF

        MNORM = M*SSUM - SUM*SUM

        IF (PFLAG) THEN
            WRITE (3,5500) (B(I), I=0,D1)
5500        FORMAT (' PGEN: NEW B PATTERN',20I3)
            WRITE (3,5600) SUM,SSUM,MNORM
5600        FORMAT (' PGEN: SUM,SSUM,MNORM',3I8)
          ENDIF


C       CHECK OF X LYING WITHIN THE SHELL TWEEN LBOUND & UBOUND

        IF ( ( MNORM .LE. LBOUND ) .OR. ( MNORM .GT. UBOUND ) ) THEN
C           THIS PATTERN UNACCEPTABLE
C           GET ANOTHER
            GOTO 200
```

```
          ENDIF

        A(0) = B(0) + 1
        DO 14, J=1, D1
            A(J) = B(J)
14        CONTINUE

        IF (PFLAG) THEN
            WRITE (3,5700) (A(I), I=0,D1)
5700        FORMAT (' PGEN: NEW PATTERN',20I3)
            WRITE (3,5800)
            WRITE (3,5800)
5800        FORMAT ('  ')
          ENDIF
        RETURN

        END


        SUBROUTINE SCELL (M,D,A,SUM,FLAG,PFLAG)

C       GIVEN A POINT X'S PATTERN A WHERE
C               X = SUM \( X(I)*E(I)/D \)
C               A(I) = #\(X(J): X(J)=I\)
C               SUM = SUM \( X(I) \)

C       THIS SUBROUTINE WILL RETURN:
C       FLAG = .TRUE.   IF POINT IS IN THE FUNDAMENTAL CELL
C       FLAG = .FALSE.  IF POINT IS NOT IN THE FUNDAMENTAL CELL.

C       NEW VERSION 16 \MARCH 1988
C               USES CONCAVITY & ONLY PERFORMS D-1 CHECKS
C               THE VARIABLE H IS D*G FROM CHAPTER 2


C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        IMPLICIT INTEGER (A-Z)
        DIMENSION A(0:20)
        LOGICAL FLAG,PFLAG
        SAVE

        M1 = M-1
        D1 = D-1

        RETURN

        ENTRY CELL (M,D,A,SUM,FLAG,PFLAG)

        IF (PFLAG) THEN
            WRITE (3,5000) (A(I), I=0,D1)
5000        FORMAT (' CELL: CHECKING PATTERN',20I3)
          ENDIF

        K = 0
        H = 0
```

```
      DO 10, J = D1, 1, -1

          K = K + A(J)
          H = ((A(J)+M)*A(J)*D)/2 + H - (M*J+D*K-SUM)*A(J)

          IF (PFLAG) THEN
              WRITE (3,5100) J,K,H
5100          FORMAT (' CELL: J/K/H',3I8)
            ENDIF

          IF ( H .LT. 0 ) THEN

C             THE POINT IS OUTSIDE OF THE CELL.
              IF (PFLAG) THEN
                  WRITE (3,5200)
                  WRITE (3,5200)
5200              FORMAT ('  ')
                ENDIF

              FLAG = .FALSE.
              RETURN

            ENDIF

10      CONTINUE


C     THE POINT HAS PASSED ALL TESTS & IS WITHIN THE CELL.

      FLAG = .TRUE.

      IF (PFLAG) THEN
          WRITE (3,5300)
          WRITE (3,5300)
5300      FORMAT ('  ')
        ENDIF

      RETURN

      END


      SUBROUTINE SPOINT (M,D,A,X,SORT,FLAG,PFLAG)

C     THIS ROUTINE CONVERTS THE PATTERN IN A INTO THE POINT IN X.
C     IT HAS BEEN MODIFIED TO NOT GENERATE UNNECESSARY POINTS.
C     IT SEARCHES FOR THE TWO DIGITS USED WITH THE
C     LEAST MULTIPLICITY AND PUTS THESE IN X(1) & X(M).


      IMPLICIT INTEGER (A-Z)
      DIMENSION A(0:20),B(0:20),X(20),SORT(20),MORT(20)
      LOGICAL FLAG,SFLAG,SSFLAG,PFLAG
      SAVE

C     SFLAG = .TRUE. MEANS NO SMALLEST YET,
C     SSFLAG = .TRUE. MEANS NO SECOND SMALLEST YET.
```

```
C234567890123456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        M1 = M-1
        D1 = D-1
        SFLAG = .TRUE.
        SSFLAG = .TRUE.


C       FIRST STORE THE PATTERN IN B SO WE CAN WORK WITH IT.

        DO 2, I=0,D1
            B(I) = A(I)
2         CONTINUE

        IF (PFLAG) THEN
            WRITE (3,5000) (A(I), I=0,D1)
5000        FORMAT (' SPOINT: A ',20I3)
            WRITE (3,5100) (B(I), I=0,D1)
5100        FORMAT (' SPOINT: B ',20I3)
          ENDIF

C       S = DIGIT WITH SMALLEST NON ZERO MULTIPLICITY
C       SS = DIGIT WITH SECOND SMALLEST.

C       START LOOP THROUGH THE DIGITS
        DO 4, I=0,D1

            IF (B(I).GT.0) THEN

C               DIGIT I HAS NON ZERO MULTIPLICITY

                IF (SFLAG) THEN

C                   NO SMALLEST SAVED YET, SO THIS IS IT.

                    S = I
                    SFLAG = .FALSE.

                  ELSE

                    IF (B(I).LE.B(S)) THEN

C                       A NEW SMALLEST.

                        SS = S
                        SSFLAG = .FALSE.
                        S = I

                      ELSE

                        IF (SSFLAG) THEN

C                           NO SECOND SMALLEST SAVED YET.

                            SS=I
                            SSFLAG = .FALSE.
```

```
                        ELSE

                          IF (B(I).LT.B(SS)) THEN
C                             A NEW SECOND SMALLEST

                              SS=I

                            ENDIF
C                           END NEW SECOND SMALLEST

                        ENDIF
C                       END NO SECOND SMALLEST

                    ENDIF
C                   END NEW SMALLEST

                ENDIF
C               END NO SMALLEST

            ENDIF
C           END DIGIT I HAS NON ZERO MULTIPLICITY

4         CONTINUE
C         END LOOP ON DIGITS

        IF (PFLAG) THEN
            WRITE (3,5200) S,SS
5200        FORMAT (' SPOINT: S/SS ',2I4)
          ENDIF

        IF (B(S).GT.1) THEN
C           SMALLEST MULTIPLICITY IS .GE. 2
            B(S) = B(S)-2
            X(1) = S
            X(M) = S
          ELSE
C           SMALLEST MULTIPLICITY IS 1
            B(S) = B(S)-1
            B(SS) = B(SS)-1
            X(1) = S
            X(M) = SS
          ENDIF
C         END FIX FIRST & LAST DIGITS

C       BEGIN INITIALIZE THE POINT

        J = 1

        DO 8, I=0,D1

            DO 6, XX=1,B(I)
                J = J+1
                X(J) = I
6             CONTINUE

8         CONTINUE
```

```
C         END INITIALIZE THE POINT (MIDDLE DIGITS)

C       NOW SORT THE POINTS SO X(SORT(1)) <= X(SORT(2)) <= ...

        DO 10, I=2,M1
            SORT(I) = I
10        CONTINUE

        I=2
11      CONTINUE
C       DO WHILE ( ( I.LE.M1 ) .AND. ( X(1).GT.X(SORT(I)) ) )
        IF ( I.LE.M1 ) THEN
            IF ( X(1).GT.X(SORT(I)) ) THEN
                SORT(I-1) = SORT(I)
                I = I+1
                GOTO 11
              ENDIF
          ENDIF
C         END DO WHILE

        SORT(I-1) = 1

        I=M1
12      CONTINUE
C       DO WHILE ( ( I.GE.1 ) .AND. ( X(M).LT.X(SORT(I)) ) )
        IF ( I.GE.1 ) THEN
            IF ( X(M).LT.X(SORT(I)) ) THEN
                SORT(I+1) = SORT(I)
                I = I-1
                GOTO 12
              ENDIF
          ENDIF
C         END DO WHILE

        SORT(I+1) = M
C       END SORTING OF POINTS

C       BEGIN MORT, THE INVERSE OF SORT.

        DO 16, I=1,M
            MORT(SORT(I)) = I
16        CONTINUE
C         END MORT, THE INVERSE OF SORT

        IF (PFLAG) THEN
            WRITE (3,5300) (X(I),I=1,M)
5300        FORMAT (' SPOINT:    X ',20I3)
            WRITE (3,5400) (SORT(I),I=1,M)
5400        FORMAT (' SPOINT: SORT ',20I3)
            WRITE (3,5500) (MORT(I),I=1,M)
5500        FORMAT (' SPOINT: MORT ',20I3)
            WRITE (3,5600)
            WRITE (3,5600)
5600        FORMAT ('  ')
          ENDIF

        FLAG = .TRUE.
```

```
C       HAVE THE FIRST POINT & RETURN


        RETURN


        ENTRY POINT (M,D,A,X,SORT,FLAG,PFLAG)

C       GET NEXT POINT

        IF (PFLAG) THEN
            WRITE (3,5700) (X(I), I=1,M)
5700        FORMAT (' POINT: X ',20I3)
          ENDIF

C       FIRST FIND X(I) SUCH THAT
C       X(I) < X(I+1) >= X(I+2) >= ... >= X(M1)

        I = M1-1

C       DO WHILE ( X(I) .GE. X(I+1) )
20      CONTINUE
        IF ( X(I) .GE. X(I+1) ) THEN

            IF (I.EQ.2) THEN
C               POINT IN INVERSE LEXICOGRAPHICAL ORDER. RETURN
                FLAG = .FALSE.
                IF (PFLAG) THEN
                    WRITE (3,5800)
                    WRITE (3,5800)
5800                FORMAT ('  ')
                  ENDIF
                RETURN
              ENDIF

            I = I-1

          GOTO 20
          ENDIF
C         REPEAT

        J = I+1

C       FIND COORDINATE X(J) SUCH THAT X(J) > X(I) AND X(J+1) <= X(I)
C       THAT IS, THE MINIMAL X(J) > X(I) WITH J > I.

C       DO WHILE (J.LT.M1)
22      CONTINUE
        IF (J.LT.M1) THEN

            IF ( X(J+1) .LE. X(I) ) THEN
C               EXIT
                GOTO 23
              ENDIF

            J=J+1

          GOTO 22
```

```
          ENDIF
C         REPEAT
23        CONTINUE

        IF (PFLAG) THEN
            WRITE (3,5900) I,J
5900        FORMAT (' POINT: I,J ',2I4)
          ENDIF

C       NOW SWAP X(I) & X(J)

C       NOTE: X(I) IS THE MORT(I)TH SMALLEST & ETC.

        K = X(I)
        X(I) = X(J)
        X(J) = K
        K = MORT(I)
        MORT(I) = MORT(J)
        MORT(J) = K
        SORT(MORT(I)) = I
        SORT(MORT(J)) = J

C       NOW INVERT TAIL OF THE LIST:  PUT IN INCREASING ORDER.

        I = I+1
        J = M1

C       DO WHILE (I.LT.J)
24      CONTINUE
        IF (I.LT.J) THEN

            K = X(I)
            X(I) = X(J)
            X(J) = K
            K = MORT(I)
            MORT(I) = MORT(J)
            MORT(J) = K
            SORT(MORT(I)) = I
            SORT(MORT(J)) = J
            I=I+1
            J=J-1

          GOTO 24
          ENDIF
C         REPEAT

        IF (PFLAG) THEN
            WRITE (3,6000) (X(I), I=1,M)
6000        FORMAT (' POINT: NEW X ',20I3)
            WRITE (3,6100) (SORT(I), I=1,M)
6100        FORMAT (' POINT: SORT  ',20I3)
            WRITE (3,6200) (MORT(I), I=1,M)
6200        FORMAT (' POINT: MORT  ',20I3)
            WRITE (3,6300)
            WRITE (3,6300)
6300        FORMAT ('  ')
          ENDIF
```

```
      RETURN

      END


      SUBROUTINE SCHEK1 (M,D,X,SORT,FLAG,COUNT,PFLAG)

      IMPLICIT INTEGER (A-Z)
      DIMENSION X(20), SORT(20)
      DIMENSION XQ(10), C(10)
      DIMENSION E(20,10), ED(20,10)
      LOGICAL FLAG, PFLAG
      REAL C, CMIN
      REAL UPPER, UBOUND, LOWER, LENGTH
      REAL TEMP1, TEMP2
      REAL LOWER4, LOWRAD, BOUND, BND2S
      COMPLEX E, ED, XQ
      DOUBLE PRECISION ANGLE, RADIUS
      DOUBLE PRECISION DTEMP, REALE, IMAGE, REALED, IMAGED
      SAVE

C     ERROR DISCOVERED IN RADIUS:      (TS, 5 JAN, 1988)
C       SINCE WE NEED USE ONLY ONE OF EACH CONJUGATE PAIR
C       OF COMPLEX ISOMORPHISMS, WE CAN DIVIDE THE RADIUS BY THE
C       SQUARE ROOT OF 2.

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

      M1 = M-1
      D1 = D-1
      S = M1/2
      ANGLE = 6.28318 53071 79586 D0 / DBLE(M)
      RADIUS = DSQRT( DBLE((M*M-1)/24) ) / DBLE(D)
      DTEMP = RADIUS / DSQRT( DBLE(S) )
      LOWER = REAL( DTEMP )
      LOWER4 = REAL( 4.0D0 * DTEMP )
      LOWRAD = REAL( 2.0D0 * DTEMP * RADIUS )
      BOUND = .99
      BND2S = BOUND * 2**S

      DO 4, I=1,M

          DO 2, J=1,S
              DTEMP = ANGLE*MOD(I*J,M)
              REALE = DCOS( DTEMP )
              IMAGE = DSIN( DTEMP )
              E(I,J) = CMPLX( REALE, IMAGE )
              REALED = REALE / DBLE(D)
              IMAGED = IMAGE / DBLE(D)
              ED(I,J) = CMPLX ( REALED, IMAGED )
2           CONTINUE

4       CONTINUE

      IF (PFLAG) THEN

          WRITE (3,5000) ANGLE,RADIUS,BND2S
```

```
5000        FORMAT (' SCHEK1: ANGLE,RADIUS,BND2S',3F12.6)

            DO 8, J=1,S

                WRITE (3,5100)
                WRITE (3,5200) J
5100            FORMAT ('  ')
5200            FORMAT (' SCHEK1: CONJUGATE NUMBER', I2)

                DO 6, I=1,M

                    WRITE (3,5300) E(I,J),ED(I,J)
5300                FORMAT (' SCHEK1: E/ED',2(2(F12.6,1X),4X))

6                 CONTINUE

8               CONTINUE

            WRITE (3,5400)
            WRITE (3,5400)
5400        FORMAT ('  ')

          ENDIF


        RETURN


        ENTRY CHECK1 (M,D,X,SORT,FLAG,COUNT,PFLAG)

C       XQ(J) = JTH CONJUGATE OF X AS A COMPLEX NUMBER

        IF (PFLAG) THEN
            WRITE (3,5500) (X(I), I=1,M)
5500        FORMAT (' CHECK1: X ',20I3)
          ENDIF

        DO 12, J=1,S

            XQ(J) = ( 0., 0. )

            DO 10, I=1,M
                XQ(J) = X(I)*ED(I,J) + XQ(J)
10            CONTINUE

12        CONTINUE

C       THE MAIN LOOP

        DO 30, I=M,1,-1

            COUNT = COUNT+1

            IF (I.LT.M) THEN

C               SUBTRACT OFF A NEARBY LATTICE POINT

                DO 20, J=1,S
```

```
                XQ(J) = XQ(J) - E(SORT(I+1),J)
20              CONTINUE

              ENDIF

C           FIND THE MINIMAL C(J) = \!XQ(J)\!

            CMIN = 999999.

            DO 22, J=1,S

                C(J) = ABS(XQ(J))

                IF ( C(J) .LT. CMIN ) THEN
                    CMIN = C(J)
                  ENDIF

22              CONTINUE

            TEMP1 = (LOWER+CMIN)*LOWER4
            LENGTH = 0.

            IF (PFLAG) THEN
                WRITE (3,5600)
5600            FORMAT ('  ')
                WRITE (3,5700) I
5700            FORMAT (' CHECK1: I ',I4)
                WRITE (3,5800) (J,XQ(J), J=1,S)
5800            FORMAT (' CHECK1:  J, XQ ',I4,2F12.6)
                WRITE (3,5900) CMIN
5900            FORMAT (' CHECK1: CMIN ',F10.6)
                WRITE (3,6000) (J,C(J), J=1,S)
6000            FORMAT (' CHECK1:  J, C ',I4,F12.6)
                WRITE (3,6100) LOWER
6100            FORMAT (' CHECK1: LOWER ',F12.6)
              ENDIF

            DO 24, J=1,S
                TEMP2 = ( SQRT( C(J)*C(J)+TEMP1 ) - C(J) )
                LENGTH = TEMP2*TEMP2 + LENGTH

                IF (PFLAG) THEN
                    WRITE (3,6200) J,TEMP2
6200                FORMAT (' CHECK1: J,TEMP2 'I4,F12.6)
                  ENDIF

24              CONTINUE

            UPPER = LOWRAD / SQRT(LENGTH)

            TEMP1 = (UPPER+CMIN)*UPPER*4.
            UBOUND = 1.

            DO 26, J=1,S
                UBOUND = ( SQRT( C(J)*C(J)+TEMP1 ) + C(J) ) * UBOUND
26              CONTINUE

            IF (PFLAG) THEN
```

```
                WRITE (3,6300) LENGTH,UPPER,UBOUND
6300            FORMAT (' CHECK1:  LENGTH,UPPER,UBOUND ',3F12.6)
              ENDIF

            IF ( UBOUND .LT. BND2S ) THEN
                FLAG = .TRUE.

                IF (PFLAG) THEN
                    WRITE (3,6400)
                    WRITE (3,6400)
6400                FORMAT ('  ')
                  ENDIF

                RETURN
              ENDIF

30      CONTINUE

        FLAG = .FALSE.

        IF (PFLAG) THEN
            WRITE (3,6500)
            WRITE (3,6500)
6500        FORMAT ('  ')
          ENDIF

        RETURN


        END



        SUBROUTINE SCHEK2 (M,D,X,FLAG,COUNT,PFLAG,CFLAG)

        IMPLICIT INTEGER (A-Z)
        LOGICAL FLAG, PFLAG, PFLAG1

        DIMENSION A(0:20),B(0:20,20),II(20)
        DIMENSION X(20),Q(20)

        SAVE

C       THIS ROUTINE HANDLES REPETITIONS IN THE DIGITS OF X

C       SLICK IS USED TO CHECK THE POINT

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        M1 = M - 1
        D1 = D - 1
        PFLAG1 = (CFLAG.GT.0)

        RETURN


        ENTRY CHECK2 (M,D,X,FLAG,COUNT,PFLAG,CFLAG)
```

```
C       INITIALIZE SLICK FOR THIS POINT
        CALL XSLICK (M,D,X,Q,FLAG,PFLAG1)

        DO 10, I=0, D1
            A(I) = 0
10        CONTINUE

C       GET PATTERN & LOCATION
C       ALSO ZERO 'NEARBY LATTICE POINT' Q
        DO 12, I=1, M
            Q(I) = 0
            J = X(I)
            A(J) = A(J)+1
            B( J, A(J) ) = I
12        CONTINUE

        IF (PFLAG) THEN

            WRITE (3,5000) (X(I), I=1,M)
5000        FORMAT (' CHECK2: X ',20I3)
            WRITE (3,5100)
5100        FORMAT (' CHECK2:  I   A    B')

            DO 14, I=0,D1
                WRITE (3,5200) I,A(I),( B(I,J), J=1,A(I) )
5200            FORMAT (8X,2I4,2X,20I3)
14            CONTINUE

            IF (PFLAG1) THEN
                WRITE (3,5250)
5250            FORMAT ('  ')
              ENDIF

          ENDIF

C       INVOKE CHECKER FOR Q = 0 FIRST!
        CALL SLICK (M,D,X,Q,FLAG,PFLAG1)

        IF (PFLAG1) THEN
            CFLAG = CFLAG-1
            PFLAG1 = (CFLAG.GT.0)
          ENDIF

C       BEGIN DOES X PASS WITH 0?
        IF (FLAG) THEN

            IF (PFLAG) THEN
                WRITE (3,6500)
6500            FORMAT (' CHECK2:  PASSES WITH 0')
                WRITE (3,6600)
                WRITE (3,6600)
6600            FORMAT ('  ')
              ENDIF

C           YES!  THE POINT PASSES!  BRAVO!
            RETURN
```

```
          ELSE
C         POINT DOES NOT PASS

            IF (PFLAG) THEN
                WRITE (3,6700)
6700            FORMAT (' CHECK2:  FLUNKS WITH 0')
                WRITE (3,6750)
6750            FORMAT ('  ')
              ENDIF

          ENDIF
C         END DOES X PASS WITH 0?

C       BEGIN COMPUTE 'NEARBY LATTICE POINT' Q
        J = D1

C       BEGIN LOOP ON DIGITS J
100     CONTINUE
        IF (J.GE.0) THEN

            DO 20, I=1,M
                II(I) = 0
20            CONTINUE

            K = 1

C           BEGIN FETCH NEXT Q
200         CONTINUE
            IF ( K.LE.A(J) ) THEN

C               INCREMENT THE KTH SPOT
                II(K) = II(K)+1

C               BEGIN HAVE NEXT Q OR NEED TO CARRY
                IF ( II(K).LE.1 ) THEN

C                   BEGIN HAVE NEXT Q

C                   INCREMENT COUNTER
                    COUNT = COUNT + 1

C                   FINALIZE THE 'NEARBY LATTICE POINT' Q
                    DO 26, I=1,A(J)
                        Q(B(J,I)) = II(I)
26                    CONTINUE

                    IF (PFLAG) THEN
                        WRITE (3,5300) (II(I), I=1,A(J))
5300                    FORMAT (' CHECK2: II ',20I3)
                        WRITE (3,5400) (Q(I), I=1,M)
5400                    FORMAT (' CHECK2:  Q ',20I3)
                        IF (PFLAG1) THEN
                            WRITE (3,5450)
5450                        FORMAT ('  ')
                          ENDIF
                      ENDIF
```

```
C                   INVOKE CHECKER
                    CALL SLICK (M,D,X,Q,FLAG,PFLAG1)

                    IF (PFLAG1) THEN
                        CFLAG = CFLAG-1
                        PFLAG1 = (CFLAG.GT.0)
                      ENDIF

C                   BEGIN DOES X PASS?
                    IF (FLAG) THEN

                        IF (PFLAG) THEN
                            WRITE (3,5500)
5500                        FORMAT (' CHECK2:  THIS POINT PASSES.')
                            WRITE (3,5600)
                            WRITE (3,5600)
5600                        FORMAT ('  ')
                          ENDIF

C                       YES! THE POINT PASSES!
                        RETURN

                      ELSE
C                     POINT DOESN'T PASS

                        IF (PFLAG) THEN
                            WRITE (3,5700)
5700                        FORMAT (' CHECK2:  FLUNKS WITH Q.')
                            WRITE (3,5750)
5750                        FORMAT ('  ')
                          ENDIF

                      ENDIF
C                     END DOES X PASS?

                    K = 1

C                   END HAVE NEXT Q

                  ELSE

C                   BEGIN NEED TO CARRY

                    II(K) = 0
                    K = K+1

C                   END NEED TO CARRY

                  ENDIF
C                 END HAVE NEXT Q OR NEED TO CARRY?

              GOTO 200
              ENDIF
C             END FETCH NEXT Q

            J = J-1

          GOTO 100
```

```
          ENDIF
C         END LOOP ON DIGITS

C       HAVE RUN THROUGH OUR LIST OF 'NEARBY LATTICE POINTS'
C       THIS POINT HAS NOT PASSED

        IF (PFLAG) THEN
            WRITE (3,5800)
5800        FORMAT (' CHECK2: THE POINT HAS FLUNKED ALL TESTS.')
            WRITE (3,5900)
            WRITE (3,5900)
5900        FORMAT ('  ')
          ENDIF

        RETURN


        END



        SUBROUTINE CHECK3 (M,D,F,X,FLAG,COUNT,PFLAG1,CFLAG2)

C       A PROGRAM TO CHECK POINTS IN THE 1/D LATTICE

C       THE METHOD IS TO GENERATE ALL INTEGERS WITH COORDINATES
C       BOUNDED ABOVE BY SOME NUMBER 'F' & TO BOUND THE
C       NORM OF X-Q ON THE 1/D CELL VIA 'SLICK'.

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        IMPLICIT INTEGER (A-Z)
        DIMENSION A(0:20), X(20), Q(20)
        LOGICAL FLAG, FLAG1, FLAG2, PFLAG1, PFLAG2
        SAVE

        PFLAG2 = (CFLAG2.GT.0)

        IF (PFLAG1) THEN
            WRITE (3,5000) (X(I), I=1, M)
5000        FORMAT (' CHECK3: X ',20I3)
          ENDIF

        CALL XSLICK (M,D,X,Q,FLAG,PFLAG2)

        CALL SPGEN3 (M,F,A,FLAG1,PFLAG1)

C       START OF PATTERN LOOP
12      CONTINUE
        IF (FLAG1) THEN

            IF (PFLAG1) THEN
                WRITE (3,5100) (A(I), I=0, F)
5100            FORMAT (' CHECK3: A ',20I3)
              ENDIF

            CALL SPOIN3 (M,F,A,Q,FLAG2,PFLAG1)
```

```
C           START OF 'NEARBY LATTICE POINT' Q LOOP
14          CONTINUE
            IF (FLAG2) THEN

                IF (PFLAG1) THEN
                    WRITE (3,5200) (Q(I), I=1, M)
5200                FORMAT (' CHECK3: Q ',20I3)
                  ENDIF

C               INCREMENT COUNTER & PERFORM CHECK
                COUNT = COUNT + 1
                CALL SLICK (M,D,X,Q,FLAG,PFLAG2)

                IF (PFLAG2) THEN
                    CFLAG2 = CFLAG2-1
                    PFLAG2 = (CFLAG2.GT.0)
                  ENDIF

                IF (FLAG) THEN

                    IF (PFLAG1) THEN
                        WRITE (3,5300)
5300                    FORMAT (' CHECK3: THIS POINT PASSES.')
                        WRITE (3,5400)
                        WRITE (3,5400)
5400                    FORMAT ('  ')
                      ENDIF

C                   POINT X PASSES -- RETURN (FLAG WILL BE .TRUE.)
                    RETURN

                  ENDIF

C               GET NEXT 'NEARBY LATTICE POINT' Q

                IF (PFLAG1) THEN
                    WRITE (3,5500)
5500                FORMAT (' CHECK3: POINT FLUNKS WITH THIS Q')
                    WRITE (3,5550)
5550                FORMAT ('  ')
                  ENDIF

                CALL POINT3 (M,F,A,Q,FLAG2,PFLAG1)

              GOTO 14
              ENDIF
C             END OF 'NEARBY LATTICE POINT' Q LOOP

C           GET NEXT PATTERN
            CALL PGEN3 (M,F,A,FLAG1,PFLAG1)

          GOTO 12
          ENDIF
C         END OF PATTERN LOOP

C       IF WE REACH HERE, WE HAVE CHECKED AS MANY Q AS
C       THE BOUND 'F' ALLOWS.  THUS, THE POINT X FAILS.
```

```
      IF (PFLAG1) THEN
          WRITE (3,5600)
5600      FORMAT (' CHECK3: POINT HAS FLUNKED ALL TESTS')
          WRITE (3,5700)
          WRITE (3,5700)
5700      FORMAT ('  ')
        ENDIF

C     POINT X FAILS -- RETURN (FLAG WILL BE .FALSE.)
      RETURN

      END



      SUBROUTINE SPGEN3 (M,F,A,FLAG,PFLAG)

C     PATTERN GENERATOR.  RETURNS THE RESULT IN THE ARRAY A(I).
C     THE MULTISET \(\( 0**A(0), 1**A(1), ... \)\) CORRESPONDS TO A
C     AND THE POINT X =  SUM \( X(I)*E(I) \), WHERE
C     A(I) = #\( J: X(J) = I \), UNDER THE ASSUMPTION THAT
C     A(0) = #\( J: X(J) = 0 \)  1.

C     METHOD USED TO GENERATE PATTERNS IS TO CARRY A 1 FROM THE
C     FIRST NON ZERO A(I) TO A(I+1) AND IF I>0, DROP THE REST
C     OF A(I) TO A(0).

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

      IMPLICIT INTEGER (A-Z)
      DIMENSION A(0:20), B(0:20)
      LOGICAL FLAG,PFLAG
      SAVE

C     INITIALIZE

      A(0) = M
      B(0) = M - 1
      DO 6, I=1, F
          A(I) = 0
          B(I) = 0
6       CONTINUE

      FLAG = .TRUE.
      RETURN

      ENTRY PGEN3 (M,F,A,FLAG,PFLAG)

      IF (PFLAG) THEN
          WRITE (3,5000) (A(I), I=0, F)
5000      FORMAT (' PGEN3: OLD PATTERN',20I3)
        ENDIF

      I=0

C     FIND A NON ZERO B(I)
```

```
C       DO WHILE (B(I).EQ.0)
12      CONTINUE
        IF (B(I).EQ.0) THEN

            I = I+1

          GOTO 12
          ENDIF
C         REPEAT

        IF (I.EQ.F) THEN

            IF (PFLAG) THEN
                WRITE (3,5100) I
5100            FORMAT (' PGEN3: INDEX I',I4)
                WRITE (3,5200)
5200            FORMAT (' PGEN3:  NO MORE PATTERNS')
                WRITE (3,5300)
                WRITE (3,5300)
5300            FORMAT ('  ')
              ENDIF

C           OUT OF PATTERNS
            FLAG = .FALSE.
            RETURN

          ENDIF

C       CARRY THE 1 UPWARDS AND DROP THE REST DOWN.

        B(I+1) = B(I+1)+1
        A(I+1) = B(I+1)
        B(0) = B(I)-1
        A(0) = B(0)+1

C       IF I .GT. 0 THEN ZERO A(I) & B(I)

        IF (I.NE.0) THEN
            B(I) = 0
            A(I) = 0
          ENDIF

        IF (PFLAG) THEN
            WRITE (3,5400) I
5400        FORMAT (' PGEN3: INDEX I',I4)
            WRITE (3,5500) (A(I), I=0,F)
5500        FORMAT (' PGEN3: NEW PATTERN',20I3)
            WRITE (3,5600)
            WRITE (3,5600)
5600        FORMAT ('  ')
          ENDIF

        RETURN

        END
```

```
        SUBROUTINE SPOIN3 (M,F,A,X,FLAG,PFLAG)

C       HIS ROUTINE CONVERTS THE PATTERN IN A INTO THE POINT IN X.

C234567890123456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        IMPLICIT INTEGER (A-Z)
        DIMENSION A(0:20), X(20)
        LOGICAL FLAG, PFLAG
        SAVE

        M1 = M-1

C       INITIALIZE THE POINT

        J = 0

        DO 8, I=0,F

            DO 6, XX=1, A(I)
                J = J+1
                X(J) = I
6             CONTINUE

8         CONTINUE

        IF (PFLAG) THEN
            WRITE (3,5000) (A(I), I = 0, F)
5000        FORMAT (' SPOIN3: A ',20I3)
            WRITE (3,5200) (X(I),I=1, M)
5200        FORMAT (' SPOIN3: X ',20I3)
            WRITE (3,5300)
            WRITE (3,5300)
5300        FORMAT ('  ')
          ENDIF

        FLAG = .TRUE.
C       HAVE THE FIRST POINT & RETURN
        RETURN

        ENTRY POINT3 (M,F,A,X,FLAG,PFLAG)

C       GET NEXT POINT

        IF (PFLAG) THEN
            WRITE (3,5400) (X(I), I=1,M)
5400        FORMAT (' POINT3: OLD X ',20I3)
          ENDIF

C       FIRST FIND X(I) SUCH THAT
C       X(I) < X(I+1) >= X(I+2) >= ... >= X(M)

        I = M1

C       DO WHILE ( X(I) .GE. X(I+1) )
20      CONTINUE
```

```
        IF ( X(I) .GE. X(I+1) ) THEN

            IF (I.EQ.1) THEN

                IF (PFLAG) THEN
                    WRITE (3,5500)
5500                FORMAT (' POINT3: END POINT GENERATION')
                    WRITE (3,5600)
                    WRITE (3,5600)
5600                FORMAT ('  ')
                  ENDIF

C               POINT IN INVERSE LEXICOGRAPHICAL ORDER. RETURN
                FLAG = .FALSE.
                RETURN

              ENDIF

            I = I-1

          GOTO 20
          ENDIF
C         REPEAT

        J = I+1

C       FIND J SUCH THAT X(J) > X(I) AND X(J+1) <= X(I)
C       THAT IS, THE MINIMAL X(J) > X(I) WITH J > I.

C       DO WHILE ( (J.LT.M) .AND. (X(I).LT.X(J+1)) )
22      CONTINUE
        IF (J.LT.M) THEN

            IF ( X(I).LT.X(J+1) ) THEN

                J=J+1
                GOTO 22
C               REPEAT

              ENDIF

          ENDIF

        IF (PFLAG) THEN
            WRITE (3,5700) I,J
5700        FORMAT (' POINT3: POINTERS I,J', 2I4)
          ENDIF

C       NOW SWAP X(I) & X(J)

        K = X(I)
        X(I) = X(J)
        X(J) = K

C       NOW INVERT TAIL OF THE LIST: PUT IN INCREASING ORDER.

        I = I+1
        J = M
```

```
C       DO WHILE (I.LT.J)
24      CONTINUE
        IF (I.LT.J) THEN

            K = X(I)
            X(I) = X(J)
            X(J) = K
            I=I+1
            J=J-1

          GOTO 24
          ENDIF
C         REPEAT

        IF (PFLAG) THEN
            WRITE (3,5800) (X(I), I=1,M)
5800        FORMAT (' POINT3: NEW X ',20I3)
            WRITE (3,5900)
            WRITE (3,5900)
5900        FORMAT ('  ')
          ENDIF

        RETURN

        END


        SUBROUTINE SSLICK (M,D,X,Q,FLAG,PFLAG)

        IMPLICIT INTEGER (A-Z)
        LOGICAL FLAG,PFLAG

        DIMENSION X(20), Q(20), C(10)
        DIMENSION REALX(10), IMAGX(10)
        DIMENSION REALE(20,10), IMAGE(20,10)
        DIMENSION REALED(20,10), IMAGED(20,10)

        DOUBLE PRECISION RM, RD, ANGLE, C, CMIN, RADIUS, RADSQD
        DOUBLE PRECISION REALE, IMAGE, REALED, IMAGED, REALX, IMAGX
        DOUBLE PRECISION APPROX, BOUND, LENGTH, DIFF
        DOUBLE PRECISION TEMP1, TEMP2
        DOUBLE PRECISION OKBND, NOKBND, DIFBND

        SAVE

C       THE ARITHMETIC IS SLICKED UP TO DOUBLE PRECISION &
C       COMPLEX NUMBERS HAVE REAL & IMAGINARY. PARTS SEPARATED.

C       THIS ROUTINE IS UPGRADED WITH AN ITERATIVE PROCEDURE TO
C       FIND THE MAXIMUM.

C23456789012345678901234567890123456789012345678901234567890123456
C        1         2         3         4         5         6

        RM = DBLE(M)
        RD = DBLE(D)
        M1 = M - 1
```

```
      D1 = D - 1
      S = M1 / 2
      ANGLE = 6.28318 53071 79586 D0 / RM
      RADSQD = DBLE( ( M*M - 1 ) / 24 )
      RADIUS = DSQRT( RADSQD ) / RD
      RADSQD = RADSQD / ( RD * RD )

      DIFBND = 1D-10
      CNTBND = 100
      OKBND = 1 - 1D-6
      NOKBND = 1 + 1D-6

C     NOTE    E = ZETA ** (I*J)
C             ED = E / D

      DO 4, I=1,M

          DO 2, J=1,S
              REALE(I,J) = DCOS( ANGLE * MOD(I*J,M) )
              IMAGE(I,J) = DSIN( ANGLE * MOD(I*J,M) )
              REALED(I,J) = REALE(I,J) / RD
              IMAGED(I,J) = IMAGE(I,J) / RD
2           CONTINUE

4       CONTINUE

      IF (PFLAG) THEN

          WRITE (3,5000) RADIUS, RADSQD
5000      FORMAT (' SSLICK: RADIUS, RADSQD ',2F16.12)
          WRITE (3,5100) OKBND,NOKBND
5100      FORMAT (' SSLICK: OKBND,NOKBND ',2F16.12)
          WRITE (3,5200)
5200      FORMAT ('  ')

          DO 6, J=1,S

              WRITE (3,5300) J
5300          FORMAT (' SSLICK: CONJUGATE NUMBER', I4)
              WRITE (3,5400) ( REALE(I,J), IMAGE(I,J),
     *          REALED(I,J), IMAGED(I,J), I=1,M )
5400          FORMAT (' SSLICK: E,ED',4F16.12)
              WRITE (3,5500)
5500          FORMAT ('  ')

6           CONTINUE

          WRITE (3,5600)
          WRITE (3,5600)
5600      FORMAT ('  ')

        ENDIF

      RETURN


      ENTRY XSLICK (M,D,X,Q,FLAG,PFLAG)
```

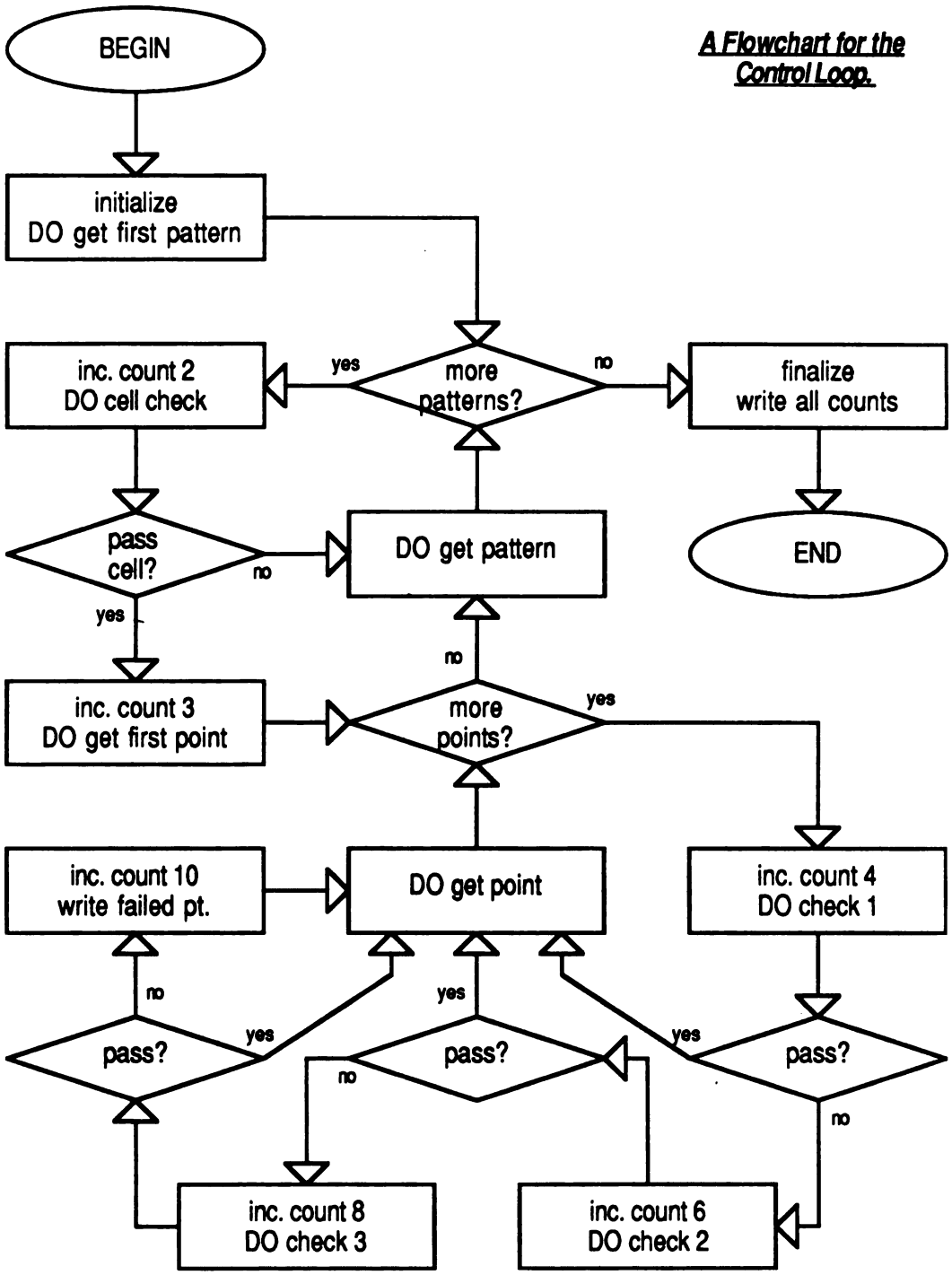
```
C       REALX(J) = REAL PART OF JTH CONJUGATE OF X.
C       IMAGX(J) = IMAGINARY PART OF JTH CONJUGATE OF X.

        DO 18, J=1,S

            REALX(J) = 0D0
            IMAGX(J) = 0D0

            DO 16, I=1,M
                REALX(J) = X(I)*REALED(I,J) + REALX(J)
                IMAGX(J) = X(I)*IMAGED(I,J) + IMAGX(J)
16            CONTINUE

18        CONTINUE

        IF (PFLAG) THEN
            WRITE (3,5700) (X(I), I=1,M)
5700        FORMAT (' XSLICK: X ',20I3)
            WRITE (3,5800) (J,REALX(J),IMAGX(J), J=1,S)
5800        FORMAT (' XSLICK: J,REALX,IMAGX ',I4,2F16.12)
            WRITE (3,5900)
            WRITE (3,5900)
5900        FORMAT ('  ')
          ENDIF

        RETURN


        ENTRY SLICK (M,D,X,Q,FLAG,PFLAG)

        IF (PFLAG) THEN
            WRITE (3,6000) (X(I), I=1,M)
6000        FORMAT (' SLICK: X ',20I3)
            WRITE (3,6100) (Q(I), I=1,M)
6100        FORMAT (' SLICK: Q ',20I3)
          ENDIF

C       DETERMINE THE C(J) = \!Z(J)\! & FIND THE MINIMAL ONE.
        CMIN = 1D10
        DO 22, J=1,S

            TEMP1 = REALX(J)
            TEMP2 = IMAGX(J)

            DO 20, I=1,M

                TEMP1 = TEMP1 - Q(I)*REALE(I,J)
                TEMP2 = TEMP2 - Q(I)*IMAGE(I,J)

20            CONTINUE

            C(J) = DSQRT( TEMP1*TEMP1 + TEMP2*TEMP2 )
            IF ( C(J) .LT. CMIN ) THEN
                CMIN = C(J)
              ENDIF

            IF (PFLAG) THEN
```

```
                WRITE (3,6200) J,TEMP1,TEMP2,C(J)
6200            FORMAT (' SLICK: J,TEMP1,TEMP2,C',I4,3F16.12)
              ENDIF

22          CONTINUE

          IF (PFLAG) THEN
              WRITE (3,6300) CMIN
6300          FORMAT (' SLICK: CMIN ',F16.12)
            ENDIF

          APPROX = RADIUS
          COUNT = 0

C         ITERATION LOOP TO COMPUTE MAXIMUM
C         DO WHILE (COUNT.LT.CNTBND)
40        CONTINUE
          IF (COUNT.LT.CNTBND) THEN

              COUNT = COUNT + 1

              TEMP1 = (APPROX+CMIN)*APPROX*4D0
              LENGTH = 0D0
              BOUND = 1D0

C             COMPUTE A NEW BOUND & LENGTH
              DO 42, J=1,S
                  TEMP2 = (DSQRT(C(J)*C(J)+TEMP1)-C(J))*.5D0
                  BOUND = ( TEMP2+C(J) ) * BOUND
                  LENGTH = TEMP2*TEMP2 + LENGTH
42              CONTINUE

              DIFF = LENGTH - RADSQD

              IF (PFLAG) THEN
                  WRITE (3,6400) APPROX, BOUND
6400              FORMAT (' SLICK: APPROX, BOUND',2F16.12)
                  WRITE (3,6450) LENGTH, DIFF
6450              FORMAT (' SLICK: LENGTH, DIFF ',2F16.12)
                ENDIF

              IF (DIFF.GE.-DIFBND) THEN

                  IF (BOUND.LT.OKBND) THEN

                      IF (PFLAG) THEN
                          WRITE (3,6500)
6500                      FORMAT (' SLICK: THIS POINT PASSES.')
                          WRITE (3,6600)
                          WRITE (3,6600)
6600                      FORMAT ('  ')
                        ENDIF

                      FLAG = .TRUE.
                      RETURN

                    ENDIF
```

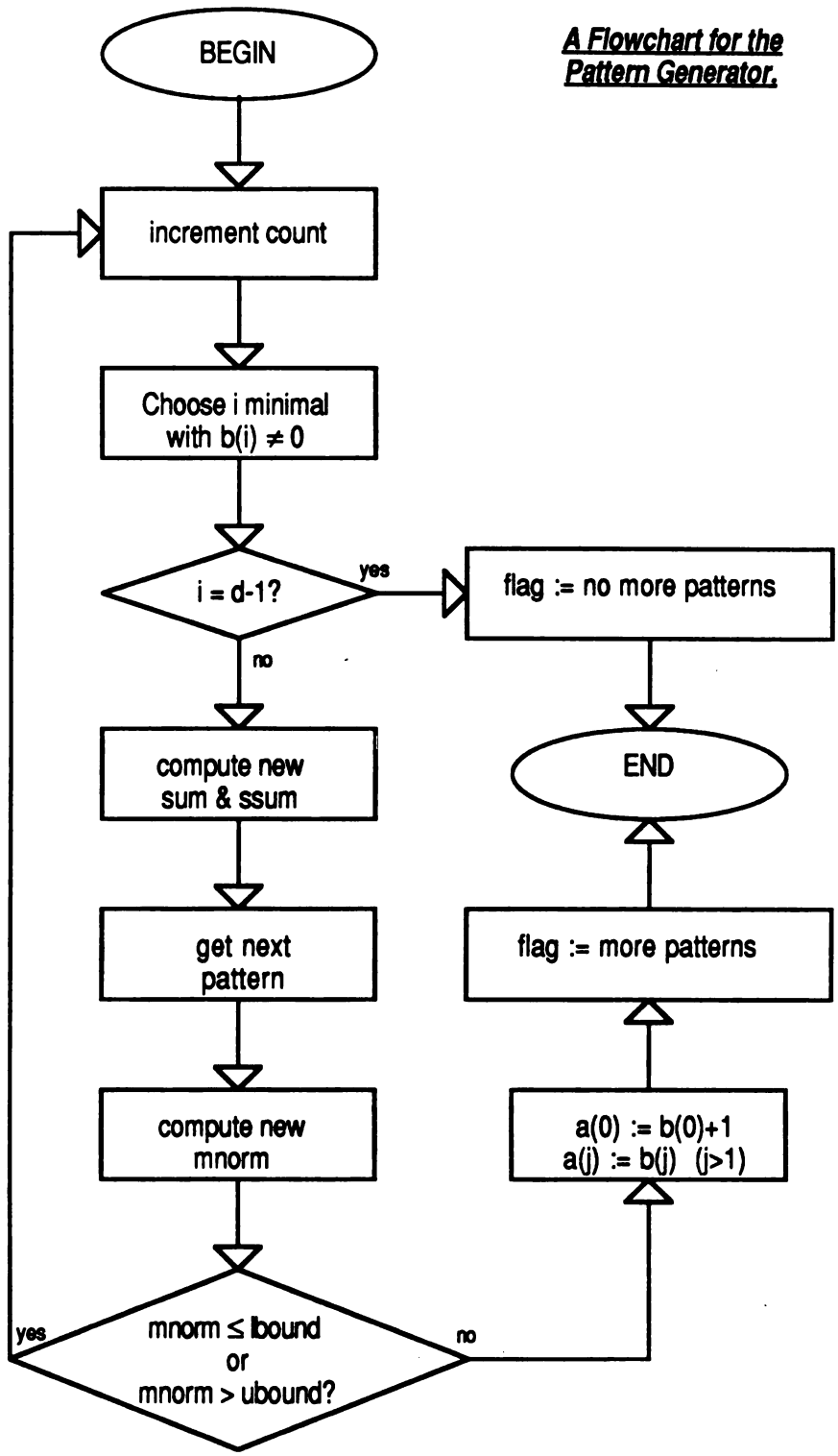
```
              ENDIF

            IF (DIFF.LE.DIFBND) THEN

                IF (BOUND.GT.NOKBND) THEN

                    IF (PFLAG) THEN
                        WRITE (3,6800)
6800                    FORMAT (' SLICK: THIS POINT FLUNKS.')
                        WRITE (3,6900)
                        WRITE (3,6900)
6900                    FORMAT ('  ')
                      ENDIF

C                     THIS ONE FLUNKS.  RETURN.
                      FLAG = .FALSE.
                      RETURN

                    ENDIF

              ENDIF

            APPROX= APPROX*RADIUS/DSQRT(LENGTH)

C           LOOP BACK & REITERATE BOUNDS
            IF ( ABS(DIFF).GT.DIFBND ) THEN
                GOTO 40
              ENDIF
          ENDIF

        IF (PFLAG) THEN
            WRITE (3,7000)
7000        FORMAT (' SLICK: THIS POINT FLUNKS.')
            WRITE (3,7100)
            WRITE (3,7100)
7100        FORMAT ('  ')
          ENDIF

        FLAG = .FALSE.

        RETURN


C23456789012345678901234567890123456789012345678901234567890123456
C       1         2         3         4         5         6

        END
```

Appendix B

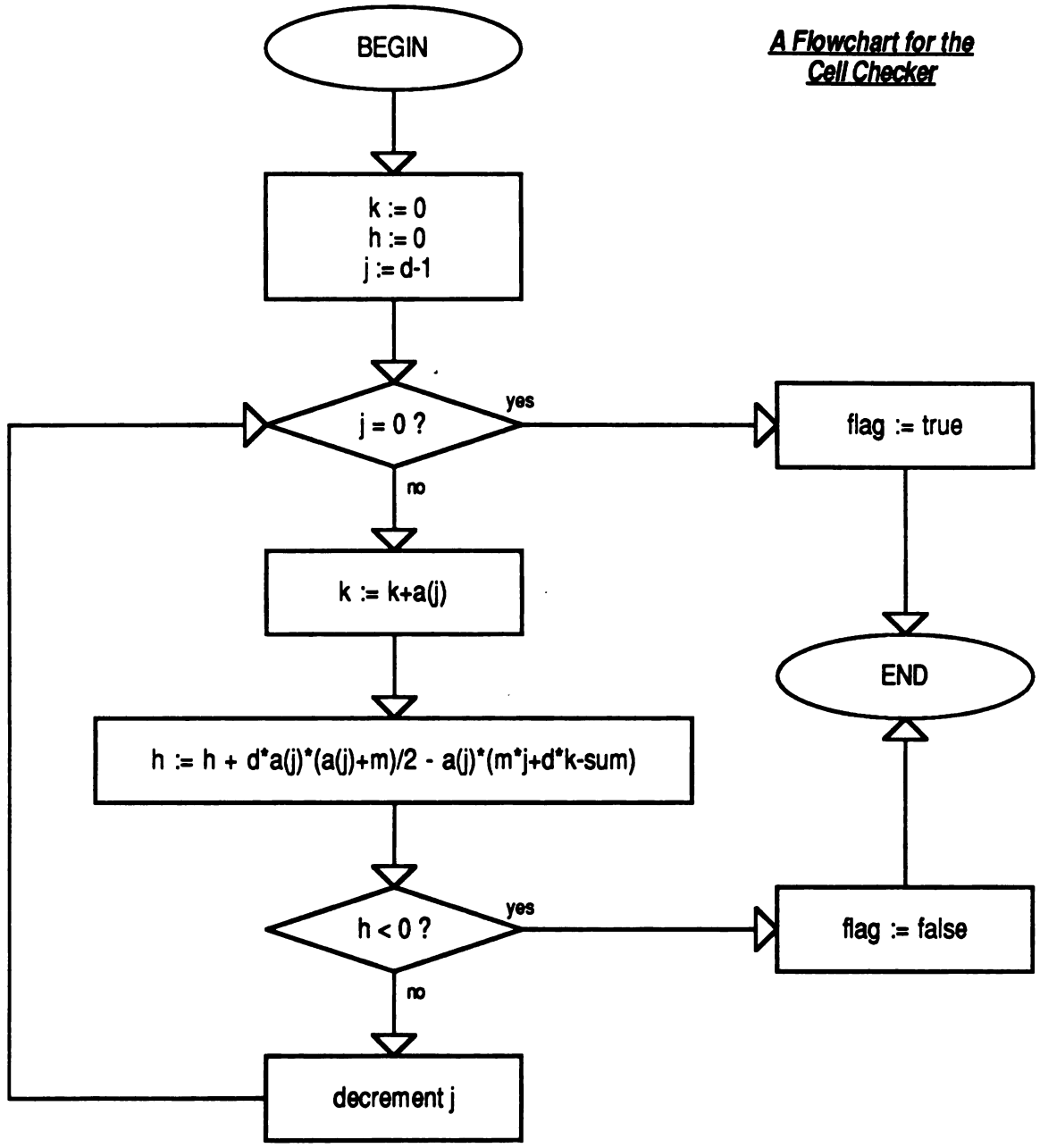
Flowcharts



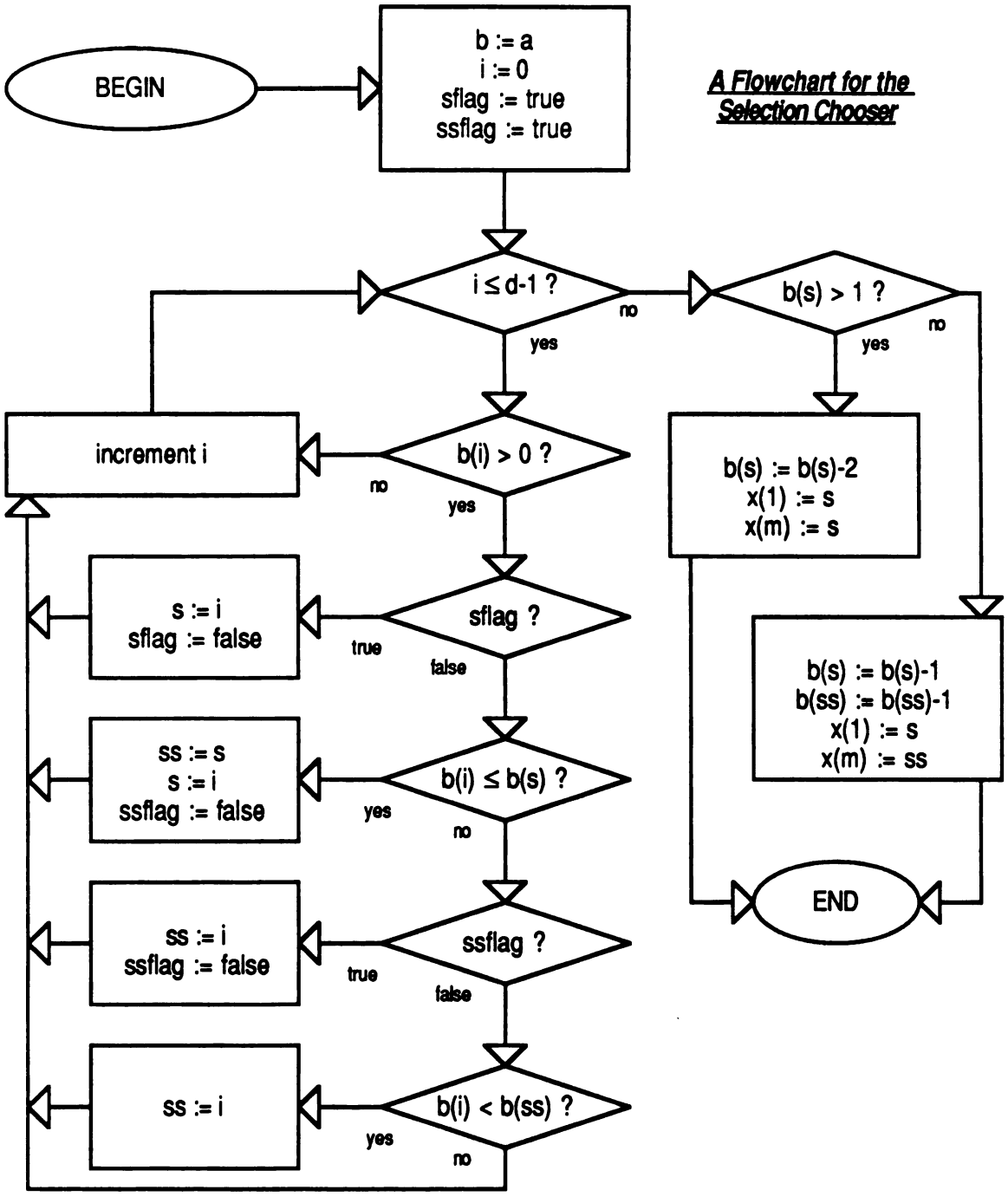
A Flowchart for the Control Loop.



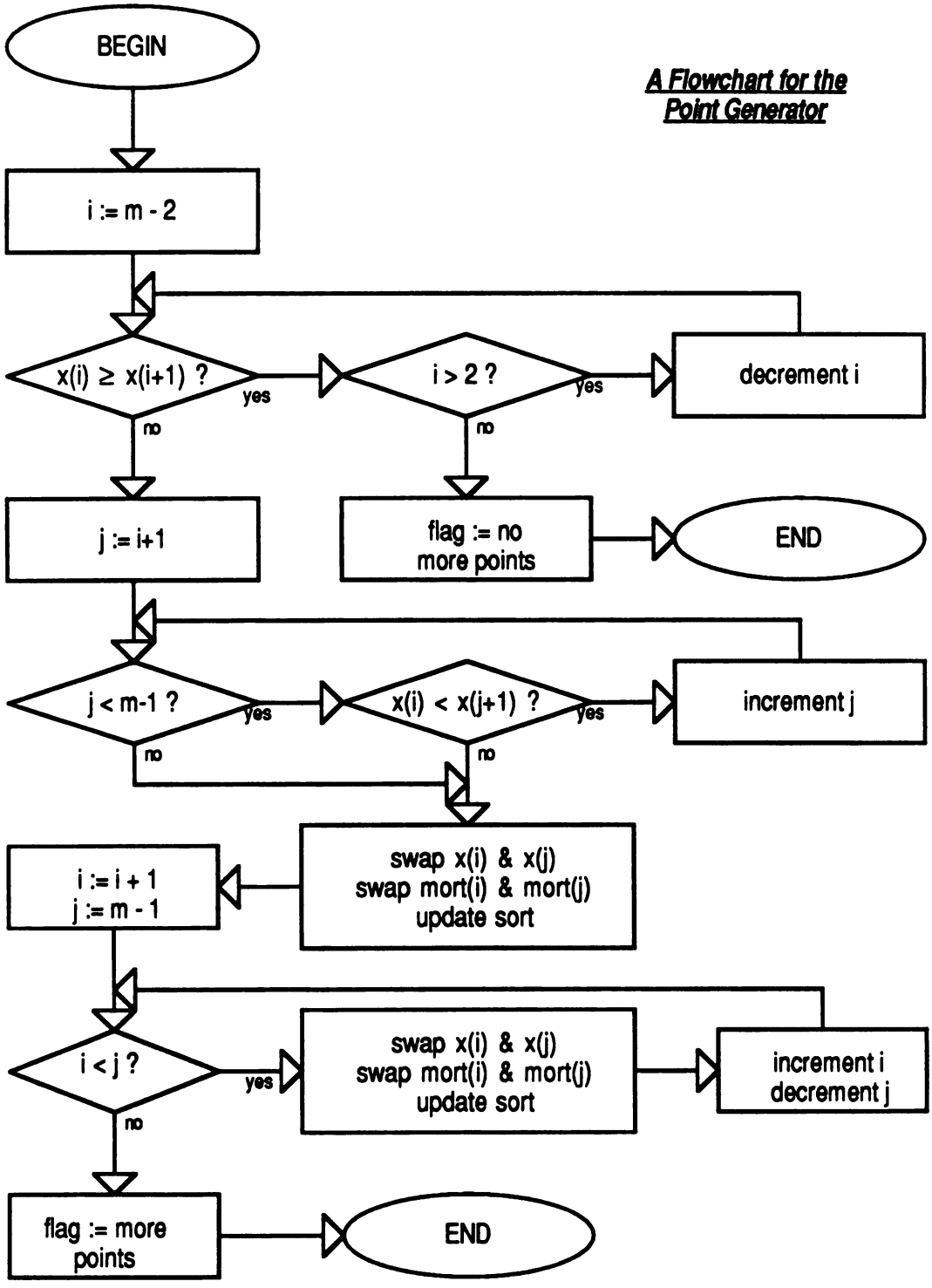
A Flowchart for the Pattern Generator.



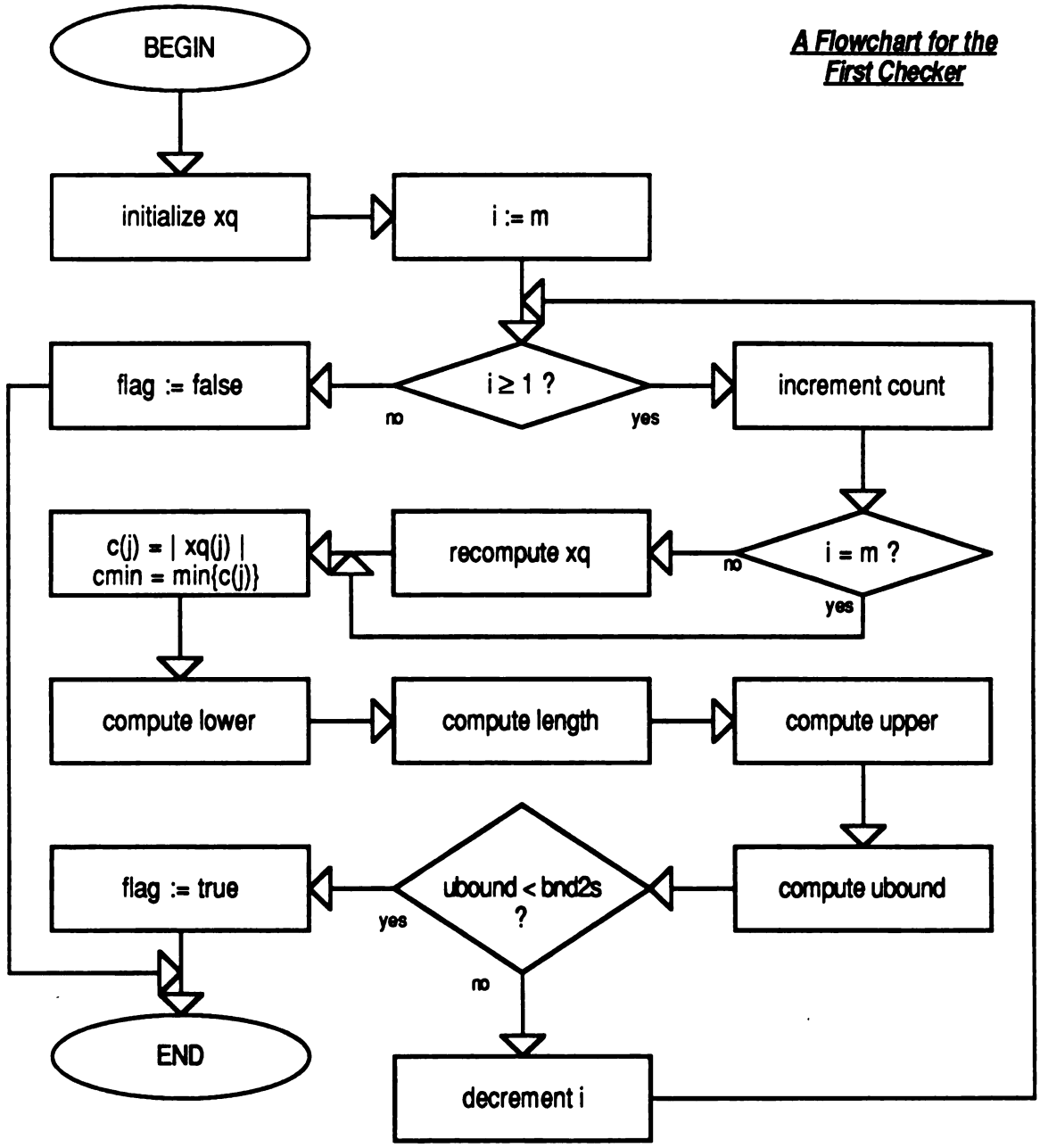
A Flowchart for the Cell Checker



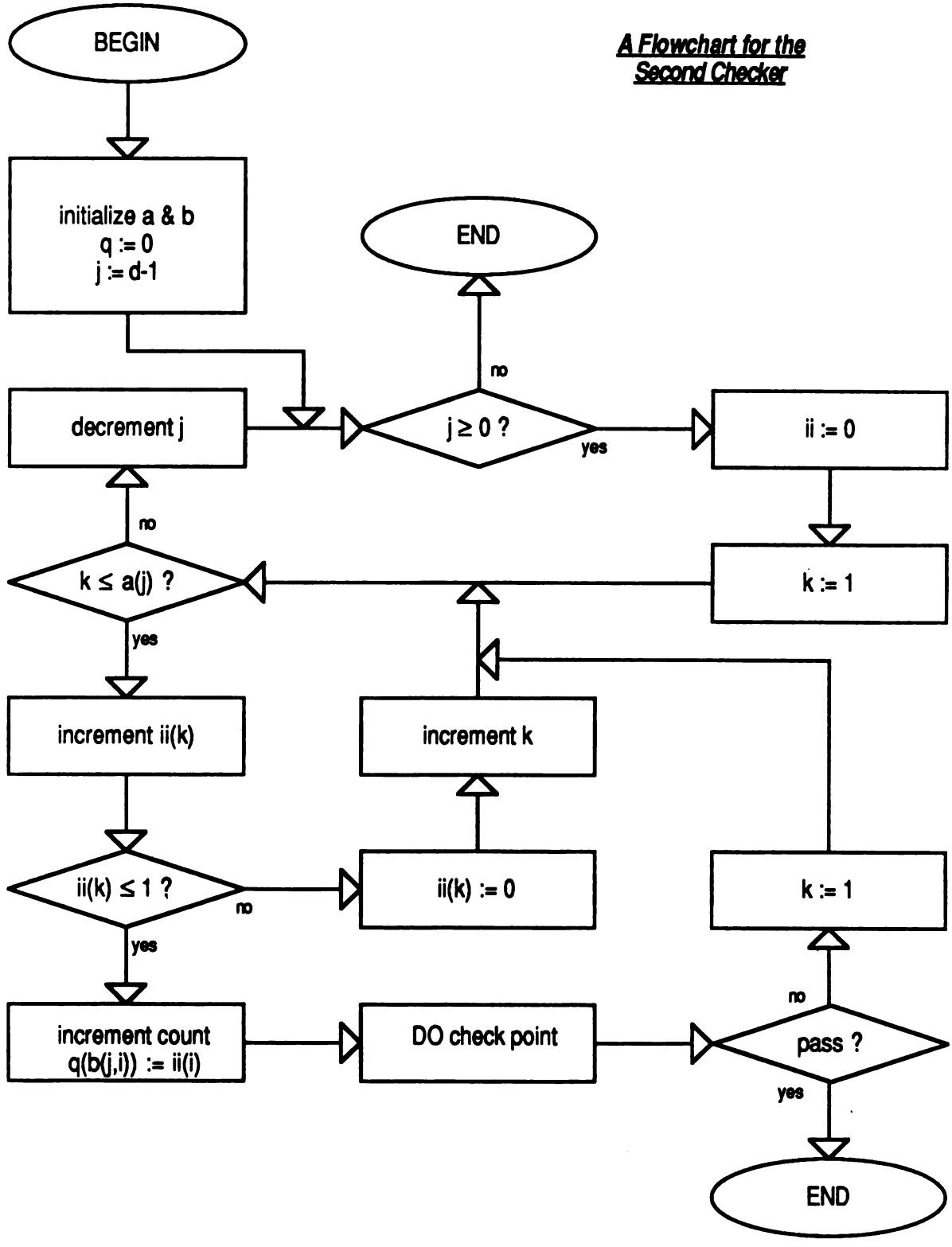
A Flowchart for the Selection Chooser



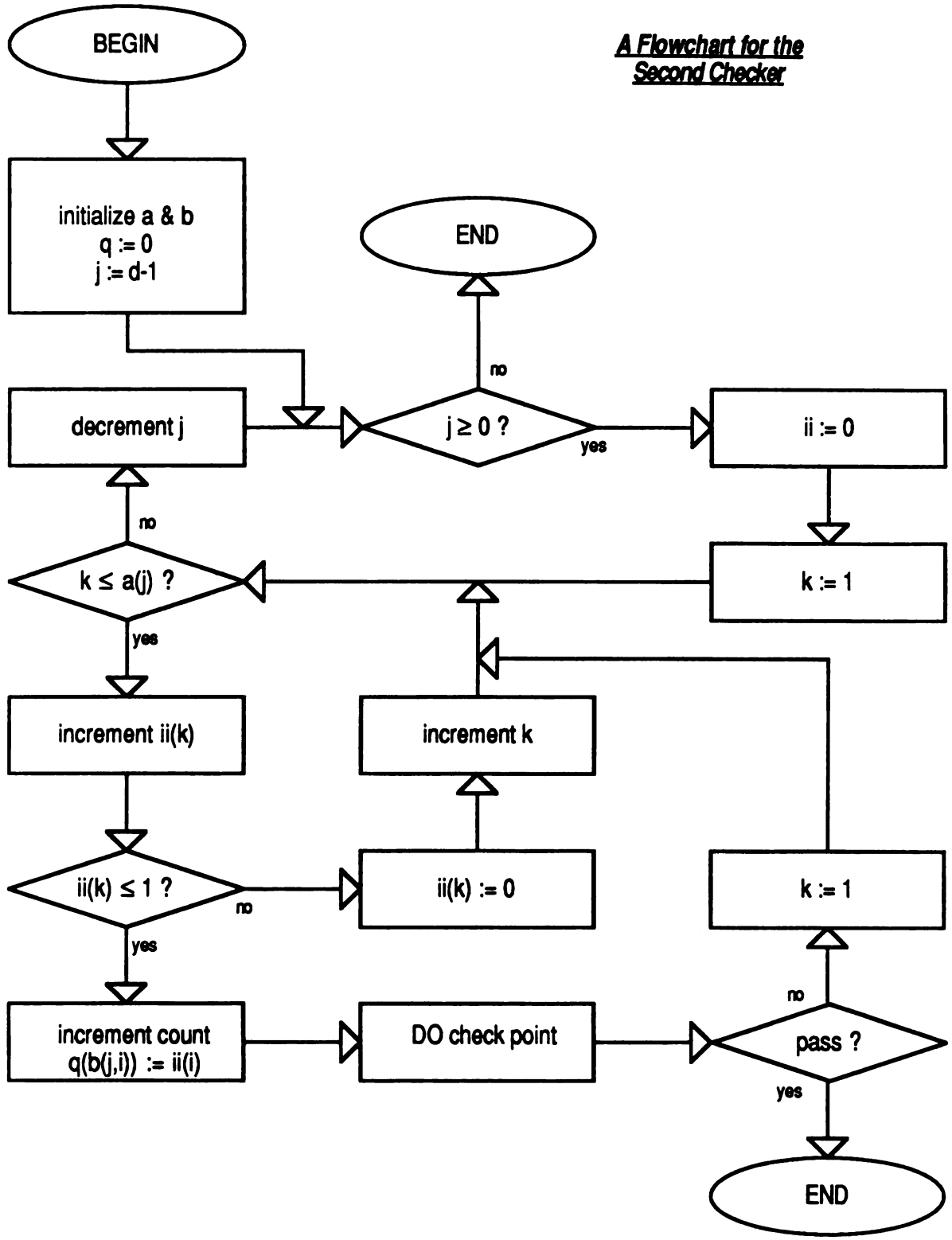
A Flowchart for the Point Generator



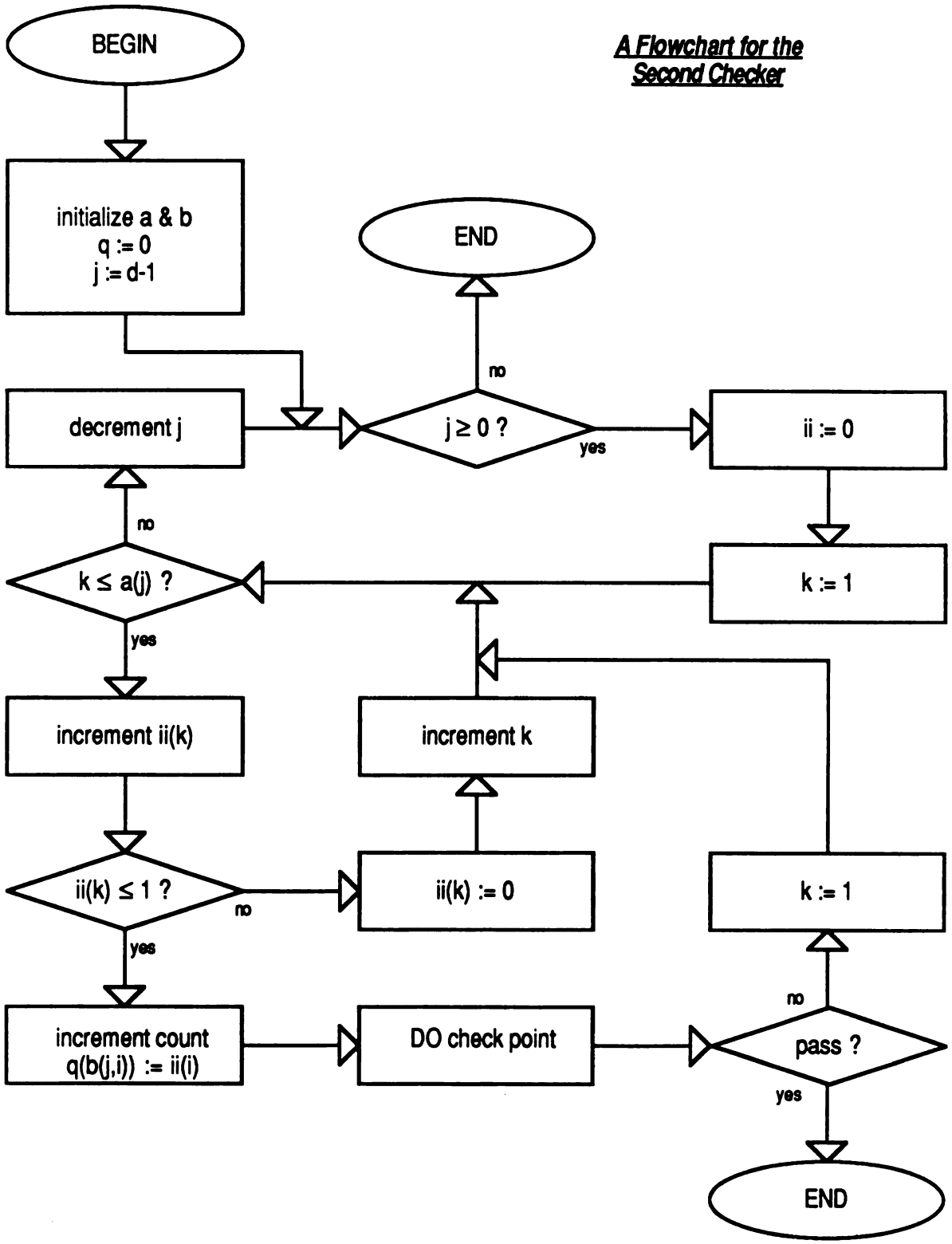
A Flowchart for the First Checker



A Flowchart for the Second Checker



A Flowchart for the Second Checker



A Flowchart for the Second Checker

BIBLIOGRAPHY

BIBLIOGRAPHY

[B] E. S. Barnes, H. P. F. Swinnerton-Dyer, *The inhomogeneous minima of binary quadratic forms*, Acta Math. **87** (1952) 259-323.

[Ch1] H. Chatland, *On the euclidean algorithm in quadratic number fields*, Bull. AMS **55** (1949) 948-953.

[Ch2] H. Chatland, H. Davenport, *Euclid's algorithm in real quadratic fields*, Canad. J. Math. **2** (1950) 289-296.

[Co] H. Cohn, *Advanced number theory*, Dover Publications, Inc., New York, 1962.

[Cu] C. W. Curtis, I. Reiner, *Representation Theory of Finite Groups and Associative Algebras*, Interscience Publishers, New York, 1962.

[Ei] G. Eisenstein, *Mathematische Werke* Band II, Chelsea Publishing Company, New York, 1975.

[Eu] Euclid, *Elements*.

[Ga] K. F. Gauss, *Werke* Zweiter Band, Georg Olms Verlag, Hildesheim and New York, 1973.

[Gu] R. Gupta, M. R. Murty, V. K. Murty, *The Euclidean algorithm for S-integers*, Number theory (Montreal, Que., 1985), CMS Conference Proceedings, **7**, (A.M.S., 1987).

[La] R. B. Lakein, *Euclid's algorithm in complex quartic fields*, Acta Arith. **20** (1972), 393-400.

[Le1] H. W. Lenstra, Jr., *Lectures on euclidean rings*, Bielfeld, 1974.

[Le2] H. W. Lenstra, Jr., *Euclid's algorithm in cyclotomic fields*, J. London Math. Soc. (2) **10** (1975), 457-465.

[Le3] H. W. Lenstra, Jr., *Quelques exemples d'anneaux euclidiens*, C. R. Acad. Sc. Paris, Sér. A, **286** (1978), 683-685.

[Le4] H. W. Lenstra, Jr., *Euclidean number fields I*, Math. Intell. **2** (1979), 6-15.

[Le5] H. W. Lenstra, Jr., *On Artin's Conjecture and Euclid's Algorithm in Global Fields*, Inventiones Math. **42** (1977), 201-224

[Ma] J. M. Masley, H. L. Montgomery, *Cyclotomic fields with unique factorization*, J. Reine Angew. Math. **286/287** (1976), 248-256.

[Mo] T. Motzkin, *The Euclidean algorithm,* Bull. A. M. S. **55** (1949), 1142-1146.

[N] A. Nijenhuis, H. S. Wilf, *Combinatorial Algorithms,* Academic Press, New York, 1975.

[Oj] T. Ojala, *Euclid's algorithm in the cyclotomic field* $\mathbf{Q}(\zeta_{16})$, Math. Comp. **31** (1977), 268-273.

[Ou] J. Ouspensky, *Note sur les nombres entiers dépendant d'une racine cinquième de l'unité,* Math. Ann. **66** (1909), 109-112.

[R] F. S. Roberts, *Applied Combinatorics,* Prentice-Hall, New Jersey, 1975.

[S] H. M. Stark, *A complete determination fo the complex quadratic fields of calss-number one,* Mich. Math. J. **14** (1967), 1-24.

[Wa] P. L. Wantzel, Comptes Rendus l'Académe des Sciences **24** (1847).

[We] P. J. Weinberger, *On euclidean rings of algebraic integers,* Proc. Symp. Pure Math., **24** (Analytic Number Theory), 321-332 (A.M.S., 1973).