THE INTEGER CHEBYSHEV PROBLEM: COMPUTATIONAL EXPLORATIONS

by

Alan Meichsner

M.Sc. Mathematics, Simon Fraser University, 2001

B.Sc. Mathematical Sciences, Okanagan University College, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the Department
of
Mathematics

SIMON FRASER UNIVERSITY
Spring 2009

APPROVAL

Name:	Alan Meichsner
Degree:	Doctor of Philosophy
Title of thesis:	The Integer Chebyshev Problem: computational explorations
Examining Committee:	Dr. Tom Archibald Chair
	Dr. Peter Borwein, Senior Supervisor
	Dr. Petr Lisonek, Supervisor
	Dr. Stephen Choi, Supervisor
	Dr. Vahid Dabbaghian, Internal Examiner
	Dr. Chris Smyth, External Examiner, University of Edinburgh, Scotland
Date Approved:	March 30, 2009

Declaration of
Partial Copyright Licence

Abstract

The integer Chebyshev problem deals with finding polynomials of degree at most n with integer coefficients having minimal supremum norm on a domain D and then analyzing the nth root behavior of the supremum norm of these polynomials as n tends to infinity. The limiting value of the nth root of the supremum norms is called the integer Chebyshev constant for D and the polynomials are called nth integer Chebyshev polynomials on D. The problem has a long history on the intervals $[0, 1]$ and $[0, 1/4]$, and although the structure of such polynomials is not well understood, it is known that certain critical polynomials must be factors of relatively high multiplicity. Progress is made here by extending the method of Wu to include all known critical polynomials on these intervals which results in an increase in the best known lower bounds for the degree to which many of these critical polynomials must divide an nth integer Chebyshev polynomial. For the unrestricted case on $[0, 1]$, this results in an increase for all known critical polynomials. On the interval $[0, 1/4]$, it results in an increase for all known nonlinear critical polynomials. Towards finding nth integer Chebyshev polynomials, each stage of the method of Habsieger and Salvy is improved and nth integer Chebyshev polynomials for the unit interval are given for all n less than or equal to 145. As a final result in the single variable case, the upper bound on the integer Chebyshev constant of the unit interval is decreased to $1/2.36482727$. The move to the bivariate case on the unit square is then made in two ways and the basic results, which include existence of the limits, initial bounds on the integer Chebyshev constants, symmetry conditions, and the method of computation are extended. For the computation of nth integer Chebyshev polynomials on the unit square, a new application of the integer relation algorithm PSLQ is used to take advantage of symmetry.

In memory of my father, Paul Franz Meichsner.

Contents

List of Tables

List of Figures

Chapter 1

The Integer Chebyshev Problem

1.1 Introduction

For any nonnegative integer n, the standard Chebyshev polynomials on $[-1,1]$ and their transformations to other intervals give rise, upon normalization, to the class of monic polynomials of degree n with minimal supremum norm.

Definition 1.1 *For each integer $n \geq 0$, let $T_n(x) = \cos(n \arccos(x))$ with the restriction that $x \in [-1,1]$. We call $T_n(x)$ the nth Chebyshev polynomial.*

Using the notation $\|p(x)\|_{[a,b]}$ for the supremum norm of the polynomial $p(x)$ on the interval $[a,b]$, it is easily seen that $\|T_n(x)\|_{[-1,1]}$ equals 1, $|T_n(x)| = 1$ at $n+1$ points in the interval $[-1,1]$, and that for $n \geq 1$, $T_n(x)$ has exactly n simple zeros.

Lemma 1.1 *For $n \geq 1$, the nth Chebyshev polynomial $T_n(x)$ takes the value 0 at the n points*

$$x_k = \cos\left(\left(\frac{2k+1}{2n}\right)\pi\right), \quad k = 0, 1, \ldots, n-1$$

and assumes its absolute extrema at the $n+1$ points

$$x'_k = \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, \ldots, n$$

with $T_n(x'_k) = (-1)^k$.

Proof: If $T_n(x) = \cos(n \arccos(x)) = 0$ then $n \arccos(x) = k\pi + \pi/2$ for some $k \in \mathbb{Z}$ and so

$$x = \cos\left(\left(\frac{2k+1}{2n}\right)\pi\right), \quad k = 0, 1, \ldots, n-1$$

as $(2k+1)\pi/2n$ must lie in the interval $[0,\pi]$.
For the second part, if $T_n'(x) = n\sin(n\arccos(x))/\sqrt{1-x^2} = 0$ then $n\arccos(x) = k\pi$ or

$$x = \cos\left(k\pi/n\right), \quad k = 0, 1, \ldots, n.$$

As $T_n(x_k') = \cos(n\arccos(\cos(k\pi/n))) = \cos(k\pi) = (-1)^k$, we are done.

□

To show that we can extend the definition of $T_n(x)$ to all values of x, and that the resulting functions are in fact polynomials, we make the substitution $\theta = \arccos(x)$.

$$\begin{aligned} T_n(\theta) &= \cos(n\theta) \text{ for } \theta \in [0,\pi] \\ T_{n+1}(\theta) &= \cos((n+1)\theta) = \cos(n\theta)\cos(\theta) - \sin(n\theta)\sin(\theta) \\ T_{n-1}(\theta) &= \cos((n-1)\theta) = \cos(n\theta)\cos(\theta) + \sin(n\theta)\sin(\theta) \end{aligned}$$

This gives

$$T_{n+1}(\theta) = 2\cos(n\theta)\cos(\theta) - T_{n-1}(\theta) = 2\cos(\theta)T_n(\theta) - T_{n-1}(\theta)$$

and so upon returning to the variable x, we get

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \text{ for each integer } n \geq 1.$$

Now as $T_0(x) = \cos(0\arccos(x)) = 1$ and $T_1(x) = \cos(1\arccos(x)) = x$ we have the following iterative definition for the Chebyshev polynomials which allows us to extend the domain from $[-1,1]$ to $\mathbb{R}$.

Definition 1.2 *The Chebyshev polynomials can be defined iteratively as follows.*

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned}$$

From this second definition it is clear that $T_n(x)$ is a polynomial of degree n with leading coefficient 2^{n-1} when $n \geq 1$. If we divide $T_n(x)$ by its leading coefficient, then the resulting polynomial can be shown to be the monic polynomial of degree n with minimal supremum norm on the interval $[-1,1]$.

Theorem 1.1 *Let* $\widetilde{T}_n(x) = \frac{1}{2^{n-1}} T_n(x)$ *and let* $\widetilde{\mathcal{P}}_n$ *be the set of all monic polynomials of degree* n *with coefficients in* $\mathbb{R}$*. Then for* $n \geq 1$

$$\frac{1}{2^{n-1}} = \left\| \widetilde{T}_n(x) \right\|_{[-1,1]} \leq \left\| \widetilde{p}(x) \right\|_{[-1,1]} \text{ for any } \widetilde{p}(x) \in \widetilde{\mathcal{P}}_n.$$

Proof: By way of contradiction, suppose $\widetilde{p}(x) \in \widetilde{\mathcal{P}}_n$ and $\left\| \widetilde{p}(x) \right\|_{[-1,1]} < 1/2^{n-1}$. Let the polynomial $q(x) = \widetilde{T}_n(x) - \widetilde{p}(x)$. As both $\widetilde{T}_n(x)$ and $\widetilde{p}(x)$ are monic polynomials of degree n, $q(x)$ is a polynomial of degree at most $n-1$. Let the $(n+1)$ points x'_k be as in Lemma 1.1 so that $\widetilde{T}_n(x'_k) = (-1)^k/2^{n-1}$. Then as $|\widetilde{p}(x'_k)| < 1/2^{n-1}$, $q(x'_k) \neq 0$. In fact $q(x'_k) < 0$ when k is odd and $q(x'_k) > 0$ when k is even. Now since $q(x)$ is continuous, the Intermediate Value Theorem ensures $q(x)$ has a zero somewhere in the interval (x'_k, x'_{k+1}) for each $k = 0, 1, \ldots, n-1$. As $q(x)$ has n zeros and is of degree less than or equal to $n-1$ it follows that $q(x)$ is the zero polynomial. This implies that $\widetilde{p}(x) = \widetilde{T}_n(x)$. As $\left\| \widetilde{p}(x) \right\|_{[-1,1]} < \left\| \widetilde{T}_n(x) \right\|_{[-1,1]}$ we have a contradiction.

□

With a little more work one can show that if $\widetilde{p}(x) \in \widetilde{\mathcal{P}}_n$ and $\left\| \widetilde{p}(x) \right\|_{[-1,1]} = \left\| \widetilde{T}_n(x) \right\|_{[-1,1]}$ then $\widetilde{p}(x) = \widetilde{T}_n(x)$. In this case, if $\widetilde{T}_n(x) - \widetilde{p}(x) = 0$ at one of the points x'_k for $1 \leq k \leq n-1$, then we may lose a point of intersection, but we gain a point where $q(x)$ has a double zero. This enables us to still claim $q(x)$ has n zeros when we count multiplicities and show $q(x) = 0$.

The above properties can be used to show that for an arbitrary interval $[a, b]$

$$P_n(x) = 2 \left(\frac{b-a}{4} \right)^n T_n \left(\frac{2x-a-b}{b-a} \right)$$

is the monic polynomial of degree n with minimal sup norm. As $P_n(x) = \left(\frac{b-a}{2}\right)^n \frac{1}{2^{n-1}} T_n(u)$ where $u = (2x-a-b)/(b-a)$ and $x \in [a, b]$ implies that $u \in [-1, 1]$, we see that

$$\left\| P_n(x) \right\|_{[a,b]} = \left(\frac{b-a}{2} \right)^n \left\| \widetilde{T}_n(u) \right\|_{[-1,1]} = \left(\frac{b-a}{2} \right)^n \frac{1}{2^{n-1}} = 2 \left(\frac{b-a}{4} \right)^n.$$

Any other monic polynomial $r(x)$ of degree n can be written in the form

$$\left(\frac{b-a}{2} \right)^n (u^n + c_{n-1} u^{n-1} + \cdots + c_0) = \left(\frac{b-a}{2} \right)^n \widetilde{r}(u) \text{ where } u = \frac{2x-a-b}{b-a}.$$

Now if $\left\| r(x) \right\|_{[a,b]} < \left\| P_n(x) \right\|_{[a,b]}$ then this would imply that $\left\| \widetilde{r}(u) \right\|_{[-1,1]} < \left\| \widetilde{T}_n(u) \right\|_{[-1,1]}$ which cannot happen. We have arrived at the following result.

Theorem 1.2 *Let $\widetilde{\mathcal{P}}_n$ be the set of all monic polynomials of degree n with coefficients in $\mathbb{R}$. Then*

$$\min_{p(x)\in\widetilde{\mathcal{P}}_n} \|p(x)\|_{[a,b]} = 2\left(\frac{b-a}{4}\right)^n.$$

1.2 The Integer Chebyshev Problem

The problem becomes much harder if we restrict ourselves to considering only polynomials of degree n with integer coefficients. This gives rise to the Integer Chebyshev Problem, or the problem of minimal Diophantic deviation from zero.

Definition 1.3 *Let $\mathcal{Z}_n[x]$ be the set of polynomials of degree at most n with integer coefficients. For $n > 0$, define $\Omega_n[a,b]$ as*

$$\Omega_n[a,b] = \left(\min_{p\in\mathcal{Z}_n[x]\setminus\{0\}} \|p(x)\|_{[a,b]}\right)^{1/n}$$

and let

$$\Omega[a,b] = \lim_{n\to\infty} \Omega_n[a,b].$$

We call $\Omega[a,b]$ the integer transfinite diameter or integer Chebyshev constant for the interval $[a,b]$. Any polynomial $p(x) \in \mathcal{Z}_n[x]$ that satisfies $(\|p(x)\|_{[a,b]})^{1/n} = \Omega_n[a,b]$ is called an nth integer Chebyshev polynomial on $[a,b]$.

Since an nth integer Chebyshev polynomial on $[0,1]$ must be of degree n for $n \geq 2$ and an nth integer Chebyshev polynomial on $[0,1/4]$ must be of degree n for any n, the term integer Chebyshev polynomial of degree n is also used when working on these intervals.

Markov's inequality [26] bounding the mth derivative of a polynomial p of degree n with real coefficients can be used to show the minimum value is achieved in the definition of $\Omega_n[a,b]$. Since

$$\left\|p^{(m)}(x)\right\|_{[a,b]} \leq \frac{2^m}{(b-a)^m}\frac{n^2(n^2-1^2)(n^2-2^2)\cdots(n^2-(m-1)^2)}{(2m-1)!!}\|p(x)\|_{[a,b]}$$

where $(2m-1)!! = 1\cdot 3\cdot 5\cdots(2m-1)$, there are only finitely many polynomials with integer coefficients of supremum norm less than or equal to a given constant k on $[a,b]$. If $p(x) \in \mathcal{Z}_n[x]$ and $\|p(x)\|_{[a,b]} \leq k$, then applying the inequality to the nth derivative of $p(x)$ gives a finite number of choices for the coefficient of x^n. For each of these possibilities, Markov's inequality again shows there are only a finite number of choices for the coefficient

of x^{n-1}. Continuing in this fashion, we arrive at the fact that only finitely many polynomials in $\mathcal{Z}_n[x]$ have supremum norm less than k on $[a,b]$.

The existence of the limit $\Omega[0,1]$ is shown in [25] and the method is easily modified to show the existence of $\Omega[a,b]$ as follows. Given any positive integers n and N, let $p(x)$ be an nth integer Chebyshev polynomial for the interval $[a,b]$ and write $N = nq + r$ with $0 \leq r < n$ using the division algorithm. Then $p(x)^q x^r$ is a polynomial in $\mathcal{Z}_N[x]$ and so

$$\Omega_N[a,b]^N \leq \|p(x)^q x^r\|_{[a,b]} \leq \|p(x)\|^q_{[a,b]} \|x\|^r_{[a,b]} \leq \Omega_n[a,b]^{qn} t^r = \Omega_n[a,b]^{N-r} t^r$$

where $t = \max\left(\|x\|_{[a,b]}, \Omega_n[a,b]\right)$. Since $t/\Omega_n[a,b] \geq 1$, this gives

$$\Omega_N[a,b] \leq \Omega_n[a,b]^{1-r/N} t^{r/N} = \Omega_n[a,b] \left(\frac{t}{\Omega_n[a,b]}\right)^{r/N} \leq \Omega_n[a,b] \left(\frac{t}{\Omega_n[a,b]}\right)^{n/N}.$$

It follows that $\limsup_{N\to\infty} \Omega_N[a,b] \leq \Omega_n[a,b]$. On the other hand, as n can be chosen so that $\Omega_n[a,b]$ is arbitrarily close to the limit inferior of the sequence $\{\Omega_N[a,b]\}$, we must have that $\limsup\ \Omega_N[a,b] \leq \liminf\ \Omega_N[a,b]$ and so $\Omega[a,b] = \lim_{n\to\infty} \Omega_n[a,b]$ exists.

1.2.1 The Gelfond-Schnirelman approach to the Prime Number Theorem

One of the initial reasons for studying integer Chebyshev polynomials was due to their connection to the Prime Number Theorem. Ferguson [17, pg 143] attributes this to I. G. Schnirelman and A. O. Gelfond. If we let $\mathrm{I} = \int_0^1 p^2(x)dx$ where $p(x) \in \mathcal{Z}_n[x] \setminus \{0\}$ and define d_n as $d_n = \mathrm{lcm}\{1, 2, \ldots, n\}$, then the degree of $p^2(x)$ is at most $2n$ so $\mathrm{I}\, d_{2n+1}$ must be a positive integer. Additionally, for any prime q dividing d_{2n+1}, q must be less than $2n+1$ and the power of q dividing d_{2n+1} is $\lfloor \log_q(2n+1) \rfloor \leq \ln(2n+1)/\ln(q)$. This gives us the following inequality:

$$1 \ \leq\ \mathrm{I}\, d_{2n+1} \ \leq\ \mathrm{I} \prod_{\substack{q \leq 2n+1 \\ q \text{ prime}}} q^{\frac{\ln(2n+1)}{\ln q}}.$$

Taking the natural logarithm of the outermost quantities yields

$$0 \ \leq\ \ln(\mathrm{I}) + \sum_{\substack{q \leq 2n+1 \\ q \text{ prime}}} \frac{\ln(2n+1)}{\ln q} \ln q \ =\ \ln(\mathrm{I}) + \ln(2n+1)\pi(2n+1).$$

If we now note that $\mathrm{I} = \int_0^1 p^2(x)dx \leq \left\|p^2(x)\right\|_{[0,1]}$, we see

$$\ln(\mathrm{I}) \leq 2n \ln\left(\left\|\left(p^2(x)\right)\right\|^{1/2n}_{[0,1]}\right)$$

or

$$2n \ln \left(\frac{1}{\|p(x)\|_{[0,1]}^{1/n}} \right) \leq -\ln(I) \leq \ln(2n+1)\pi(2n+1).$$

Letting $p(x)$ be chosen so that $\|p(x)\|_{[0,1]}^{1/n} = \Omega_n[0,1]$ gives

$$\ln \frac{1}{\Omega[0,1]+\epsilon} \;\leq\; \ln \frac{1}{\Omega_n[0,1]} \;\leq\; \frac{\ln(2n+1)\pi(2n+1)}{2n}$$

for any $\epsilon > 0$ provided n is large enough and so

$$\ln \frac{1}{\Omega[0,1]} \;\leq\; \liminf \frac{\ln(n)\pi(n)}{n}.$$

If we could show that $\Omega[0,1] = 1/e$, this would lead to an alternate proof of the lower bound required in the Prime Number Theorem. Unfortunately, this is not the case as will be shown in Section 1.2.3.

1.2.2 The main tools of the single variable case

The following lemmas are the basis for a large proportion of known results in the single variable case. The first is an easy consequence of having two expressions for the resultant of two polynomials [20], and the second follows directly from the work done in Section 1.2.

Lemma 1.2 *Suppose $p_n(x) \in \mathcal{Z}_n[x]$ is of degree n and $q_k(x) = a_k x^k + \cdots + a_0 \in \mathcal{Z}_k[x]$ has k roots in $[a,b]$. If $p_n(x)$ and $q_k(x)$ do not have common factors, then*

$$\left(\|p_n(x)\|_{[a,b]} \right)^{1/n} \geq |a_k|^{-1/k}.$$

Proof: Let β_1, β_2, ..., β_k be the roots of $q_k(x)$. Then the resultant of $p_n(x)$ and $q_k(x)$ is given by the expression $a_k^n p_n(\beta_1)p_n(\beta_2)\cdots p_n(\beta_k)$ which is not zero since $p_n(x)$ and $q_k(x)$ do not have common factors. As the resultant is also given by the determinant of the Sylvester matrix of $p_n(x)$ and $q_k(x)$, which is an integer in this case since both polynomials have integer coefficients,

$$1 \leq |a_k|^n \left| p_n(\beta_1)p_n(\beta_2)\cdots p_n(\beta_k) \right|.$$

Since each $\beta_i \in [a,b]$, $|p_n(\beta_i)| \leq \|p_n(x)\|_{[a,b]}$ and so

$$1 \leq |a_k|^n \left(\|p_n(x)\|_{[a,b]} \right)^k.$$

The result now follows.

□

Lemma 1.3 *For any polynomial $p(x)$ with integer coefficients, $\Omega[a,b] \leq \|p(x)\|_{[a,b]}^{1/n}$ where n is the degree of $p(x)$.*

Proof: Let n be the degree of the polynomial $p(x)$. As in the proof of the existence of $\Omega[a,b]$ given in Section 1.2, $\Omega[a,b] = \limsup_{N\to\infty} \Omega_N[a,b] \leq \Omega_n[a,b] \leq \|p(x)\|_{[a,b]}^{1/n}$.

□

1.2.3 Bounds on $\Omega[a,b]$

From Theorem 1.2, it is clear that any nonconstant polynomial with integer coefficients must have supremum norm at least $2\left(\frac{b-a}{4}\right)^n$. Since $1 \in \mathcal{Z}_n[x]$ for all n, the *Integer Chebyshev Problem* is solvable when $b-a \geq 4$. In this case 1 is an nth integer Chebyshev polynomial for the interval for any n and so $\Omega[a,b] = 1$. For intervals of length less than 4, an early result of Fekete [6] ensures that for any integer n, there exists a polynomial $p_n(x)$ of degree n with integer coefficients for which $\|p_n(x)\|_{[a,b]} < 2(n+1)((b-a)/4)^{n/2}$. This gives the following bounds on $\Omega[a,b]$ when $[a,b]$ is an interval of length less than 4.

$$\frac{b-a}{4} \leq \Omega[a,b] \leq \sqrt{\frac{b-a}{4}} \quad \text{when} \quad b-a < 4.$$

Lower bounds on $\Omega[0,1]$

The case when $[a,b] = [0,1]$ has been extensively studied and the above lower bound of $1/4$ has been improved multiple times. The initial improvement comes from the Prime Number Theorem and Section 1.2.1. Since $\ln(1/\Omega[0,1]) \leq \lim_{n\to\infty} \pi(n)\ln(n)/n = 1$, $\Omega[0,1] \geq 1/e$. The next set of improvements comes from studying irreducible polynomials with integer coefficients having all their roots in the interval $[0,1]$. We begin by looking at the scaled, translated Chebyshev polynomials from Section 1.1. When $b-a = 4$, the polynomials

$$P_n(x) = 2\left(\frac{b-a}{4}\right)^n T_n\left(\frac{2x-a-b}{b-a}\right)$$

have integer coefficients. This follows from the definition of the Chebyshev polynomials $T_n(x)$. Since $b-a=4$, $b+a$ is an even integer and so $P_n(x) = 2T_n\left(\frac{2x-(b+a)}{4}\right) = 2T_n\left(\frac{x-k}{2}\right)$ for some integer k. The definition of $T_0(x)$ and $T_1(x)$ now give $P_0(x) = 2T_0\left(\frac{x-k}{2}\right) = 2$ and $P_1(x) = 2T_1\left(\frac{x-k}{2}\right) = x-k$ which have integer coefficients. As

$$P_{n+1}(x) = 2T_{n+1}\left(\frac{x-k}{2}\right) = 2\left(2\left(\frac{x-k}{2}\right)T_n\left(\frac{x-k}{2}\right) + T_{n-1}\left(\frac{x-k}{2}\right)\right),$$

$P_n(x)$ must have integer coefficients for any nonnegative integer n. The polynomials $P_n(x)$ can be used to generate a sequence of irreducible polynomials with integer coefficients having all roots in $[0,1]$.

On the interval $[0,4]$, the polynomials $P_n(x)$ reduce to $P_n(x) = 2T_n\left((x/2)-1\right)$ which oscillate between -2 and 2 a total of n times on $[0,4]$ and can be generated by the following iteration.

$$\begin{aligned} P_0(x) &= 2 \\ P_1(x) &= x-2 \\ P_{n+1}(x) &= (x-2)P_n(x) - P_{n-1}(x) \end{aligned}$$

From these defining equations, we can see that the $P_n(x)$ are monic polynomials for $n \geq 1$, are of degree n, and have constant term equal to $(-1)^n 2$. If we scale these polynomials by a factor of 2 and then shift them up or down by one unit, the resulting polynomials $2P_n(x) - (-1)^n$ still have n roots in the interval $[0,4]$. Defining $r_n(x)$ by

$$r_n(x) = x^n \left(2P_n\left(\frac{1}{x}\right) - (-1)^n\right)$$

gives a sequence of irreducible polynomials with integer coefficients with the property that each $r_n(x)$ is of degree n and has n positive roots. The fact that the $r_n(x)$ do not factor over $\mathbb{Q}$ follows from Eisenstein's condition for irreducibility [19] since the leading coefficient of $r_n(x)$ is $(-1)^n 3$, the coefficients of x^m are divisible by 2 for $m < n$, and the constant term 2 is not divisible by 4. It follows that the polynomials $r_n(x-1)$ must also be irreducible and so must the polynomials

$$s_n(x) = x^n r_n\left(\frac{1}{x} - 1\right) = (1-x)^n \left(4T_n\left(\frac{3x-2}{2(1-x)}\right) - (-1)^n\right).$$

This last transformation gives a sequence of irreducible polynomials with integer coefficients where each $s_n(x)$ is of degree n and has all of its roots in $[0,1]$ and provides us with a concrete family of polynomials illustrating the following.

Lemma 1.4 *For any nonnegative integer n, there exists an irreducible polynomial of degree n with integer coefficients having all its roots in $[0,1]$.*

The leading coefficients of the $s_n(x)$ can be found by looking at the constant terms of the polynomials

$$x^n s_n(1/x) = (x-1)^n \left(4T_n\left(\frac{3-2x}{2(x-1)}\right) - (-1)^n\right).$$

Evaluating these last polynomials at $x = 0$ and letting a_n denote the leading coefficient of $s_n(x)$ leads to $a_n = (-1)^n 4t_n - 1$ where $t_n = T_n(-3/2)$. Since $t_0 = 1, t_1 = -3/2$, and $t_{n+1} = -(3t_n + t_{n-1})$ for $n \geq 1$, to solve for the t_n we note that

$$\begin{bmatrix} t_{n+1} \\ t_n \end{bmatrix} = \begin{bmatrix} -3 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t_n \\ t_{n-1} \end{bmatrix} = \begin{bmatrix} -3 & -1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} t_1 \\ t_0 \end{bmatrix}.$$

Diagonalizing the 2×2 matrix using its eigenvalues and eigenvectors so that the powers can be easily found, and then looking at the last row of the resulting equation gives

$$t_n = \frac{(-1)^n(3+\sqrt{5})^n}{2^{n+1}} + \frac{(-3+\sqrt{5})^n}{2^{n+1}}$$

and so

$$a_n = 2\left(\frac{3+\sqrt{5}}{2}\right)^n + 2\left(\frac{3-\sqrt{5}}{2}\right)^n - 1.$$

Since $(3-\sqrt{5})/2 < 1$,

$$\lim_{n\to\infty} a_n^{1/n} = \frac{3+\sqrt{5}}{2}.$$

We are now in a position to improve the lower bound on $\Omega[0,1]$. Any polynomial $p(x)$ of degree n in $\mathcal{Z}[x]$ is divisible by at most a finite number of the polynomials $s_k(x)$ and so for all but finitely many k, Lemma 1.2 gives

$$\left(\|p_n(x)\|_{[0,1]}\right)^{1/n} \geq |a_k|^{-1/k}.$$

Taking the limit as k tends to infinity shows that

$$\left(\|p_n(x)\|_{[0,1]}\right)^{1/n} \geq \frac{2}{3+\sqrt{5}}$$

so we must have $\Omega[0,1] \geq 2/(3+\sqrt{5}) > 1/2.618034$. Montgomery [25] and Chudnovksky [9] both conjecture that if $\mathcal{F}_k$ denotes the set of polynomials with integer coefficients of degree k that are irreducible over $\mathbb{Q}$ and have all their roots in $[0,1]$, and if we define s as

$$s = \limsup_{k\to\infty}\left(\sup_{q_k\in\mathcal{F}_k} |a_k|^{-1/k}\right)$$

where a_k denotes the leading coefficient of q_k, then $s = \Omega[0,1]$. Repeating the above argument with the limit superior in place of the limit gives $\Omega[0,1] \geq s$. Currently, the best lower bound for s comes from the Gorshkov-Wirsing polynomials.

Definition 1.4 *The Gorshkov-Wirsing polynomials are defined iteratively as follows.*

$$\begin{aligned} q_0(x) &= 2x-1 \\ q_1(x) &= 5x^2-5x+1 \\ q_{n+1}(x) &= q_n(x)^2 + q_n(x)q_{n-1}(x)^2 - q_{n-1}(x)^4 \end{aligned}$$

The Gorshkov-Wirsing polynomial $q_k(x)$ is an irreducible polynomial of degree 2^k, has all its roots in $[0,1]$, and satisfies the relation

$$q_k(x) = (-1+3x-3x^2)^{2^{k-1}} q_{k-1}(u(x))$$

for $k \geq 1$ where $u(x)$ is the rational function

$$u(x) = \frac{q_0(x)^2 - q_1(x)}{2q_0(x)^2 - q_1(x)} = \frac{x(1-x)}{1-3x(1-x)}.$$

Full details are given in [25] where it is also shown that if a_{2^k} denotes the leading coefficient of $q_k(x)$, then

$$a_{2^k}^{1/2^k} = 2\prod_{j=0}^{k-1}\left(1+\frac{1}{c_j^2}\right)^{1/2^{j+1}}$$

where $c_0 = 2$ and $c_{k+1} = c_k + 1/c_k$. Since each $c_j \geq 2$, the terms $a_{2^k}^{1/2^k}$ form a bounded increasing sequence and so must converge to a limit which we will call s_0. A simple computation gives $2.376841706263926188\ldots$ as the value of s_0. Montgomery attributes the proof that the Gorshkov-Wirsing polynomials are irreducible over $\mathbb{Q}$ to Wirsing. Armed with these results, it is now an easy matter to improve the lower bound on $\Omega[0,1]$. If $p_n(x)$ is an nth integer Chebyshev polynomial, then at most a finite number of the Gorshkov-Wirsing polynomials can divide $p_n(x)$. For any Gorshkov-Wirsing polynomial $q_k(x)$ that has no common roots with $p_n(x)$, Lemma 1.2 gives

$$\frac{1}{a_{2^k}^{1/2^k}} \leq \left(\|p_n\|_{[a,b]}\right)^{1/n}.$$

Since the terms $a_{2^k}^{-1/2^k}$ are all greater than $1/s_0$, this gives

$$\Omega[0,1] = \lim_{n\to\infty}\left(\|p_n\|_{[a,b]}\right)^{1/n} \geq \frac{1}{s_0} = \frac{1}{2.376841706263926188\ldots}.$$

If the value of $s = 1/s_0$, then the conjecture of Montgomery and Chudnovsky is incorrect since Borwein and Erdelyi [8] show that $\Omega[0,1] > (1/s_0) + \epsilon$ for some $\epsilon > 0$. This is done

by first showing that an nth integer Chebyshev polynomial must be divisble by x^k where $k/n > 0.26$ if n is large enough (see Section 1.2.5) and then constructing a sequence of polynomials $\{R_k(x)\}$ where each $R_k(x)$ is of degree 2^k, has all its roots in $[0,1]$, and has leading coefficient a_{2^k} satisfying $\lim_{k\to\infty} a_{2^k}^{1/2^k} = s_0$. Claiming that each $R_k(x)$ has a positive proportion of its roots in $[0,\delta]$ for any $\delta \in (0,1)$ provided k is large enough, they then use a result of Saff and Varga [28] which implies that if $p_n(x) = x^{k(n)} p_{n-k(n)}(x)$ and $\theta < k(n)/n$ for sufficiently large n, then

$$\limsup_{n\to\infty} \left(\left\| \frac{p_n(x)}{\|p_n(x)\|_{[0,1]}} \right\|_{[0,\theta^2]} \right)^{1/n} < 1.$$

Using $\theta^2 = 1/16 < 0.26^2$ and letting $p_n(x)$ be an nth integer Chebyshev polynomial where n is sufficiently large, $c \in (0,1)$ is chosen so that $\left(\|p_n(x)\|_{[0,\theta^2]}\right)^{1/n} < c\left(\|p_n(x)\|_{[0,1]}\right)^{1/n}$. The proof of Lemma 1.2 is modified for the polynomials $p_n(x)$ and $R_k(x)$ by replacing the the terms $|p_n(\beta_i)|$ with $c^n \|p_n(x)\|_{[0,1]}$ for all roots β_i in $[0,\theta^2]$. Letting k tend to infinity then gives $1/s_0 \le c^\eta \|p(x)\|_{[0,1]}^{1/n}$ where η is a lower bound on the proportion of roots of $R_k(x)$ in $[0,\theta^2]$ and the result follows by letting n tend to infinity. The lower bound is further improved to $1/2.3736 \le \Omega[0,1]$ by Pritsker [27] using methods of weighted potential theory.

Upper bounds on $\Omega[0,1]$

Lemma 1.3 shows that for any polynomial $p(x) \in \mathcal{Z}[x]$, $\Omega[0,1] \le \left(\|p(x)\|_{[0,1]}\right)^{1/n}$. Using this inequality, we can obtain bounds on $\Omega[0,1]$ by computational methods. The following example, intially found by Borwein and Erdelyi [8], illustrates this point. Let

$$\begin{aligned}
c_0(x) &= x \\
c_1(x) &= 1 - x \\
c_2(x) &= 2x - 1 \\
c_3(x) &= 5x^2 - 5x + 1 \\
c_4(x) &= 13x^3 - 19x^2 + 8x - 1 \\
c_5(x) &= 13x^3 - 20x^2 + 9x - 1 \\
c_6(x) &= 29x^4 - 58x^3 + 40x^2 - 11x + 1 \\
c_7(x) &= 31x^4 - 61x^3 + 41x^2 - 11x + 1
\end{aligned}$$

$$\begin{aligned} c_8(x) &= 31x^4 - 63x^3 + 44x^2 - 12x + 1 \\ c_9(x) &= 941x^8 - 3764x^7 + 6349x^6 - 5873x^5 + 3243x^4 - 1089x^3 + 216x^2 - 23x + 1 \end{aligned}$$

and let

$$p_{210}(x) = c_0(x)^{67} \cdot c_1(x)^{67} \cdot c_2(x)^{24} \cdot c_3(x)^9 \cdot c_4(x) \cdot c_5(x) \cdot c_6(x)^3 \cdot c_7(x) \cdot c_8(x) \cdot c_9(x).$$

It is a simple calculus exercise to find the maximum value obtained by $|p_{210}(x)|$ on the interval $[0,1]$ and is best done with the aid of a computer algebra package such as *Maple*. Doing so reveals that $\|p_{210}\|_{[0,1]} = 0.8074175067967268\ldots \times 10^{-78}$ and so

$$\Omega[0,1] \leq \left(\|p_{210}(x)\|_{[0,1]}\right)^{1/210} = \frac{1}{2.3543496486\ldots}.$$

It will be shown in Section 1.3.3 that $p_{210}(x)$ is in fact an integer Chebyshev polynomial of degree 210 on the interval $[0,1]$.

The upper bound given above can be improved upon by means of the simplex algorithm. Given a set of polynomials $\{p_i\}$, we can consider polynomials of the form $\prod p_i^{\alpha_i}$. Since the goal is to choose integer values for the α_i that minimize the supremum norm of this product on the interval $[0,1]$, we first take the logarithm of the absolute value of the above product and evaluate it at multiple points $x_j \in [0,1]$ to give inequalities of the form

$$\sum \frac{\alpha_i}{2} \log\left(p_i(x_j)^2\right) < c.$$

The simplex algorithm is then used to minimize c under the above system of constraints together with $\sum \alpha_i = 1$ and $\alpha_i \geq 0$ for each α_i. The resulting α_i are then multiplied by a large power of 10 and rounded to give a new polynomial $\prod p_i^{\beta_i}$ of very large degree N. The Nth root of the supremum norm of this new polynomial then gives an upper bound on $\Omega[0,1]$. This is a reasonable way to search for a polynomial of large degree with small supremum norm since Markov's inequality on the rth derivative of a polynomial $f(x)$, which is stated in Section 1.2, gives

$$\left\|f^{(r)}(x)\right\|_{[a,b]} \leq (2n^2)^r \left\|f(x)\right\|_{[a,b]}$$

or

$$\left(\left\|f^{(r)}(x)\right\|_{[a,b]}\right)^{1/n} \leq \left(\sqrt{2}n\right)^{2r/n} \left(\|f(x)\|_{[a,b]}\right)^{1/n}.$$

If $q_k(x)$ is a polynomial as in Lemma 1.2, $(\|f(x)\|_{[0,1]})^{1/n} < |a_k|^{-1/k}$, and n is sufficiently large in relation to r, then $(\|f^{(r)}(x)\|_{[0,1]})^{1/n}$ will also be smaller than $|a_k|^{-1/k}$. If this is the case, then $q_k(x)$ must also divide the first r derivatives of $f(x)$, or $q_k(x)^{r+1}$ divides $f(x)$.

This method was used by Borwein and Erdelyi [8] to show $\Omega[0,1] \leq 1/2.3605$, by Habsieger and Salvy [21] to show $\Omega[0,1] \leq 1/2.3613$, by Wu [33] to show $\Omega[0,1] \leq 1/2.3631$, and will be used again in Section 1.4.3 to show $\Omega[0,1] \leq 1/2.36482727$. The difficult part is in finding a good set of polynomials $\{p_i\}$.

1.2.4 Symmetry conditions and the intervals $[-1,1]$, $[0,1]$, and $[0,1/4]$

By using polynomials that satisfy certain symmetry properties, it can be shown that

$$\Omega[-1,1]^2 = \Omega[0,1] = \Omega[0,1/4]^{1/2}.$$

The easier case of $\Omega[-1,1]^2 = \Omega[0,1]$ is handled first.

Given an integer Chebyshev polynomial $p_n(x)$ of degree n for the interval $[0,1]$, the polynomial $p_n(x^2)$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[-1,1]$. To see this, note that $\|p_n(x^2)\|_{[-1,1]} = \|p_n(x)\|_{[0,1]}$ and so $\|q_{2n}(x)\|_{[-1,1]}^{1/n} \leq \|p_n(x)\|_{[0,1]}^{1/n}$ for at least one polynomial $q_{2n}(x)$ of degree at most $2n$. This gives

$$\Omega_{2n}[-1,1]^2 \leq \Omega_n[0,1].$$

On the other hand, suppose $q_{2n}(x)$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[-1,1]$. Considering the even part $e(x)$ of $q_{2n}(x)$, which is a nonzero polynomial of degree $2n$ with integer coefficients, gives

$$\|e(x)\|_{[-1,1]} = \left\| \frac{q_{2n}(x) + q_{2n}(-x)}{2} \right\|_{[-1,1]} \leq \frac{1}{2}\|q_{2n}(x)\|_{[-1,1]} + \frac{1}{2}\|q_{2n}(x)\|_{[-1,1]} = \|q_{2n}(x)\|_{[-1,1]}.$$

Since $q_{2n}(x)$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[-1,1]$, the above inequality must in fact be an equality and so $e(x)$ is also an integer Chebyshev polynomial of degree $2n$ for the interval $[-1,1]$. Since $e(x)$ has only even powers of x, on the interval $[0,1]$ the function $e(\sqrt{x})$ is a polynomial of degree n with integer coefficients so

$$\Omega_n[0,1] \leq \|e(\sqrt{x})\|_{[0,1]}^{1/n} = \|e(x)\|_{[-1,1]}^{1/n} = \Omega_{2n}[-1,1]^2.$$

Since $\Omega_n[0,1] = \Omega_{2n}[-1,1]^2$, this justifies the statement that the above polynomial $p_n(x^2)$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[-1,1]$. Furthermore, if we take the limit as n tends to infinity, this gives $\Omega[0,1] = \Omega[-1,1]^2$.

To show $\Omega[0,1]^2 = \Omega[0,1/4]$, we again use symmetric polynomials, but now we consider polynomials with symmetry around $x = 1/2$. If $p_n(x)$ is an integer Chebyshev polynomial of degree n for the interval $[0,1/4]$, then $p_n(x(1-x))$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[0,1]$. To see this we note that $\|p_n(x)\|_{[0,1/4]} = \|p_n(x(1-x))\|_{[0,1]}$ and so

$$\Omega_n[0,1/4] \geq \Omega_{2n}[0,1]^2.$$

To reverse the inequality, we need the following result which can be found in [21].

Lemma 1.5 *Let $E_k = \{p(x) \in \mathcal{Z}_k[x] : p(1-x) = (-1)^k p(x)\}$. For any nonnegative integer k, we have*

$$E_{2k} = \mathcal{Z}_k[x(1-x)] \quad \text{and} \quad E_{2k+1} = (1-2x)\mathcal{Z}_k[x(1-x)].$$

Proof: Showing that $E_{2k} = \mathcal{Z}_k[x(1-x)]$ is easily accomplished using induction. The case when $k = 0$ holds since $E_0 = \mathcal{Z}_0[x(1-x)] = \mathbb{Z}$. Now suppose the statement holds for some nonnegative integer $k-1$ and $p(x)$ is a polynomial in the set E_{2k}. If we consider the polynomial $q(x) = p(x) - p(0)$, then $q(0) = 0$ and since $p(x) \in E_{2k}$, $q(1)$ also equals 0 so $q(x)$ is divisible by $x(1-x)$. Since $p(x) \in E_{2k}$, $q(x)/(x(1-x)) = q(1-x)/((1-x)x)$ and so by the inductive hypothesis, $q(x) = x(1-x)r(x(1-x))$ for some polynomial $r(x) \in \mathcal{Z}_{k-1}[x]$. It now follows that $p(x) = x(1-x)r(x(1-x)) + p(0) \in \mathcal{Z}_k[x(1-x)]$.

To show that $E_{2k+1} = (1-2x)\mathcal{Z}_k[x(1-x)]$, suppose $p(x) \in E_{2k+1}$. Then $1-2x$ divides $p(x)$ since $p(1/2) = -p(1/2) = 0$. Applying the earlier result to the function $p(x)/(1-2x) \in E_{2k}$ gives $p(x) = (1-2x)r(x(1-x))$ for some function $r(x) \in \mathcal{Z}_k[x]$.

□

We are now in a position to reverse the last inequality above. Suppose $q(x)$ is an integer Chebyshev polynomial of degree m for the interval $[0,1]$ and define the polynomials $Q_1(x)$ and $Q_2(x)$ as follows.

$$\begin{aligned} Q_1(x) &= xq(x) + (-1)^m(1-x)q(1-x) \\ Q_2(x) &= (1-x)q(x) + (-1)^m xq(1-x) \end{aligned}$$

Then $Q_1(x)$ and $Q_2(x)$ are both polynomials with integer coefficients of degree at most m and satisfy the property that $Q_i(x) = (-1)^m Q_i(1-x)$. Furthermore, at least one of the $Q_i(x)$ must be nonzero because

$$xQ_1(x) - (1-x)Q_2(x) = (2x-1)q(x) \neq 0.$$

Let $Q_0(x)$ be a nonzero $Q_i(x)$. Then $Q_0(x)$ must be an integer Chebyshev polynomial of degree m for the interval $[0,1]$ since $q(x)$ is a polynomial with minimal supremum norm on $[0,1]$ and for any $x \in [0,1]$,

$$|Q_0(x)| \leq x \left\|q(x)\right\|_{[0,1]} + (1-x)\left\|q(x)\right\|_{[0,1]} = \left\|q(x)\right\|_{[0,1]}.$$

If $m = 2n+1$ is odd, then $Q_0(x) \in E_{2n+1}$ and so $Q_0(x) = (2x-1)p(x(1-x))$ for some $p(x) \in \mathcal{Z}_n[x]$. If $m = 2n$ is even, then $Q_0(x) \in E_{2n}$ so $Q_0(x) = p(x(1-x))$ for some polynomial $p(x) \in \mathcal{Z}_n[x]$. In this last case, since $\left\|Q_0(x)\right\|_{[0,1]} = \left\|p(x)\right\|_{[0,1/4]}$,

$$\Omega_{2n}[0,1]^2 \geq \Omega_n[0,1/4].$$

Since $\Omega_n[0,1/4] = \Omega_{2n}[0,1]^2$, this justifies the statement that if $p_n(x)$ is an integer Chebyshev polynomial of degree n for the interval $[0,1/4]$, then $p_n(x(1-x))$ is an integer Chebyshev polynomial of degree $2n$ for the interval $[0,1]$. Again, if we let n tend to infinity, we see that $\Omega[0,1] = \Omega[-1,1]^2$.

The fact that there must exist a symmetric nth integer Chebyshev polynomial on $[0,1]$ for any n, and that this symmetric polynomial must be of the form $p(x(1-x))$ when n is even and of the form $(1-2x)p(x(1-x))$ when n is odd will prove to be very useful in Section 1.3 where the problems of finding integer Chebyshev polynomials and the value of $\Omega_n[0,1]$ are considered.

1.2.5 Critical polynomials and statements of structure

From Lemma 1.2, if $c_i(x) = a_k x^k + \cdots + a_0 \in \mathcal{Z}_k[x]$ has all its roots in $[0,1]$ and if $\Omega[0,1] < |a_k|^{-1/k}$, then $c_i(x)$ must divide an nth integer Chebyshev polynomial provided n is sufficiently large. Irreducible polynomials of this form are called *critical polynomials* for the interval $[0,1]$. As indicated earlier, Markov's inequality can be used to show that critical polynomials for the interval $[0,1]$ must be factors of an nth integer Chebyshev polynomial with relatively high multiplicity. By applying Stirling's formula, or even just weaker bounds on the size of $n!$, we can give lower bounds on the multiplicity of these factors. We begin by noting that $(2r-1)!!$, the product of the first r odd integers, can be written as $\frac{(2r)!}{2^r r!}$ and that $n^2(n^2-1^2)(n^2-2^2)\cdots(n^2-(r-1)^2) = \frac{n}{n+r}\frac{(n+r)!}{(n-r)!}$ where r is an integer between 1 and $n-1$. To justify this last statement note that for any integer $n \geq 2$, if r=1 then both sides simplify to n^2. If the statement holds for some integer $n \geq 2$ and an integer r between 1

and $n-2$, then it also hold for n and $r+1$ since

$$\begin{aligned}\frac{n}{n+(r+1)}\frac{(n+(r+1))!}{(n-(r+1))!} &= \frac{n+r}{n+(r+1)}(n+(r+1))(n-r)\left(\frac{n}{n+r}\frac{(n+r)!}{(n-r)!}\right)\\ &= \left(n^2-r^2\right)\left(n^2(n^2-1^2)(n^2-2^2)\cdots(n^2-(r-1)^2)\right).\end{aligned}$$

Markov's inequality applied to the function $p_n^{(r)}(x)/r!$ on the interval $[0,1]$ now gives

$$\begin{aligned}\left\|\frac{p_n^{(r)}(x)}{r!}\right\|_{[0,1]} &\le 2^r\frac{n^2(n^2-1^2)(n^2-2^2)\cdots(n^2-(r-1)^2)}{(2r-1)!!}\left\|\frac{p_n(x)}{r!}\right\|_{[0,1]}\\ &= 4^r\frac{n}{n+r}\frac{(n+r)!}{(n-r)!(2r)!}\left\|p_n(x)\right\|_{[0,1]}.\end{aligned}$$

Using the fact [12] that

$$2n^{n+\frac{1}{2}}e^{-n} < \left(\frac{2e}{3}\right)^{3/2}n^{n+\frac{1}{2}}e^{-n} < n! < en^{n+\frac{1}{2}}e^{-n} < 3n^{n+\frac{1}{2}}e^{-n}$$

allows us to say

$$\begin{aligned}\left\|\frac{p_n^{(r)}(x)}{r!}\right\|_{[0,1]} &\le 4^r\frac{n}{n+r}\frac{3(n+r)^{n+r}e^{-(n+r)}}{2(n-r)^{n-r}e^{-(n-r)}2(2r)^{2r}e^{-2r}}\sqrt{\frac{n+r}{(n-r)2r}}\left\|p_n(x)\right\|_{[0,1]}\\ &= \sqrt{\frac{9n^2}{32r(n^2-r^2)}}\frac{(n+r)^{n+r}}{(n-r)^{n-r}r^{2r}}\left\|p_n(x)\right\|_{[0,1]}\\ &< \frac{(n+r)^{n+r}}{(n-r)^{n-r}r^{2r}}\left\|p_n(x)\right\|_{[0,1]}\quad\text{for}\ \ 1\le r\le n-1.\end{aligned}$$

If $p_n(x)$ is a polynomial with integer coefficients of degree n, then $p_n^{(r)}(x)/r!$ is a polynomial of degree $n-r$ with integer coefficients and so to apply Lemma 1.2 with the above inequality, we need to take the $(n-r)$th root of $p_n^{(r)}(x)/r!$. Doing so and making the substitution $r=\lambda n$ leads to

$$\begin{aligned}\left(\left\|\frac{p_n^{(r)}(x)}{r!}\right\|_{[0,1]}\right)^{\frac{1}{n-r}} &< \left(\frac{(n+r)^{n+r}}{(n-r)^{n-r}r^{2r}}\left\|p_n(x)\right\|_{[0,1]}\right)^{\frac{1}{n}\frac{n}{n-r}}\\ &= \left(\frac{(1+\lambda)^{1+\lambda}}{(1-\lambda)^{1-\lambda}\lambda^{2\lambda}}\left(\left\|p_n(x)\right\|_{[0,1]}\right)^{1/n}\right)^{\frac{1}{1-\lambda}}\end{aligned}$$

where $1/n\le\lambda\le(n-1)/n$.

For sufficiently large n, if $p_n(x)$ is an integer Chebyshev polynomial on the interval $[0,1]$, then $p_n(x)$ must be of degree n and $\left(\left\|p_n(x)\right\|_{[0,1]}\right)^{1/n} \le 1/2.36482727$ so given a

critical polynomial $c_i(x) = a_k x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0$ of degree k, if we can find a λ_i so that

$$\frac{(1+\lambda)^{1+\lambda}}{(1-\lambda)^{1-\lambda}\lambda^{2\lambda}} \frac{1}{2.36482727} < \left(\frac{1}{|a_k|}\right)^{\frac{1-\lambda}{k}}$$

for all positive $\lambda \leq \lambda_i$, then Lemma 1.2 will allow us to say $c_i(x)$ divides the derivatives $p_n^{(r)}(x)$ for all positive integers r less than or equal to $\lfloor \lambda_i n \rfloor$. Since c_i is a critical polynomial for the interval $[0,1]$, it must also divide $p_n(x)$ and so $c_i(x)$ divides $p_n^{(r)}(x)$ for all integers r between 0 and $\lfloor \lambda_i n \rfloor$. If $p_n(x)$ is an integer Chebyshev polynomial for the interval $[0,1]$, and if n is sufficiently large, then $(c_i(x))^{\lfloor \lambda_i n \rfloor + 1}$ divides $p_n(x)$. Applying this result to the critical polynomials listed in Section 1.2.3 gives the following result.

Theorem 1.3 *Let $p_n(x)$ be a polynomial of degree n with integer coefficients. If $p_n(x)$ is an integer Chebyshev polynomial for the interval $[0,1]$ of sufficiently large degree, then $p_n(x)$ satisfies the inequality $\left(\|p_n(x)\|_{[0,1]}\right)^{1/n} \leq 1/2.36482727$ and*

$$\begin{aligned} p_n(x) &= (x(1-x))^{\lambda_1 n} (2x-1)^{\lambda_2 n} (5x^2-5x+1)^{\lambda_3 n} \\ &\cdot (29x^4 - 58x^3 + 40x^2 - 11x + 1)^{\lambda_4 n} \\ &\cdot \left((13x^3 - 19x^2 + 8x - 1)(13x^3 - 20x^2 + 9x - 1)\right)^{\lambda_5 n} \\ &\cdot \left((31x^4 - 61x^3 + 41x^2 - 11x + 1)(31x^4 - 63x^3 + 44x^2 - 12x + 1)\right)^{\lambda_6 n} \\ &\cdot (941x^8 - 3764x^7 + 6349x^6 - 5873x^5 + 3243x^4 - 1089x^3 + 216x^2 - 23x + 1)^{\lambda_7 n} \\ &\cdot q(x) \end{aligned}$$

where $\lambda_1 > 0.148071$, $\lambda_2 > 0.017919$, $\lambda_3 > 0.004698$, $\lambda_4 > 0.001307$, $\lambda_5 > 0.000333$, $\lambda_6 > 0.000114$, $\lambda_7 > 0.000275$, and $q(x)$ is a polynomial with integer coefficients.

This is essentially the method used by Aparicio in [5] where it is shown that if $p_n(x)$ is an nth integer Chebyshev polynomial for the interval $[0,1]$ and n is sufficiently large, then

$$p_n(x) = (x(1-x))^{\lambda_1 n} (2x-1)^{\lambda_2 n} (5x^2-5x+1)^{\lambda_3 n} q(x)$$

where $\lambda_1 \geq 0.1456$, $\lambda_2 \geq 0.0166$, and $\lambda_3 \geq 0.0037$.

These bounds can be improved when $p(x)$ is a symmetric integer Chebyshev polynomial of even degree for the interval $[0,1]$. In this case, $p(x) = r(x(1-x))$ for some integer Chebyshev polynomial $r(x)$ of degree $n/2$ for the interval $[0,1/4]$ and $\Omega[0,1/4] = \Omega[0,1]^2 \leq (1/2.36482727)^2$. Since the above argument extends without change to the interval $[0,1/4]$, we have the following result.

Theorem 1.4 *Let $r_n(x)$ be a polynomial of degree n with integer coefficients. If $r_n(x)$ is an integer Chebyshev polynomial for the interval $[0, 1/4]$ of sufficiently large degree, then $r_n(x)$ satisfies the inequality $\left(\|r_n(x)\|_{[0,1/4]}\right)^{1/n} \leq 1/5.592408$ and*

$$\begin{aligned} r_n(x) &= x^{\mu_1 n}(4x-1)^{\mu_2 n}(5x-1)^{\mu_3 n} \\ &\quad \cdot(29x^2-11x+1)^{\mu_4 n}(169x^3-94x^2+17x-1)^{\mu_5 n} \\ &\quad \cdot(961x^4-712x^3+194x^2-23x+1)^{\mu_6 n} \\ &\quad \cdot(941x^4-703x^3+193x^2-23x+1)^{\mu_7 n} q(x) \end{aligned}$$

where $\mu_1 > 0.583649$, $\mu_2 > 0.051069$, $\mu_3 > 0.012157$, $\mu_4 > 0.003198$, $\mu_5 > 0.000784$, $\mu_6 > 0.000263$, $\mu_7 > 0.000645$, and $q(x)$ is a polynomial with integer coefficients.

When $p_n(x)$ is a symmetric integer Chebyshev polynomial of even degree for the interval $[0, 1]$, this last result gives $\lambda_1 > 0.291824$, $\lambda_2 > 0.051069$, $\lambda_3 > 0.006078$, $\lambda_4 > 0.001599$, $\lambda_5 > 0.000392$, $\lambda_6 > 0.000131$, $\lambda_7 > 0.000322$.

The bounds on the λ_i in the nonsymmetric case can be improved by starting with a specific sequence of Muntz-Legendre polynomials as opposed to starting with Markov's inequality. In most cases, this will either match or improve the bounds found above in the symmetric case. This was done first by Borwein and Erdelyi [8] to improve the bound on λ_1. Applying the Gram-Schmidt orthogonalization process to the polynomials $x^n, x^{n-1}, x^{n-2}, \ldots, x^k$ with $k \leq n$ and the inner product $\langle f(x), g(x)\rangle = \int_0^1 f(x)g(x)dx$ leads to the basis

$$L_i(x) = \sum_{j=i}^{n} c_{i,j}x^j = \sum_{j=i}^{n}(-1)^{n-j}\binom{n+1+j}{n-i}\binom{n-i}{n-j}x^j \quad \text{where } k \leq i \leq n \text{ [33].}$$

Since the $L_i(x)$ have norm equal to $1/\sqrt{2i+1}$ and form an orthogonal basis for the space of functions of the form $\sum_{i=k}^{n} a_i x^i$, if $x^k q_{n-k}(x) = \sum_{i=k}^{n} \lambda_i L_i(x)$ where $q_{n-k}(x) \in \mathcal{Z}_{n-k}[x]$, then $\lambda_k = q_{n-k}(0)/c_{k,k}$ and we must have

$$\frac{|q_{n-k}(0)|}{|c_{k,k}|\sqrt{2k+1}} = \sqrt{\left\langle \frac{q_{n-k}(0)}{c_{k,k}}L_k(x), \frac{q_{n-k}(0)}{c_{k,k}}L_k(x)\right\rangle} \leq \sqrt{\langle x^k q_{n-k}(x), x^k q_{n-k}(x)\rangle}\ .$$

As the 2-norm is less than or equal to the supremum norm, if $q_{n-k}(0) \neq 0$, we must therefore have

$$1 \leq |q_{n-k}(0)| \leq \sqrt{2k+1}\,|c_{k,k}|\left\|x^k q_{n-k}(x)\right\|_{[0,1]}\ .$$

Given that

$$|c_{k,k}| = \binom{n+1+k}{n-k}\binom{n-k}{n-k} = \frac{n+k+1}{2k+1}\binom{n+k}{n-k}$$

and

$$((n-k)+2k)^{n+k} \geq \binom{n+k}{n-k}(n-k)^{n-k}(2k)^{2k},$$

this leads to

$$|q_{n-k}(0)| \leq \frac{n+k+1}{\sqrt{2k+1}} \frac{(n+k)^{n+k}}{(n-k)^{n-k}(2k)^{2k}} \left\| x^k q_{n-k}(x) \right\|_{[0,1]}.$$

Upon taking the nth root and making the substitution $k = \lambda n$ we obtain the inequality

$$1 \leq \left(\frac{n+k+1}{\sqrt{2k+1}}\right)^{1/n} \frac{(1+\lambda)^{1+\lambda}}{(1-\lambda)^{1-\lambda}(2\lambda)^{2\lambda}} \left(\left\| x^k q_{n-k}(x) \right\|_{[0,1]} \right)^{1/n}.$$

By considering the Nth power of $x^k q_{n-k}(x)$, repeating the above argument, and letting N tend to infinity [18] we see that we can drop the $((n+k+1)/\sqrt{2k+1})^{1/n}$ term since we will still have $\lambda = (Nk/Nn) = k/n$, $\left(\left\| (x^k q_{n-k}(x))^N \right\|_{[0,1]} \right)^{1/nN} = \left(\left\| x^k q_{n-k}(x) \right\|_{[0,1]} \right)^{1/n}$, and $\lim_{N\to\infty}((nN+kN+1)/\sqrt{2kN+1})^{1/nN} = 1$. If $q_{n-k}(x) \neq 0$, then

$$\left(\left\| x^k q_{n-k}(x) \right\|_{[0,1]} \right)^{-1/n} \leq \frac{(1+\lambda)^{1+\lambda}}{(1-\lambda)^{1-\lambda}(2\lambda)^{2\lambda}}.$$

Borwein and Erdelyi use this to show that if n is sufficiently large and

$$p_n(x) = (x(1-x))^{\lambda_1 n}(2x-1)^{\lambda_2 n}(5x^2-5x+1)^{\lambda_3 n} q(x)$$

is an nth integer Chebyshev polynomial, then $2.36 \leq \left(\|p_n(x)\|_{[0,1]} \right)^{-1/n}$ and so $\lambda_1 \geq 0.26$. By starting with a slightly improved bound of $\Omega[0,1] \leq 1/2.3611$ and modifying the above procedure to work with polynomials other than x, Flamming, Rhin, and Smyth [18] show $\lambda_1 \geq 0.26415$, $\lambda_2 \geq 0.02196$, $\lambda_3 \geq 0.00528$, $\lambda_4 \geq 0.00106$, $\lambda_5 \geq 0.00023$, $\lambda_6 \geq 0.00002$, and $\lambda_7 \geq 0.00013$. Previously, these were the best known bounds on λ_i for $4 \leq i \leq 7$.

A better way to generalize the work of Borwein and Erdelyi is to use what Wu [33] calls generalized Muntz-Legendre polynomials which are constructed by using the inner product $\langle f(x), g(x) \rangle = \int_0^1 f(x)g(x)dx$ and applying the Gram-Schmidt orthogonalization process to the ordered polynomials $x^n F(x), x^{n-1}F(x), \ldots, x^k F(x)$. Wu gives a detailed proof that if $F(x) = (1-x)^q$, then an orthogonal basis for the space of functions spanned by $x^n(1-x)^q, x^{n-1}(1-x)^q, \ldots, x^k(1-x)^q$ is given by

$$W_i(x) = \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} x^j (1-x)^q \qquad \text{for } k \leq i \leq n$$

and the norm of each $W_i(x)$ is

$$\langle W_i(x), W_i(x)\rangle^{1/2} = \sqrt{\binom{n+i+2q+1}{n-i+2q}\left((2i+1)\binom{n+i+1}{n-i}\right)^{-1}}.$$

The case $q = 0$ gives the Muntz-Legendre polynomials $L_i(x)$ of Borwein and Erdelyi. Using the bound $\Omega[0,1] \leq 1/2.363149$, Wu applied the polynomials $W_i(x)$ to show $\lambda_1 \geq 0.2907$ and that if $p(x)$ is of even degree and symmetric on the interval $[0,1]$, then $\lambda_1 \geq 0.2976$ and $\lambda_2 \geq 0.09662$. The method, and its extension to the other known critical polynomials follows below.

1.2.6 An extension of Wu's method to all known critical polynomials on $[0,1]$

Let $c(x) = c_d x^d + c_{d-1}x^{d-1} + \ldots + c_1 x + c_0$ be a critical polynomial other than x for the interval $[0,1]$, let α be a root of $c(x)$, and set $F(x) = (1 - \frac{x}{\alpha})^q$ in the definition of Wu's generalized Muntz-Legendre polynomials with inner product $\langle f(x), g(x)\rangle = \int_0^\alpha f(x)g(x)dx$. Then an orthogonal basis for the space of functions generated by $x^i(1-\frac{x}{\alpha})^q$ where $k \leq i \leq n$ is given by

$$M_{\alpha,i}(x) = W_i(x/\alpha) = \sum_{j=i}^{n}(-1)^{n-j}\binom{n+2q+j+1}{n-i+2q}\binom{n-i}{n-j}\frac{x^j}{\alpha^j}\left(1-\frac{x}{\alpha}\right)^q \quad \text{for } k \leq i \leq n.$$

The orthogonality of the $W_i(x)$ can be used to show the $M_{\alpha,i}(x)$ are orthogonal to each other as

$$\begin{aligned}
\langle M_{\alpha,i}(x), M_{\alpha,j}(x)\rangle &= \int_0^\alpha M_{\alpha,i}(x)M_{\alpha,j}(x)dx \\
&= \int_0^\alpha W_i(x/\alpha)W_j(x/\alpha)dx \\
&= \alpha\int_0^1 W_i(y)W_j(y)dy \\
&= \begin{cases} \alpha\binom{n+i+2q+1}{n-i+2q}\left((2i+1)\binom{n+i+1}{n-i}\right)^{-1} & i = j \\ 0 & i \neq j. \end{cases}
\end{aligned}$$

If $Q(x) = x^k c(x)^q p(x) = a_k x^k\left(1-\frac{x}{\alpha}\right)^q + \ldots + a_n x^n\left(1-\frac{x}{\alpha}\right)^q = \sum_{i=k}^n \mu_i M_{\alpha,i}(x) \in \mathcal{Z}_{n+q}[x]$ and x does not divide $p(x)$, then a_k is a nonzero integer because the constant term of $p(x)$

is a nonzero integer and the constant term of $c(x)/(1-x/\alpha)$ is c_0. This means

$$|\mu_k| = \left| \frac{a_k \alpha^k}{\binom{n+2q+k+1}{n+2q-k}} \right| \geq \frac{\alpha^k}{\binom{n+2q+k+1}{n+2q-k}}.$$

Since $\int_0^\alpha Q(x)^2 dx \leq \alpha \|Q\|^2_{[0,\alpha]} \leq \alpha \|Q\|^2_{[0,1]}$,

$$\begin{aligned} \sqrt{\alpha}\,\|Q\|_{[0,1]} \geq \langle Q(x), Q(x)\rangle^{1/2} &\geq \langle \mu_k M_{\alpha,k}(x), \mu_k M_{\alpha,k}(x)\rangle^{1/2} \\ &= \sqrt{\mu_k^2 \alpha \binom{n+k+2q+1}{n-k+2q}\left((2k+1)\binom{n+k+1}{n-k}\right)^{-1}}. \end{aligned}$$

Substituting the above bound for $|\mu_k|$ and rearranging yields

$$1 \leq \frac{\sqrt{(2k+1)\binom{n+k+2q+1}{n-k+2q}\binom{n+k+1}{n-k}}}{\alpha^k} \|Q(x)\|_{[0,1]}.$$

At this point, we have an inequality that no longer depends on the polynomials $M_{\alpha,i}(x)$. Since we wish to look at the nth root of polynomials of degree n, it is no longer advantageous to have the degree of $Q(x)$ equal to $n+q$. Instead, we can view $Q(x)$ as a polynomial of degree n and substitute $n-q$ for n in the above inequality. This leads to the following statement.

If $Q(x) = x^k c(x)^q p(x) \in \mathcal{Z}_n[x]$ where $c(x)$ is a critical polynomial for the interval $[0,1]$, and if x does not divide $p(x)$, then

$$\begin{aligned} 1 &\leq \frac{\sqrt{(2k+1)\binom{n+k+q+1}{n-k+q}\binom{n-q+k+1}{n-q-k}}}{\alpha^k} \|Q(x)\|_{[0,1]} \\ &= \sqrt{\frac{(n+k+q+1)(n-q+k+1)}{2k+1}\binom{n+k+q}{n-k+q}\binom{n-q+k}{n-q-k}} \frac{\|Q(x)\|_{[0,1]}}{\alpha^k}. \end{aligned}$$

Using the facts that

$$((n-k+q)+2k)^{n+k+q} \geq \binom{n+k+q}{n-k+q}(n-k+q)^{n-k+q}(2k)^{2k}$$

and

$$((n-k-q)+2k)^{n+k-q} \geq \binom{n+k-q}{n-k-q}(n-k-q)^{n-k-q}(2k)^{2k}$$

gives the following.

$$1 \leq \sqrt{\frac{(n+k+q+1)(n-q+k+1)}{2k+1}} \sqrt{\frac{(n+k+q)^{n+k+q}(n+k-q)^{n+k-q}}{(n-k+q)^{n-k+q}(n-k-q)^{n-k-q}(2k)^{4k}}} \frac{\|Q(x)\|_{[0,1]}}{\alpha^k}$$

Taking nth roots and making the substitutions $\beta = k/n$ and $\gamma = q/n$ leads to

$$1 \leq \left(\frac{(n+k+q+1)(n-q+k+1)}{2k+1}\right)^{1/2n} \sqrt{\frac{(1+\beta+\gamma)^{1+\beta+\gamma}(1+\beta-\gamma)^{1+\beta-\gamma}}{(1-\beta+\gamma)^{1-\beta+\gamma}(1-\beta-\gamma)^{1-\beta-\gamma}(2\beta)^{4\beta}}} \frac{\|Q(x)\|_{[0,1]}^{1/n}}{\alpha^{\beta}}.$$

As in the previous case with the Muntz-Legendre Polynomials, if we consider powers of $Q(x)$, say $Q(x)^N$, then the ratios $\beta = kN/(nN)$ and $\gamma = qN/(nN)$ do not change and the value $\left\|Q(x)^N\right\|_{[0,1]}^{1/(nN)} = \|Q(x)\|_{[0,1]}^{1/n}$ remains the same. Since the limit as N tends to infinity of $\left((nN+kN+qN+1)(nN-qN+kN+1)(2kN+1)^{-1}\right)^{1/(2nN)}$ is 1, repeating the above derivation with $Q(x)^N$ and letting $N \to \infty$ gives the result we are after.

Lemma 1.6 *Let $Q(x) = x^k c(x)^q p(x)$ be a polynomial of degree n with integer coefficients for which x does not divide $p(x)$. If $\gamma = q/n$, $\beta = k/n$, and $c(x)$ is a critical polynomial for the interval $[0,1]$ with $c(x) \neq x$, then*

$$1 \leq \sqrt{\frac{(1+\beta+\gamma)^{1+\beta+\gamma}(1+\beta-\gamma)^{1+\beta-\gamma}}{(1-\beta+\gamma)^{1-\beta+\gamma}(1-\beta-\gamma)^{1-\beta-\gamma}(2\beta)^{4\beta}}} \frac{\|Q(x)\|_{[0,1]}^{1/n}}{\alpha^{\beta}}$$

where α is a root of $c(x)$.

Given that we know a certain power of $c(x)$ must divide $Q(x)$, the least nonnegative value of β for which the above inequality holds gives a lower bound on the value of k.

In the other direction, if we know a certain power of x divides $Q(x)$, the above argument can be modified to find a lower bound on the degree to which $c(x)$ must also divide $Q(x)$. On the interval $[0, \alpha]$ where α is a root of $c(x)$, consider the polynomials $N_{\alpha,i}(x)$ defined by

$$\begin{aligned} N_{\alpha,i}(x) &= \alpha^q M_{\alpha,i}(\alpha - x) \\ &= \alpha^q \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} \frac{(\alpha-x)^j}{\alpha^j} \left(1 - \frac{(\alpha-x)}{\alpha}\right)^q \\ &= \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} x^q \left(1 - \frac{x}{\alpha}\right)^j \quad \text{for } k \leq i \leq n \end{aligned}$$

with inner product $\langle f(x), g(x) \rangle = \int_0^{\alpha} f(x)g(x)dx$. The polynomials $N_{\alpha,i}(x)$ form an orthogonal basis for the space of functions generated by $x^q(1 - \frac{x}{\alpha})^i$ where $k \leq i \leq n$ since

$$\begin{aligned} \langle N_{\alpha,i}(x), N_{\alpha,j}(x) \rangle &= \langle \alpha^q M_{\alpha,i}(\alpha - x), \alpha^q M_{\alpha,j}(\alpha - x) \rangle \\ &= \alpha^{2q} \int_0^{\alpha} M_{\alpha,i}(\alpha - x) M_{\alpha,j}(\alpha - x) dx \end{aligned}$$

$$\begin{aligned} &= \alpha^{2q}\int_0^\alpha M_{\alpha,i}(y)M_{\alpha,j}(y)dy \\ &= \begin{cases} \alpha^{2q+1}\binom{n+i+2q+1}{n-i+2q}\left((2i+1)\binom{n+i+1}{n-i}\right)^{-1} & i=j \\ 0 & i\neq j. \end{cases} \end{aligned}$$

To proceed, if $Q(x) = x^q c(x)^k p(x) = a_k x^q \left(1-\frac{x}{\alpha}\right)^k + \cdots + a_n x^q \left(1-\frac{x}{\alpha}\right)^n = \sum_{i=k}^n \eta_i N_{\alpha,i}(x)$ is a polynomial in $\mathcal{Z}_{n+q}[x]$, we need to find a lower bound on $|\eta_k|$. We need to look at the coefficients of $(1-\frac{x}{\alpha})^0$ in the expansions of $c(x)/(1-\frac{x}{\alpha})$ and $p(x)$ as linear combinations of powers of $(1-\frac{x}{\alpha})$. If $p(\alpha)\neq 0$, then since $c(x)$ is irreducible, the resultant of $c(x)$ and $p(x)$ must be a nonzero integer. If the set of roots of $c(x)$ is represented by $\{\alpha_i\}_{i=1}^d$, then

$$\left| c_d^{n-kd}\prod_{i=1}^d p(\alpha_i)\right| \geq 1$$

and so for at least one value of i, we must have $|p(\alpha_i)| \geq |c_d|^{k-\frac{n}{d}}$. Expressing $p(x)$ in the form $p(x) = \tilde{p}_0 + \tilde{p}_1(1-\frac{x}{\alpha}) + \cdots + \tilde{p}_{n-kd}(1-\frac{x}{\alpha})^{n-kd}$ allows us to see that if α is the root α_i referred to above, then

$$|\tilde{p}_0| = |p(\alpha)| \geq |c_d|^{k-\frac{n}{d}}.$$

Similarly, if $c(x)/(1-\frac{x}{\alpha}) = -c_d\alpha\prod_{\alpha_i\neq\alpha}(x-\alpha_i) = \tilde{c}_0 + \tilde{c}_1(1-\frac{x}{\alpha}) + \cdots + \tilde{c}_{d-1}(1-\frac{x}{\alpha})^{d-1}$, then

$$|\tilde{c}_0| = |c_d\alpha|\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|$$

where the empty product is taken to be 1 in the case when $c(x)$ is linear and $d=1$. If $Q(x) = x^q c(x)^k p(x) = \sum_{i=k}^n \eta_i N_{\alpha,i}(x)$, then

$$|\eta_k| = \frac{\left|\tilde{c}_0^k\tilde{p}_0\right|}{\binom{n+2q+k+1}{n+2q-k}} \geq \frac{\left|c_d^{2k-\frac{n}{d}}\alpha^k\right|\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k}{\binom{n+2q+k+1}{n+2q-k}}.$$

As in the previous case, since $\int_0^\alpha Q(x)^2dx \leq \alpha\|Q\|_{[0,\alpha]}^2 \leq \alpha\|Q\|_{[0,1]}^2$,

$$\begin{aligned} \sqrt{\alpha}\|Q\|_{[0,1]} &\geq \langle Q(x),Q(x)\rangle^{1/2} \geq \langle \eta_k N_{\alpha,k}(x), \eta_k N_{\alpha,k}(x)\rangle^{1/2} \\ &= \sqrt{\eta_k^2\alpha^{2q+1}\binom{n+k+2q+1}{n-k+2q}\left((2k+1)\binom{n+k+1}{n-k}\right)^{-1}}. \end{aligned}$$

Substituting the above bound for η_k and rearranging now yields

$$1 \leq \sqrt{(2k+1)\binom{n+k+2q+1}{n-k+2q}\binom{n+k+1}{n-k}}\frac{|c_d|^{\frac{n}{d}-2k}\|Q(x)\|_{[0,1]}}{\alpha^{k+q}\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k}.$$

As in the previous case, we have the result needed from the polynomials $N_{\alpha,i}(x)$ and it is no longer advantageous to view $Q(x)$ as a polynomial of degree $n+q$. Substituting $n-q$ for n so that we can view $Q(x)$ as a polynomial of degree n. If $Q(x) = x^q c(x)^k p(x) \in \mathcal{Z}_n[x]$ where $c(x)$ is a critical polynomial for the interval $[0,1]$, and if $p(\alpha) \neq 0$, then

$$\begin{aligned} 1 &\leq \sqrt{(2k+1)\binom{n+k+q+1}{n-k+q}\binom{n-q+k+1}{n-q-k}}\frac{|c_d|^{\frac{n-q}{d}-2k}\left\|Q(x)\right\|_{[0,1]}}{\alpha^{k+q}\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k} \\ &= \sqrt{\frac{(n+k+q+1)(n-q+k+1)}{2k+1}}\sqrt{\binom{n+k+q}{n-k+q}\binom{n-q+k}{n-q-k}}\frac{|c_d|^{\frac{n-q}{d}-2k}\left\|Q(x)\right\|_{[0,1]}}{\alpha^{k+q}\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k}. \end{aligned}$$

This final inequality is dealt with in the same way as before. Using the inequalities that arise from the binomial theorem, taking nth roots, making the substitutions $\beta = k/n$ and $\gamma = q/n$, and then considering the limit as N tends to infinity after applying the above result to $Q(x)^N$ leads to the following lemma.

Lemma 1.7 *Let $Q(x) = x^q c(x)^k p(x)$ be a polynomial of degree n with integer coefficients for which $c(x)$ does not divide $p(x)$. If $\gamma = q/n$, $\beta = k/n$, and $c(x)$ is a critical polynomial for the interval $[0,1]$ of degree d with leading coefficient c_d and $c(x) \neq x$, then*

$$1 \leq \sqrt{\frac{(1+\beta+\gamma)^{1+\beta+\gamma}(1+\beta-\gamma)^{1+\beta-\gamma}}{(1-\beta+\gamma)^{1-\beta+\gamma}(1-\beta-\gamma)^{1-\beta-\gamma}(2\beta)^{4\beta}}}\frac{|c_d|^{\frac{1-\gamma}{d}-2\beta}}{\left|\prod_{\alpha_i\neq\alpha}(\alpha-\alpha_i)\right|^{\beta}\alpha^{\beta+\gamma}}\left\|Q(x)\right\|_{[0,1]}^{1/n}$$

for at least one root α of $c(x)$.

Given that we know a certain power of x must divide $Q(x)$, for each root α_i of $c(x)$ we can find smallest nonnegative value of β for which the above inequality holds. The minimum value of β found across the α_i's gives a lower bound on the value of k.

When $c(x) = 1-x$, Lemmas 1.6 and 1.7 can be combined to give an interval in which both the exponents of x and $1-x$ must lie. Writing $Q(x)$ in form $Q(x) = x^a(1-x)^b\tilde{p}(x)$ where neither x nor $1-x$ divide $\tilde{p}(x)$ and letting $k = \min(a,b)$ leads to one of the following situations. If $k = a$, then $Q(x) = x^k(1-x)^k p(x)$ where x does not divide $p(x)$ and so Lemma 1.6 leads to

$$0 \leq \sqrt{\frac{(1+2\beta)^{1+2\beta}}{(1-2\beta)^{1-2\beta}}}\frac{\left\|Q(x)\right\|_{[0,1]}^{1/n}}{(2\beta)^{2\beta}} - 1.$$

If $k = b$ then $Q(x) = x^k(1-x)^k p(x)$ where $1-x$ does not divide $p(x)$ in which case Lemma 1.7 leads to the same inequality. Considering the right hand side of the inequality as a function

of β, we see the sign of the derivative is determined by the sign of $\ln((1/(4\beta^2)) - 1)$. The right hand side is an increasing function of β for $0 < \beta < 1/\sqrt{8}$ and a decreasing function of β for $1/\sqrt{8} < \beta < 1/2$. For any value of $\|Q(x)\|_{[0,1]}^{1/n}$ that lies in the allowable interval for $\Omega[0,1]$ we will have equality at two points in $(0,1/2)$ and the inequality will be satisfied for all β between these two values. In particular, if $\|Q_n(x)\|_{[0,1]}^{1/n} \leq 1/2.36482727$, the inequality fails to be satisfied when $\beta \leq 0.291824$ or when $\beta \geq 0.411773$.

When $c(x) = 2x - 1$, Lemma 1.7 can be used to show that if $Q(x) = x^q(2x-1)^k p(x)$ is a polynomial of degree n with integer coefficients and $2x - 1$ does not divide $p(x)$, then the inequality

$$1 \leq \sqrt{\frac{(1+\beta+\gamma)^{1+\beta+\gamma}(1+\beta-\gamma)^{1+\beta-\gamma}}{(1-\beta+\gamma)^{1-\beta+\gamma}(1-\beta-\gamma)^{1-\beta-\gamma}(2\beta)^{4\beta}}} 2^{1-\beta} \|Q(x)\|_{[0,1]}^{1/n}$$

must be satisfied where $\gamma = q/n$ and $\beta = k/n$. Using $1/2.36482727$ in place of $\|Q_n(x)\|_{[0,1]}^{1/n}$ and 0.291824 in place of γ leads to $\beta > 0.022613$. By viewing the right hand side of this inequality as the function $f(\gamma,\beta)$, we can use the gradient of $f(\gamma,\beta)$ to show that the implicitly defined curve $f(\gamma,\beta) = 1$ gives β as an increasing function of γ in a neighborhood of the point $(\gamma,\beta) = (0.291824, 0.022613)$. Even though we would like to find a lower bound for β when γ is some value greater than or equal to $\lceil 0.291824n \rceil / n$, using $\gamma = 0.291824$ gives a smaller value and so still gives a valid lower bound. Similarly, since replacing the value $1/2.36482727$ with the smaller value $\|Q_n(x)\|_{[0,1]}^{1/n}$ results in an increased lower bound, these substitutions for γ and $\|Q_n(x)\|_{[0,1]}^{1/n}$ lead to a valid lower bound on β.

Theorem 1.5 *Let $Q_n(x)$ be a polynomial of degree n with integer coefficients. If $Q_n(x)$ is an integer Chebyshev polynomial for the interval $[0,1]$ of sufficiently large degree, then $Q_n(x)$ satisfies the inequality $\left(\|Q_n(x)\|_{[0,1]}\right)^{1/n} \leq 1/2.36482727$ and*

$$Q_n(x) = (x(1-x))^{\lambda_1 n} (2x-1)^{\lambda_2 n} p(x)$$

where $0.291824 < \lambda_1 < 0.411773$, $0.022613 < \lambda_2$, and $p(x)$ is a polynomial with integer coefficients.

For any other known critical polynomial $c(x)$, Lemma 1.7 can also be used to determine a lower bound on the degree to which $c(x)$ must divide an nth integer Chebyshev polynomial. However, since we must use the minimum value taken over the set of all roots of $c(x)$, and since Lemma 1.7 tends to return relatively weak bounds when α is close to zero, it is

worthwhile to construct another inequality by considering the interval $[\alpha, 1]$. It is important to keep in mind that we are working with the root α of $c(x)$ for which $p(\alpha) \geq |c_d|^{k-\frac{n}{d}}$.

On the interval $[\alpha, 1]$, we consider the polynomials $\widetilde{N}_{\alpha,i}(x)$ defined as

$$\begin{aligned}
\widetilde{N}_{\alpha,i}(x) &= N_{1-\alpha,i}(1-x) \\
&= \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} (1-x)^q \left(1 - \frac{1-x}{1-\alpha}\right)^j \\
&= \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} \left(\frac{-\alpha}{1-\alpha}\right)^j (1-x)^q \left(1 - \frac{x}{\alpha}\right)^j
\end{aligned}$$

where $k \leq i \leq n$. Under the inner product $\langle f(x), g(x) \rangle = \int_\alpha^1 f(x)g(x)dx$, the $\widetilde{N}_{\alpha,i}(x)$ form an an orthogonal basis for the space of functions generated by $(1-x)^q(1-\frac{x}{\alpha})^i$ where $k \leq i \leq n$ since

$$\begin{aligned}
\left\langle \widetilde{N}_{\alpha,i}(x), \widetilde{N}_{\alpha,j}(x) \right\rangle &= \langle N_{1-\alpha,i}(1-x), N_{1-\alpha,j}(1-x) \rangle \\
&= \int_\alpha^1 N_{1-\alpha,i}(1-x) N_{1-\alpha,j}(1-x) dx \\
&= \int_0^{1-\alpha} N_{1-\alpha,i}(y) N_{1-\alpha,j}(y) dy \\
&= \begin{cases} (1-\alpha)^{2q+1} \binom{n+i+2q+1}{n-i+2q} \left((2i+1)\binom{n+i+1}{n-i}\right)^{-1} & i = j \\ 0 & i \neq j. \end{cases}
\end{aligned}$$

In the same fashion as before, we would like to express $Q(x) = x^q c(x)^k p(x)$ in the form $Q(x) = a_k(1-x)^q \left(1-\frac{x}{\alpha}\right)^k + \cdots + a_n(1-x)^q \left(1-\frac{x}{\alpha}\right)^n = \sum_{i=k}^n \eta_i \widetilde{N}_{\alpha,i}(x)$. If x^q divides $Q(x)$ were $q/n \leq 0.291824$, then Lemma 1.7 can be used to show $(1-x)^q$ also divides $Q(x)$. Given that $Q(x) = x^q c(x)^k p(x) \in \mathcal{Z}_{n+q}[x]$ where $c(x)$ is a critical polynomial for the interval $[0,1]$ other than x or $1-x$, we may rewrite $Q(x)$ as $Q(x) = (1-x)^q c(x)^k \left(x^q \tilde{p}(x)\right)$. As before, we need to look at the coefficients of $\left(1-\frac{x}{\alpha}\right)^0$ in the expansions of $c(x)/\left(1-\frac{x}{\alpha}\right)$ and $x^q\tilde{p}(x)$ as linear combinations of powers of $\left(1-\frac{x}{\alpha}\right)$. Expressing $x^q\tilde{p}(x)$ in the form

$$x^q \tilde{p}(x) = \tilde{p}_0 + \tilde{p}_1 \left(1 - \frac{x}{\alpha}\right) + \cdots + \tilde{p}_{n-kd} \left(1 - \frac{x}{\alpha}\right)^{n-kd},$$

we see that if $p(\alpha) \geq |c_d|^{k-\frac{n}{d}}$, then

$$\tilde{p}_0 = \alpha^q \tilde{p}(\alpha) = (\alpha/(1-\alpha))^q p(\alpha) \geq (\alpha/(1-\alpha))^q |c_d|^{k-\frac{n}{d}}$$

and in exactly the same fashion as before, if

$$c(x)/\left(1-\frac{x}{\alpha}\right) = -c_d\alpha \prod_{\alpha_i\neq\alpha}(x-\alpha_i) = \tilde{c}_0 + \tilde{c}_1\left(1-\frac{x}{\alpha}\right) + \cdots + \tilde{c}_{d-1}\left(1-\frac{x}{\alpha}\right)^{d-1},$$

then

$$|\tilde{c}_0| = |c_d\alpha| \prod_{\alpha_i\neq\alpha} |\alpha-\alpha_i|$$

where the empty product is taken to be 1 in the case when $c(x)$ is linear and $d = 1$. If $Q(x) = \sum_{i=k}^{n} \eta_i \widetilde{N}_{\alpha,i}(x)$, then

$$|\eta_k| = \frac{\left|\tilde{c}_0^k \tilde{p}_0\right|}{\binom{n+2q+k+1}{n+2q-k}\left(\frac{\alpha}{1-\alpha}\right)^k} \geq \frac{|c_d|^{2k-\frac{n}{d}} \prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k\,\alpha^q}{\binom{n+2q+k+1}{n+2q-k}(1-\alpha)^{q-k}}.$$

In a similar fashion as before, the fact that $\int_\alpha^1 Q(x)^2dx \leq (1-\alpha)\|Q\|^2_{[\alpha,1]} \leq (1-\alpha)\|Q\|^2_{[0,1]}$ leads to

$$\begin{aligned}\sqrt{1-\alpha}\,\|Q\|_{[0,1]} &\geq \langle Q(x),Q(x)\rangle^{1/2} \geq \left\langle \eta_k\widetilde{N}_{\alpha,k}(x), \eta_k\widetilde{N}_{\alpha,k}(x)\right\rangle^{1/2} \\ &= |\eta_k|\sqrt{\frac{(1-\alpha)^{2q+1}\binom{n+k+2q+1}{n-k+2q}}{(2k+1)\binom{n+k+1}{n-k}}}.\end{aligned}$$

Substituting the above bound for η_k and rearranging now yields

$$1 \leq \sqrt{(2k+1)\binom{n+k+2q+1}{n-k+2q}\binom{n+k+1}{n-k}}\,\frac{|c_d|^{\frac{n}{d}-2k}\,\|Q(x)\|_{[0,1]}}{\alpha^q(1-\alpha)^k\prod_{\alpha_i\neq\alpha}|\alpha-\alpha_i|^k}.$$

The only difference to the previous case at this step is that we now have an $\alpha^q(1-\alpha)^k$ in the denominator as opposed to an α^{q+k} term. Applying the same procedure, this difference carries through giving an $\alpha^\gamma(1-\alpha)^\beta$ term as opposed to an $\alpha^{\gamma+\beta}$ term in the final result. Since we know $p(\alpha) \geq |c_d|^{k-\frac{n}{d}}$ must be satisfied for at least one root α of $c(x)$, this last result can be combined with Lemma 1.7 to give a lower bound on $\beta = k/n$.

Lemma 1.8 *Let $Q(x) = x^q c(x)^k p(x)$ be a polynomial of degree n with integer coefficients for which $c(x)$ does not divide $p(x)$ and let $\gamma = q/n$. If $c(x)$ is a critical polynomial for the interval $[0,1]$ of degree d with leading coefficient c_d and $c(x)$ is not equal to x or $1-x$, then for at least one root α of $c(x)$ we must have $\beta = k/n \geq \max(\beta_1,\beta_2)$ where β_1 is the smallest positive value for which the inequality*

$$1 \leq \sqrt{\frac{(1+\beta_1+\gamma)^{1+\beta_1+\gamma}(1+\beta_1-\gamma)^{1+\beta_1-\gamma}}{(1-\beta_1+\gamma)^{1-\beta_1+\gamma}(1-\beta_1-\gamma)^{1-\beta_1-\gamma}(2\beta_1)^{4\beta_1}}}\,\frac{|c_d|^{\frac{1-\gamma}{d}-2\beta_1}}{\left|\prod_{\alpha_i\neq\alpha}(\alpha-\alpha_i)\right|^{\beta_1}\alpha^{\gamma+\beta_1}}\,\|Q(x)\|^{1/n}_{[0,1]}$$

holds and β_2 is the smallest positive value for which the inequality

$$1 \leq \sqrt{\frac{(1+\beta_2+\gamma)^{1+\beta_2+\gamma}(1+\beta_2-\gamma)^{1+\beta_2-\gamma}}{(1-\beta_2+\gamma)^{1-\beta_2+\gamma}(1-\beta_2-\gamma)^{1-\beta_2-\gamma}(2\beta_2)^{4\beta_2}}} \frac{|c_d|^{\frac{1-\gamma}{d}-2\beta_2}}{\left|\prod_{\alpha_i \neq \alpha}(\alpha-\alpha_i)\right|^{\beta_2}\alpha^{\gamma}(1-\alpha)^{\beta_2}} \|Q(x)\|_{[0,1]}^{1/n}$$

holds.

Even though we do not know which root α of $c(x)$ must be used, if we take the minimal value of β given over the set of all roots of $c(x)$, this will give a lower bound on k/n. Any upper bound on $\|Q(x)\|_{[0,1]}^{1/n}$ can be used during the computations since decreasing this value has the effect of increasing the resulting bound on β, but since the inequality $p(\alpha) \geq |c_d|^{k-\frac{n}{d}}$ is dependent on the value of q, the same value of $\gamma = q/n$ must be used throughout the calculations.

For the known nonlinear critical polynomials that are symmetric on the interval $[0,1]$, the optimal value of γ to use is $\gamma = 0$. For the two pairs of non symmetric critical polynomials we can improve the bound slightly by using a marginally larger value of γ. For all roots of the polynomial $c_5(x) = 13x^3 - 20x^2 + 9x - 1$, we find that $\beta > 0.000415$ whenever $\gamma \in [0.00052, 0.00053]$. For sufficiently large n, there will exist an allowable q that puts γ in this interval and so the exponent k on $c_5(x)$ must be greater than $0.000415n$. If the above lemma is used to show $Q(x) = c_5(x)^k p(x)$, then it can also be used to show that $Q(1-x) = c_5(x)^k q(x)$ and so $Q(x) = c_4(x)^k \tilde{q}(x)$ since $c_4(x) = 13x^3 - 19x^2 + 8x - 1 = -c_5(1-x)$. The same bound holds for both polynomials in this nonsymmetric pair. A similar statement can be made for the pair $c_7(x) = 31x^4 - 61x^3 + 41x^2 - 11x + 1$ and $c_8(x) = 31x^4 - 63x^3 + 44x^2 - 12x + 1$. Combining the results with those from Theorem 1.5 leads to the following.

Theorem 1.6 *Let $Q(x)$ be a polynomial of degree n with integer coefficients. If $Q(x)$ is an integer Chebyshev polynomial for the interval $[0,1]$ of sufficiently large degree, then $Q(x)$ satisfies the inequality $\|Q(x)\|_{[0,1]}^{1/n} \leq 1/2.36482727$ and*

$$\begin{aligned} Q(x) \;=\; & (x(1-x))^{\lambda_1 n}\,(2x-1)^{\lambda_2 n}(5x^2-5x+1)^{\lambda_3 n} \\ & \cdot(29x^4-58x^3+40x^2-11x+1)^{\lambda_4 n} \\ & \cdot\left((13x^3-19x^2+8x-1)(13x^3-20x^2+9x-1)\right)^{\lambda_5 n} \\ & \cdot\left((31x^4-61x^3+41x^2-11x+1)(31x^4-63x^3+44x^2-12x+1)\right)^{\lambda_6 n} \\ & \cdot(941x^8-3764x^7+6349x^6-5873x^5+3243x^4-1089x^3+216x^2-23x+1)^{\lambda_7 n} \\ & \cdot p(x) \end{aligned}$$

where $0.411773 > \lambda_1 > 0.291824$, $\lambda_2 > 0.022613$, $\lambda_3 > 0.006518$, $\lambda_4 > 0.001749$, $\lambda_5 > 0.000415$, $\lambda_6 > 0.000140$, $\lambda_7 > 0.000351$, *and* $p(x)$ *is a polynomial with integer coefficients.*

These are the best known bounds on the λ_i for the unrestricted case on the unit interval.

1.2.7 An easy extension to the interval $[0, 1/4]$

It is worth noting that if $c(x)$ is a critical polynomial for $[0, 1/4]$ and α is a root of $c(x)$, then the arguments of the previous section can be repeated with $\|Q\|_{[0,\alpha]} \leq \|Q\|_{[0,1/4]}$ in place of $\|Q\|_{[0,\alpha]} \leq \|Q\|_{[0,1]}$. This leads to equivalent statements to Lemmas 1.6 and 1.7 with all references to the interval $[0, 1]$ replaced with $[0, 1/4]$. As done by Wu [33], this allows us to set up an iterative process to find lower bounds on the degree to which x and $1 - 4x$ must divide an nth integer Chebyshev polynomial for the interval $[0, 1/4]$. Beginning with an exponent of zero for $1 - 4x$, the equivalent statement to Lemma 1.6 gives a lower bound on the exponent of x which in turn can be used with the equivalent statement to Lemma 1.7 to give a lower bound on the exponent of $1 - 4x$. This new lower bound on the exponent of $1 - 4x$ can then be used to increase the lower bound on the exponent of x and the process can be repeated until the lower bounds stabilize.

For the other critical polynomials, this iterative process does not lead to optimal values for the lower bounds on the degrees to which they must divide an nth integer Chebyshev polynomial for the interval $[0, 1/4]$. For the polynomial $1-5x$, it was found to be best to use as large a value as possible for the exponent of x in the equivalent statement to Lemma 1.7. For the nonlinear critical polynomials, the best value to use for the exponent of x is zero. Initial attempts to work on the interval $[\alpha, 1/4]$ with the polynomials

$$\widetilde{N}_{4\alpha,i}(4x) = \sum_{j=i}^{n} (-1)^{n-j} \binom{n+2q+j+1}{n-i+2q} \binom{n-i}{n-j} \left(\frac{-4\alpha}{1-4\alpha}\right)^j (1-4x)^q \left(1 - \frac{x}{\alpha}\right)^j$$

did not improve the results.

Theorem 1.7 *Let $Q(x)$ be a polynomial of degree n with integer coefficients. If $Q(x)$ is an integer Chebyshev polynomial for the interval $[0, 1/4]$ of sufficiently large degree, then $Q(x)$ satisfies the inequality $\|Q(x)\|_{[0,1/4]}^{1/n} \leq 1/5.592408$ and*

$$\begin{aligned} Q(x) &= x^{\mu_1 n}(4x-1)^{\mu_2 n}(5x-1)^{\mu_3 n} \\ &\quad \cdot(29x^2-11x+1)^{\mu_4 n}(169x^3-94x^2+17x-1)^{\mu_5 n} \end{aligned}$$

$$\cdot(961x^4 - 712x^3 + 194x^2 - 23x + 1)^{\mu_6 n}$$
$$\cdot(941x^4 - 703x^3 + 193x^2 - 23x + 1)^{\mu_7 n} q(x)$$

where $\mu_1 > 0.598219$, $\mu_2 > 0.098017$, $\mu_3 > 0.016556$, $\mu_4 > 0.004035$, $\mu_5 > 0.000954$, $\mu_6 > 0.000325$, $\mu_7 > 0.000828$, *and* $q(x)$ *is a polynomial with integer coefficients.*

These are the best known bounds on the μ_i for $i \geq 4$. For the linear critical polynomials, the best bounds are due to Pritsker [27].

Converting these results to the interval $[0, 1]$ by sending x to $x(1-x)$ shows that if $Q(x)$ is a polynomial of even degree and is symmetric around $x = 1/2$, then the λ_i in Theorem 1.5 can be improved to $\lambda_1 > 0.299109$, $\lambda_2 > 0.098017$, $\lambda_3 > 0.008278$, $\lambda_4 > 0.002017$, $\lambda_5 > 0.000477$, $\lambda_6 > 0.000162$, and $\lambda_7 > 0.000414$. Using methods of weighted potential theory, Pritsker has shown that in the even symmetric case, $0.34 \geq \lambda_1 \geq 0.31$, $0.14 \geq \lambda_2 \geq 0.11$, and $0.057 \geq \lambda_3 \geq 0.035$.

1.3 The computation of Integer Chebyshev polynomials on the unit interval

The process of finding an nth integer Chebyshev polynomial for the interval $[0, 1]$ has been described as complicated, even for polynomials of low degree [8]. Although we can reduce the number of polynomials that must be considered to a finite set for any n, the size of this set grows very quickly with the degree and it is hard to keep the problem tractable. The underlying idea behind the process is shared by all the current methods and breaks down into the following three steps. Begin by finding a good upper bound on $\Omega_n[0, 1]$, then attempt to find necessary factors, and finally do an exhaustive search for the remaining terms. The difference between the methods is in how each of these steps is implemented.

An example of an nth integer Chebyshev polynomial is given by Montgomery in [25] for each n from 1 to 5, and this is extended to the case $n = 6$ by Borwein and Erdelyi in [8]. The authors of [8] gave a list of open problems of which the following three referred specifically to the interval $[0, 1]$.

Q1. Find a reasonable algorithm for exactly computing integer Chebyshev polynomials on $[0, 1]$ that would work up to, say, degree 200.

Q3. Do the integer Chebyshev polynomials on $[0, 1]$ have all their zeros in $[0, 1]$?

Q6. Are all the irreducible factors of the integer Chebyshev polynomials on $[0,1]$ forced to be factors as in Lemma 1.2? That is, are all irreducible factors q of the form

$$q(x) = a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0,$$

with all their zeros in $[0,1]$, and with

$$|a_k|^{1/k} < \Omega[0,1]^{-1}?$$

A partial response to the first problem was given by Habsieger and Salvy in [21] where they improved the basic procedure. This allowed them to find symmetric integer Chebyshev polynomials up to degree 75. The 70th integer Chebyshev polynomial for the interval $[0,1]$ found by them gives a negative answer to the last two questions since it has an irreducible factor with complex roots (see Table 1.1). The second step of their method was improved slightly by Wu [33] who attempted to extend the list out to degree 100. Unfortunately, the polynomial he gives of degree 80 is not an integer Chebyshev polynomial for the interval $[0,1]$. The correct symmetric polynomial gives a second counterexample to the last two questions above as it also has a factor with complex roots. By examining the list of factors of known integer Chebyshev polynomials given in Section 1.3.3, one can see that even factors with all real roots don't necessarily satisfy problem Q6 as given above.

It is worthwhile to illustrate the three basic steps in the first nontrivial case which occurs when $n = 4$. For the first step, we note that the polynomial $x^2(1-x)^2$ gives the upper bound $\Omega_4[0,1] \le 1/16$. This is sufficient to force two necessary factors. If $p(x)$ is a polynomial of degree at most 4 for which $\|p(x)\|_{[0,1]} \le 1/16$, then $|p(0)|$ and $|p(1)|$ must be integers of magnitude less than $1/16$ and so $p(0) = p(1) = 0$. Both x and $1-x$ must divide $p(x)$. Letting $p(x) = x(1-x)(ax^2+bx+c) \in \mathcal{Z}_4[x]$, we move on to the final stage. Noting that $2^4\,|p(1/2)| \le 2^4(1/16)$ leads to $|a+2b+4c| \le 1$. Similarly, by considering the points $x = 1/4$ and $x = 3/4$, we get the inequalities $|a+4b+16c| \le 5$ and $|9a+12b+16c| \le 5$. A little algebraic manipulation, together with the fact that an integer linear combination of a, b, and c must be an integer, gives the following constraints.

$$-2 \le c \le 2, \quad -3-6c \le b \le 3-6c, \quad -1-2b-4c \le a \le 1-2b-4c$$

Checking the supremum norm of the resulting polynomial on the interval $[0,1]$ for all allowable combinations of a, b, and c shows that up to sign, there are three distinct nth

integer Chebyshev polynomials for $n = 4$. They are $x^2(1-x)^2$, $x(1-x)(2x-1)^2$, and $x(1-x)(5x^2-5x+1)$. The only known degrees for which there are more than one nth integer Chebyshev polynomial on unit interval are $n = 1$ and $n = 4$.

Even from this small example, it is clear that the final stage becomes the bottleneck in the method when the degree of the unknown factor is too large. For this reason, it is important and worthwhile spending the time to find both a good upper bound on $\Omega_n[0,1]$ and as many necessary factors as possible.

1.3.1 The methods of Habsieger, Salvy, and Wu

Habsieger and Salvy were the first to make significant progress in the computation of integer Chebyshev polynomials on the unit interval. This was done by improving the method for finding repeated divisors, using symmetry conditions to halve the degree of the unknown factor, and incorporating the simplex algorithm into the final stage in order to trim the space to be searched. Although the use of symmetry means one can no longer claim to have found all nth integer Chebyshev polynomials, it still allows one to find an example of such a polynomial as well as the value $\Omega_n[0,1]$. The question as to whether nth integer Chebyshev polynomials on the unit interval are eventually unique for sufficiently large n is still open.

The method of Habsieger and Salvy [21] begins with the computation of

$$c_n = \min_{0<k<n} \|p_k(x)p_{n-k}(x)\|_{[0,1]}$$

where $p_i(x)$ is an ith integer Chebyshev polynomial on the interval $[0,1]$. The value c_n is then used as an upper bound on $\|p_n(x)\|_{[0,1]}$. Since integer Chebyshev polynomials on the unit interval tend to have many repeated factors and the introduction of a new unknown factor is relatively rare, this is a reasonable method to obtain a decent upper bound. In many cases, one actually finds a polynomial of minimal supremum norm of degree n, but this is not known until the end of the final stage.

The second step is iterative and is done on the interval $[0,1/4]$. As shown in Section 1.2.4, there must exist an nth integer Chebyshev polynomial of the form $p_n(x) = G(x(1-x))$ when n is even, and one of the form $p_n(x) = (1-2x)G(x(1-x))$ when n is odd. By setting $F(x) = 1$ when n is even and $F(x) = 1-2x$ when n is odd, we can write $p_n(x) = F(x)G(x(1-x))$ and then attempt to find factors of $G(x)$. Each time we find a new required divisor, $F(x)$ and $G(x)$ are updated so that $F(x)$ always represents the known divisors and $G(x)$ represents the unknown factor.

The initial value of the exponent of x can quickly be found by using the inequality of Borwein and Erdelyi from Section 1.2.5. If $p_n(x) = x^k\left((1-x)^k q_{n-2k}(x)\right)$ where $q_{n-2k}(0) \neq 0$, then

$$1 \leq |q_{n-k}(0)| \leq \sqrt{2k+1}\binom{n+k+1}{n-k}c_n$$

where c_n is the bound on $\|p_n(x)\|_{[0,1]}$ found above. Using the smallest nonnegative integer k for which this holds, we set $F(x)$ equal to $(x(1-x))^k F(x)$.

To work on the interval $[0, 1/4]$, we note that $|F(x)G(x(1-x))| \leq c_n$ on $[0,1]$ is equivalent to

$$|F(u(x))G(x)| \leq c_n \text{ on } [0,1/4] \text{ where } u(x) = \frac{1-\sqrt{1-4x}}{2}$$

provided $F(x) = \pm F(1-x)$. For each critical polynomial, or pair of critical polynomials in Theorem 1.6 that does not yet divide $F(x)$, we can check to see if the corresponding polynomial $d(x)$ on $[0,1/4]$ divides $G(x)$ by bounding its resultant with $G(x)$. For each root α_i of $d(x)$, the fact that

$$|G(\alpha_i)| \leq \frac{c_n}{|F(u(\alpha_i))|}$$

is used to compute a bound on the magnitude of the resultant of $d(x)$ and $G(x)$. If this bound is less than one, we set $F(x)$ equal to $F(x)c(x)$. Using Markov's bound on the magnitude of the mth derivative as given in Section 1.2, this can be extended to find multiple factors. However, this typically only gives a few additional factors of the linear polynomials $d(x) = 4x-1$ and $5x-1$.

The main technique used by Habsieger and Salvy to increase the exponent of a critical polynomial already dividing $F(x)$ is based on Lagrange interpolation. Given the $g+1$ distinct points $x_0, x_1, \ldots, x_g$ where g is the degree of $G(x)$, we may write

$$G(x) = \sum_{i=0}^{g} G(x_i) \prod_{i \neq j} \frac{x - x_j}{x_i - x_j}.$$

Since $|G(x_i)| \leq c_n / |F(u(x_i))|$, for each root α_k of $d(x)$ we have the inequality

$$|G(\alpha_k)| \leq \sum_{i=0}^{g} \frac{c_n}{|F(u(x_i))|} \prod_{i \neq j} \left| \frac{\alpha_k - x_j}{x_i - x_j} \right|.$$

For each root α_k of $d(x)$, a set of $\{x_i\}$ that minimize this upper bound on $|G(\alpha_k)|$ is chosen in an attempt to show

$$|a_m|^g \, |G(\alpha_1)| \, |G(\alpha_2)| \cdots |G(\alpha_m)| < 1$$

where m and a_m are the degree and leading coefficient of $d(x)$. Since the resultant of $G(x)$ and $d(x)$ must be an integer, each time it is shown to be less than one in magnitude we know $d(x)$ divides $G(x)$. It should be pointed out that although one would like to find values of $\{x_i\}$ that give an absolute minimum for the above upper bound on $|G(\alpha_k)|$, this becomes computationally prohibitive as the number of distinct factors of $F(x)$ increases, even for modest sizes of g. Luckily, a few iterations of any decent optimizing scheme appear to produce adequate results. This procedure is carried out for each of the critical polynomials $d(x)$ in Theorem 1.7. Each time the resultant of $G(x)$ with one of the polynomials $d(x)$ is shown to be zero, $F(x)$ is updated by setting it equal to $F(x)c(x)$ where $c(x)$ is the critical polynomial or pair of critical polynomials on $[0,1]$ corresponding to the polynomial $d(x)$ and the process is restarted. This continues until no more required factors can be found.

For the final stage of the algorithm, an exhaustive search is done to determine the coefficients of the unknown factor $G(x) = a_g x^g + a_{g-1}x^{g-1} + \cdots + a_1 x + a_0$. The fact that $|G(x)| \leq c_n / |F(u(x))|$ for any $x \in [0, 1/4]$ is used once again, but here the purpose is to set up a system of linear inequalities. By evaluating $c_n/F(u(x))$ at sufficiently many points in the interval $[0, 1/4]$, a convex polytope can be constructed that contains the point $(a_0, a_1, \ldots, a_g)$ corresponding to $G(x)$. To find this point, we treat the a_i as variables and find all points with integer coordinates that are not in the exterior of the bounding polytope. For each such point, the corresponding polynomial $G(x)$ is constructed and we compute $\|F(u(x))G(x)\|_{[0,1]}$. Any polynomial $G(x)$ that gives a minimal supremum norm is one we are looking for.

The method used to find the integer coordinate points makes use of the simplex algorithm. Initially, the simplex algorithm is used to find the maximum and minimum value of each a_i on the bounding polytope. These extreme coordinate values define a polyrectangle and if the number of integer coordinate points contained within it is not too large, each is checked to see if it lies within the bounding polytope. For larger polyrectangles, the coordinate with minimal variation is chosen and the method is applied recursively to each of its allowable values. Habsieger and Salvy mention that it appears to be better to work on the interval $[4, \infty)$ and compute the coefficients of the reciprocal polynomial of $G(x)$ in the basis $1, (x-4), (x-4)(x-5), \ldots$ instead of the coefficients of $G(x)$ itself. Using this method allowed them to find nth integer Chebyshev polynomials for all n from 0 to 75. For the final step, the largest degree in the unknown factor $G(x)$ they were able to handle was degree 12.

Wu's contribution

Wu's main contribution [32] [33] to the initial method comes in the form of an improvement in the second step. By incorporating the ideas from the section on generalized Muntz-Legendre polynomials, Wu manages to find additional factors of the linear critical polynomials on $[0, 1/4]$. The first and third steps are also modified slightly.

As with Habsieger and Salvy, Wu begins with the computation of

$$c_n = \min_{0<k<n} \|p_k(x)p_{n-k}(x)\|_{[0,1]}$$

where $p_i(x)$ is an ith integer Chebyshev polynomial on the interval $[0, 1]$ and writes $p_n(x)$ as $F(x)G(x(1-x))$ where $F(x)$ represents the known factors and $G(x(1-x))$ represents the unknown portion of $p_n(x)$. In an effort to reduce this bound, Wu considers the polynomials $F(x), F(x)(x(1-x)), F(x)(x(1-x))^2, \ldots, F(x)(x(1-x))^g$ where g is the degree of $G(x)$, and then constructs the lattice generated by the vectors whose components are the coefficients of these polynomials. Using the LLL algorithm, a new basis for the space of integer linear combinations of these polynomials can be constructed. In the event that any of the new basis vectors has supremum norm less than c_n on the unit interval, the value of c_n is updated. These polynomials are also used in the final step when performing the exhaustive search for $G(x)$. Since each of the new basis polynomials is of the form $F(x)B_i(x(1-x))$ and $F(x)G(x(1-x))$ can be written as an integer linear combination of these, we note that $G(x) = a_0B_0(x) + a_1B_1(x) + \cdots + a_gB_g(x)$. This allows for the method of Habsieger and Salvy to be used in order to determine the coefficients a_i.

The main improvement in Wu's method comes from applying the ideas used in the section on generalized Muntz-Legendre polynomials in order to increase the exponents of the linear polynomials x, $4x-1$ and $5x-1$ while working on the interval $[0, 1/4]$. Working from left to right and applying the Gram-Schmidt orthogonalization process to the functions $(ax+b)^gF(u(x)), (ax+b)^{g-1}F(u(x)), \ldots, (ax+b)^0F(u(x))$ with inner product $\langle f(x), g(x)\rangle = \int_0^{1/4} f(x)g(x)dx$ results in the orthogonal functions $L_g(x)$, $L_{g-1}(x)$ $\ldots$, $L_0(x)$ and allows for the expression

$$p_n(u(x)) = F(u(x))G(x) = \sum_{i=0}^{g} a_i(ax+b)^iF(u(x)) = \sum_{i=0}^{g} \lambda_iL_i(x).$$

Noting that $G(x) = \sum_{i=0}^{g} a_i(ax+b)^i$ shows that $a_0 = G(-b/a)$ and so $a^ga_0 \in \mathbb{Z}$. Furthermore, since the coefficient of the $(ax+b)^0F(u(x))$ term in $L_0(x)$ is 1, $\lambda_0 = a_0$. The

orthogonality of the functions $L_i(x)$ and the fact that $p_n(u(x)) \leq c_n$ on $[0, 1/4]$ now gives

$$\frac{c_n}{2} \geq \sqrt{\int_0^{1/4} p_n(u(x))p_n(u(x))dx} = \sqrt{\sum_{i=0}^{g} \lambda_i^2 \|L_i(x)\|_{2,[0,1/4]}^2} \geq |\lambda_0| \, \|L_0(x)\|_{2,[0,1/4]}$$

or

$$|a^g a_0| \leq \frac{|a^g| \, c_n}{2\|L_0(x)\|_{2,[0,1/4]}}.$$

If the right hand side of this last inequality is less than 1, then $a_0 = 0$ and $ax + b$ divides $G(x)$. Transferring this factor to $F(u(x))$, relabelling $L_1(x),\ldots,L_g(x)$ as $L_0(x),\ldots,L_{g-1}(x)$, and decreasing g by 1 allows for the process to be repeated with a minimal amount of work until no additonal factors of $ax + b$ are found.

These modifications to the method of Habsieger and Salvy allowed Wu to extend the list of known symmetric integer Chebyshev polynomials for the interval $[0, 1]$ out to degree 100. Unfortunately, there is an error in the case $n = 80$. Wu gives the polynomial

$$(x(1-x))^{27}(1-2x)^{10}(5x^2-5x+1)^4(29x^4-58x^3+40x^2-11x+1)^2$$

but the correct symmetric $80th$ integer Chebyshev polynomial on the unit interval is

$$\begin{aligned}(x(1-x))^{26}&(1-2x)^{10}(5x^2-5x+1)^3(29x^4-58x^3+40x^2-11x+1)\\ &\cdot(821x^8-3284x^7+5555x^6-5171x^5+2886x^4-985x^3+200x^2-22x+1).\end{aligned}$$

It is unclear as to exactly where Wu's error was made, but the comment made in [32] about numerical inaccuracies in the pivoting procedure suggests that inexact arithmetic was used.

1.3.2 An improved method

The method of finding an nth integer Chebyshev polynomial can be improved significantly by introducing the simplex algorithm in the second step and basing the final search on the resultants of the unknown factor with many linear polynomials having their roots in the interval $[0, 1/4]$. The process begins in the same way as the method of Habsieger and Salvy. Starting with the upper bound on $\|p_n(x)\|_{[0,1]}$ of

$$c_n = \min_{0<k<n} \|p_k(x)p_{n-k}(x)\|_{[0,1]}$$

where $p_i(x)$ is an ith integer Chebyshev polynomial on the interval $[0, 1]$, the inequality of Borwein and Erdelyi from Section 1.2.5 is used to find the initial exponent k for $x(1 - x)$.

Setting $F(x) = (x(1-x))^k$ when n is even and $F(x) = (x(1-x))^k(1-2x)$ when n is odd, $p_n(x)$ is then written as $p_n(x) = F(x)G(x(1-x))$ and we move to the interval $[0, 1/4]$ in an attempt to find additional factors of $G(x)$. For each critical polynomial $d(x)$ in Theorem 1.7 and for the polynomial $6x-1$, let m, b_m, and $\{\alpha_i\}_{i=1}^m$ be the degree, leading coefficient, and roots of $d(x)$. If $F(u(\alpha_i)) \neq 0$, then $|G(\alpha_i)| \leq c_n / |F(u(\alpha_i))|$ and so the resultant of $G(x)$ and $d(x)$ can be bounded by the inequality

$$|\mathrm{res}_x(G(x), d(x))| = \left| b_m^g \prod_{i=1}^{m} G(\alpha_i) \right| \leq |b_m|^g \, c_n^m \prod_{i=1}^{m} \frac{1}{|F(u(\alpha_i))|}.$$

It is when $d(x(1-x))$ divides $F(x)$ that the process differs from that of Habsieger and Salvy. In this case, when α_i is rational, the simplex algorithm is used to bound $|G(\alpha_i)|$ directly. Writing $G(x) = a_g x^g + a_{g-1}x^{g-1} + \cdots + a_1 x + a_0$ and using the fact that $|G(x)| \leq c_n / |F(u(x))|$ for all $x \in [0, 1/4]$ at which $F(u(x)) \neq 0$, we can evaluate the right hand side at sufficiently many points in $[0, 1/4]$ in order to construct a convex polytope which contains all allowable points $(a_0, a_1, \ldots, a_g)$. Finding the maximum and minimum values of the linear objective function $b_m^g G(\alpha_i)$ under the above constraints then gives bounds on the resultant. Due to observed numerical instabilities in the simplex algorithm, exact rational arithmetic was used with an iterative process in order to select the points $x_i \in [0, 1/4]$. Starting with sufficiently many equally spaced points so that the feasible set was bounded, the simplex algorithm was used to find a point $\tilde{a} = (\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_g)$ in the feasible set at which $b_m^g G(\alpha_i)$ was maximized. The number of equally spaced points $\{x_i\}$ was then doubled and those x_i giving rise to a hyperplane either containing $\tilde{a}$ or removing $\tilde{a}$ from the feasible set were used during the next iteration. In order to avoid questions of numerical stability, when the α_i are not rational the simplex algorithm was used to find bounds on the values of $G(w_i/v_i)$ where w_i/v_i is a continued fraction convergent of α_i. The w_i/v_i and bounds on $|G(w_i, v_i)|$ were then used with the method of Lagrange interpolation of Habsieger and Salvy in an attempt to show that the resultant of $G(x)$ and $d(x)$ must be zero. The method of Wu was not used as it provided no additional factors.

To improve the final step of the process, the search was based on the resultants of $G(x)$ with $g+1$ linear polynomials $y = v_i x - w_i$ having roots in $[0, 1/4]$. Letting $r_i = v_i^g G(w_i/v_i)$, the system of equations $\left\{ r_i = v_i^g \sum_{k=0}^{g} a_k (w_i/v_i)^k \right\}_{i=1}^{g+1}$ can be used to express the coefficients of $G(x)$ in terms of the integers r_i. Although we do not know the values of the r_i which correspond to the desired polynomial beforehand, we can reduce the number of possible combinations to consider to a finite set and then run through this set in order to find the

combination giving rise to a polynomial $F(x)G(x(1-x))$ of minimal supremum norm on the interval $[0,1]$. To reduce the size of the set of allowable combinations, we first bound the r_i with the expression $|v_i^g G(w_i/v_i)| \leq v_i^g c_n / |F(u(w_i/v_i))|$ when $F(u(w_i/v_i)) \neq 0$, and by the use of the simplex algorithm as above when $F(u(w_i/v_i)) = 0$. The important thing to note in order to further reduce the size of this set is that the coefficients a_k of $G(x)$, which are integers, are given as rational linear combinations of the r_i. This allows us to set up a system of congruences that the r_i must satisfy. Rather than using the Chinese Remainder theorem, the following method was used to solve the system of congruences. Not only is it simple and easy to implement, but it also allows for a very convenient and compact description for the set of solutions.

Writing the rational linear combination of the r_i corresponding to each a_j as

$$\frac{1}{m_j} \sum_{i=1}^{g+1} t_{j,i} r_i = a_j$$

where the m_j and $t_{j,i}$ are integers gives rise to the $g+1$ congruence relations

$$\sum_{i=1}^{g+1} t_{j,i} r_i \equiv 0 \pmod{m_j}.$$

Letting $M = \operatorname{lcm}(m_1, m_2, \ldots, m_{g+1})$ gives the equivalent form

$$\sum_{i=1}^{g+1} \frac{M}{m_j} t_{j,i} r_i \equiv 0 \pmod{M}$$

which can be written as

$$S\,\mathbf{r} = \begin{bmatrix} s_{1,1} & s_{1,2} & s_{1,3} & \cdots & s_{1,g+1} \\ s_{2,1} & s_{2,2} & & & \vdots \\ s_{3,1} & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ s_{g+1,1} & \cdots & \cdots & \cdots & s_{g+1,g+1} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ \vdots \\ r_{g+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \pmod{M}$$

where $s_{j,i} = (M/m_j) t_{j,i}$. Note that interchanging rows of S or replacing row k with row k plus an integer multiple of row l for any $l \neq k$ does not change the set of solutions. This allows for a form of Gaussian elimination to be applied to the matrix S using the Euclidean algorithm. After the elimination step, back substitution using the resulting simplified congruences and the initial bounds on the r_i allows one to step through all allowable

combinations of the r_i in an orderly fashion. In the author's implementation, the elimination step was arranged so that the back substitution resulted in considering the r_i in the order r_1, r_2, ..., r_{g+1}. Each choice of r_1 through r_{i-1} then resulted in a congruence of the form

$$\tilde{a}r_i \equiv \tilde{b} \pmod{\tilde{m}_i}$$

which was easily handled. As in the method of Habsieger and Salvy, the simplex algorithm can be used to further reduce the bounds on r_i when r_1, r_2, ..., r_{i-1} are fixed. In the event that $r_i = 0$, it was found that for $g \geq 11$, it was worthwhile to consider the subcase where $F(x)$ is multiplied by the factor $v_i x(1-x) - w_i$, the degree of $G(x)$ is decreased by 1, and the values of r_k for $1 \leq k < i$ are updated appropriately. As a final time saving technique, since multiplying a polynomial by -1 does not change the value of its supremum norm on an interval, we can restrict our search to combinations of the r_i for which the first nonzero r_i is negative.

With the full procedure in hand, one can now improve the first step as well. By adding suspected divisors to the known divisors at the end of the second stage, one can execute the final step in an attempt to reduce the bound c_n. After completion, the suspected factors added without justification must be removed. If the bound on $\Omega_n[0,1]$ was reduced, one returns to the second step with an improved value for c_n in an attempt to find more necessary factors. If the bound was not decreased, one simply continues on to the third step.

These improvements to the general method allow for significantly better results. Habsieger and Salvy managed to find nth integer Chebyshev polynomials on $[0,1]$ for all n up to and including $n = 75$ and could only handle cases where the degree of the unknown factor was at most equal to 12. With the new method, the maximum degree of the unknown factor after termination of the second step was 7 for $n \leq 75$, and an unknown of degree greater than 12 was not encountered in the final stage until $n = 127$. The new method of finding the unknown factor $G(x)$ is significantly better as well. It handled all cases where the degree of $G(x)$ was less than or equal to 14, handled many cases where the degree of $G(x)$ equaled 15, and even managed to handle a few degree 16 cases. This allowed for the computation of $\Omega_n[0,1]$ and the discovery of a corresponding nth integer Chebyshev polynomials on $[0,1]$ for all $n \leq 145$, and for all but 27 of the cases where $n \leq 230$.

1.3.3 The results of the improved method

The results of the new method for finding nth integer Chebyshev polynomials on $[0, 1]$ are presented in Tables 1.1, 1.2, and 1.3. The following polynomials are the factors referred to in the tables.

$$
\begin{aligned}
h_1(x) &= x(1-x) \\
h_2(x) &= 2x-1 \\
h_3(x) &= 5x^2-5x+1 \\
h_4(x) &= 6x^2-6x+1 \\
h_5(x) &= 29x^4-58x^3+40x^2-11x+1 \\
h_6(x) &= 33x^4-66x^3+45x^2-12x+1 \\
h_7(x) &= 34x^4-68x^3+46x^2-12x+1 \\
h_8(x) &= (7x^2-6x+1)(7x^2-8x+2) \\
h_9(x) &= 161x^6-483x^5+575x^4-345x^3+109x^2-17x+1 \\
h_{10}(x) &= (13x^3-19x^2+8x-1)(13x^3-20x^2+9x-1) \\
h_{11}(x) &= 181x^6-543x^5+644x^4-383x^3+119x^2-18x+1 \\
h_{12}(x) &= 193x^6-579x^5+683x^4-401x^3+122x^2-18x+1 \\
h_{13}(x) &= 821x^8-3284x^7+5555x^6-5171x^5+2886x^4-985x^3+200x^2-22x+1 \\
h_{14}(x) &= 941x^8-3764x^7+6349x^6-5873x^5+3243x^4-1089x^3+216x^2-23x+1 \\
h_{15}(x) &= (31x^4-61x^3+41x^2-11x+1)(31x^4-63x^3+44x^2-12x+1) \\
h_{16}(x) &= 4921x^{10}-24605x^9+53804x^8-67586x^7+53866x^6-28388x^5+9995x^4 \\
&\quad -2317x^3+338x^2-28x+1 \\
h_{17}(x) &= 31169x^{12}-187014x^{11}+502099x^{10}-796200x^9+828936x^8-595698x^7 \\
&\quad +302334x^6-108945x^5+27600x^4-4783x^3+537x^2-35x+1 \\
h_{18}(x) &= 43609x^{12}-261654x^{11}+704777x^{10}-1125390x^9+1184854x^8-865270x^7 \\
&\quad +448776x^6-166327x^5+43659x^4-7905x^3+936x^2-65x+2 \\
h_{19}(x) &= 161429x^{14}-1130003x^{13}+3599830x^{12}-6908941x^{11}+8913112x^{10} \\
&\quad -8165339x^9+5470288x^8-2718775x^7+1005970x^6-275399x^5+54846x^4 \\
&\quad -7697x^3+719x^2-40x+1
\end{aligned}
$$

$$\begin{aligned}
h_{20}(x) &= 161813x^{14} - 1132691x^{13} + 3608246x^{12} - 6924493x^{11} + 8931952x^{10} \\
&\quad -8181043x^9 + 5479474x^8 - 2722543x^7 + 1007031x^6 - 275594x^5 + 54867x^4 \\
&\quad -7698x^3 + 719x^2 - 40x + 1 \\
h_{21}(x) &= 161929x^{14} - 1133503x^{13} + 3610755x^{12} - 6928991x^{11} + 8937122x^{10} \\
&\quad -8185014x^9 + 5481532x^8 - 2723251x^7 + 1007185x^6 - 275613x^5 + 54868x^4 \\
&\quad -7698x^3 + 719x^2 - 40x + 1 \\
h_{22}(x) &= 887981x^{16} - 7103848x^{15} + 26189139x^{14} - 59006633x^{13} + 90856296x^{12} \\
&\quad -101276631x^{11} + 84454852x^{10} - 53688009x^9 + 26265936x^8 - 9911593x^7 \\
&\quad +2872148x^6 - 631701x^5 + 103263x^4 - 12115x^3 + 961x^2 - 46x + 1 \\
h_{23}(x) &= 907201x^{16} - 7257608x^{15} + 26750188x^{14} - 60243176x^{13} + 92693614x^{12} \\
&\quad -103221560x^{11} + 85965780x^{10} - 54562008x^9 + 26643715x^8 - 10032840x^7 \\
&\quad +2900545x^6 - 636399x^5 + 103781x^4 - 12149x^3 + 962x^2 - 46x + 1
\end{aligned}$$

1.4 The search for candidate critical polynomials

The technique referred to in Section 1.2.3 to reduce the upper bound on $\Omega[0,1]$ relies on having a good set of polynomials to work with. What is classified as a good polynomial here is one that is likely to be a divisor of an nth integer Chebyshev polynomial on $[0,1]$ for sufficiently large n. The factors $h_i(x)$ from the previous section, being factors of integer Chebyshev polynomials on the unit interval for at least some n are such polynomials. Three other sources of such polynomials are the Schur-Siegel-Smyth trace problem, the function $u(x)$ related to the Gorshkov-Wirsing polynomials, and factors of polynomials with small supremum norm found in the previous section that are not nth integer Chebyshev polynomials for the unit interval. Of this last type, the following polynomials do not show up in the next two subsections.

$$\begin{aligned}
r_1(x) &= 30689x^{12} - 184134x^{11} + 494483x^{10} - 784520x^9 + 817442x^8 - 588122x^7 \\
&\quad +298951x^6 - 107936x^5 + 27408x^4 - 4762x^3 + 536x^2 - 35x + 1 \\
r_2(x) &= 179213x^{14} - 1254491x^{13} + 3994456x^{12} - 7658353x^{11} + 9863413x^{10} \\
&\quad -9014198x^9 + 6019243x^8 - 2978749x^7 + 1096048x^6 - 297935x^5 \\
&\quad +58801x^4 - 8158x^3 + 751x^2 - 41x + 1
\end{aligned}$$

Table 1.1: Integer Chebyshev polynomials for the interval $[0,1]$ ($n = 0$ to 76)

n	$\Omega_n[0,1]^{-1}$	Polynomial(s)	n	$\Omega_n[0,1]^{-1}$	Polynomial
0	–	1			
1	1	$1,\ x,\ 1-x,\ 2x-1$	39	2.33704596	$h_1^{13}h_2^5h_3^2h_5$
2	2	h_1	40	2.31744217	$h_1^{13}h_2^6h_3^2h_5$
3	2.18224727	h_1h_2	41	2.33149140	$h_1^{14}h_2^5h_3^2h_5$
4	2	$h_1^2,\ h_1h_2^2,\ h_1h_3$	42	2.32573112	$h_1^{14}h_2^6h_32h_5$
5	2.23606797	$h_1^2h_2$	43	2.32794984	$h_1^{14}h_2^5h_3^3h_5$
6	2.18224727	$h_1^2h_2^2$	44	2.32932654	$h_1^{15}h_2^6h_3^2h_5$
7	2.22372632	$h_1^3h_2$	45	2.31596596	$h_1^{15}h_2^7h_3^2h_5$
8	2.22782986	$h_1^3h_2^2$	46	2.33234040	$h_1^{16}h_2^6h_3^2h_5$
9	2.29702748	$h_1^3h_2h_3$	47	2.32349270	$h_1^{15}h_2^5h_3^2h_5^2$
10	2.23606797	$h_1^4h_2^2$	48	2.33770356	$h_1^{16}h_2^6h_3^3h_5$
11	2.25009568	$h_1^4h_2h_3$	49	2.32967805	$h_1^{16}h_2^7h_3^2h_4h_5$
12	2.27856669	$h_1^4h_2^2h_3$	50	2.32754248	$h_1^{16}h_2^6h_3^2h_5^2$
13	2.24573490	$h_1^4h_2h_3^2$	51	2.33340214	$h_1^{17}h_2^7h_3^3h_5$
14	2.31766354	$h_1^5h_2^2h_3$	52	2.33142077	$h_1^{17}h_2^8h_3^2h_4h_5$
15	2.26211527	$h_1^5h_2^3h_3$	53	2.33829879	$h_1^{18}h_2^7h_3^3h_5$
16	2.28609047	$h_1^6h_2^2h_3$	54	2.32603479	$h_1^{18}h_2^8h_3^3h_5$
17	2.29584896	$h_1^6h_2^3h_3$	55	2.33247963	$h_1^{19}h_2^7h_3^3h_5$
18	2.29702748	$h_1^6h_2^2h_3^2$	56	2.32968376	$h_1^{19}h_2^8h_3^3h_5$
19	2.31639873	$h_1^7h_2^3h_3$	57	2.33130232	$h_1^{19}h_2^7h_3^4h_5$
20	2.29070109	$h_1^6h_2^2h_3h_5$	58	2.33290562	$h_1^{20}h_2^8h_3^3h_5$
21	2.29422160	$h_1^8h_2^3h_3$	59	2.33220983	$h_1^{19}h_2^7h_3^3h_5^2$
22	2.29855818	$h_1^8h_2^4h_3$	60	2.33574336	$h_1^{21}h_2^8h_3^3h_5$
23	2.31301167	$h_1^8h_2^3h_3^2$	61	2.33096907	$h_1^{20}h_2^7h_3^3h_5^2$
24	2.31149862	$h_1^9h_2^4h_3$	62	2.33692684	$h_1^{21}h_2^8h_3^4h_5$
25	2.32754248	$h_1^8h_2^3h_3h_5$	63	2.33105072	$h_1^{20}h_2^7h_3^3h_5h_{10}$
26	2.30208977	$h_1^9h_2^4h_3^2$	64	2.33433366	$h_1^{21}h_2^8h_3^3h_5^2$
27	2.32440612	$h_1^9h_2^3h_3h_5$	65	2.33465253	$h_1^{22}h_2^9h_3^4h_5$
28	2.31766354	$h_1^{10}h_2^4h_3^2$	66	2.33716820	$h_1^{22}h_2^{10}h_3^3h_4h_5$
29	2.30626874	$h_1^{11}h_2^5h_3$	67	2.33779974	$h_1^{23}h_2^9h_3^4h_5$
30	2.31610249	$h_1^{10}h_2^4h_3h_5$	68	2.33222376	$h_1^{23}h_2^{10}h_3^3h_4h_5$
31	2.30666526	$h_1^{11}h_2^5h_3^2$	69	2.33309478	$h_1^{22}h_2^9h_3^3h_5h_{10}$
32	2.31918334	$h_1^{11}h_2^4h_3h_5$	70	2.33216994	$h_1^{22}h_2^8h_3^2h_5h_{16}$
33	2.31794438	$h_1^{12}h_2^5h_3^2$	71	2.33324154	$h_1^{24}h_2^{11}h_3^3h_4h_5$
34	2.33071440	$h_1^{11}h_2^4h_3^2h_5$	72	2.33942227	$h_1^{23}h_2^8h_3^4h_5h_{10}$
35	2.31371757	$h_1^{11}h_2^5h_3h_4h_5$	73	2.33945780	$h_1^{24}h_2^9h_3^4h_5^2$
36	2.31862425	$h_1^{12}h_2^4h_3^2h_5$	74	2.33658654	$h_1^{23}h_2^8h_3^4h_5h_{15}$
37	2.32370076	$h_1^{12}h_2^5h_3^2h_5$	75	2.33666483	$h_1^{24}h_2^9h_3^4h_5h_{10}$
38	2.31639873	$h_1^{14}h_2^6h_3^2$	76	2.33815431	$h_1^{24}h_2^{10}h_3^4h_4h_5h_6$

Table 1.2: Integer Chebyshev polynomials for the interval $[0,1]$ ($n = 77$ to 154)

n	$\Omega_n[0,1]^{-1}$	Polynomial	n	$\Omega_n[0,1]^{-1}$	Polynomial
77	2.33683660	$h_1^{25}h_2^9h_3^3h_5h_{14}$	116	2.34795393	$h_1^{38}h_2^{14}h_3^6h_5^2h_{10}$
78	2.33704596	$h_1^{26}h_2^{10}h_3^4h_5^2$	117	2.34500490	$h_1^{38}h_2^{13}h_3^5h_5h_{10}h_{14}$
79	2.34098055	$h_1^{26}h_2^9h_3^3h_5h_{14}$	118	2.34577006	$h_1^{37}h_2^{14}h_3^4h_5^2h_{10}h_{14}$
80	2.33896527	$h_1^{26}h_2^{10}h_3^3h_5h_{13}$	119	2.34477067	$h_1^{39}h_2^{15}h_3^6h_5^2h_{10}$
81	2.33860591	$h_1^{26}h_2^9h_3^5h_5h_{10}$	120	2.34423737	$h_1^{38}h_2^{14}h_3^4h_4h_5^2h_{10}h_{11}$
82	2.33908234	$h_1^{27}h_2^{12}h_3^3h_4h_5^2$	121	2.34448487	$h_1^{40}h_2^{15}h_3^6h_5^2h_{10}$
83	2.33703405	$h_1^{27}h_2^{11}h_3^4h_5h_{10}$	122	2.34378508	$h_1^{39}h_2^{14}h_3^4h_5^2h_{19}$
84	2.33578895	$h_1^{27}h_2^{10}h_3^5h_5h_{10}$	123	2.34439044	$h_1^{39}h_2^{13}h_3^5h_5^2h_{12}h_{14}$
85	2.34066658	$h_1^{26}h_2^9h_3^3h_5h_{10}h_{14}$	124	2.34960324	$h_1^{39}h_2^{14}h_3^5h_5^2h_{10}h_{14}$
86	2.34034898	$h_1^{28}h_2^{10}h_3^5h_5h_{10}$	125	2.34786626	$h_1^{41}h_2^{15}h_3^7h_5^2h_{10}$
87	2.34203352	$h_1^{27}h_2^9h_3^3h_5h_{10}h_{14}$	126	2.35016259	$h_1^{40}h_2^{14}h_3^5h_5^2h_{10}h_{14}$
88	2.34009517	$h_1^{29}h_2^{10}h_3^4h_5h_{14}$	127	2.34551883	$h_1^{40}h_2^{15}h_3^5h_5^2h_{10}h_{14}$
89	2.34159749	$h_1^{29}h_2^{11}h_3^5h_5h_{10}$	128	2.34611715	$h_1^{40}h_2^{16}h_3^5h_4h_5^2h_{17}$
90	2.34215408	$h_1^{28}h_2^{10}h_3^3h_5h_{10}h_{14}$	129	2.34905859	$h_1^{42}h_2^{15}h_3^6h_5^3h_{10}$
91	2.33789130	$h_1^{30}h_2^{13}h_3^4h_4h_5^2$	130	2.34719016	$h_1^{43}h_2^{16}h_3^7h_5^2h_{10}$
92	2.34027671	$h_1^{30}h_2^{12}h_3^3h_4h_5h_{14}$	131	2.34938969	$h_1^{42}h_2^{15}h_3^5h_5^2h_{10}h_{14}$
93	2.34124790	$h_1^{30}h_2^{13}h_3^5h_4h_5h_6$	132	2.34708501	$h_1^{43}h_2^{16}h_3^6h_5^3h_{10}$
94	2.34344270	$h_1^{29}h_2^{10}h_3^4h_5h_{10}h_{14}$	133	2.34785751	$h_1^{42}h_2^{15}h_3^6h_5^2h_{10}h_{14}$
95	2.34002364	$h_1^{31}h_2^{11}h_3^3h_5h_{18}$	134	2.34628359	$h_1^{43}h_2^{16}h_3^5h_4h_5^2h_{10}h_{11}$
96	2.34339293	$h_1^{32}h_2^{14}h_3^4h_4h_5^2$	135	2.34773453	$h_1^{43}h_2^{15}h_3^6h_5^2h_{10}h_{14}$
97	2.34027461	$h_1^{31}h_2^{11}h_3^4h_5^2h_{14}$	136	2.34711093	$h_1^{44}h_2^{16}h_3^5h_5^2h_{10}h_{14}$
98	2.34205718	$h_1^{32}h_2^{12}h_3^6h_5h_{10}$	137	2.34802942	$h_1^{43}h_2^{15}h_3^5h_5^3h_{10}h_{14}$
99	2.34250303	$h_1^{32}h_2^{13}h_3^6h_4h_5h_6$	138	2.34709892	$h_1^{44}h_2^{16}h_3^6h_5^2h_{10}h_{14}$
100	2.34163023	$h_1^{34}h_2^{12}h_3^4h_5h_{14}$	139	2.34543668	$h_1^{46}h_2^{17}h_3^8h_5^2h_{10}$
101	2.34525665	$h_1^{32}h_2^{11}h_3^4h_5h_{10}h_{14}$	140	2.35086471	$h_1^{44}h_2^{16}h_3^5h_5^3h_{10}h_{14}$
102	2.34570028	$h_1^{33}h_2^{12}h_3^5h_5^2h_{10}$	141	2.34703414	$h_1^{46}h_2^{17}h_3^7h_5^3h_{10}$
103	2.34332934	$h_1^{32}h_2^{11}h_3^5h_5h_{10}h_{14}$	142	2.34870654	$h_1^{45}h_2^{16}h_3^5h_5^3h_{10}h_{14}$
104	2.34245745	$h_1^{34}h_2^{12}h_3^5h_5^2h_{10}$	143	2.34960986	$h_1^{47}h_2^{17}h_3^7h_5^3h_{10}$
105	2.34317801	$h_1^{34}h_2^{13}h_3^5h_5^2h_{10}$	144	2.34803496	$h_1^{46}h_2^{18}h_3^6h_4h_5^2h_9h_{10}$
106	2.34321049	$h_1^{34}h_2^{12}h_3^4h_5h_{10}h_{14}$	145	2.34788883	$h_1^{46}h_2^{17}h_3^5h_5^3h_{10}h_{14}$
107	2.34273972	$h_1^{35}h_2^{13}h_3^5h_5^2h_{10}$	146	≥ 2.346298	
108	2.34392465	$h_1^{34}h_2^{12}h_3^5h_5h_{20}$	147	≥ 2.348766	
109	2.34122988	$h_1^{37}h_2^{13}h_3^5h_5h_{14}$	148	2.34928812	$h_1^{46}h_2^{16}h_3^5h_5^2h_{10}h_{14}h_{15}$
110	2.34487956	$h_1^{37}h_2^{16}h_3^5h_4h_5^2$	149	≥ 2.348586	
111	2.34410344	$h_1^{36}h_2^{13}h_3^6h_5^2h_{10}$	150	2.34889729	$h_1^{47}h_2^{16}h_3^5h_5^2h_{10}h_{14}h_{15}$
112	2.34602159	$h_1^{35}h_2^{12}h_3^4h_5^2h_{10}h_{14}$	151	2.35089600	$h_1^{48}h_2^{17}h_3^6h_5^3h_{10}h_{14}$
113	2.34567385	$h_1^{37}h_2^{13}h_3^6h_5^2h_{10}$	152	≥ 2.348660	
114	2.34294116	$h_1^{37}h_2^{14}h_3^4h_4h_5^2h_{14}$	153	≥ 2.350315	
115	2.34786857	$h_1^{36}h_2^{13}h_3^4h_5^2h_{10}h_{14}$	154	≥ 2.350246	

Table 1.3: Integer Chebyshev polynomials for the interval $[0,1]$ ($n = 155$ to 230)

n	$\Omega_n[0,1]^{-1}$	Polynomial	n	$\Omega_n[0,1]^{-1}$	Polynomial
155	≥ 2.347764		193	2.35154999	$h_1^{61}h_2^{23}h_3^7h_4h_5^3h_9h_{10}h_{14}$
156	2.35167936	$h_1^{50}h_2^{18}h_3^6h_5^3h_{10}h_{14}$	194	≥ 2.351954	
157	2.35298690	$h_1^{49}h_2^{17}h_3^6h_5^2h_{10}h_{14}h_{15}$	195	2.35216904	$h_1^{62}h_2^{25}h_3^7h_4h_5^3h_7h_{10}h_{14}$
158	≥ 2.350933		196	2.35592087	$h_1^{62}h_2^{22}h_3^8h_5^3h_{10}h_{14}h_{15}$
159	≥ 2.349342		197	2.35438234	$h_1^{62}h_2^{23}h_3^7h_4h_5^3h_{10}h_{14}h_{15}$
160	2.35262727	$h_1^{50}h_2^{18}h_3^6h_5^2h_{10}h_{14}h_{15}$	198	≥ 2.352664	
161	≥ 2.349767		199	2.35325444	$h_1^{63}h_2^{23}h_3^8h_5^3h_{10}h_{14}h_{15}$
162	2.35494623	$h_1^{51}h_2^{18}h_3^6h_5^2h_{10}h_{14}h_{15}$	200	≥ 2.352393	
163	2.35176777	$h_1^{51}h_2^{19}h_3^6h_5^2h_{10}h_{14}h_{15}$	201	2.35358771	$h_1^{64}h_2^{23}h_3^8h_5^3h_{10}h_{14}h_{15}$
164	2.35129166	$h_1^{52}h_2^{18}h_3^6h_5^2h_{10}h_{14}h_{15}$	202	≥ 2.352159	
165	2.35263602	$h_1^{52}h_2^{19}h_3^6h_5^2h_{10}h_{14}h_{15}$	203	2.35391172	$h_1^{65}h_2^{23}h_3^8h_5^3h_{10}h_{14}h_{15}$
166	2.35122436	$h_1^{52}h_2^{18}h_3^7h_5^2h_{10}h_{14}h_{15}$	204	2.35467398	$h_1^{65}h_2^{26}h_3^8h_4h_5^3h_7h_{10}h_{14}$
167	2.35230229	$h_1^{53}h_2^{19}h_3^6h_5^2h_{10}h_{14}h_{15}$	205	2.35445302	$h_1^{65}h_2^{23}h_3^9h_5^3h_{10}h_{14}h_{15}$
168	2.35100107	$h_1^{53}h_2^{22}h_3^6h_4h_5^2h_7h_{10}h_{14}$	206	2.35362212	$h_1^{66}h_2^{26}h_3^8h_4h_5^3h_7h_{10}h_{14}$
169	2.35305597	$h_1^{53}h_2^{19}h_3^7h_5^2h_{10}h_{14}h_{15}$	207	≥ 2.352999	
170	2.35259616	$h_1^{54}h_2^{22}h_3^6h_4h_5^2h_7h_{10}h_{14}$	208	2.35346196	$h_1^{66}h_2^{24}h_3^9h_5^3h_{10}h_{14}h_{15}$
171	2.35236551	$h_1^{54}h_2^{19}h_3^7h_5^2h_{10}h_{14}h_{15}$	209	2.35557985	$h_1^{67}h_2^{27}h_3^8h_4h_5^3h_7h_{10}h_{14}$
172	2.35218622	$h_1^{54}h_2^{20}h_3^7h_5^2h_{10}h_{14}h_{15}$	210	2.35434964	$h_1^{67}h_2^{24}h_3^9h_5^3h_{10}h_{14}h_{15}$
173	≥ 2.350620		211	2.35354763	$h_1^{68}h_2^{27}h_3^8h_4h_5^3h_7h_{10}h_{14}$
174	2.35380903	$h_1^{55}h_2^{20}h_3^7h_5^2h_{10}h_{14}h_{15}$	212	2.35370011	$h_1^{68}h_2^{24}h_3^9h_5^3h_{10}h_{14}h_{15}$
175	≥ 2.350605		213	≥ 2.353051	
176	2.35200375	$h_1^{56}h_2^{20}h_3^7h_5^2h_{10}h_{14}h_{15}$	214	2.35411349	$h_1^{68}h_2^{26}h_3^8h_4h_5^3h_{10}h_{14}h_{15}$
177	≥ 2.350977		215	≥ 2.352712	
178	2.35333824	$h_1^{56}h_2^{20}h_3^8h_5^2h_{10}h_{14}h_{15}$	216	2.35401353	$h_1^{69}h_2^{26}h_3^{10}h_4h_5^3h_6h_{10}h_{14}$
179	2.35188343	$h_1^{57}h_2^{21}h_3^7h_5^2h_{10}h_{14}h_{15}$	217	2.35397104	$h_1^{69}h_2^{27}h_3^8h_4h_5^3h_{10}h_{14}h_{15}$
180	≥ 2.350799		218	2.35423461	$h_1^{70}h_2^{28}h_3^9h_4h_5^3h_7h_{10}h_{14}$
181	2.35246637	$h_1^{57}h_2^{21}h_3^8h_5^2h_{10}h_{14}h_{15}$	219	2.35439365	$h_1^{70}h_2^{25}h_3^{10}h_5^3h_{10}h_{14}h_{15}$
182	2.35247684	$h_1^{57}h_2^{20}h_3^7h_5^3h_{10}h_{14}h_{15}$	220	2.35450391	$h_1^{69}h_2^{24}h_3^8h_5^3h_{10}h_{14}h_{22}$
183	2.35298504	$h_1^{58}h_2^{21}h_3^8h_5^2h_{10}h_{14}h_{15}$	221	2.35480670	$h_1^{71}h_2^{25}h_3^8h_5^3h_8h_{10}h_{14}h_{15}$
184	≥ 2.350104		222	≥ 2.353061	
185	2.35167425	$h_1^{58}h_2^{21}h_3^7h_5^3h_{10}h_{14}h_{15}$	223	2.35353721	$h_1^{72}h_2^{29}h_3^9h_4h_5^3h_7h_{10}h_{14}$
186	≥ 2.351068		224	≥ 2.352907	
187	2.35319807	$h_1^{59}h_2^{21}h_3^7h_5^3h_{10}h_{14}h_{15}$	225	2.35456816	$h_1^{71}h_2^{25}h_3^9h_5^3h_{10}h_{14}h_{21}$
188	2.35208180	$h_1^{60}h_2^{22}h_3^8h_5^2h_{10}h_{14}h_{15}$	226	2.35462498	$h_1^{72}h_2^{28}h_3^9h_4h_5^3h_{10}h_{14}h_{15}$
189	2.35333286	$h_1^{60}h_2^{21}h_3^7h_5^3h_{10}h_{14}h_{15}$	227	2.35559409	$h_1^{72}h_2^{25}h_3^9h_5^3h_{10}h_{14}h_{21}$
190	2.35280474	$h_1^{59}h_2^{20}h_3^7h_5^2h_{10}h_{14}h_{23}$	228	2.35358328	$h_1^{72}h_2^{26}h_3^9h_5^3h_{10}h_{14}h_{21}$
191	≥ 2.352230		229	≥ 2.35396	
192	2.35296330	$h_1^{61}h_2^{24}h_3^7h_4h_5^3h_7h_{10}h_{14}$	230	2.35487542	$h_1^{73}h_2^{26}h_3^9h_5^3h_{10}h_{14}h_{21}$

$$\begin{aligned} r_3(x) &= 890381x^{16} - 7123048x^{15} + 26258899x^{14} - 59158953x^{13} + 91079142x^{12} \\ &\quad -101507147x^{11} + 84628171x^{10} - 53783834x^9 + 26304860x^8 - 9923055x^7 \\ &\quad +2874524x^6 - 632029x^5 + 103290x^4 - 12116x^3 + 961x^2 - 46x + 1 \\ r_4(x) &= 5287361x^{18} - 47586249x^{17} + 199139568x^{16} - 514494900x^{15} + 918909397x^{14} \\ &\quad -1203880783x^{13} + 1198032039x^{12} - 925178633x^{11} + 561591777x^{10} \\ &\quad -269728267x^9 + 102663046x^8 - 30867601x^7 + 7271422x^6 - 1322862x^5 \\ &\quad +181649x^4 - 18155x^3 + 1243x^2 - 52x + 1 \end{aligned}$$

1.4.1 The Schur-Siegel-Smyth trace problem

The mean trace of an algebraic integer α of degree d is

$$\mathrm{tr}(\alpha) = \frac{1}{d}\sum_{i=1}^{d} \alpha_i$$

where $\{\alpha_i\}_{i=1}^{d}$ represents the set of conjugates of α. If we define $\mathcal{T}$ as

$$\mathcal{T} = \{\mathrm{tr}(\alpha) : \alpha \text{ is a totally positive agebraic integer}\},$$

then it is an open question as to whether 2 is the smallest limit point of this set. Recent attacks on this problem come in the form of showing that for all but finitely many totally positive algebraic integers α, $\mathrm{tr}(\alpha) > 2-\delta$ for some $\delta > 0$ and use the method of Smyth [29]. If it can be shown that $x - a\log(|q(x)|) \geq b$ for all $x > 0$ where $q(x)$ is a polynomial with integer coefficients and $a > 0$, then for a totally positive algebraic integer α of degree d with conjugates $\{\alpha_i\}_{i=1}^{d}$ and minimal polynomial $p(x)$,

$$\frac{1}{d}\left(\sum_{i=1}^{d} \alpha_i - a\log(|q(\alpha_i)|)\right) = \mathrm{tr}(\alpha) - \frac{a}{d}\log(|\mathrm{resultant}(p(x), q(x))|) \geq b.$$

If $q(\alpha) \neq 0$, then the resultant in this last expression is a nonzero integer which leads to $\mathrm{tr}(\alpha) \geq b$ for all totally positive algebraic integers except those with minimal polynomial dividing $q(x)$. By constructing $q(x)$ from polynomials $q_i(x)$ with small mean trace and selecting rational a_i in an attempt to maximize the minimum value of $x - a_i\log(|q_i(x)|)$ on $x > 0$, one can try to increase the value of b. An account of recent results can be found in [1], which also contains Serre's proof that this method cannot increase the constant b beyond the value 1.898302. The interest here lies in the polynomials $q_i(x)$ used to construct

$q(x)$. Since a totally positive algebraic integer α has all positive roots, if $q_i(x)$ is the minimal polynomial of α and is of degree d, then

$$s_i(x) = (x(1-x))^d q_i\left(\frac{1-4x(1-x)}{x(1-x)}\right)$$

is a polynomial of degree $2d$ with all roots in the interval $[0,1]$ satisfying the symmetry condition $s_i(x) = s_i(1-x)$. Since the lead coefficients tend not to be large after the transformation for polynomials of small mean trace, this gives an alternate source of polynomials to use when attempting to reduce the upper bound on $\Omega[0,1]$. The following polynomials $q_i(x)$ from the Schur-Siegel-Smyth trace problem gave rise to additional polynomials $s_i(x)$ that were found to be useful. With the exception of the last polynomial $q_{28}(x)$, they are all minimal polynomials of totally positive algebraic integers.

$$\begin{aligned}
q_1(x) &= x^3 - 6x^2 + 9x - 3 \\
q_2(x) &= x^4 - 8x^3 + 16x^2 - 9x + 1 \\
q_3(x) &= x^5 - 9x^4 + 26x^3 - 29x^2 + 11x - 1 \\
q_4(x) &= x^5 - 9x^4 + 27x^3 - 31x^2 + 12x - 1 \\
q_5(x) &= x^5 - 9x^4 + 27x^3 - 32x^2 + 13x - 1 \\
q_6(x) &= x^6 - 11x^5 + 42x^4 - 68x^3 + 46x^2 - 12x + 1 \\
q_7(x) &= x^6 - 11x^5 + 42x^4 - 68x^3 + 47x^2 - 13x + 1 \\
q_8(x) &= x^6 - 11x^5 + 43x^4 - 72x^3 + 51x^2 - 14x + 1 \\
q_9(x) &= x^7 - 13x^6 + 61x^5 - 131x^4 + 136x^3 - 66x^2 + 14x - 1 \\
q_{10}(x) &= x^7 - 13x^6 + 61x^5 - 133x^4 + 142x^3 - 71x^2 + 15x - 1 \\
q_{11}(x) &= x^7 - 13x^6 + 62x^5 - 135x^4 + 140x^3 - 67x^2 + 14x - 1 \\
q_{12}(x) &= x^7 - 13x^6 + 62x^5 - 136x^4 + 144x^3 - 71x^2 + 15x - 1 \\
q_{13}(x) &= x^7 - 13x^6 + 62x^5 - 137x^4 + 146x^3 - 72x^2 + 15x - 1 \\
q_{14}(x) &= x^7 - 13x^6 + 62x^5 - 137x^4 + 147x^3 - 73x^2 + 15x - 1 \\
q_{15}(x) &= x^7 - 13x^6 + 62x^5 - 137x^4 + 148x^3 - 75x^2 + 16x - 1 \\
q_{16}(x) &= x^7 - 13x^6 + 62x^5 - 138x^4 + 150x^3 - 76x^2 + 16x - 1 \\
q_{17}(x) &= x^7 - 13x^6 + 63x^5 - 143x^4 + 157x^3 - 78x^2 + 16x - 1 \\
q_{18}(x) &= x^7 - 13x^6 + 63x^5 - 143x^4 + 158x^3 - 80x^2 + 16x - 1
\end{aligned}$$

$$
\begin{aligned}
q_{19}(x) &= x^7 - 13x^6 + 63x^5 - 143x^4 + 159x^3 - 82x^2 + 17x - 1 \\
q_{20}(x) &= x^7 - 13x^6 + 63x^5 - 144x^4 + 160x^3 - 80x^2 + 16x - 1 \\
q_{21}(x) &= x^8 - 15x^7 + 83x^6 - 220x^5 + 303x^4 - 220x^3 + 83x^2 - 15x + 1 \\
q_{22}(x) &= x^8 - 15x^7 + 84x^6 - 228x^5 + 323x^4 - 240x^3 + 91x^2 - 16x + 1 \\
q_{23}(x) &= x^8 - 15x^7 + 85x^6 - 233x^5 + 333x^4 - 250x^3 + 96x^2 - 17x + 1 \\
q_{24}(x) &= x^8 - 15x^7 + 86x^6 - 241x^5 + 353x^4 - 270x^3 + 104x^2 - 18x + 1 \\
q_{25}(x) &= x^{10} - 18x^9 + 134x^8 - 538x^7 + 1273x^6 - 1822x^5 + 1560x^4 \\
&\quad -766x^3 + 200x^2 - 24x + 1 \\
q_{26}(x) &= x^{10} - 18x^9 + 135x^8 - 549x^7 + 1320x^6 - 1920x^5 + 1662x^4 \\
&\quad -813x^3 + 206x^2 - 24x + 1 \\
q_{27}(x) &= x^{12} - 22x^{11} + 204x^{10} - 1050x^9 + 3322x^8 - 6752x^7 + 8944x^6 \\
&\quad -7677x^5 + 4177x^4 - 1388x^3 + 265x^2 - 26x + 1 \\
q_{28}(x) &= x^6 - 10x^5 + 35x^4 - 52x^3 + 33x^2 - 9x + 1
\end{aligned}
$$

1.4.2 Fixed points of the function $u(x)$ and the Gorshkov-Wirsing polynomials

As mentioned in Section 1.2.3, the Gorshkov-Wirsing polynomials are defined iteratively as

$$
\begin{aligned}
q_0(x) &= 2x - 1 \\
q_1(x) &= 5x^2 - 5x + 1 \\
q_{n+1}(x) &= q_n^2(x) + q_n(x)q_{n-1}^2(x) - q_{n-1}^4(x)
\end{aligned}
$$

and satisfy the relationship $q_{k+1}(x) = (-1 + 3x - 3x^2)^{2^k} q_k(u(x))$ where $u(x)$ is the rational function

$$u(x) = \frac{q_0^2(x) - q_1(x)}{2q_0^2(x) - q_1(x)} = \frac{x(1-x)}{1 - 3x(1-x)}.$$

The Gorshkov-Wirsing polynomials $q_k(x)$ for $k \leq 5$ are useful in reducing the upper bound on $\Omega[0,1]$. Additionally, since $u(x)$ increases from 0 to 1 on $[0, 1/2]$, decreases form 1 to 0 on $[1/2, 1]$, and satisfies the property $u(x) = u(1-x)$, if $g_i(x)$ is a symmetric polynomial of degree d with all roots in the interval $[0,1]$ then so is $t_i(x) = (3x^2 - 3x + 1)^d g_i(u(x))$. Applying this to the polynomials of the previous sections resulted in finding the polynomial

$$t_1(x) = (3x^2 - 3x + 1)^{14} s_9(u(x))$$

which also proved to be useful.

As a further source of polynomials, we can consider the fixed points of $u(x)$. If we let $u^1(x) = u(x)$ and define $u^{(n+1)}(x)$ by

$$u^{(n+1)}(x) = u(u^n(x)) = \frac{u^n(x)(1-u^n(x))}{1-3u^n(x)(1-u^n(x))},$$

then an easy induction [25] shows

$$u^{(n)}(x) = \frac{q_{n-1}^2(x) - q_n(x)}{2q_{n-1}^2(x) - q_n(x)} \quad \text{for } n \geq 1.$$

The key to the induction is to note that $\left(q_{n-1}^2(x) - q_n(x)\right) q_{n-1}^2(x) = q_n^2(x) - q_{n+1}(x)$. Now, since $u(x)$ increases from 0 to 1 on $[0, 1/2]$, and decreases from 1 to 0 on $[1/2, 1]$, $u(x)$ maps the interval $[0, 1]$ onto itself twice, and $u^n(x)$ maps the unit interval onto itself 2^n times. This gives 2^n points of intersection between the curves $y = u^n(x)$ and $y = x$ and accounts for 2^n zeros of the function $u^n(x) - x$ in the interval $[0, 1]$. As the numerator

$$m_n(x) = (q_{n-1}^2(x) - q_n(x)) - x(2q_{n-1}^2(x) - q_n(x))$$

of $u^n(x) - x$ is of degree $2^n + 1$ and is divisible by x^2, at least for $n \leq 13$, $m_n(x)$ has a double zero at $x = 0$, and $2^n - 1$ simple zeros in the interval $(0, 1)$. The irreducible factors of $m_n(x)$ provide a source of irreducible polynomials with integer coefficients having all their roots in the interval $[0, 1]$. For $1 \leq n \leq 5$, the irreducible factors are x, $3x - 2$, $7x^2 - 8x + 2$, $13x^3 - 19x^2 + 8x - 1$, $31x^4 - 61x^3 + 41x^2 - 11x + 1$, and

$$\begin{aligned} v_1(x) \;=\; & 414157x^{15} - 3082633x^{14} + 10506317x^{13} - 21730488x^{12} + 30475127x^{11} \\ & -30666643x^{10} + 22852753x^9 - 12829629x^8 + 5465583x^7 - 1765190x^6 \\ & +428258x^5 - 76576x^4 + 9759x^3 - 836x^2 + 43x - 1. \end{aligned}$$

Noting that four of these polynomials are factors of the symmetric pairs $h_1(x)$, $h_8(x)$, $h_{10}(x)$, and $h_{15}(x)$ leads to the consideration of the polynomials $(3x-1)(3x-2)$ and $v_1(x)v_1(1-x)$. The polynomial $v_1(x)v_1(1-x)$ proved to be useful in reducing the upper bound for $\Omega[0, 1]$.

As a side note, it is worth mentioning that if we take the irreducible factors $v(x)$ of $m_n(x)$ for larger values of n and consider the polynomials $(x+1)^d v(1/(x+1))$ where d is the degree of $v(x)$, we get the minimal polynomials of totally positive algebraic integers with trace equal to $2d - k$ for values of k as large as 6. This may be worthy of further study.

The first few terms of these polynomials are presented below.

$$\begin{aligned}
x^{15} - 28x^{14} + 339x^{13} - \ldots \qquad & (n = 5, 2d-2) \\
x^{24} - 46x^{23} + 964x^{22} - \ldots \qquad & (n = 6, 2d-2) \\
x^{30} - 58x^{29} + 1559x^{28} - \ldots \qquad & (n = 6, 2d-2) \\
x^{63} - 123x^{62} + 7290x^{61} - \ldots \qquad & (n = 7, 2d-3) \\
x^{112} - 221x^{111} + 23910x^{110} - \ldots \qquad & (n = 8, 2d-3) \\
x^{252} - 501x^{251} + 124296x^{250} - \ldots \qquad & (n = 9, 2d-3) \\
x^{480} - 956x^{479} + 454648x^{478} - \ldots \qquad & (n = 10, 2d-4) \\
x^{510} - 1016x^{509} + 513652x^{508} - \ldots \qquad & (n = 10, 2d-4) \\
x^{1023} - 2041x^{1022} + 2077818x^{1021} - \ldots \qquad & (n = 11, 2d-5) \\
x^{1980} - 3956x^{1979} + 7815176x^{1978} - \ldots \qquad & (n = 12, 2d-4) \\
x^{2040} - 4076x^{2039} + 8296782x^{2038} - \ldots \qquad & (n = 12, 2d-4) \\
x^{4095} - 8184x^{4094} + 33468579x^{4093} - \ldots \qquad & (n = 13, 2d-6)
\end{aligned}$$

1.4.3 An improved upper bound on $\Omega[0,1]$

To improve the upper bound on $\Omega[0,1]$, the standard method referred to in Section 1.2.3 was used. Given a set of polynomials $\{p_i(x)\}$, we can consider polynomials of the form $\prod p_i(x)^{m_i}$ and attempt to find values for the m_i that minimize the supremum norm of this product on the interval $[0,1]$. This is done by taking the natural logarithm of the absolute value of the above product and evaluating it at multiple points $x_j \in [0,1]$ which leads to the system of constraints

$$\left\{ \sum \frac{m_i}{2} \ln\left(p_i(x_j)^2\right) < c \right\}$$

where e^c is an upper bound on $\left\| \prod p_i(x)^{m_i} \right\|_{[0,1]}$. Adding the additional constraints $\sum m_i = 1$ and $m_i \geq 0$ for each m_i, the simplex algorithm can then be used to minimize c. Multiplying the returned values m_i by a large power of 10 and rounding to the nearest integer results in a polynomial $Q(x)$ of very large degree with relatively small supremum norm on $[0,1]$. The initial result can be improved upon by adding additional control points $x_j \in [0,1]$ at which $Q(x)$ attains local extrema larger than the value of c returned by the simplex algorithm, and then repeating the procedure.

Applying this procedure with the polynomials $h_i(x)$, $q_i(x)$, $r_i(x)$, $s_i(x)$, $t_i(x)$, and $v_i(x)$ from the preceding sections resulted in the polynomial

$$\begin{aligned} Q(x) \;=\; & h_1(x)^{3136689596} h_2(x)^{1117706483} h_3(x)^{378371211} h_4(x)^{16801958} h_5(x)^{131164392} \\ & \cdot h_6(x)^{8943338} h_7(x)^{1574854} h_8(x)^{1117548} h_{10}(x)^{33466589} h_{12}(x)^{1377045} h_{14}(x)^{31611818} \\ & \cdot h_{15}(x)^{17250649} h_{16}(x)^{1733381} h_{17}(x)^{3054552} h_{21}(x)^{2793536} h_{22}(x)^{1603001} q_5(x)^{269851} \\ & \cdot r_1(x)^{288868} r_2^{339997} r_3(x)^{434242} r_4(x)^{653128} s_1(x)^{1994437} s_2(x)^{9371} s_3(x)^{4232059} \\ & \cdot s_4(x)^{1798114} s_5(x)^{271961} s_6(x)^{1366536} s_7(x)^{1099530} s_8(x)^{93315} s_9(x)^{1035078} \\ & \cdot s_{10}(x)^{1368043} s_{11}(x)^{248638} s_{12}(x)^{648325} s_{14}(x)^{5099012} s_{15}(x)^{647549} s_{16}(x)^{993218} \\ & \cdot s_{17}(x)^{2195681} s_{18}(x)^{66269} s_{19}(x)^{289394} s_{20}(x)^{2545157} s_{21}(x)^{4432963} s_{22}(x)^{110295} \\ & \cdot s_{23}(x)^{1525872} s_{24}(x)^{915404} s_{25}(x)^{423575} s_{26}(x)^{386416} s_{27}(x)^{1317066} s_{28}(x)^{174142} \\ & \cdot t_1(x)^{126275} \left(v_1(x) v_1(1-x)\right)^{340742} \end{aligned}$$

of degree 10000000027 for which $\|Q(x)\|_{[0,1]}^{1/10000000027} = (2.36482727\ldots)^{-1}$. By Lemma 1.3, this gives the best known upper bound on $\Omega[0,1]$ of

$$\Omega[0,1] \leq \frac{1}{2.36482727}.$$

It should be pointed out that finding $\|Q(x)\|_{[0,1]}$ is not as daunting of a task as it may initially appear. To find the extreme values of a polynomial of the form $\prod p_i(x)^{m_i}$ on $[0,1]$ that is divisible by $x(1-x)$, we need only evaluate $\exp\left(\sum m_i \ln(p_i(x))\right)$ at the roots of $\sum \left(\left(\prod_{i \neq j} p_j(x)\right) m_i p_i'(x)\right)$ that lie in the interior of the interval $[0,1]$.

Chapter 2

Integer Relation Algorithms

To take advantage of symmetry conditions when extending the method used to compute integer Chebyshev polynomials to the bivariate case, it is useful to have a method for constructing a basis for the lattice of all simultaneous integer relations satisfied by a given set of input vectors. The method used to construct this basis is an extension of a standard integer relation algorithm, three of which are presented in this chapter. The presentation of these standard integer relation algorithms follows the author's earlier work [24].

Definition 2.1 *An integer relation is said to exist between the numbers $x_1, x_2, \ldots, x_n$ if there exist integers $a_1, a_2, \ldots, a_n$, not all zero, such that $\sum_{i=1}^{n} a_i x_i = 0$. For the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, a nonzero vector $\mathbf{a} \in \mathbb{Z}^n$ is an integer relation for $\mathbf{x}$ if $\mathbf{a} \cdot \mathbf{x} = 0$.*

Definition 2.2 *The lattice L spanned by the n linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ is the set of vectors $L = \{\sum_{i=1}^{n} r_i \mathbf{b}_i : r_i \in \mathbb{Z}, i = 1, 2, \ldots, n\}$. The vectors $\mathbf{b}_i$ are said to form a basis for L.*

The following fact is useful to keep in mind when working with lattices. It lets us know that each successive set of vectors we consider form a basis for the original lattice.

Theorem 2.1 *If $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ form a basis for the lattice L, then the vectors $\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_n$ form a basis for L if and only if there exists an $n \times n$ invertible matrix A with integer coefficients and determinant ± 1 such that $B' = BA$ where $B = [\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n]$ (the matrix for which the ith column is the vector $\mathbf{b}_i$) and $B' = [\mathbf{b}'_1, \mathbf{b}'_2, \cdots, \mathbf{b}'_n]$.*

Proof:($\Rightarrow$) Suppose $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ and $\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_n$ both form a basis for L. Then as each $\mathbf{b}'_i \in L$, each $\mathbf{b}'_i$ can be expressed as an integer combination of the $\mathbf{b}_i$'s. It follows that

$B' = BA$ for some $n \times n$ matrix A with integer entries. Similarly, as each $\mathbf{b}_i \in L$, $B = B'C$ for some $n \times n$ matrix C with integer entries. As $B = B'C = BAC$ and $B' = BA = B'CA$, $AC = CA = I$. We see that $B' = BA$ for some invertible matrix A with integer entries and determinant ± 1.
($\Leftarrow$) Suppose $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ form a basis for L and A is an invertible $n \times n$ matrix with integer entries and determinant ± 1 such that $B' = BA$. Clearly, each column vector $\mathbf{b}'_i$ of B' can be written as an integer combination of the $\mathbf{b}_i$'s. Now as A is invertible, has integer entries and determinant ± 1, we see by the formula $A^{-1} = \frac{1}{\det(A)}\text{adj}(A)$ that A^{-1} also has integer entries and determinant ± 1. Thus we see $B = B'A^{-1}$ and so each $\mathbf{b}_i$ can be written as an integer combination of the $\mathbf{b}'_i$'s. The vectors $\mathbf{b}'_i$ also form a basis for the lattice L.

□

Integer Relation Algorithms have also been called Multidimensional Continued Fraction Algorithms and Generalized Euclidean Algorithms. As one can show that applying the Continued Fraction Algorithm to the value r_0/r_1 is equivalent to running the Extended Euclidean Algorithm on the real numbers r_0 and r_1, and as both can be used to determine either a shortest integer relation for the vector $\mathbf{x} = [r_0, r_1]$ or a lower bound on the norm of any possible integer relation for $\mathbf{x}$, this suggests that these names are appropriate. In fact, the following method for showing that these two algorithms are equivalent and how they can be used to find a lower bound on the norm of any possible integer relation for $\mathbf{x}$ also suggests how we should proceed in developing integer relation algorithms for $n \geq 3$. Details of both algorithms are presented briefly for the sake of notation.

The Extended Euclidean Algorithm is applied to the values r_0 and r_1 as follows. Begin by setting $s_0 = 1$, $t_0 = 0$, $s_1 = 0$, $t_1 = 1$ and then repeat the following process starting with k=1:

1. Set $Q_k = \lfloor r_{k-1}/r_k \rfloor$ so that $r_{k-1} = Q_k r_k + r_{k+1}$ where $0 \leq r_{k+1} < r_k$
2. Set $s_{k+1} = s_{k-1} - Q_k s_k$ and $t_{k+1} = t_{k-1} - Q_k t_k$.

At each stage we have

$$r_k = s_k r_0 + t_k r_1.$$

This obviously holds for $k = 0$ and $k = 1$ and so by induction, as

$$r_k = r_{k-2} - Q_{k-1} r_{k-1} = (s_{k-2} r_0 + t_{k-2} r_1) - Q_{k-1}(s_{k-1} r_0 + t_{k-1} r_1)$$

$$= (s_{k-2} - Q_{k-1}s_{k-1})r_0 + (t_{k-2} - Q_{k-1}t_{k-1})r_1 = s_k r_0 + t_k r_1,$$

it holds for all k. As $r_k < \frac{1}{2}r_{k-2}$ for $k \geq 3$, either this algorithm terminates with $r_k = s_k r_0 + t_k r_1 = 0$ or it constructs an infinite sequence of pairs (s_k, t_k) such that $s_k r_0 + t_k r_1$ decreases monotonically to zero.

The Continued fraction algorithm is applied to the value r_0/r_1 in the following manner. Begin by setting $\alpha_1 = r_0/r_1$, $a_1 = \lfloor r_0/r_1 \rfloor$, $p_0 = 0$, $q_0 = 1$, $p_1 = 1$, $q_1 = 0$ and then repeat the following process beginning with $k = 2$:

1. Set $\alpha_k = \frac{1}{\alpha_{k-1} - a_{k-1}}$ and $a_k = \lfloor \alpha_k \rfloor$.
2. Set $p_k = a_k p_{k-1} + p_{k-2}$ and $q_k = a_k q_{k-1} + q_{k-2}$.

The *nth* convergent of the continued fraction is p_{n+2}/q_{n+2}. Note that this is different from what is usually given, the difference in the subscripts has been introduced to simplify what follows. Either this algorithm terminates with $\alpha_k \in \mathbb{Z}$, in which case $r_0/r_1 = p_k/q_k$, or this algorithm constructs an infinite sequence of convergents that converge to r_0/r_1.

To show the equivalence of the Euclidean and Continued Fraction algorithms, we consider the lines $\mathbb{R}\mathbf{x} = \mathbb{R}[r_0, r_1]^T$ and $\mathbf{x}^\perp = \mathbb{R}[r_1, -r_0]^T$. The task is to construct a sequence of bases for the lattice $\mathbb{Z}^2$ that converge to $\mathbf{x}^\perp$ by examining the projections of these basis vectors onto $\mathbb{R}\mathbf{x}$. Starting with the standard basis for $\mathbb{Z}^2$, we let $\mathbf{b}_1^{(1)} = [1, 0]^T$ and $\mathbf{b}_2^{(1)} = [0, 1]^T$. If we set $B^{(1)} = [\mathbf{b}_1^{(1)}, \mathbf{b}_2^{(1)}]$, the matrix with column vectors $\mathbf{b}_1^{(1)}$ and $\mathbf{b}_2^{(1)}$, then

$$B^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} s_0 & s_1 \\ t_0 & t_1 \end{bmatrix}$$

and

$$A = \left(B^{(1)}\right)^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} p_1 & q_1 \\ p_0 & q_0 \end{bmatrix}.$$

The following properties hold for $k = 1$:

1. $\alpha_k = r_{k-1}/r_k$ and so $Q_k = a_k$
2. $\mathbf{b}_1^{(k)} = [s_{k-1}, t_{k-1}]^T$ and $\mathbf{b}_2^{(k)} = [s_k, t_k]^T$ are the column vectors of $B^{(k)}$
3. $(\mathbf{a}_1^{(k)})^T = [p_k, q_k]$ and $(\mathbf{a}_2^{(k)})^T = [p_{k-1}, q_{k-1}]$ are the row vectors of $A^{(k)} = \left(B^{(k)}\right)^{-1}$

Now suppose these three conditions hold for some k. Then $\mathbf{b}_1^{(k)} = [s_{k-1}, t_{k-1}]^T$ and $\mathbf{b}_2^{(k)} = [s_k, t_k]^T$ and so as $r_m = s_m r_0 + t_m r_1$, their projections onto $\mathbb{R}\mathbf{x}$ have norms equal to

$$\frac{1}{\sqrt{r_0^2 + r_1^2}} r_{k-1} \quad \text{and} \quad \frac{1}{\sqrt{r_0^2 + r_1^2}} r_k.$$

As our goal is to construct a sequence of bases for $\mathbb{Z}^2$ that converge to $\mathbf{x}^\perp$, we set $\mathbf{b}_1^{(k+1)} = \mathbf{b}_2^{(k)}$ and $\mathbf{b}_2^{(k+1)} = \mathbf{b}_1^{(k)} - Q_k \mathbf{b}_2^{(k)}$. As $r_{k+1} = r_{k-1} - Q_k r_k$, this results in a new basis for $\mathbb{Z}^2$ with projections onto $\mathbb{R}\mathbf{x}$ having norms equal to

$$\frac{1}{\sqrt{r_0^2 + r_1^2}} r_k \quad \text{and} \quad \frac{1}{\sqrt{r_0^2 + r_1^2}} r_{k+1}.$$

To maintain $B^{(k+1)} = [\mathbf{b}_1^{(k+1)}, \mathbf{b}_2^{(k+1)}]$ with each iteration, we must update the matrices $B^{(k)}$ and $A^{(k)} = \left(B^{(k)}\right)^{-1}$ as follows:

$$B^{(k+1)} = \begin{bmatrix} s_{k-1} & s_k \\ t_{k-1} & t_k \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -Q_k \end{bmatrix} = \begin{bmatrix} s_k & s_{k+1} \\ t_k & t_{k+1} \end{bmatrix}$$

$$A^{(k+1)} = \left(B^{(k+1)}\right)^{-1} = \begin{bmatrix} a_k & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_k & q_k \\ p_{k-1} & q_{k-1} \end{bmatrix} = \begin{bmatrix} p_{k+1} & q_{k+1} \\ p_k & q_k \end{bmatrix}.$$

As $r_{k+1} = (r_{k-1}/r_k - Q_k) r_k = r_k/\alpha_{k+1}$, we also have $\alpha_{k+1} = r_k/r_{k+1}$ and so by induction the above three conditions hold for all k. By forcing the vectors $\mathbf{b}_i^{(k)}$ to converge to $\mathbf{x}^\perp$ in this fashion, the relation between the values p_k and q_k from the Continued Fraction algorithm and the values s_k and t_k from the Extended Euclidean algorithm becomes clear.

Now for each k, $[s_k, t_k]^T$ is orthogonal to $[p_k, q_k]^T$ and so $p_k/q_k = -t_k/s_k$. As it is easily shown that $1 = \gcd(p_k, q_k) = \gcd(s_k, t_k)$ by showing both $p_k q_{k+1} - p_{k+1} q_k$ and $s_k t_{k+1} - s_{k+1} t_k$ equal ± 1, it follows that either $p_k = t_k$ and $q_k = -s_k$ or $p_k = -t_k$ and $q_k = s_k$. This, coupled with the fact that $\alpha_k = r_{k-1}/r_k$ and $Q_k = a_k$ for all k shows that each step of the Extended Euclidean Algorithm coincides with each step of the Continued Fraction Algorithm. The two algorithms are equivalent.

Viewed in this fashion, if $r_k \neq 0$ then it is straightforward to give a lower bound on the size of any possible integer relation for $\mathbf{x}$. As $\mathbf{a}_1^{(k)} = [p_k, q_k]^T$ and $r_k = s_k r_0 + t_k r_1$,

$$\left\| \text{proj}_{\mathbf{x}^\perp} \mathbf{a}_1^{(k)} \right\| = \left| \frac{p_k r_1 - q_k r_0}{\sqrt{r_0^2 + r_1^2}} \right| = \left| \frac{\pm(t_k r_1 + s_k r_0)}{\sqrt{r_0^2 + r_1^2}} \right| = \frac{|r_k|}{\sqrt{r_0^2 + r_1^2}} \neq 0.$$

Now if $\mathbf{m}$ is an integer relation for $\mathbf{x}$, then $\mathbf{m}$ lies in $\mathbf{x}^{\perp}$ and so it follows that $\left|\mathbf{a}_1^{(k)} \cdot \mathbf{m}\right| = \left\|\text{proj}_{\mathbf{x}^{\perp}} \mathbf{a}_1^{(k)}\right\| \|\mathbf{m}\|$ is a nonzero integer. It follows that if $r_k \neq 0$, then

$$\|\mathbf{m}\| \geq \frac{1}{\left\|\text{proj}_{\mathbf{x}^{\perp}} \mathbf{a}_1^{(k)}\right\|} = \frac{\sqrt{r_0^2 + r_1^2}}{|r_k|}.$$

In the event that $r_k = 0$, $[s_k, t_k]^T$ is an integer relation for $\mathbf{x}$ and so $r_0/r_1 = -t_k/s_k = p_k/q_k \in \mathbb{Q}$. Since any other integer relation $[u, v]$ must also satisfy $-v/u = r_0/r_1$ and $1 = \gcd(s_k, t_k) = \gcd(p_k, q_k)$, we see that in this case we have found a smallest possible integer relation for $\mathbf{x}$.

Although the Euclidean and continued fraction algorithms solve the problem of finding integer relations for the vector $[x_1, x_2, \ldots, x_n]$ when $n = 2$, until recently there were no known polynomial time algorithms that solved the problem for $n \geq 3$. A breakthrough was made in 1977 with Ferguson and Forcade's *Generalized Euclidean Algorithm* [13][14], a recursive algorithm that was guaranteed to find an integer relation when one existed. Following this, a number of non-recursive algorithms were developed, among which are the PSLQ algorithm, the HJLS algorithm, and a method based on the LLL algorithm that shall be covered in this chapter. Following their presentations, it will be shown that the HJLS algorithm can be viewed as a special case of the later PSLQ algorithm with the parameter $\gamma = \sqrt{2}$, both of which stem from extending the ideas presented earlier.

The above method of showing the equivalence of the Euclidean and Continued Fraction algorithms and how to apply them to finding integer relations between the values $x_1, x_2, \ldots, x_n$ for $n = 2$ indicates how to proceed for $n \geq 3$. Given the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, we would like to let $\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \ldots, \mathbf{b}_n^{(k)}$ be a basis for $\mathbb{Z}^n$ and then with each successive iteration, choose a new basis in such a way that an upper bound on the values $\left\|\text{proj}_{\mathbf{x}} \mathbf{b}_i^{(k)}\right\|$ decreases. However, as it is currently not known how to select a new basis in such a way that guarantees finding a small integer relation when one exists by looking only at the values $\left\|\text{proj}_{\mathbf{x}} \mathbf{b}_i^{(k)}\right\|$, we must do something different. Noting that the bound on the size of a smallest possible integer relation in the case $n = 2$ came from looking at the projection of the vectors $[p_k, q_k]^T$ onto $\mathbf{x}^{\perp}$ gives the correct course of action. We will construct a sequence of bases $\{(\mathbf{a}_1^{(k)}, \mathbf{a}_2^{(k)}, \ldots, \mathbf{a}_n^{(k)})\}$ for the lattice $\mathbb{Z}^n$ which converge to the line $\mathbb{R}\mathbf{x}$ by examining their projections onto $\mathbf{x}^{\perp}$. We will let $B^{(k)} = [\mathbf{b}_1^{(k)}, \mathbf{b}_2^{(k)}, \ldots, \mathbf{b}_n^{(k)}]$, $A^{(k)} = \left(B^{(k)}\right)^{-1}$, and let $\mathbf{a}_1^{(k)}, \mathbf{a}_2^{(k)}, \ldots, \mathbf{a}_n^{(k)}$ be the column vectors of $(A^{(k)})^T$. By choosing a new basis $\mathbf{a}_1^{(k)}, \mathbf{a}_2^{(k)}, \ldots, \mathbf{a}_n^{(k)}$ for $\mathbb{Z}^n$ in such a way that the projections of the $\mathbf{a}_i^{(k)}$ onto $\mathbf{x}^{\perp}$

tend to zero, we shall indirectly force the vectors $\mathbf{b}_i^{(k)}$ to converge to $\mathbf{x}^\perp$ and raise a lower bound on the norm of any possible integer relation for $\mathbf{x}$. If $\mathbf{x}$ has integer relations, then this bound can only be made so large before one of the $\mathbf{b}_i^{(k)}$ lies in $\mathbf{x}^\perp$ and an integer relation is found. This is the general idea behind both the PSLQ and HJLS algorithms that are presented later.

2.1 The LLL algorithm

It is often desirable to find a basis for a lattice L that is in some sense reduced. The obvious choice for a reduced basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ is to let $\mathbf{b}_i$ be the shortest vector in L that is independent of the vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1}$. Although Gaussian reduction finds such a basis for $n = 2$ [10, pg. 23] and a method due to Vallée [31] finds such a basis for $n = 3$, currently there is no known algorithm that will construct such a basis in a reasonable amount of time for $n > 3$. The following alternate definition of a reduced basis, due to Lenstra, Lenstra and Lovász [23], is useful as there is an algorithm (LLL) for finding such a reduced basis.

Definition 2.3 *Let* $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ *be a basis for the lattice* L *and let* $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ *where* $\mu_{i,j} = (\mathbf{b}_i \cdot \mathbf{b}_j^*)/\|\mathbf{b}_j^*\|^2$ (*This is the Gram-Schmidt orthogonalization process*). *We call the vectors* $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ LLL reduced *if*

1. $|\mu_{i,j}| \leq 1/2$ *for* $1 \leq j < i \leq n$
2. $\|\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*\|^2 \geq 3/4\|\mathbf{b}_{i-1}^*\|^2$ *for* $1 < i \leq n$

Condition 1 states that the vectors $\mathbf{b}_i$ must be close to orthogonal. Condition 2, along with 1, allows us to bound the values $\|\mathbf{b}_j\|$ in terms of the norms of the shortest vectors in the lattice L.

Theorem 2.2 *Suppose* $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ *form an LLL reduced basis for a lattice* L. *Then for every nonzero vector* $\mathbf{x} \in L$, *we have* $\|\mathbf{b}_1\| \leq 2^{(n-1)/2}\|\mathbf{x}\|$. *In particular,* $\|\mathbf{b}_1\|$ *is no larger than* $2^{(n-1)/2}$ *times as large as the norm of a shortest nonzero vector in* L.

Proof: As the vectors $\mathbf{b}_i^*$ and $\mathbf{b}_{i-1}^*$ are orthogonal, condition 2 tells us that

$$(\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*) \cdot (\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*) = \|\mathbf{b}_i^*\|^2 + |\mu_{i,i-1}|^2 \|\mathbf{b}_{i-1}^*\|^2 \geq 3/4\|\mathbf{b}_{i-1}^*\|^2.$$

By condition 1, this implies $\|\mathbf{b}_i^*\|^2 \geq 1/2\|\mathbf{b}_{i-1}^*\|^2$ and so by induction

$$\|\mathbf{b}_i^*\|^2 \geq \frac{1}{2^{i-j}}\|\mathbf{b}_j^*\|^2.$$

Now for any nonzero vector $\mathbf{x} \in L$, we can write $\mathbf{x}$ as $\mathbf{x} = \sum_{i=1}^{k} a_i \mathbf{b}_i$ with $1 \leq k \leq n$, $a_k \neq 0$, and each $a_i \in \mathbb{Z}$. Replacing $\mathbf{b}_i$ with $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$ allows us to write $\mathbf{x} = \sum_{i=1}^{k} s_i \mathbf{b}_i^*$ with each $s_i \in \mathbb{R}$ and $s_k = a_k \in \mathbb{Z}$. This gives

$$\begin{aligned} \|\mathbf{x}\|^2 &= \sum_{i=1}^{k} |s_i|^2 \|\mathbf{b}_i^*\|^2 \geq |a_k|^2 \|\mathbf{b}_k^*\|^2 \geq \|\mathbf{b}_k^*\|^2 \\ &\geq \|\mathbf{b}_1^*\|^2 2^{1-k} \geq \|\mathbf{b}_1^*\|^2 2^{1-n} = \|\mathbf{b}_1\|^2 2^{1-n} \end{aligned}$$

or equivalently,

$$\|\mathbf{b}_1\| \leq 2^{(n-1)/2} \|\mathbf{x}\| \text{ for any } \mathbf{x} \in L.$$

□

As shown in [23], one can prove that for an LLL reduced basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ and for any set of t linearly independent vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t \in L$, we have the inequality $\|\mathbf{b}_j\| \leq 2^{(n-1)/2} \max(\|\mathbf{x}_1\|, \|\mathbf{x}_2\|, \ldots, \|\mathbf{x}_t\|)$ for $1 \leq j \leq t$. The details of the algorithm used to construct an LLL reduced basis from an arbitrary basis for L are presented in Figure 2.1. Referring to Figure 2.1 we see that the body of the main loop first ensures condition 1 in the definition of an LLL reduced basis is satisfied for the vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k$ and then checks to see if condition 2 holds. Note that at the beginning of the main loop, the vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{k-1}$ form an LLL reduced basis for the lattice that they span. For proof of termination and improvements on the basic algorithm one is referred to [23] and [10].

2.1.1 Finding integer relations with LLL

One use of the LLL algorithm is to find small integer relations between a set of nonzero values $x_1, x_2, \ldots, x_n$. To use the LLL algorithm to find an integer relation for $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, define the $(n+1) \times n$ matrix B as

$$B = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ Nx_1 & Nx_2 & \cdots & \cdots & Nx_n \end{bmatrix} \text{ where } N \text{ is a large number}$$

The LLL Algorithm
This algorithm takes an arbitrary basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ for the lattice L as input and uses it to construct an LLL reduced basis.

Step 1 Initialization
Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a basis for L.
for i to n do
set $\mathbf{b}_i^* := \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$
calculate $\|\mathbf{b}_i^*\|^2$ and $\mu_{j,i}$ for $i+1 \le j \le n$ $\quad (\mu_{j,i} := (\mathbf{b}_j \cdot \mathbf{b}_i^*)/\|\mathbf{b}_i^*\|^2)$
end do
set $k := 2$

Step 2 Main Loop
Repeat
for j from $(k-1)$ downto 1 do
$q := \lfloor \mu_{k,j} \rceil$ (nearest integer)
$\mathbf{b}_k := \mathbf{b}_k - q\mathbf{b}_j$
for i to j do $\mu_{k,i} := \mu_{k,i} - q\mu_{j,i}$ end do
end do
If $\|\mathbf{b}_k^*\|^2 \ge (3/4 - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$
then set $k := k+1$
else interchange $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$.
Update $\mathbf{b}_k^*$, $\mathbf{b}_{k-1}^*$, $\|\mathbf{b}_k^*\|^2$, $\|\mathbf{b}_{k-1}^*\|^2$ and the $\mu_{i,j}$'s as follows:
set $\mathbf{b}_{k-1}^{*\prime} := \mathbf{b}_k^* + \mu_{k,k-1}\mathbf{b}_{k-1}^*$
$\|\mathbf{b}_{k-1}^{*\prime}\|^2 := \|\mathbf{b}_k^*\|^2 + (\mu_{k,k-1})^2\|\mathbf{b}_{k-1}^*\|^2$
$m := \mu_{k,k-1}\|\mathbf{b}_{k-1}^*\|^2/\|\mathbf{b}_{k-1}^{*\prime}\|^2$
$\mathbf{b}_k^{*\prime} := \mathbf{b}_k^* - m\mathbf{b}_{k-1}^{*\prime} = \frac{\|\mathbf{b}_k^*\|^2}{\|\mathbf{b}_{k-1}^{*\prime}\|^2}\mathbf{b}_{k-1}^* - m\mathbf{b}_k^*$
$\|\mathbf{b}_k^{*\prime}\|^2 := \frac{\|\mathbf{b}_k^*\|^4}{\|\mathbf{b}_{k-1}^{*\prime}\|^4}\|\mathbf{b}_{k-1}^*\|^2 + m^2\|\mathbf{b}_k^*\|^2 = \|\mathbf{b}_k^*\|^2\|\mathbf{b}_{k-1}^*\|^2/\|\mathbf{b}_{k-1}^{*\prime}\|^2$
interchange $\mu_{k,i}$ with $\mu_{k-1,i}$ for $1 \le i \le k-2$
for i from $k+1$ to n do
$t := \mu_{i,k}$, $\mu_{i,k} := \mu_{i,k-1} - \mu_{k,k-1}\mu_{i,k}$, $\mu_{i,k-1} := t + m\mu_{i,k}$
end do
set $\mu_{k,k-1} := m$
set $\mathbf{b}_k^* := \mathbf{b}_k^{*\prime}$, $\|\mathbf{b}_k^*\|^2 := \|\mathbf{b}_k^{*\prime}\|^2$, $\mathbf{b}_{k-1}^* := \mathbf{b}_{k-1}^{*\prime}$, $\|\mathbf{b}_{k-1}^*\|^2 := \|\mathbf{b}_{k-1}^{*\prime}\|^2$.
set $k := \max(2, k-1)$
end if
until $k = n+1$
At this point, the vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ form an LLL reduced basis for the lattice L.

Figure 2.1: Pseudocode implementation of the LLL algorithm

and let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be the column vectors of B. If we now consider the vectors in the lattice L spanned by $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ we see they are of the form

$$\mathbf{m}' = \sum_{i=1}^{n} m_i \mathbf{b}_i = \left[m_1, m_2, \ldots, m_n, N \sum_{i=1}^{n} m_i x_i \right]^T .$$

We may view the last term in $\mathbf{m}'$, $N \sum m_i x_i$, as a penalty term. If the vector $\mathbf{m} = [m_1, m_2, \ldots, m_n]$ is an integer relation for $\mathbf{x}$ then this term will be zero. However, if $\mathbf{m}$ is not an integer relation for $\mathbf{x}$ then this term will be large provided N is large enough. The penalty for not being an integer relation depends on the choice of N. If N is taken large enough and $\mathbf{m}$ is a short integer relation for $\mathbf{x}$, then $\mathbf{m}'$ will be one of the shortest vectors in L. With this in mind, to find an integer relation for $\mathbf{x}$ we choose a suitably large value of N and run the LLL algorithm on the vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$. The first vector, $\mathbf{b}'_1$, in the returned basis will be one of the smallest vectors in L. (Note that $\mathbf{b}'_1$ is not necessarily the shortest vector in the returned basis. We may also wish to consider the other $\mathbf{b}'_i$ as well). From $\mathbf{b}'_1$ we can pick off the coefficients of the suspected integer relation $\mathbf{m}$ for $\mathbf{x}$. It is important to realize that $\mathbf{m}$ may not be an integer relation for $\mathbf{x}$. This method can fail to return a valid relation for one of two reasons, the first being that $\mathbf{x}$ has no integer relations, and the second being that N was not large enough.

Lemma 2.1 *Suppose there are integer relations for $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$. Then the method presented above will find one provided N is large enough.*

Proof: Let M be the norm of a smallest integer relation for $\mathbf{x}$ and consider the finite set of vectors $\left\{ \mathbf{y} \in \mathbb{Z}^n : \|\mathbf{y}\| < 2^{n/2} M, (\mathbf{y} \cdot \mathbf{x}) \neq 0 \right\}$. From this nonempty set, choose a vector $\mathbf{y}$ with the property that $|\mathbf{y} \cdot \mathbf{x}| = |\sum y_i x_i|$ is minimal. For this $\mathbf{y}$, choose N so that $N |\sum y_i x_i| > 2^{n/2} M$. Now for any $\mathbf{m}' \in L$, if $\sum_{i=1}^{n} m_i x_i \neq 0$ then $\|\mathbf{m}'\| > 2^{n/2} M$. If $\mathbf{m} = [m_1, m_2, \ldots, m_n]^T$ is not an integer relation for $\mathbf{x}$, then the norm of the vector $\mathbf{m}' = [m_1, m_2, \ldots, m_n, N \sum m_i x_i]^T$ is greater than $2^{((n+1)-1)/2}$ times as large as the norm of a shortest nonzero vector in L and hence cannot be the first vector in an LLL reduced basis by Theorem 2.2.

□

Although this shows that an integer relation will be found if one exists and N is large enough, we do not know beforehand how large N must be.

2.2 The PSLQ algorithm

Following the *Generalized Euclidean Algorithm*[13], Ferguson developed a sequence of non-recursive integer relation algorithms [15][3][16], each an improvement on the previous ones. In this section we cover the latest incarnation of these, a simplified statement of the PSLQ algorithm. Although the general outline of [16] is followed, additional motivation for each step of the algorithm is presented and a greater emphasis is placed on $\mathbf{x}^\perp$. This gives a better picture of how the steps fit together and allows for a clearer statement of the proofs of Theorem 2.3 and Lemma 2.2. The proof of termination given in [16] has also been extended to cover the case when $\mathbf{x}$ has no integer relations of norm less than some constant T.

As before, suppose we wish to find a small integer relation between the nonzero values $x_1, x_2, \ldots, x_n$. Rather than using the method of the previous section, based upon the LLL algorithm, we instead consider the ideas laid out at the end of the introduction to this chapter.

Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and suppose we have a set of n linearly independent vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n \in \mathbb{Z}^n$ such that the projection of each $\mathbf{a}_i$ onto $\mathbf{x}^\perp$ is small (we say each vector $\mathbf{a}_i$ is close to $\mathbf{x}$). If we define A to be the matrix such that the ith row of A is $\mathbf{a}_i^T$, then A is invertible. Let $B = A^{-1}$ and let $\mathbf{b}_j$ be the jth column vector of B. Now as $(\mathbf{a}_i \cdot \mathbf{b}_j) = 0$ for $i \neq j$ and each $\mathbf{a}_i$ is close to $\mathbf{x}$, we would expect that each $\mathbf{b}_j$ lies close to $\mathbf{x}^\perp$. The idea behind the PSLQ algorithm is to start with the standard basis for the lattice $\mathbb{Z}^n$ and with each iteration, construct a new basis $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ for $\mathbb{Z}^n$ in which the $\mathbf{a}_i$ are closer to $\mathbf{x}$. In doing so, we hope to force the vectors $\mathbf{b}_j$ closer to $\mathbf{x}^\perp$. We will see that as an upper bound on the values $\left\|\text{proj}_{\mathbf{x}^\perp} \mathbf{a}_i\right\|$ decreases, a lower bound on the size of any possible integer relation for $\mathbf{x}$ increases. Throughout the PSLQ algorithm we work with the following matrices:

A: An $n \times n$ invertible matrix. The column vectors $\mathbf{a}_i$ of A^T form a basis for $\mathbb{Z}^n$.

H: An $n \times (n-1)$ matrix with column vectors $\mathbf{h}_j$ that form an orthonormal basis for $\mathbf{x}^\perp$.

H': The matrix AH. Each entry $h'_{i,j}$ in H' is the inner product of $\mathbf{a}_i$ with $\mathbf{h}_j$. Note that the projection of $\mathbf{a}_i$ onto $\mathbf{x}^\perp$ is $\sum_{j=1}^{n-1} (\mathbf{a}_i \cdot \mathbf{h}_j)\mathbf{h}_j$. Each time we begin the main iteration of the algorithm, H' will be lower trapezoidal (see Definition 2.4) and we will have $\left|h'_{i,j}\right| \leq 1/2 \left|h'_{j,j}\right|$ for $1 \leq i < j$. This will give

$$\left\|\text{proj}_{\mathbf{x}^\perp} \mathbf{a}_i\right\|^2 \leq 1/4 \sum_{j=1}^{i-1} \left|h'_{j,j}\right|^2 + \left|h'_{i,i}\right|^2 \leq \sum_{j=1}^{i} \left|h'_{j,j}\right|^2 .$$

By reducing the $\left|h'_{i,i}\right|$, we will reduce an upper bound on $\left\|\text{proj}_{\mathbf{x}^\perp}\mathbf{a}_i\right\|$ for each i.

B: $B = A^{-1}$. The column vectors $\mathbf{b}_j$ of B will be forced closer to $\mathbf{x}^\perp$ by forcing the vectors $\mathbf{a}_i$ closer to $\mathbf{x}$.

Definition 2.4 *The $m \times n$ matrix C is said to be lower trapezoidal if $m > n$ and each entry $c_{i,j}$ of C equals zero if $j > i$.*

Although forcing the vectors $\mathbf{a}_i$ closer to $\mathbf{x}$ is not sufficient to guarantee one of the $\mathbf{b}_j$ will eventually lie in $\mathbf{x}^\perp$, termination of the algorithm and a bound on the size of the relation found will follow with some work from Theorem 2.3. From this theorem, we will see that reducing the values $\left|h'_{i,i}\right|$ also increases a lower bound on the norm of the smallest possible integer relation for $\mathbf{x}$.

Theorem 2.3 *Let A be an invertible $n \times n$ matrix with integer coefficients, $\mathbf{x}$ a vector in $\mathbb{R}^n$, and H an $n \times (n-1)$ matrix with column vectors that form an orthonormal basis for $\mathbf{x}^\perp$. If $H' = AH$ is lower trapezoidal with each diagonal $h'_{i,i} \neq 0$ then*

$$\frac{1}{\max\left|h'_{i,i}\right|} \leq \|\mathbf{m}\| \quad \text{for any integer relation } \mathbf{m} \text{ of } \mathbf{x}.$$

Proof: For any integer relation $\mathbf{m}$, $HH^T\mathbf{m} = \mathbf{m}$ as HH^T is the projection matrix onto $\mathbf{x}^\perp$. Thus $A\mathbf{m} = H'(H^T\mathbf{m})$. Let $\mathbf{a}_i^T$ be the ith row vector of A, $\mathbf{h}_i^T$ be the ith row vector of H^T, and $h'_{j,j}$ the jth diagonal element of H'. As A is invertible, $A\mathbf{m} \neq \mathbf{0}$. Let j be the least integer such that $\mathbf{a}_j^T\mathbf{m} \neq 0$. Then $\mathbf{a}_k^T\mathbf{m} = 0$ for $1 \leq k < j$ and so by recursion and the fact that H' is lower trapezoidal with nonzero diagonal elements, $\mathbf{h}_k^T\mathbf{m} = 0$ for $1 \leq k < j$ and $\mathbf{a}_j^T\mathbf{m} = h'_{j,j}(\mathbf{h}_j^T\mathbf{m})$. As $\mathbf{a}_j^T\mathbf{m}$ is a nonzero integer,

$$1 \leq \left|h'_{j,j}\right| \left|\mathbf{h}_j^T\mathbf{m}\right| \leq \left|h'_{j,j}\right| \|\mathbf{m}\|.$$

The last inequality comes from the fact that the norm of the projection of $\mathbf{m}$ onto the unit vector $\mathbf{h}_j$ cannot be larger than the norm of $\mathbf{m}$. The result now follows.

□

The details of the PSLQ algorithm are presented in Figure 2.2. Although one can implement the algorithm in such a way that requires only the matrices B and H', the matrices A and H

The PSLQ Algorithm

This algorithm takes a vector $\mathbf{x}^T = [x_1, x_2, \ldots, x_n]$ and a constant $T \geq 1$ as input. It either returns an integer relation for $\mathbf{x}$ along with a lower bound on the norm of the shortest integer relation or it returns a lower bound ($\geq T$) on the norm of any possible relation for $\mathbf{x}$.

Step 1: Initialization

Fix the constant γ so $\gamma > \sqrt{4/3}$.

Let $A = B = I$, $\mathbf{a}_i^T$ the ith row of A, $\mathbf{b}_j$ the jth column of B.

Let H and H' be the $n \times (n-1)$ lower trapezoidal matrices with entries

$$h'_{i,j} = h_{i,j} = \begin{cases} 0 & 1 \leq i < j \leq n-1 \\ s_{i+1}/s_i & 1 \leq i = j \leq n-1 \\ -x_i x_j / s_j s_{j+1} & 1 \leq j < i \leq n \end{cases} \quad \text{where } s_j^2 = \sum_{k=j}^{n} x_k^2$$

Let $\mathbf{h}'_i$ be the ith row vector of H' and let $\mathbf{h}_i$ be the ith column vector of H.

Step 2: Size Reduce H'

For i from 2 to n do, for j from $i-1$ downto 1 do

set $t = \lfloor h'_{i,j}/h'_{j,j} \rceil$

replace $\mathbf{a}_i$ with $\mathbf{a}_i - t\mathbf{a}_j$, $\mathbf{b}_j$ with $\mathbf{b}_j + t\mathbf{b}_i$, and $\mathbf{h}'_i$ with $\mathbf{h}'_i - t\mathbf{h}'_j$

end do, end do.

Step 3: The Main Iteration

Choose r so that $\gamma^i \left|h'_{i,i}\right|$ is maximal when $i = r$.

Repeat the following until either $\frac{1}{\max|h'_{i,i}|} \geq T$ or both $h'_{n,n-1} = 0$ and $r = n-1$:

1. Let $\alpha = h'_{r,r}$, $\beta = h'_{r+1,r}$, and $\lambda = h'_{r+1,r+1}$. Then interchange rows $\mathbf{a}_r^T$ and $\mathbf{a}_{r+1}^T$ of A, columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B, and rows $\mathbf{h}'_r$ and $\mathbf{h}'_{r+1}$ of H'.
2. If $r = n-1$ then H' is still lower trapezoidal. In this case the value of $|h_{n-1,n-1}|$ was reduced by at least a factor of 2.
   If $r < n-1$ then H' is no longer trapezoidal. Remedy this by modifying the basis for $\mathbf{x}^\perp$. Rotate $\mathbf{h}_r$ and $\mathbf{h}_{r+1}$ in the plane they define so that the projection of $\mathbf{a}_r$ onto $\mathbf{h}_{r+1}$ is 0. This is done by replacing H by HQ and H' by $H'Q$ where Q is the $(n-1)\times(n-1)$ unitary matrix defined as follows:
   Set $Q = I_{n-1}$ and let $\delta = \sqrt{\beta^2 + \lambda^2}$.
   Then set $q_{r,r} = \beta/\delta$, $q_{r+1,r} = \lambda/\delta$, $q_{r,r+1} = -\lambda/\delta$, and $q_{r+1,r+1} = \beta/\delta$.
   In addition to setting $h'_{r,r+1}$ to 0, this sets $h'_{r,r} = \delta$ and $h'_{r+1,r+1} = -\alpha\lambda/\delta$.
3. Size reduce H' as in Step 2.
4. Choose r so that $\gamma^i \left|h'_{i,i}\right|$ is maximal as above.

Step 4: Return $1/\max \left|h'_{i,i}\right|$ as a lower bound on the norm of any integer relation for $\mathbf{x}$.

If $r = n-1$ and $h'_{n,n-1} = 0$ then return $\mathbf{b}_{n-1}$ as an integer relation for $\mathbf{x}$.

Figure 2.2: Pseudocode implementation of the PSLQ algorithm

are included as they make it easier to follow the reasoning behind the various steps. While the version presented here is valid only for real vectors $\mathbf{x}$, it can easily be extended to work with complex vectors as well [16].

Note that in step 1, partial sums of squares of the x_i are used to construct the matrix H. It can be seen that the column vectors $\mathbf{h}_i$ of H form an orthonormal basis for $\mathbf{x}^\perp$ by considering $(\mathbf{x} \cdot \mathbf{h}_i)$ and $(\mathbf{h}_i \cdot \mathbf{h}_j)$ for $1 \leq i, j \leq n-1$. By examining the definitions of the h_i and s_i in step 1 of the algorithm we see the following.
For $1 \leq i \leq n-1$

$$(\mathbf{x} \cdot \mathbf{h}_i) = x_i h_{i,i} + \sum_{k=i+1}^{n} x_k h_{k,i} = x_i \frac{s_{i+1}}{s_i} + \sum_{k=i+1}^{n} x_k \frac{-x_k x_i}{s_i s_{i+1}}$$

$$= \frac{x_i s_{i+1}}{s_i} - \frac{x_i}{s_i s_{i+1}} \sum_{k=i+1}^{n} x_k^2 = \frac{x_i s_{i+1}}{s_i} - \frac{x_i s_{i+1}^2}{s_i s_{i+1}} = 0.$$

For $1 \leq i < j \leq n-1$

$$(\mathbf{h}_i \cdot \mathbf{h}_j) = h_{j,i} h_{j,j} + \sum_{k=j+1}^{n} h_{k,i} h_{k,j} = \frac{-x_j x_i}{s_i s_{i+1}} \frac{s_{j+1}}{s_j} + \sum_{k=j+1}^{n} \frac{-x_k x_i}{s_i s_{i+1}} \frac{-x_k x_j}{s_j s_{j+1}}$$

$$= \frac{-x_i x_j s_{j+1}}{s_i s_{i+1} s_j} + \frac{x_i x_j}{s_i s_{i+1} s_j s_{j+1}} \sum_{k=j+1}^{n} x_k^2 = \frac{-x_i x_j s_{j+1}}{s_i s_{i+1} s_j} + \frac{x_i x_j s_{j+1}^2}{s_i s_{i+1} s_j s_{j+1}} = 0.$$

For $1 \leq i \leq n-1$

$$(\mathbf{h}_i \cdot \mathbf{h}_i) = h_{i,i}^2 + \sum_{k=i+1}^{n} h_{k,i}^2 = \left(\frac{s_{i+1}}{s_i} \right)^2 + \sum_{k=i+1}^{n} \left(\frac{-x_k x_i}{s_i s_{i+1}} \right)^2$$

$$= \frac{s_{i+1}^2}{s_i^2} + \frac{x_i^2}{s_i^2 s_{i+1}^2} \sum_{k=i+1}^{n} x_k^2 = \frac{s_i^2 - x_i^2}{s_i^2} + \frac{x_i^2}{s_i^2 s_{i+1}^2} s_{i+1}^2 = 1.$$

The matrix H we start with has the desired property, its columns form an orthonormal basis for $\mathbf{x}^\perp$.

At the beginning of step 1 of the algorithm we set the constant $\gamma > \sqrt{4/3}$. This requires an explanation. As stated above, if we reduce the values $\left| h'_{i,i} \right|$ for each i, then we reduce an upper bound on the values $\left\| \text{proj}_{\mathbf{x}^\perp} \mathbf{a}_i \right\|$ and increase a lower bound on the size of the smallest possible norm for any integer relation of $\mathbf{x}$. Now as r is chosen so that $\gamma^r \left| h'_{r,r} \right|$ is as large as possible, if $r < n-1$ then $\left| h'_{r+1,r+1} \right| \leq \frac{1}{\gamma} \left| h'_{r,r} \right|$. In this case we let $\alpha = h'_{r,r}$, $\beta = h'_{r+1,r}$,

$\lambda = h'_{r+1,r+1}$, and set $\delta = \sqrt{\beta^2 + \lambda^2}$. We then replace $h'_{r,r}$ with δ. From the reduction of H' we have that $\left|h'_{r+1,r}\right| \leq \frac{1}{2}\left|h'_{r,r}\right|$ which then gives

$$\delta = \sqrt{\beta^2 + \lambda^2} < \sqrt{\frac{\alpha^2}{4} + \frac{\alpha^2}{\gamma^2}} = |\alpha| \sqrt{\frac{1}{4} + \frac{1}{\gamma^2}}. \tag{2.1}$$

Thus $\left|h'_{r,r}\right|$ is reduced as long as $\sqrt{\frac{1}{4} + \frac{1}{\gamma^2}} < 1$ or $\gamma > \sqrt{4/3}$. Although this also increases $\left|h'_{r+1,r+1}\right|$ (as $h'_{r+1,r+1}$ is replaced with $-\alpha\lambda/\delta$ and $\left|h'_{r,r}h'_{r+1,r+1}\right| = |\delta \cdot \alpha\lambda/\delta| = |\alpha\lambda|$ remains unchanged, we see that $\left|h'_{r+1,r+1}\right|$ increases), this is not a significant problem. At each step we are forcing the larger diagonal elements of H' towards $h'_{n-1,n-1}$ where their size can be reduced by at least a factor of 2 when $r = n - 1$.

Even though we strive to reduce the diagonal entries of H', none will ever equal zero. From the fact that $h'_{i,i} \neq 0$ initially for any i, and as $h'_{r,r}$ and $h'_{r+1,r+1}$ are replaced with nonzero values when $r < n-1$, the only way a value $h'_{i,i}$ may become zero is if we interchange rows h'_{n-1} and h'_n of H' when $h_{n,n-1} = 0$. However, as we would exit the algorithm before this interchange occurred, the diagonal elements of H' are always nonzero. We need not concern ourselves about divisions by zero when computing $\lfloor h'_{i,j}/h'_{j,j} \rceil$ during the reductions of H'.

Now, in the event that the algorithm terminates with $h_{n,n-1} = 0$, the column vector $\mathbf{b}_{n-1}$ of B is an integer relation for $\mathbf{x}$. To see this, recall that $\mathbf{x}^T H = \mathbf{0}, BA = I, AH = H'$ and $h'_{n-1,n-1} \neq 0$. This gives $\mathbf{0} = \mathbf{x}^T BH' = [\mathbf{x}^T B\mathbf{h}'_1, \mathbf{x}^T B\mathbf{h}'_2, \ldots, \mathbf{x}^T B\mathbf{h}'_{n-1}]$ where $\mathbf{h}'_i$ is the ith column vector of H'. As the only nonzero entry in $\mathbf{h}'_{n-1}$ is $h'_{n-1,n-1}$, we have $0 = \mathbf{x}^T \mathbf{b}_{n-1} h_{n-1,n-1}$ which yields $\mathbf{x}^T \mathbf{b}_{n-1} = 0$. The $(n-1)$th column vector of B is an integer relation for $\mathbf{x}$.

2.2.1 A bound on the relation found by PSLQ

We have just shown that if the PSLQ algorithm terminates with $h'_{n,n-1} = 0$, then the column vector $\mathbf{b}_{n-1}$ of B is an integer relation for $\mathbf{x}$. Since we are looking for a small integer relation, we would like to show that the relation found is not much larger than the smallest possible integer relation for $\mathbf{x}$. To do this, we need the following lemma.

Lemma 2.2 *If the PSLQ algorithm terminates with $h'_{n,n-1}$ equaling zero, then the norm of $\mathbf{b}_{n-1}$, the integer relation found, is*

$$\|\mathbf{b}_{n-1}\| = 1/\left|h'_{n-1,n-1}\right|$$

Proof: Let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ be the standard orthonormal basis for $\mathbb{R}^n$ and $\mathbf{e}'_1, \mathbf{e}'_2, \ldots, \mathbf{e}'_{n-1}$ be the standard orthonormal basis for $\mathbb{R}^{n-1}$. As $\mathbf{b}_{n-1}$ is an integer relation for the vector $\mathbf{x}$, $(HH^T)\mathbf{b}_{n-1} = \mathbf{b}_{n-1}$. Using the facts that $AH = H'$ and $\mathbf{b}_{n-1}$ is the $(n-1)$th column of $B = A^{-1}$ we see that

$$H'H^T\mathbf{b}_{n-1} = AHH^T\mathbf{b}_{n-1} = A\mathbf{b}_{n-1} = \mathbf{e}_{n-1} = [0, 0, \ldots, 1, 0]^T$$

or

$$H^T\mathbf{b}_{n-1} = (H')^\dagger\mathbf{e}_{n-1}$$

where $(H')^\dagger$ is the left inverse of H'. We know H' has a left inverse because the row vectors of the lower trapezoidal matrix H' span $\mathbb{R}^{n-1}$. However, as the only nonzero element in the last column of H' is $h'_{n-1,n-1}$, the $(n-1)$th column of the $(n-1)\times n$ matrix $(H')^\dagger$ must be equal to $\frac{1}{h'_{n-1,n-1}}\mathbf{e}'_{n-1} = [0, 0, \ldots, 0, \frac{1}{h'_{n-1,n-1}}]^T$. The linear combination of the rows of H' required to construct $\mathbf{e}_i'^T$ cannot use the $(n-1)$th row of H' when $i \neq n-1$ and must have a multiple of $\frac{1}{h'_{n-1,n-1}}$ times the $(n-1)$th row when $i = n-1$. It follows that

$$\|H^T\mathbf{b}_{n-1}\| = \|H'^\dagger\mathbf{e}_{n-1}\| = \|[0, 0, \ldots, 0, \frac{1}{h'_{n-1,n-1}}]^T\| = \frac{1}{\left|h'_{n-1,n-1}\right|}.$$

Now, as $\mathbf{b}_{n-1}$ lies in $\mathbf{x}^\perp$ and the rows of H^T form an orthonormal basis for $\mathbf{x}^\perp$, we see that $\|H^T\mathbf{b}_{n-1}\| = \|\mathbf{b}_{n-1}\|$ which gives the result we are after:

$$\|\mathbf{b}_{n-1}\| = \frac{1}{\left|h'_{n-1,n-1}\right|}.$$

□

Armed with this lemma it becomes a simple matter to find a bound on the size of an integer relation found by the PSLQ algorithm.

Theorem 2.4 *Let M be the norm of the smallest integer relation for $\mathbf{x}$. If the PSLQ algorithm terminates because $h'_{n,n-1} = 0$ and $r = n-1$, then*

$$\|\mathbf{b}_{n-1}\| \leq \gamma^{n-2}M.$$

Proof: By Lemma 2.2, we know that $\|\mathbf{b}_{n-1}\| = \frac{1}{|h'_{n-1,n-1}|}$. As $r = n-1$, we have $\gamma^{n-1}\left|h'_{n-1,n-1}\right| \geq \gamma^i\left|h'_{i,i}\right|$ for $1 \leq i \leq n-1$. From Theorem 2.3, as none of the diagonal elements of H' are zero, $M \geq \frac{1}{|h'_{j,j}|}$ for some j and so

$$M\gamma^{n-1} \geq \frac{\gamma^{n-1}}{\left|h'_{j,j}\right|} \geq \frac{\gamma^j}{\left|h'_{n-1,n-1}\right|} \geq \frac{\gamma}{\left|h'_{n-1,n-1}\right|} = \gamma\|\mathbf{b}_{n-1}\|$$

or

$$\|\mathbf{b}_{n-1}\| \leq \gamma^{n-2} M.$$

□

We cannot guarantee the PSLQ algorithm will return a smallest integer relation for $\mathbf{x}$. However, if we have $\max \left|h'_{i,i}\right| = \left|h'_{n-1,n-1}\right|$ upon termination, then we have found an integer relation for $\mathbf{x}$ of smallest possible norm. It should be noted that if we stop the algorithm when $h'_{n,n-1} = 0$ but before $r = n-1$, then the above bound does not apply. Although $\mathbf{b}_{n-1}$ will still be an integer relation for $\mathbf{x}$ with norm equal to $1/\left|h'_{n-1,n-1}\right|$, this norm may not be as small as we can make it. If we continue until $r = n-1$, then we may increase the value $|h_{n-1,n-1}|$ (it will not decrease) and hence decrease the norm of $\mathbf{b}_{n-1}$.

2.2.2 Termination of the algorithm

We first consider the case when the vector $\mathbf{x}$ has integer relations. In this case, termination of the algorithm rests upon the result of Theorem 2.3 and the method used to reduce the diagonal elements of H'. To begin, define τ so that

$$\frac{1}{\tau} = \sqrt{\frac{1}{4} + \frac{1}{\gamma^2}}.$$

In step 1 of the algorithm, γ was chosen so that $\gamma > \sqrt{4/3}$ and thus we see that $\tau > 1$. To show that the algorithm terminates we show that $\Pi(k)$, a function of the diagonal elements of H', is bounded and increases with each iteration.

Definition 2.5 *Let $h'_{i,i}(k)$ be the ith diagonal element of H' at the end of the kth iteration for $k \geq 1$ and let $h'_{i,i}(0)$ be the ith diagonal element of H' at the beginning of the first iteration. Let γ be as chosen in step 1 and let M be the norm of a smallest integer relation for the n dimensional vector $\mathbf{x}$. Then define $\Pi(k)$ to be*

$$\Pi(k) = \prod_{i=1}^{n-1} [\min(\gamma^{n-1} M, \frac{1}{\left|h'_{i,i}(k)\right|})]^{n-i}.$$

The following lemma shows that the function $\Pi(k)$ is bounded.

Lemma 2.3 *At the end of the kth iteration ($k \geq 0$), the following inequalities are satisfied.*

$$1 \leq \Pi(k) \leq (\gamma^{n-1} M)^{\binom{n}{2}}$$

Proof: From step 1 of the algorithm we see that

$$\left|h'_{i,i}(0)\right| = \left|\frac{s_{i+1}}{s_i}\right| < 1 \quad \text{for } 1 \le i \le n-1.$$

Now suppose that at the beginning of the kth iteration $\left|h'_{i,i}(k)\right| \le 1$ for each i. Then if $r = n-1$ only $\left|h'_{n-1,n-1}(k)\right|$ changes and is reduced by at least a factor of 2. If $r < n-1$ then only $\left|h'_{r,r}(k)\right|$ and $\left|h'_{r+1,r+1}(k)\right|$ change. As was shown previously in equation (2.1), $\left|h'_{r,r}(k)\right|$ is replaced with the smaller value $|\delta|$ and $\left|h'_{r+1,r+1}(k)\right|$ is replaced with $\left|\frac{\alpha\lambda}{\delta}\right| < |\alpha| = \left|h'_{r,r}(k)\right| \le 1$. Thus $\left|h'_{i,i}(k+1)\right| \le 1$ for $1 \le i \le n-1$. We see that for each i and $k \ge 0$, $\left|h'_{i,i}(k)\right| \le 1$. As both γ and M are larger than 1 it follows that for any $k \ge 0$, $\min(\gamma^{n-1}M, \frac{1}{\left|h'_{i,i}(k)\right|}) \ge 1$ for $1 \le i \le n-1$. This establishes one of the desired inequalities, $1 \le \Pi(k)$ for each $k \ge 0$.

For the second inequality, note that

$$\gamma^{n-1}M \ge \min(\gamma^{n-1}M, \frac{1}{\left|h'_{i,i}(k)\right|})$$

and so

$$\Pi(k) \le \prod_{i=1}^{n-1} \left(\gamma^{n-1}M\right)^{n-i} = \left(\gamma^{n-1}M\right)^{\sum_{i=1}^{n-1} i}.$$

As $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \binom{n}{2}$ we have the required result.

□

Before showing that $\Pi(k)$ increases by at least a factor of τ with each iteration, we need the following lemma.

Lemma 2.4 *Suppose the positive constants a, b and t satisfy the following inequalities:*

$$a \ge b, \quad a \ge t, \quad \text{and} \quad 1 \ge t.$$

Then

$$\frac{\min(a,1)\min(b,t)}{\min(a,t)\min(b,1)} \ge 1.$$

Proof: The proof is a simple verification. If one lists off all 24 possible orderings of a, b, 1, and t and then crosses off those orderings where it is possible to have $a < b$, $a < t$, or $1 < t$,

then only the following 5 orderings remain:

$$\begin{array}{ccccccc} a & \geq & b & \geq & 1 & \geq t \\ a & \geq & 1 & \geq & b & \geq t \\ a & \geq & 1 & \geq & t & \geq b \\ 1 & \geq & a & \geq & b & \geq t \\ 1 & \geq & a & \geq & t & \geq b. \end{array}$$

With these remaining orderings one easily checks that the desired inequality holds.

□

Lemma 2.5 *For any $k \geq 0$, $\Pi(k+1) \geq \tau\Pi(k)$.*

Proof: We will show that the quotient $\Pi(k+1)/\Pi(k)$ is greater than or equal to τ. When $r < n-1$, the following 2×2 submatrix of H'

$$\begin{bmatrix} h'_{r,r}(k) & 0 \\ h'_{r+1,r}(k) & h'_{r+1,r+1}(k) \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \beta & \lambda \end{bmatrix} \text{ becomes } \begin{bmatrix} \delta & 0 \\ \alpha\beta/\delta & -\alpha\lambda/\delta \end{bmatrix}$$

where $\delta = \sqrt{\beta^2 + \lambda^2}$. All other diagonal elements of H' remain unchanged. It follows that in this case

$$\frac{\Pi(k+1)}{\Pi(k)} = \frac{\min(\gamma^{n-1}M, \frac{1}{|\delta|})^{n-r} \cdot \min(\gamma^{n-1}M, \left|\frac{\delta}{\alpha\lambda}\right|)^{n-r-1}}{\min(\gamma^{n-1}M, \frac{1}{|\alpha|})^{n-r} \cdot \min(\gamma^{n-1}M, \frac{1}{|\lambda|})^{n-r-1}}.$$

If we make the substitutions $a = \gamma^{n-1}M\,|\delta|$ and $b = \gamma^{n-1}M\,|\lambda|$, then we have

$$\frac{\Pi(k+1)}{\Pi(k)} = \frac{\min(a,1)}{\min(a, \left|\frac{\delta}{\alpha}\right|)} \left(\frac{\min(a,1)}{\min(a, \left|\frac{\delta}{\alpha}\right|)} \frac{\min(b, \left|\frac{\delta}{\alpha}\right|)}{\min(b,1)} \right)^{n-r-1}.$$

Now as $\delta = \sqrt{\beta^2 + \lambda^2} \geq |\lambda|$, $a \geq b$ and as $\gamma > \sqrt{4/3}$, equation (2.1) shows that $|\delta/\alpha| < 1$. By Theorem 2.3 and the choice of r we also have that $M \geq 1/\left|h'_{j,j}\right|$ for some j and $\gamma^r\,|\alpha| \geq \gamma^i \left|h'_{i,i}\right|$ for $1 \leq i \leq n-1$. As $\gamma > 1$ this implies that

$$M\gamma^{n-1} \geq \frac{\gamma^r}{\left|h'_{j,j}\right|} \geq \frac{\gamma^j}{|\alpha|} \geq \frac{\gamma}{|\alpha|} \geq \frac{1}{|\alpha|} \tag{2.2}$$

or equivalently that

$$a = M\gamma^{n-1}\delta \geq \frac{\delta}{|\alpha|}.$$

As the conditions of Lemma 2.4 are satisfied, we see that

$$\frac{\Pi(k+1)}{\Pi(k)} \geq \frac{\min(a,1)}{\min(a, \left|\frac{\delta}{\alpha}\right|)}.$$

If $a \geq 1$, then since $|\beta| \leq |\alpha|/2$ and $|\alpha| \geq \gamma |\lambda|$ by choice of r,

$$\frac{\min(a,1)}{\min(a, \left|\frac{\delta}{\alpha}\right|)} = \left|\frac{\alpha}{\delta}\right| \geq \frac{|\alpha|}{\sqrt{\frac{\alpha^2}{4} + \frac{\alpha^2}{\gamma^2}}} = \frac{1}{\sqrt{\frac{1}{4} + \frac{1}{\gamma^2}}} = \tau.$$

Otherwise we have $1 > a \geq |\delta/\alpha|$ and so

$$\frac{\min(a,1)}{\min(a, \left|\frac{\delta}{\alpha}\right|)} = M\gamma^{n-1}|\alpha| \geq \gamma$$

from equation (2.2) above. From the definition of τ we see that $1/\tau^2 > 1/\gamma^2$ or that $\gamma > \tau$. Thus if $r < n-1$ then

$$\Pi(k+1) \geq \tau\Pi(k).$$

If $r = n-1$ then the only diagonal of H' that changes is $\left|h'_{n-1,n-1}\right| = |\alpha|$. In this case we have $\left|h'_{n-1,n-1}(k+1)\right| \leq |\alpha|/2$ and so

$$\frac{\Pi(k+1)}{\Pi(k)} \geq \frac{\min(\gamma^{n-1}M, \frac{1}{|\alpha|/2})}{\min(\gamma^{n-1}M, \frac{1}{|\alpha|})} = \frac{\min(\gamma^{n-1}M|\alpha|, 2)}{\min(\gamma^{n-1}M|\alpha|, 1)}.$$

Again, from equation (2.2) we have $\gamma^{n-1}M|\alpha| \geq \gamma \geq 1$. Now as $1/\tau^2 = 1/4 + 1/\gamma^2 > 1/4$, $2 > \tau$ and so if $\gamma^{n-1}M|\alpha| \geq 2$ then

$$\frac{\Pi(k+1)}{\Pi(k)} = 2 > \tau.$$

On the other hand, if $2 > \gamma^{n-1}M|\alpha| \geq 1$ we have

$$\frac{\Pi(k+1)}{\Pi(k)} = \gamma^{n-1}M|\alpha| \geq \gamma > \tau.$$

If $r = n-1$ we also have

$$\Pi(k+1) \geq \tau\Pi(k).$$

□

We are now in a position to give bounds on both the number of iterations required to find an integer relation for $\mathbf{x}$ and the number of exact arithmetic operations required to find this relation.

Theorem 2.5 *If the vector* $\mathbf{x}$ *has integer relations, then the PSLQ algorithm will find one in less than*

$$\binom{n}{2}\frac{\log(\gamma^{n-1}M)}{\log\tau}$$

iterations where M *is the norm of a shortest relation for* $\mathbf{x}$*,* γ *is as chosen in step 1 of the algorithm, and* $\tau > 1$ *is defined by* $1/\tau^2 = 1/4 + 1/\gamma^2$*.*

Proof: Suppose we have completed k iterations of the algorithm and have not yet found an integer relation for $\mathbf{x}$. Then from Lemma 2.5 we see that

$$\Pi(k) \geq \tau\Pi(k-1) \geq \cdots \geq \tau^k\Pi(0),$$

and so Lemma 2.3 gives

$$\left(\gamma^{n-1}M\right)^{\binom{n}{2}} > \Pi(k) \geq \tau^k.$$

Now as $\tau > 1$ we have

$$\frac{\binom{n}{2}\log(\gamma^{n-1}M)}{\log\tau} > k.$$

□

Corollary 2.1 *If* $\mathbf{x}$ *has integer relations then the PSLQ algorithm can be made to find one using*

$$O\left(n^4 + n^3\log M\right)$$

exact arithmetic operations.

Proof: The above theorem shows that the PSLQ algorithm will find an integer relation, if one exists, in less than $\frac{n^2+n}{2}\frac{((n-1)\log\gamma+\log M)}{\log\tau}$ or $O\left(n^3 + n^2\log M\right)$ iterations. Examining the algorithm we see that parts 1, 2, and 4 of the main iteration can be completed using $O(n)$ exact arithmetic operations and part 3, the size reduction of H', requires $O(n^3)$. Thus the algorithm as given requires $O(n^6 + n^5\log M)$ exact arithmetic operations. However, if we examine the proof of Lemma 2.5 we see that the full reduction of the matrix H' is not necessary. All that is required for this proof to go through is that $\left|h'_{i+1,i}\right| \leq 1/2\left|h'_{i,i}\right|$ for $1 \leq i \leq n-1$. If we make this change to the algorithm then part 3 can be completed in $O(n)$ exact arithmetic operations as well. Thus if $\mathbf{x}$ has integer relations then the PSLQ algorithm can be made to find one in $O(n^4 + n^3\log M)$ exact arithmetic operations.

□

Although we can modify part 3 of the main iteration so that it requires only $O(n)$ exact arithmetic operations, this was not done as our final goal is to implement the PSLQ algorithm using inexact arithmetic. If we do not do a full reduction of the matrix H', but instead only do a partial reduction then the algorithm becomes unstable. More is said on this matter when we discuss the HJLS algorithm and its relation to PSLQ.

The proof of termination of the algorithm when $\mathbf{x}$ has no integer relations of norm less than T is very similar and requires only cosmetic changes. In this case we define a new function $\Pi^*(k)$ as

$$\Pi^*(K) = \prod_{i=1}^{n-1} [\min(\gamma^{n-1}T, \frac{1}{\left|h'_{i,i}(k)\right|})]^{n-1}.$$

Exactly as in Lemma 2.3 we have that

$$1 \leq \Pi^*(k) \leq (\gamma^{n-1}T)^{\binom{n}{2}}.$$

Now rather than using Theorem 2.3 in Lemma 2.5, we instead use the fact that if the algorithm has not terminated after the kth iteration, then there is at least one j such that

$$\left|\frac{1}{h'_{j,j}}\right| < T.$$

If we redefine a and b so that

$$a = \gamma^{n-1}T\delta \quad \text{and} \quad b = \gamma^{n-1}T\,|\lambda|$$

then equation (2.2) becomes

$$T\gamma^{n-1} \geq \frac{\gamma^r}{\left|h'_{j,j}\right|} \geq \frac{\gamma^j}{|\alpha|} \geq \frac{\gamma}{|\alpha|} \geq \frac{1}{|\alpha|}$$

or equivalently,

$$a = T\gamma^{n-1}\delta \geq \frac{\delta}{|\alpha|}.$$

Everything carries through as before. We see that

$$\Pi^*(k+1) \geq \tau\Pi^*(k).$$

The results of this section are summarized in the following theorem.

Theorem 2.6 *Let $\mu = \min(M, T)$ where T is the bound passed to the PSLQ algorithm and M is the norm of a smallest integer relation for $\mathbf{x}$ (if $\mathbf{x}$ has no integer relations then $M = \infty$). Then the PSLQ algorithm will terminate in less than*

$$\binom{n}{2} \frac{\log\left(\gamma^{n-1}\mu\right)}{\log \tau}$$

iterations. Upon termination, the algorithm will either return an integer relation for $\mathbf{x}$ of norm no larger than $\gamma^{n-2}M$ or return a lower bound $\geq T$ on the norm of any integer relation for $\mathbf{x}$.

The fact that the PSLQ algorithm can return a lower bound on the size of an integer relation for $\mathbf{x}$ is very useful. For instance, Bailey and Plouffe use this to show that if Euler's constant satisfies an integer polynomial of degree 50 or less, then the Euclidean norm of the coefficients must exceed 7×10^{17} [4]. This is an advantage that the PSLQ algorithm has over LLL.

2.3 The HJLS algorithm

Initially given the preferable name of *The Small Integer Relation Algorithm*, HJLS is an integer relation algorithm developed by Hastad, Just, Lagarias, and Schnorr [22]. As with the PSLQ algorithm, it is based on work stemming from Ferguson and Forcade's *Generalized Euclidean Algorithm* [13]. The idea behind HJLS is again to construct a sequence of bases for the lattice $\mathbb{Z}^n$ in such a way that a lower bound on the size of any possible integer relation for $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ increases. Following the initial paper, we present the details of the algorithm in Figure 2.3 and give a brief proof of the algorithm's correctness. The proof of termination and a bound on the number of iterations required is essentially the same as that given for the PSLQ algorithm and will be omitted.

Theorem 2.7 *The HJLS algorithm correctly returns either an integer relation for $\mathbf{x}$ or the value 2^k as a lower bound on the norm of any possible relation. If an integer relation is found, it is no more than $\sqrt{2}^{n-2}$ times as large as the smallest possible relation for $\mathbf{x}$.*

Proof: Suppose the HJLS algorithm terminates with $\|\mathbf{a}_n^*\| \neq 0$. This implies that $\|\mathbf{a}_i^*\| = 0$ for some $i < n$ and so the vectors $\mathbf{x} = \mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}$ form a linearly dependent set. As the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n-1}$ are linearly independent, we see that $\mathbf{x} \in \text{span}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n-1})$

The HJLS Algorithm
This algorithm takes a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and a constant k as input. It either returns an integer relation for $\mathbf{x}$ or shows that $\mathbf{x}$ has no integer relations of norm less than 2^k.

Step 1 Initialization

Set $A = B = I_n$.
Let $\mathbf{a}_i^T$ be the ith row vector of A and $\mathbf{b}_i$ be the ith column vector of B.
Set $\mathbf{a}_0^* = \mathbf{x}$ and $\mathbf{a}_i^* = \mathbf{a}_i - \sum_{j=0}^{i-1} \mu_{i,j}\mathbf{a}_j^*$ for $i = 1, \ldots, n$

$$\text{where } \mu_{i,j} = \begin{cases} \frac{\mathbf{a}_i \cdot \mathbf{a}_j^*}{\|\mathbf{a}_j^*\|^2} & \|\mathbf{a}_j^*\| \neq 0 \\ 0 & \|\mathbf{a}_j^*\| = 0 \end{cases}$$

The vectors $\mathbf{a}_1^*, \mathbf{a}_2^*, \ldots, \mathbf{a}_n^*$ span $\mathbf{x}^\perp$.
If $\|\mathbf{a}_n^*\| = 0$ then return $\mathbf{b}_n$ as an integer relation for $\mathbf{x}$.

Step 2

Repeat
  Choose the value r that maximizes $2^r\|\mathbf{a}_r^*\|^2$ for $1 \leq r \leq n$.
  Partial Reduction: (Ensure $|\mu_{r+1,r}| \leq 1/2$)
    Set $\mathbf{a}_{r+1} = \mathbf{a}_{r+1} - \lceil \mu_{r+1,r} \rfloor \mathbf{a}_r$.
    Set $\mathbf{b}_{r+1} = \mathbf{b}_{r+1} + \lceil \mu_{r+1,r} \rfloor \mathbf{b}_r$ to maintain $B = A^{-1}$.
    Update the values $\mu_{r+1,i}, 1 \leq i \leq r$
  Exchange and Update:
    Exchange rows $\mathbf{a}_r^T$ and $\mathbf{a}_{r+1}^T$ of A.
    Exchange columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B.
    Update $\mathbf{a}_r^*, \|\mathbf{a}_r^*\|, \mathbf{a}_{r+1}^*, \|\mathbf{a}_{r+1}^*\|$, $\mu_{r+1,r}$,
    and the values $\mu_{i,r}$ and $\mu_{i,r+1}$ for $r + 2 \leq i \leq n$.
Until $\|\mathbf{a}_n^*\| \neq 0$ or $\|\mathbf{a}_i^*\| \leq 2^{-k}$ for all i with $1 \leq i \leq n$.

If $\|\mathbf{a}_n^*\| \neq 0$ then return $\mathbf{b}_n$ as an integer relation for $\mathbf{x}$. Otherwise return 2^k as a lower bound on the norm of any possible integer relation for $\mathbf{x}$.

Figure 2.3: Pseudocode implementation of the HJLS algorithm

and so $\mathbf{b}_n$ lies in $\mathbf{x}^\perp$ as $\mathbf{b}_n$ is orthogonal to each $\mathbf{a}_i$ for $1 \le i \le n-1$. In the event that $\|\mathbf{a}_n^*\| \neq 0$, HJLS correctly returns an integer relation for $\mathbf{x}$.

If HJLS terminates with $\|\mathbf{a}_n^*\| = 0$, then $\|\mathbf{a}_i^*\| \le 2^{-k}$ for each i. Now if $\mathbf{m}$ is an integer relation for $\mathbf{x}$, then $\mathbf{m} \in \mathbf{x}^\perp = \text{span}(\mathbf{a}_1^*, \mathbf{a}_2^*, \ldots, \mathbf{a}_{n-1}^*)$ and so $\mathbf{m} \cdot \mathbf{a}_i^* \neq 0$ for at least one i with $1 \le i \le n-1$. Choose the smallest i such that $\mathbf{m} \cdot \mathbf{a}_i^* \neq 0$. Then $\mathbf{m} \cdot \mathbf{a}_i^* = \mathbf{m} \cdot \mathbf{a}_i \in \mathbb{Z}$ and so as $\|\mathbf{m}\|\|\mathbf{a}_i^*\| \ge |\mathbf{m} \cdot \mathbf{a}_i^*| \ge 1$, we have $\|\mathbf{m}\| \ge 1/\|\mathbf{a}_i^*\|$. Thus the norm of any integer relation for $\mathbf{x}$ must be greater than $1/\max\|\mathbf{a}_i^*\| \ge 2^k$.

For the last portion of the proof, suppose that $\mathbf{b}_n$ is an integer relation found by the HJLS algorithm. Then as above, both $\mathbf{a}_n^*$ and $\mathbf{b}_n$ are orthogonal to the vectors $\mathbf{x}, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}$ and so $\mathbf{a}_n^* = c\mathbf{b}_n$ for some constant c. It follows that $\|\mathbf{b}_n\|\|\mathbf{a}_n^*\| = |\mathbf{b}_n \cdot \mathbf{a}_n^*| = |\mathbf{b}_n \cdot \mathbf{a}_n| = 1$ or $\|\mathbf{b}_n\| = 1/\|\mathbf{a}_n^*\|$. Now in the event that $\mathbf{a}_n^* \neq 0$ initially, $\|\mathbf{b}_n\| = 1$ and so $\mathbf{b}_n$ is a smallest integer relation for $\mathbf{x}$. If $\mathbf{a}_n^* = 0$ initially, then let $\mathbf{a}_1^{(k)}, \ldots, \mathbf{a}_n^{(k)}$, $\mathbf{a}_1^{*(k)}, \ldots, \mathbf{a}_n^{*(k)}$ be the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, $\mathbf{a}_1^*, \ldots, \mathbf{a}_n^*$ at the beginning of the final iteration, let $\mathbf{a}_1^{(k+1)}, \ldots, \mathbf{a}_n^{(k+1)}$, $\mathbf{a}_1^{*(k+1)}, \ldots, \mathbf{a}_n^{*(k+1)}$ be the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, $\mathbf{a}_1^*, \ldots, \mathbf{a}_n^*$ at the end of the final iteration, and let $\mathbf{m}$ be any integer relation for $\mathbf{x}$. We know that $\|\mathbf{m}\| \ge 1/\max\|\mathbf{a}_i^{*(k)}\|$ and that for the final iteration when a relation is found, $r = n-1$. With this choice of r, the partial reduction step sets $\mathbf{a}_n^{(k+1)} = \mathbf{a}_{n-1}^{(k)}$ and $\mathbf{a}_{n-1}^{(k+1)} = \mathbf{a}_n^{(k)} + \lceil \mu_{n,n-1}^{(k)} \rfloor \mathbf{a}_{n-1}^{(k)}$. As this results in $\mathbf{a}_n^{*(k+1)} \neq 0$ and $\mathbf{x}, \mathbf{a}_1^{*(k+1)}, \ldots, \mathbf{a}_{n-2}^{*(k+1)}$ do not change, we have that $\mathbf{a}_{n-1}^{*(k+1)} = 0$ and so $\mathbf{a}_n^{*(k+1)} = \mathbf{a}_{n-1}^{*(k)}$. Now by choice of r, $2\|\mathbf{a}_i^{*(k)}\|^2 \le 2^i\|\mathbf{a}_i^{*(k)}\|^2 \le 2^{n-1}\|\mathbf{a}_{n-1}^{*(k)}\|^2$ and so

$$\|\mathbf{a}_i^{*(k)}\|^2 \le 2^{n-2}\|\mathbf{a}_{n-1}^{*(k)}\|^2 = 2^{n-2}\|\mathbf{a}_n^{*(k+1)}\|^2 = 2^{n-2}\frac{1}{\|\mathbf{b}_n\|^2} \quad \text{for } 1 \le i \le n.$$

As $\|\mathbf{m}\| \ge 1/\max\|\mathbf{a}_i^{*(k)}\|$ we have the desired result, $\|\mathbf{b}_n\| \le \sqrt{2}^{n-2}\|\mathbf{m}\|$. A relation found by the HJLS algorithm is no more than $\sqrt{2}^{n-2}$ times as large as the smallest integer relation for $\mathbf{x}$.

□

Note that there is nothing special that requires having a power of 2 for the lower bound found by HJLS. This lower bound on the norm of any possible integer relation for $\mathbf{x}$ is only a consequence of the termination condition which can be modified.

As in the proof of termination for the PSLQ algorithm, the HJLS algorithm will terminate provided we always have $|\mu_{r+1,r}| \le 1/2$ before exchanging $\mathbf{a}_r$ and $\mathbf{a}_{r+1}$. Due to this fact, the authors claim that having $|\mu_{i,j}| \le 1/2$ for all $i > j$ is unnecessary in the real number model of computation and so implement their algorithm with only a partial reduction done

at each step. This is unfortunate as it leads to numerical instability in the algorithm when implemented with inexact arithmetic operations. As they explain that the objective of their algorithm is to construct a sequence of bases for the lattice $\mathbb{Z}^n$ that converge strongly to the line $\mathbb{R}\mathbf{x}$ (the sequence of vectors $\{\mathbf{a}_i\}$ converges strongly to the line $\mathbb{R}\mathbf{x}$ if $\left\|\text{proj}_{\mathbf{x}^\perp}\mathbf{a}_i\right\| \to 0$), it is fair to say that if we implement HJLS with full reductions, which actually forces the projections of the basis vectors to tend to zero, then we simply have another implementation of the HJLS algorithm. The authors of PSLQ must agree with this as to get the better bound on the number of iterations required, they work with an implementation of PSLQ that only performs a partial reduction at each step and still call it the PSLQ algorithm.

2.3.1 The relation between HJLS and PSLQ

In [16], the authors claim that HJLS is not a special case of the later PSLQ algorithm with $\gamma = \sqrt{2}$. To support their claim they give two examples where the performance of PSLQ and HJLS differ. If we consider the algebraic number $\alpha = 3^{1/4} - 2^{1/4}$, then the vector

$$[1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1]^T$$

is an integer relation for the vector

$$[1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}]^T$$

as the minimal polynomial for α is $x^{16} - 20x^{12} - 666x^8 - 3860x^4 + 1$. They state that PSLQ, with $\gamma = \sqrt{2}$, finds this relation with a working precision of 85 decimal digits while HJLS requires more than 10,000. As further evidence, they consider finding an integer relation for the vector $[11, 27, 31]^T$, a case when exact arithmetic can easily be used. Here, they show that for PSLQ with $\gamma = \sqrt{2}$, the successive iterations $k = 0, 1, 2, 3, 4$ yield the five A^{-1} matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 3 & 8 & 1 \\ -3 & -7 & -1 \end{bmatrix}, \begin{bmatrix} -2 & 1 & 0 \\ 2 & 3 & 1 \\ -1 & -3 & -1 \end{bmatrix},$$

$$\begin{bmatrix} 3 & -2 & 0 \\ 1 & 2 & 1 \\ -2 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -8 & -2 \\ 5 & 9 & 2 \\ -4 & -5 & -1 \end{bmatrix}.$$

The first two columns of the last matrix are integer relations for $[11, 27, 31]^T$. The HJLS algorithm requires 6 iterations for which iterations $k = 0, 1, 2, 3, 4, 5, 6$ produce the seven A^{-1} matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix},$$

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

With exact arithmetic, the PSLQ algorithm constructs two relations after only 4 iterations while the HJLS algorithm constructs only one and requires 6 iterations. As this appears to show that the PSLQ and HJLS algorithms behave differently, both in the case when inexact arithmetic is used and when exact arithmetic is used, the authors of [16] claim HJLS and PSLQ are different. They suggest that the difference may be due to the fact that PSLQ uses an orthogonal decomposition based on QR factorization while HJLS uses a decomposition which follows the classical method of Gram-Schmidt. This is not the case. The differences observed here are due to the partial reduction rather than full reduction done in the given implementation of HJLS. We shall show that HJLS with full reductions is equivalent to PSLQ with $\gamma = \sqrt{2}$ and then comment on these examples.

Following shortly, we will give the details for implementing HJLS with a full reduction at the end of each iteration, ensuring $|\mu_{i,j}| \leq 1/2$ for all $i > j$, $j = 1, \ldots, n-1$. With the intent of making it easier to show the two algorithms are equivalent, we make the reasonable assumption that no component of $\mathbf{x}$ equals zero and at each stage we normalize the vectors $\mathbf{a}_1^*, \mathbf{a}_2^*, \ldots, \mathbf{a}_{n-1}^*$. The vectors $\mathbf{h}_i = \mathbf{a}_i^*/\|\mathbf{a}_i^*\|$ for $1 \leq i \leq n-1$ form an orthonormal basis for $\mathbf{x}^\perp$ at the beginning of each iteration as none of the $\mathbf{a}_i^*$ are $\mathbf{0}$ for $1 \leq i \leq n-1$ until the end of the final iteration when $\mathbf{a}_{n-1}^* = \mathbf{0}$. Let H be the $n \times n$ matrix such that the ith column vector of H is $\mathbf{h}_i$ for $1 \leq i \leq n-1$ and the nth column of H is the zero vector. Then let M be the lower triangular matrix AH where $\mathbf{a}_i^T$ is the ith row vector of A. For each entry $m_{i,j}$ of M,

$$m_{i,j} = (\mathbf{a}_i \cdot \mathbf{h}_j) = \begin{cases} (\mathbf{a}_i \cdot \mathbf{a}_j^*/\|\mathbf{a}_j^*\|) & 1 \leq j \leq n-1 \\ 0 & j = n \end{cases}$$

and so

$$\mu_{i,j} = \begin{cases} m_{i,j}/\|\mathbf{a}_j^*\| & 1 \le j \le n-1 \\ m_{i,j} & j = n. \end{cases}$$

As $m_{i,i} = \|\mathbf{a}_i^*\|$ and $\|\mathbf{a}_n^*\| = 0$, selecting a value of r that maximizes $2^r\|\mathbf{a}_j^*\|^2$ for $1 \le i \le n$ is equivalent to selecting an r that maximizes $(\sqrt{2})^r m_{i,i}$ for $1 \le i \le n$–1 and ensuring $|\mu_{i,j}| \le 1/2$ is equivalent to ensuring $|m_{i,j}/m_{j,j}| \le 1/2$. Now with this choice of r, after exchanging $\mathbf{a}_r$ with $\mathbf{a}_{r+1}$ we need to update $\mathbf{h}_r$, $\mathbf{h}_{r+1}$ and the values $m_{i,j}$ to maintain $\mathbf{h}_i = \mathbf{a}_i^*/\|\mathbf{a}_i^*\|$ for $1 \le i \le n-1$ and $m_{i,j} = (\mathbf{a}_i \cdot \mathbf{h}_j)$.

First, suppose $r < n-1$. Let $\mathbf{a}_i^{(k)}$, $\mathbf{a}_i^{*(k)}$, $\mathbf{h}_i^{(k)}$, $\mu_{i,j}^{(k)}$, and $m_{i,j}^{(k)}$ equal $\mathbf{a}_i$, $\mathbf{a}_i^*$, $\mathbf{h}_i$, $\mu_{i,j}$, and $m_{i,j}$ at the beginning of the kth iteration and let a superscript $^{(k')}$ denote the variable after interchanging $\mathbf{a}_r$ with $\mathbf{a}_{r+1}$. Define α, β, λ and δ as follows:

$$\alpha = \|\mathbf{a}_r^{*(k)}\| = m_{r,r}^{(k)}, \quad \beta = \frac{(\mathbf{a}_{r+1}^{(k)} \cdot \mathbf{a}_r^{*(k)})}{\|\mathbf{a}_r^{*(k)}\|} = \mu_{r+1,r}^{(k)}\|\mathbf{a}_r^{*(k)}\| = m_{r+1,r}^{(k)},$$

$$\lambda = \|\mathbf{a}_{r+1}^{*(k)}\| = m_{r+1,r+1}^{(k)}, \quad \text{and} \quad \delta = \sqrt{\beta^2 + \lambda^2}.$$

After exchanging $\mathbf{a}_r$ with $\mathbf{a}_{r+1}$, $\mathbf{a}_r^{(k')} = \mathbf{a}_{r+1}^{(k)}$ and $\mathbf{a}_{r+1}^{(k')} = \mathbf{a}_r^{(k)}$. This gives

$$\begin{aligned}
\mathbf{a}_r^{*(k')} &= \mathbf{a}_{r+1}^{*(k)} + \mu_{r+1,r}^{(k)}\mathbf{a}_r^{*(k)} \\
\|\mathbf{a}_r^{*(k')}\| &= \sqrt{\|\mathbf{a}_{r+1}^{*(k)}\|^2 + (m_{r+1,r}^{(k)}/\|\mathbf{a}_r^{*(k)}\|)^2\|\mathbf{a}_r^{*(k)}\|^2} = \sqrt{\lambda^2+\beta^2} = \delta \\
\mathbf{h}_r^{(k')} &= \frac{\mathbf{a}_r^{*(k')}}{\delta} = \frac{\mathbf{h}_{r+1}^{*(k)}\|\mathbf{a}_{r+1}^{*(k)}\| + \mu_{r+1,r}^{(k)}\mathbf{h}_r^{(k)}\|\mathbf{a}_r^{*(k)}\|}{\delta} = \frac{\lambda\mathbf{h}_{r+1}^{(k)}}{\delta} + \frac{\beta\mathbf{h}_r^{(k)}}{\delta}
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{a}_{r+1}^{*(k')} &= \mathbf{a}_r^{*(k)} - \frac{(\mathbf{a}_{r+1}^{(k')} \cdot \mathbf{a}_r^{*(k')})}{\|\mathbf{a}_r^{*(k')}\|^2}\mathbf{a}_r^{*(k')} \\
&= \mathbf{a}_r^{*(k)} - \frac{(\mathbf{a}_r^{(k)} \cdot (\mathbf{a}_{r+1}^{*(k)} + \mu_{r+1,r}^{(k)}\mathbf{a}_r^{*(k)}))}{\|\mathbf{a}_{r+1}^{*(k)}\|^2 + (\mu_{r+1,r}^{(k)})^2\|\mathbf{a}_r^{*(k)}\|^2}(\mathbf{a}_{r+1}^{*(k)} + \mu_{r+1,r}^{(k)}\mathbf{a}_r^{*(k)}) \\
&= \frac{\mathbf{a}_r^{*(k)}\|\mathbf{a}_{r+1}^{*(k)}\|^2 - \mu_{r+1,r}^{(k)}\|\mathbf{a}_r^{*(k)}\|^2\mathbf{a}_{r+1}^{*(k)}}{\|\mathbf{a}_{r+1}^{*(k)}\|^2 + (\mu_{r+1,r}^{(k)})^2\|\mathbf{a}_r^{*(k)}\|^2} \\
&= \frac{\mathbf{h}_r^{(k)}\alpha\lambda^2 - \beta\alpha\lambda\mathbf{h}_{r+1}^{(k)}}{\delta^2} = \frac{\alpha\lambda}{\delta}\frac{(\lambda\mathbf{h}_r^{(k)} - \beta\mathbf{h}_{r+1}^{(k)})}{\delta} \\
\|\mathbf{a}_{r+1}^{*(k')}\| &= \frac{\alpha\lambda}{\delta} \\
\mathbf{h}_{r+1}^{(k')} &= \frac{\lambda\mathbf{h}_r^{(k)}}{\delta} - \frac{\beta\mathbf{h}_{r+1}^{(k)}}{\delta}.
\end{aligned}$$

If we define the $n \times n$ matrix S so that

$$s_{i,j} = \begin{cases} 1 & i = j \neq r \text{ or } r+1 \\ \beta/\delta & i = j = r \\ -\beta/\delta & i = j = r+1 \\ \lambda/\delta & i = r+1, j = r \text{ and } i = r, j = r+1 \\ 0 & otherwise \end{cases}$$

then $H^{(k)}S=$

$$\left[\mathbf{h}_1^{(k)}, \ldots, \mathbf{h}_{r-1}^{(k)}, \left(\frac{\beta}{\delta}\mathbf{h}_r^{(k)} + \frac{\lambda}{\delta}\mathbf{h}_{r+1}^{(k)}\right), \left(\frac{\lambda}{\delta}\mathbf{h}_r^{(k)} - \frac{\beta}{\delta}\mathbf{h}_{r+1}^{(k)}\right), \mathbf{h}_{r+2}^{(k)}, \ldots, \mathbf{h}_{n-1}^{(k)}, \mathbf{0}\right]$$

$$= \left[\mathbf{h}_1^{(k')}, \ldots, \mathbf{h}_{r-1}^{(k')}, \mathbf{h}_r^{(k')}, \mathbf{h}_{r+1}^{(k')}, \mathbf{h}_{r+2}^{(k')}, \ldots, \mathbf{h}_{n-1}^{(k')}, \mathbf{0}\right] = H^{(k')}.$$

As $H^{(k')} = H^{(k)}S$ and $A^{(k')} = EA^{(k)}$ where E is the $n \times n$ elementary matrix that interchanges rows r and $r+1$, $M^{(k')} = A^{(k')}H^{(k')} = EM^{(k)}S$.

Interchanging rows r and $r+1$ of M and then multiplying on the right by S and multiplying H on the right by S correctly updates the matrices M and H, maintaining both

$$\mathbf{h}_i^{(k')} = \begin{cases} \mathbf{a}_i^{*(k')}/\|\mathbf{a}_i^{*(k')}\| & \text{for} \quad 1 \leq i \leq n-1 \\ 0 & \text{for} \quad i = n \end{cases}$$

and

$$m_{i,j}^{(k')} = (\mathbf{a}_i^{(k')} \cdot \mathbf{h}_j^{(k')}) \text{ for } 1 \leq i, j \leq n.$$

Following this, we apply a full reduction to the matrix M to maintain $|\mu_{i,j}| = |m_{i,j}/m_{j,j}| \leq 1/2$. This reduces M in the same manner used to reduce the matrix H' in the PSLQ algorithm and does not change the vectors $\mathbf{a}_i^{*(k')}$. At the start of the $(k+1)$st iteration, $\mathbf{h}_i^{(k+1)} = \mathbf{h}_i^{(k')}$ for each i.

In the event that $r = n-1$, then either $\mathbf{a}_{n-1}^{*(k')} = \mathbf{0}$ or $\mathbf{a}_{n-1}^{*(k')} \neq \mathbf{0}$. If $\mathbf{a}_{n-1}^{*(k')}$ is not $\mathbf{0}$, then this reduces to the previous case. However, if $\mathbf{a}_{n-1}^{*(k')}$ is $\mathbf{0}$ then we no longer have $\mathbf{h}_{n-1}^{(k')} = \mathbf{a}_{n-1}^{*(k')}/\|\mathbf{a}_{n-1}^{*(k')}\|$. Although this is of little consequence as the algorithm will terminate after this iteration, we can avoid this situation altogether by modifying the termination condition. As $\mathbf{a}_{n-1}^{*(k')} = \mathbf{a}_n^{*(k)} + \mu_{n,n-1}^{(k)}\mathbf{a}_{n-1}^{*(k)}$, in order for $\mathbf{a}_{n-1}^{*(k')}$ to equal zero we must have had $\mu_{n,n-1}^{(k)} = 0$, or equivalently, $m_{n,n-1}^{(k)} = 0$. Rather than waiting for $\mathbf{a}_n^* \neq \mathbf{0}$ we can save a single iteration by terminating when $m_{n,n-1} = 0$ and $r = n-1$. With this modified termination condition, at the beginning and end of each iteration $\mathbf{h}_i = \mathbf{a}_i^*/\|\mathbf{a}_i^*\|$ for $1 \leq i \leq n-1$,

The HJLS algorithm with full reductions
This algorithm takes a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and a constant T as input. It either returns an integer relation for $\mathbf{x}$ along with a lower bound on the norm of a shortest integer relation or shows $\mathbf{x}$ has no integer relations of norm less than T.

Step 1 Initialization
Set $A = B = H = I_n$.
Let $\mathbf{a}_i$ be the ith column vector of A^T and $\mathbf{b}_i$ be the ith column vector of B.
Set $\mathbf{h}_0 = \mathbf{x}/\|\mathbf{x}\|$, entry $h_{n,n}$ of H to 0, and let $\mathbf{h}_i$ be the ith row vector of H^T.
For j from 1 to $n-1$ do (Gram-Schmidt)
set $\mathbf{h}_j = \mathbf{h}_j - \sum_{k=0}^{j-1}(\mathbf{h}_j \cdot \mathbf{h}_k)\mathbf{h}_k$
set $\mathbf{h}_j = \mathbf{h}_j/\|\mathbf{h}_j\|$
end do.
Set $M = AH = H$ as $A = I_n$.
Let $\mathbf{m}_i$ be the ith row vector of M and $m_{i,j}$ the (i,j)th entry of M.

Step 2 Size Reduction of M
For i from 2 to n do, for j from $i-1$ to 1 do
set $t = \lfloor m_{i,j}/m_{j,j} \rceil$
replace $\mathbf{a}_i$ with $\mathbf{a}_i - t\mathbf{a}_j$, $\mathbf{b}_j$ with $\mathbf{b}_j + t\mathbf{b}_i$, and $\mathbf{m}_j$ with $\mathbf{m}_i - t\mathbf{m}_j$
end do, end do.

Step 3: The Main Iteration
Choose r so that $\sqrt{2}^i m_{i,i}$ is maximal when $i = r$.
Repeat the following until either $1/\max(m_{i,i}) \geq T$ or both $m_{n,n-1}=0$ and $r=n-1$.

1. Let $\alpha = m_{r,r}$, $\beta = m_{r+1,r}$, $\lambda = m_{r+1,r+1}$, $\delta = \sqrt{\beta^2 + \lambda^2}$ and $S = I_{n-1}$.
   Set $s_{r,r} = \beta/\delta$, $s_{r+1,r} = \lambda/\delta$, $s_{r,r+1} = \lambda/\delta$, and $s_{r+1,r+1} = -\beta/\delta$.
2. Interchange rows $\mathbf{a}_r^T$ and $\mathbf{a}_{r+1}^T$ of A, columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B, and rows $\mathbf{m}_r$ and $\mathbf{m}_{r+1}$ of M.
3. Replace H with HS and M with MS.
4. Size reduce M as in Step 2.
5. Choose r so that $\sqrt{2}^i m_{i,i}$ is maximal as above.

Step 4: Return $1/\max(m_{i,i})$ as a lower bound on the norm of any integer relation for $\mathbf{x}$.
If $m_{n,n-1} = 0$ then return $\mathbf{b}_{n-1}$ as an integer relation for $\mathbf{x}$.

Figure 2.4: Pseudocode implementation of the HJLS algorithm with full reductions

$\mathbf{h}_n = \mathbf{0}$, and $m_{i,j} = (\mathbf{a}_i \cdot \mathbf{h}_j) = \mu_{i,j}\|\mathbf{a}_i^*\|$. It follows that the HJLS algorithm with full reductions is equivalent to the algorithm presented in Figure 2.4. When implemented using inexact arithmetic, some care must be taken when checking for the condition $m_{n,n-1} = 0$. As $m_{n,n-1} = (\mathbf{a}_n \cdot \mathbf{h}_{n-1})$, we will claim $m_{n,n-1} = 0$ if $(\frac{\mathbf{a}_n}{\|\mathbf{a}_n\|} \cdot \frac{\mathbf{h}_{n-1}}{\|\mathbf{h}_{n-1}\|}) = m_{n,n-1}/\|\mathbf{a}_n\| < \epsilon$ where the value ϵ depends on the level of precision being used.

Comparing this algorithm to the PSLQ algorithm in Figure 2.2 with $\gamma = \sqrt{2}$, one sees many similarities. In particular we have the following:

Theorem 2.8 *For each iteration, the matrix A from the HJLS algorithm with full reductions and the matrix A from the PSLQ algorithm are the same and up to sign, the column vectors of the matrix H from PSLQ are the same as the first $n-1$ column vectors of the matrix H from HJLS.*

Proof: Let $\mathbf{a}_i^{(H,k)}$ ($\mathbf{a}_i^{(P,k)}$) denote the ith column vector of the matrix A^T from the HJLS (PSLQ) algorithm at the beginning of the kth iteration, and let $\mathbf{h}_j^{(H,k)}$ ($\mathbf{h}_j^{(P,k)}$) denote the jth column vector of the matrix H from the HJLS (PSLQ) algorithm at the beginning of the kth iteration. Let a superscript of $k = 0$ denote the vector at the end of the step 1.

Initially, both algorithms set $A = I_n$ and so $\mathbf{a}_i^{(H,0)} = \mathbf{a}_i^{(P,0)}$ for $1 \le i \le n$. Now by construction, $\mathbf{h}_{n-1}^{(P,0)}$ is orthogonal to the vectors $\mathbf{x}, \mathbf{a}_1^{(P,0)}, \ldots, \mathbf{a}_{n-2}^{(P,0)}$ and $\mathbf{h}_{n-1}^{(H,0)}$ is orthogonal to the vectors $\mathbf{x}, \mathbf{a}_1^{(H,0)}, \ldots, \mathbf{a}_{n-2}^{(H,0)}$. As each $\mathbf{a}_i^{(H,0)} = \mathbf{a}_i^{(P,0)}$ and the vectors $\mathbf{x}, \mathbf{a}_1^{(H,0)}, \ldots, \mathbf{a}_{n-2}^{(H,0)}$ are linearly independent, we see that $\mathbf{h}_{n-1}^{(P,0)}$ is a scalar multiple of $\mathbf{h}_{n-1}^{(H,0)}$. As both are unit vectors and initially the diagonal elements of H in both PSLQ and HJLS are positive, $\mathbf{h}_{n-1}^{(P,0)} = \mathbf{h}_{n-1}^{(H,0)}$. Similarly, as the vector $\mathbf{h}_{n-2}^{(H,0)}$ is orthogonal to $\mathbf{x}, \mathbf{a}_1^{(H,0)}, \ldots, \mathbf{a}_{n-3}^{(H,0)}$, and $\mathbf{h}_{n-1}^{(H,0)}$ and the vector $\mathbf{h}_{n-2}^{(P,0)}$ is orthogonal to $\mathbf{x}, \mathbf{a}_1^{(P,0)}, \ldots, \mathbf{a}_{n-3}^{(P,0)}$, and $\mathbf{h}_{n-1}^{(P,0)}$ we have $\mathbf{h}_{n-2}^{(H,0)} = \mathbf{h}_{n-2}^{(P,0)}$. Continuing in this fashion, we see that $\mathbf{h}_i^{(H,0)} = \mathbf{h}_i^{(P,0)}$ for $1 \le i \le n-1$.

As $H'{=}AH$ in PSLQ, $M{=}AH$ in HJLS, and as the reduction in step 2 is the same in both algorithms, when we enter the main iteration for the first time we have $\mathbf{a}_i^{(H,1)} = \mathbf{a}_i^{(P,1)}$ for $1 \le i \le n$ and $\mathbf{h}_i^{(H,1)} = \mathbf{h}_i^{(P,1)}$ for $1 \le i \le n-1$.

Now suppose for some $k \ge 1$, $\mathbf{a}_i^{(H,k)} = \mathbf{a}_i^{(P,k)}$ for $1 \le i \le n$ and $\mathbf{h}_i^{(H,k)} = \pm\mathbf{h}_i^{(P,k)}$ for $1 \le i \le n-1$. Then in both algorithms, the same value is chosen for r. First, suppose $r < n-1$. Let $\alpha^{(H,k)}, \beta^{(H,k)}, \lambda^{(H,k)}$, and $\delta^{(H,k)}$ be the values assigned to $\alpha, \beta\,\lambda$, and δ in the HJLS algorithm and let $\alpha^{(P,k)}, \beta^{(P,k)}, \lambda^{(P,k)}$, and $\delta^{(P,k)}$ be the values assigned to $\alpha, \beta\,\lambda$, and δ in the PSLQ algorithm. One of the four conditions holds:

1. $\mathbf{h}_r^{(H,k)} = \mathbf{h}_r^{(P,k)}$, $\alpha^{(H,k)} = \alpha^{(P,k)}$, $\beta^{(H,k)} = \beta^{(P,k)}$,

$\mathbf{h}_{r+1}^{(H,k)} = \mathbf{h}_{r+1}^{(P,k)}$, $\lambda^{(H,k)} = \lambda^{(P,k)}$, $\delta^{(H,k)} = \delta^{(P,k)}$

2. $\mathbf{h}_{r}^{(H,k)} = \mathbf{h}_{r}^{(P,k)}$, $\alpha^{(H,k)} = \alpha^{(P,k)}$, $\beta^{(H,k)} = \beta^{(P,k)}$,
   $\mathbf{h}_{r+1}^{(H,k)} = -\mathbf{h}_{r+1}^{(P,k)}$, $\lambda^{(H,k)} = -\lambda^{(P,k)}$, $\delta^{(H,k)} = \delta^{(P,k)}$

3. $\mathbf{h}_{r}^{(H,k)} = -\mathbf{h}_{r}^{(P,k)}$, $\alpha^{(H,k)} = -\alpha^{(P,k)}$, $\beta^{(H,k)} = -\beta^{(P,k)}$,
   $\mathbf{h}_{r+1}^{(H,k)} = \mathbf{h}_{r+1}^{(P,k)}$, $\lambda^{(H,k)} = \lambda^{(P,k)}$, $\delta^{(H,k)} = \delta^{(P,k)}$

4. $\mathbf{h}_{r}^{(H,k)} = -\mathbf{h}_{r}^{(P,k)}$, $\alpha^{(H,k)} = -\alpha^{(P,k)}$, $\beta^{(H,k)} = -\beta^{(P,k)}$,
   $\mathbf{h}_{r+1}^{(H,k)} = -\mathbf{h}_{r+1}^{(P,k)}$, $\lambda^{(H,k)} = -\lambda^{(P,k)}$, $\delta^{(H,k)} = \delta^{(P,k)}$.

After exchanging $\mathbf{a}_{r}^{(P,k)}$ and $\mathbf{a}_{r+1}^{(P,k)}$, the PSLQ algorithm updates the vectors $\mathbf{h}_{r}^{(P,k+1)}$ and $\mathbf{h}_{r+1}^{(P,k+1)}$ as follows:

$$\mathbf{h}_{r}^{(P,k+1)} = \frac{\beta^{(P,k)}}{\delta^{(P,k)}}\mathbf{h}_{r}^{(P,k+1)} + \frac{\lambda^{(P,k)}}{\delta^{(P,k)}}\mathbf{h}_{r+1}^{(P,k+1)},$$

$$\mathbf{h}_{r+1}^{(P,k+1)} = \frac{-\lambda^{(P,k)}}{\delta^{(P,k)}}\mathbf{h}_{r}^{(P,k+1)} + \frac{\beta^{(P,k)}}{\delta^{(P,k)}}\mathbf{h}_{r+1}^{(P,k+1)}.$$

After exchanging $\mathbf{a}_{r}^{(H,k)}$ and $\mathbf{a}_{r+1}^{(H,k)}$, the HJLS algorithm updates the vectors $\mathbf{h}_{r}^{(H,k+1)}$ and $\mathbf{h}_{r+1}^{(H,k+1)}$ as follows:

$$\mathbf{h}_{r}^{(H,k+1)} = \frac{\beta^{(H,k)}}{\delta^{(H,k)}}\mathbf{h}_{r}^{(H,k+1)} + \frac{\lambda^{(H,k)}}{\delta^{(H,k)}}\mathbf{h}_{r+1}^{(H,k+1)},$$

$$\mathbf{h}_{r+1}^{(H,k+1)} = \frac{\lambda^{(H,k)}}{\delta^{(H,k)}}\mathbf{h}_{r}^{(H,k+1)} + \frac{-\beta^{(H,k)}}{\delta^{(H,k)}}\mathbf{h}_{r+1}^{(H,k+1)}.$$

One easily checks that in all 4 cases, $\mathbf{h}_{r}^{(H,k+1)} = \mathbf{h}_{r}^{(P,k+1)}$ and $\mathbf{h}_{r+1}^{(H,k+1)} = \pm\mathbf{h}_{r+1}^{(H,k+1)}$. As the other basis vectors for $\mathbf{x}^{\perp}$ do not change, $\mathbf{h}_{i}^{(H,k+1)} = \pm\mathbf{h}_{i}^{(P,k+1)}$ for $1 \leq i \leq n-1$. Now upon entering the reduction stage, which is the same in both algorithms, the column vectors of the matrix $H' = AH$ in the PSLQ algorithm are the same, up to sign, as the first $n-1$ column vectors of $M = AH$ in the HJLS algorithm. It follows that $\lfloor m_{i,j}/m_{j,j} \rfloor = \lfloor h'_{i,j}/h'_{j,j} \rfloor$ at each step of the reduction and so $\mathbf{a}_{i}^{(H,k+1)} = \mathbf{a}_{i}^{(P,k+1)}$ for $1 \leq i \leq n$.

If $r = n-1$, then the PSLQ algorithm does not change its H matrix and so $\mathbf{h}_{i}^{(P,k)} = \mathbf{h}_{i}^{(P,k+1)}$ for $1 \leq i \leq n-1$. However, the HJLS algorithm updates $\mathbf{h}_{n-1}^{(H,k+1)}$ and $\mathbf{h}_{n}^{(H,k+1)}$ as above. In this situation, $\lambda^{(H,k)} = 0$ and $\delta^{(H,k)} = \left|\beta^{(H,k)}\right|$ and so

$$\mathbf{h}_{n-1}^{(H,k+1)} = \frac{\beta^{(H,k)}}{\left|\beta^{(H,k)}\right|}\mathbf{h}_{n-1}^{(H,k)} = \pm\mathbf{h}_{n-1}^{(H,k)} = \pm\mathbf{h}_{n-1}^{(P,k)} = \pm\mathbf{h}_{n-1}^{(P,k+1)}.$$

We again have $\mathbf{h}_{i}^{(H,k+1)} = \pm\mathbf{h}_{i}^{(P,k+1)}$ for $1 \leq i \leq n-1$ and so as above, the Reduction step maintains $\mathbf{a}_{i}^{(H,k+1)} = \mathbf{a}_{i}^{(P,k+1)}$ for $1 \leq i \leq n$.

□

As $B = A^{-1}$ in both algorithms and as $M{=}AH$ in HJLS and $H'{=}AH$ in PSLQ, we see there is a direct relation between the matrices A, B, M, and H from HJLS and the matrices A, B, H', and H from PSLQ. At each iteration, both algorithms have the same (up to sign) orthonormal basis for $\mathbf{x}^{\perp}$, the same basis for the lattice $\mathbb{Z}^n$, and both update these bases in an identical fashion with the intent of making the basis vectors for $\mathbb{Z}^n$ converge to $\mathbf{x}$. Each step of one algorithm corresponds exactly with each step of the other. The PSLQ algorithm with $\gamma = \sqrt{2}$ is equivalent to the HJLS algorithm with full reductions.

If we apply the HJLS algorithm with full reductions to the two examples given at the beginning of this section, we get the following results:
For $\alpha = 3^{1/4} - 2^{1/4}$, the HJLS algorithm successfully recovers the integer relation

$$[1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1]^T$$

for the vector

$$[1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}]^T$$

using less than 85 digits of precision which is consistent with the results from the PSLQ algorithm.
For the vector $\mathbf{x} = [11, 27, 31]^T$, the HJLS algorithm requires 5 iterations to find the integer relation $[-1, 5, -4]^T$. Iterations $k = 0, 1, 2, 3, 4$ and 5 produce the six $B = A^{-1}$ matrices

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 3 & 8 & 1 \\ -3 & -7 & -1 \end{bmatrix}, \begin{bmatrix} -2 & 1 & 0 \\ 2 & 3 & 1 \\ -1 & -3 & -1 \end{bmatrix},$$

$$\begin{bmatrix} 3 & -2 & 0 \\ 1 & 2 & 1 \\ -2 & -1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & -8 & -2 \\ 5 & 9 & 2 \\ -4 & -5 & -1 \end{bmatrix}, \begin{bmatrix} -6 & -1 & -2 \\ -1 & 5 & 2 \\ 3 & -4 & -1 \end{bmatrix}.$$

Note that the first five matrices are identical to those given by the PSLQ algorithm. The extra matrix is due to the fact that the authors of PSLQ have modified their termination condition so that the algorithm terminates as soon as one of the column vectors of $B = A^{-1}$ is an integer relation for $\mathbf{x}$. Although doing this may save a few iterations and still return an integer relation, it causes us to lose the ability to say that the norm of the recovered relation

is no more than γ^{n-2} times as large as the norm of a smallest possible integer relation for $\mathbf{x}$. For instance, in this example, the integer relation $[-8, 9, -5]^T$ has a norm equal to $\sqrt{170}$ which is greater than $\sqrt{2}\sqrt{42}$ where $\sqrt{42}$ is the norm of $[-1, 5, -4]^T$. In this example, the shorter relation is also recovered but we have no guarantee in the general case that this will happen.

2.4 Practical implementations of the PSLQ algorithm

Given a set of values $x'_1, x'_2, \ldots, x'_n$ in symbolic form, it is often far too costly to find an integer relation using exact arithmetic operations. Instead, we would like to let the values $x_1, x_2, \ldots, x_n$ be good rational approximations of $x'_1, x'_2, \ldots, x'_n$ and attempt to find an integer relation for $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ using inexact arithmetic. As stated in [2], a simple information theory argument gives a lower bound on the number of digits of precision that must be used. If we wish to recover a relation $\mathbf{a} \in \mathbb{Z}^n$ with coefficients of at most d digits in size, then the coordinates of $\mathbf{x}$ must be given to at least nd digits. Although this simple lower bound is often too low, the PSLQ algorithm usually recovers a given relation using only about 15% more digits than this bound suggests are necessary. Now in the event that we do recover a suspected relation $\mathbf{a} \in \mathbb{Z}^n$ using inexact arithmetic, there is no guarantee that it is a true integer relation for $\mathbf{x}'$. If the values $x'_1, x'_2, \ldots, x'_n$ are computed to twice the precision and $\mathbf{a}$ still appears to be an integer relation, then this gives us strong reason to believe that it is. While this cannot prove that $\mathbf{a}$ is an integer relation for $\mathbf{x}'$, it suggests that it may be worthwhile to search for a rigorous proof to show that it is.

In what follows, a selection of implementations for PSLQ using inexact arithmetic are presented. In light of the previous section, everything that is said here can also be applied to the HJLS algorithm.

2.4.1 The basic algorithm

We begin with the basic implementation of the PSLQ algorithm, similar to that given in [16], [4], and [2]. The details are presented in Figure 2.5. Note that the only significant differences between this implementation and the statement of the algorithm given in Figure 2.2 are that here the matrices A and H have been omitted and the termination condition has been modified. Rather than waiting until $r = n-1$ and $h'_{n,n-1} = 0$, termination occurs as soon as $\mathbf{x} \cdot \mathbf{b}_j = 0$ for some column vector $\mathbf{b}_j$ of B. Although any returned relations tend to be small

The Basic PSLQ Algorithm
This algorithm takes a vector $\mathbf{x}^T = [x_1, x_2, \ldots, x_n]$ and a constant $T \geq 1$ as input. It either returns a suspected integer relation for $\mathbf{x}$ or a lower bound ($\geq T$) on the norm of any possible relation for $\mathbf{x}$.

Step 1: Initialization
Fix the constant γ so that $\gamma > \sqrt{4/3}$.
Let $B = I$ and let $\mathbf{b}_j$ the jth column vector of B.
Let $\mathbf{h}'_i$ be the ith row vector of H' where the entries of H' are defined as follows:

$$h'_{i,j} = \begin{cases} 0 & 1 \leq i < j \leq n-1 \\ s_{i+1}/s_i & 1 \leq i = j \leq n-1 \\ -x_i x_j / s_j s_{j+1} & 1 \leq j < i \leq n \end{cases} \quad \text{where } s_j^2 = \sum_{k=j}^{n} x_k^2$$

Set $\mathbf{y} = \mathbf{x}/\|\mathbf{x}\|$ and let y_i be the ith component of $\mathbf{y}$.

Step 2: Size Reduce H'
For i from 2 to n do, for j from $i-1$ downto 1 do
set $t = \lfloor h'_{i,j}/h'_{j,j} \rceil$
replace $\mathbf{b}_j$ with $\mathbf{b}_j + t\mathbf{b}_i$, $\mathbf{h}'_i$ with $\mathbf{h}'_i - t\mathbf{h}'_j$, and y_j with $y_j + ty_i$
end do, end do.

Step 3: The Main Iteration
Repeat the following until either $\frac{1}{\max|h'_{i,i}|} \geq T$ or $\min \frac{y_j}{\|\mathbf{b}_j\|} < \epsilon$:

1. Choose r so that $\gamma^i \left|h'_{i,i}\right|$ is maximal when $i = r$. Then interchange columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B, rows $\mathbf{h}'_r$ and $\mathbf{h}'_{r+1}$ of H', and entries y_r and y_{r+1} of $\mathbf{y}$.
2. If $r < n-1$ then
   set $\alpha = h'_{r+1,r}$, $\beta = h'_{r,r}$, $\gamma = h'_{r,r+1}$, and $\delta = \sqrt{\beta^2 + \lambda^2}$
   for i from r to n do
   set $t = h'_{i,r}$, $h'_{i,r} = \frac{\beta}{\delta} h'_{i,r} + \frac{\lambda}{\delta} h'_{i,r+1}$, and $h'_{i,r+1} = -\frac{\lambda}{\delta} t + \frac{\beta}{\delta} h'_{i,r+1}$
   end do.
3. Size reduce H'
   For i from r+1 to n do, for j from $\min(i-1, r+1)$ downto 1 do
   set $t = \lfloor h'_{i,j}/h'_{j,j} \rceil$
   replace $\mathbf{b}_j$ with $\mathbf{b}_j + t\mathbf{b}_i$, $\mathbf{h}'_i$ with $\mathbf{h}'_i - t\mathbf{h}'_j$ and y_j with $y_j + ty_i$
   end do, end do.

Step 4: If $\min \frac{y_j}{\|\mathbf{b}_j\|} < \epsilon$ then return the corresponding $\mathbf{b}_j$ as an integer relation for $\mathbf{x}$, otherwise return $\frac{1}{\max|h'_{i,i}|}$ as a lower bound on the norm of any relation for $\mathbf{x}$.

Figure 2.5: Pseudocode implementation of the basic PSLQ algorithm

in practice, this modified termination condition causes us to lose the ability to claim that they have norm no larger than $\gamma^{n-2}M$ where M is the norm of a smallest integer relation for $\mathbf{x}$.

Again, some care must be taken when checking to see if $\mathbf{x} \cdot \mathbf{b}_j = 0$. We shall claim $\mathbf{x} \cdot \mathbf{b}_j = 0$ when $\frac{\mathbf{x}}{\|\mathbf{x}\|} \cdot \frac{\mathbf{b}_j}{\|\mathbf{b}_j\|} < \epsilon$ where ϵ depends on the level of precision being used. If we look only at the values $y_j = \mathbf{b}_j \cdot \frac{\mathbf{x}}{\|\mathbf{x}\|}$ then we may miss a relation. If $\|\mathbf{b}_j\|$ is large enough, then $\mathbf{b}_j \cdot \frac{\mathbf{x}}{\|\mathbf{x}\|}$ may be larger than the given value of ϵ. As it is undesirable to require the calculation of the values $\|\mathbf{b}_j\|$ for $1 \leq j \leq n$ with each iteration, it is noted that in practice the values y_j tend to stay within a few orders of magnitude of each other and gradually decrease until one of the $\mathbf{b}_j$ is an integer relation for $\mathbf{x}$. As one would expect, when $\mathbf{b}_j \cdot \mathbf{x} = 0$ the corresponding value y_j suddenly decreases. Rather than checking all the values $y_j/\|\mathbf{b}_j\|$ to see if one is less than ϵ, we can select the value j such that y_j is minimal and look only at the corresponding value of $y_j/\|\mathbf{b}_j\|$.

2.4.2 Periodic reductions and the multi-pair algorithm

For a first improvement to the basic algorithm, we note that the full reductions of the matrix H' are the bottle neck. As the standard HJLS algorithm shows, we cannot omit the full reductions altogether as this causes severe numerical instability. We can, however, perform them periodically and still achieve good results. Rather than fully reducing the matrix H' at each step, we will perform a full reduction only when $r = n - 1$. If $r < n - 1$ then we will perform a partial reduction prior to exchanging $\mathbf{h}_r$ and $\mathbf{h}_{r+1}$ to ensure that $\left|h'_{r+1,r}\right| \leq \left|h'_{r,r}\right|/2$. The details of the PSLQ algorithm with periodic reductions are presented in Figure 2.6.

On a similar note, but with an eye towards a parallel implementation, Bailey and Broadhurst have introduced a variant of PSLQ which they call the Multi-Pair algorithm [2]. The idea behind this variant is to first select a number of disjoint pairs $(r_i, r_i + 1)$ and then perform the normal operations of PSLQ on each pair with $r = r_i$. One can easily do this in such a way that the operations from one pair do not affect the operations of another as the pairs are disjoint. In addition, they have also reordered the steps in which the full reduction of the matrix H' are performed. Even though it has been designed for a parallel implementation, when run on a single processor the Multi-Pair algorithm offers an improvement to the basic algorithm. However, in this case it amounts to little more than the PSLQ algorithm with periodic reductions and a poor selection procedure. The selection procedure

The PSLQ Algorithm with Periodic Reductions
This algorithm takes a vector $\mathbf{x}^T = [x_1, x_2, \ldots, x_n]$ and a constant $T \geq 1$ as input. It either returns a suspected integer relation for $\mathbf{x}$ or a lower bound ($\geq T$) on the norm of any possible relation for $\mathbf{x}$.

Step 1: Initialization (see Figure 2.5)

Step 2: Size Reduce H' (see Figure 2.5)

Step 3: The Main Iteration
Repeat the following until either $\frac{1}{\max|h'_{i,i}|} \geq T$ or $\min\frac{y_j}{\|\mathbf{b}_j\|} < \epsilon$:

1. Set doFullReduction=false.
   While doFullReduction=false do
   Choose r so that $\gamma^i \left|h'_{i,i}\right|$ is maximal when $i = r$.
   Set $t = \lfloor h'_{r+1,r}/h'_{r,r} \rceil$
   and let $\mathbf{b}_r = \mathbf{b}_r + t\mathbf{b}_{r+1}$, $\mathbf{h}'_{r+1} = \mathbf{h}'_{r+1} - t\mathbf{h}'_r$, and $y_r = y_r + ty_{r+1}$.
   Interchange columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B, rows $\mathbf{h}'_r$ and $\mathbf{h}'_{r+1}$ of H',
   and entries y_r and y_{r+1} of $\mathbf{y}$.
   If $r < n - 1$ then
   set $\alpha = h'_{r+1,r}$, $\beta = h'_{r,r}$, $\gamma = h'_{r,r+1}$, and $\delta = \sqrt{\beta^2 + \lambda^2}$
   for i from r to n do
   set $t = h'_{i,r}$, $h'_{i,r} = \frac{\beta}{\delta}h'_{i,r} + \frac{\lambda}{\delta}h'_{i,r+1}$,
   and let $h'_{i,r+1} = -\frac{\lambda}{\delta}t + \frac{\beta}{\delta}h'_{i,r+1}$
   end do
   else
   doFullReduction=true
   end if
   end do.
2. Size reduce H' as in Step 2 above.

Step 4: If $\min\frac{y_j}{\|\mathbf{b}_j\|} < \epsilon$ then return the corresponding $\mathbf{b}_j$ as an integer relation for $\mathbf{x}$, otherwise return $\frac{1}{\max|h'_{i,i}|}$ as a lower bound on the norm of any relation for $\mathbf{x}$.

Figure 2.6: Pseudocode implementation of the PSLQ algorithm with periodic full reductions

used in the algorithm is as follows:

1. Sort the entries of the (n-1)-long vector $\left\{\gamma^i \left|h'_{i,i}\right|\right\}$ in decreasing order, producing the sort indices.

2. Beginning at the sort index r_1 corresponding to the largest $\gamma^i \left|h'_{i,i}\right|$, select pairs of indices $(r_i, r_i + 1)$, where r_i is the sort index. If at any step either r_i or $r_i + 1$ has already been selected, pass to the next index in the list. Continue until either the maximum number of pairs desired has been selected, or the list is exhausted.

It has been reported that for certain problems, the Multi-Pair algorithm falls into a cycle. Examining the selection criterion, we see that although we pass to the next index in the list if either r_i or $r_i + 1$ has already been selected, we do allow pairs $(r_i, r_i + 1)$ where $r_i + 1$ has already been passed over. We allow pairs $(r_i, r_i + 1)$ where $\left|h'_{r_i,r_i}\right| \leq \gamma \left|h'_{r_i+1,r_i+1}\right|$. If we restrict ourselves to pairs where either $r_i = n - 1$ or $\left|h'_{r_i,r_i}\right| > \gamma \left|h'_{r_i+1,r_i+1}\right|$ then we cannot fall into a cycle. This can be shown by considering the product $\mathbf{P} = \prod_{i=1}^{n-1} \left|h'_{i,i}\right|^{n-i}$.

First, consider the pair $(r_i, r_i + 1)$ where $r_i < n - 1$. Let $\alpha = h'_{r_i,r_i}$, $\beta = h'_{r_i+1,r_i}$, $\lambda = h'_{r_i+1,r_i+1}$, and let $\delta = \sqrt{\beta^2 + \lambda^2}$. Then as $\beta^2 \leq \alpha^2/4$, $\lambda^2 < \alpha^2/\gamma^2$, and $1/\gamma^2 < 3/4$ we have

$$\frac{|\delta|^{n-r_i} |\alpha\lambda/\delta|^{n-r_i-1}}{|\alpha|^{n-r_i} |\lambda|^{n-r_i-1}} = \frac{|\delta|}{|\alpha|} = \frac{\sqrt{\beta^2 + \lambda^2}}{|\alpha|} < \frac{\sqrt{\alpha^2/4 + \alpha^2/\gamma^2}}{|\alpha|} = \sqrt{1/4 + 1/\gamma^2} < 1.$$

As the algorithm replaces $h'_{r_i,r_i} = \alpha$ and $h'_{r_i+1,r_i+1} = \lambda$ with the values δ and $-\alpha\lambda/\delta$, at the end of the iteration the term $\left|h'_{r_i,r_i}\right|^{n-r_i} \left|h'_{r_i+1,r_i+1}\right|^{n-r_i-1}$ in the product $\mathbf{P}$ has been decreased. In the case when $r_i = n - 1$, the algorithm simply exchanges rows $\mathbf{h}'_n$ and $\mathbf{h}'_{n-1}$ of H', resulting in a decrease of the value $\left|h'_{n-1,n-1}\right|$ by at least a factor of 2. Here, the term $\left|h'_{n-1,n-1}\right|$ of the product $\mathbf{P}$ decreases as well.

This shows that $\mathbf{P} = \prod_{i=1}^{n-1} |h'_i|^{n-i}$ decreases with each iteration of the multi-pair algorithm provided we add the restriction $\left|h'_{r_i,r_i}\right| > \gamma \left|h'_{r_i+1,r_i+1}\right|$ to the selection procedure. As the product $\mathbf{P}$ strictly decreases with each iteration, we cannot fall into a cycle.

2.4.3 A multi-level implementation

Although using periodic full reductions can reduce the time required to execute the PSLQ algorithm, run times can still be excessively long for large problems. As a further improvement we consider a multi-level implementation [2], an implementation in which the majority

of the calculations are done using low precision arithmetic. Even though this introduces a significant amount of additional overhead, it drastically reduces the time spent within the main iteration and results in an overall saving.

To apply such a scheme, we first perform the usual initialization and reduction steps to produce the full precision versions of B, H', and $\mathbf{y}$. We then let $\overline{H'}$ be the double precision equivalent of $H'/\max\left|h'_{i,j}\right|$, $\overline{\mathbf{y}}$ be the double precision equivalent of $\mathbf{y}/\min|y_i|$ and repeat the following until either a suspected relation is found or the desired bound on the norm of any possible integer relation is achieved:

1. Using only double precision arithmetic, set both $\overline{A}$ and $\overline{B}$ equal to I, size reduce $\overline{H'}$ as in Figure 2.5, and repeat the main iteration of PSLQ with periodic full reductions on the matrices $\overline{B}$, $\overline{H'}$, $\overline{A}$, and the vector $\overline{\mathbf{y}}$. Note that to maintain $\overline{A} = (\overline{B})^{-1}$, when we interchange columns $\overline{\mathbf{b}}_r$ and $\overline{\mathbf{b}}_{r+1}$ of $\overline{B}$, we must interchange rows $\overline{\mathbf{a}}_r^T$ and $\overline{\mathbf{a}}_{r+1}^T$ of $\overline{A}$ and when we set $\overline{\mathbf{b}}_j = \overline{\mathbf{b}}_j + t\overline{\mathbf{b}}_i$ we must set $\overline{\mathbf{a}}_i = \overline{\mathbf{a}}_i - t\overline{\mathbf{a}}_j$.
   Stop when either $\min|\overline{y}_i| < \overline{\epsilon}$, or $\max|\overline{a}_{i,j}| > \overline{\mu}$. A good choice for $\overline{\mu}$ is 10^{12} and a good choice for $\overline{\epsilon}$ is the maximum of 10^{-12} and the factor one must multiply the minimum $|y_i|$ value from the previous level of precision by in order for it to be less than the previous level's ϵ. The reason for requiring a bound on the integral entries of $\overline{A}$ is that if they become too large, then they can no longer be accurately stored.

2. Returning to full precision arithmetic, let $B = B\overline{B}$, $H' = \overline{A}H'$, and $\mathbf{y} = \mathbf{x}B$. Then let $\overline{\mathbf{y}}$ be the double precision equivalent of $\mathbf{y}/\min|y_i|$ and let $\overline{H'}$ be the double precision equivalent of $H'/\max\left|h'_{i,j}\right|$. Using double precision, perform an LQ factorization of the matrix $\overline{H'}$ and set $\overline{H'}$ equal to L.

When computing with double precision arithmetic, it is important to keep backup copies of the matrices $\overline{A}$ and $\overline{B}$. If at any stage the integer entries of either one of these matrices can no longer be stored accurately, both must be reset to their previous state before returning to the higher level of precision. After updating as in step 2 and size reducing H', a single run through the main iteration of iteration of PSLQ should be applied before returning to the lower level of precision.

It is of interest to note that the code is in some sense self-correcting. Even if some errors are introduced within the double precision loop, causing incorrect choices of t and r, we may recover from this. As long as the matrices $\overline{A}$ and $\overline{B}$ have integral entries and LQ factorization of $\overline{A}H'$ produces a matrix L for which the maximum diagonal entry has decreased, then we

have increased a lower bound on the norm of any possible integer relation for $\mathbf{x}$ and have everything we need to proceed. Even with small errors at the double precision level, we may still move forward and recover an integer relation or an appropriate lower bound.

This two-level scheme extends to a multi-level scheme with intermediate levels of precision in an easy fashion. To drop down one level of precision, define the lower precision versions of H' and $\mathbf{y}$ as above, set the lower precision version of $\mathbf{x}$ equal to the initial lower precision version of $\mathbf{y}$, and set the lower level versions of A and B equal to I. Then drop down to the lower level of precision, size reduce H' and continue as though it was the top-most level. When the stopping condition of step 1 is satisfied, but with values of ϵ and μ that more appropriately reflect the current level of precision, return to the previous level of precision and update as in step 2. The only difference being that if we are returning to an intermediate level of precision, we must also update the current matrix A by multiplying it on the left by the lower precision level's version of A.

The author's implementation of a multi-level scheme for real valued vectors $\mathbf{x}$ is the default version of PSLQ currently found in the computer algebra package *Maple*. The author's multi-level version for both complex and real valued vectors appears in Appendix B.6 and is set to appear in a future release of *Maple*.

2.4.4 A selection of timings for the various algorithms

To give a brief idea of the relative efficiency of the algorithms considered, we present a small selection of timings. The three algebraic numbers $\alpha_1 = 3^{1/4} - 2^{1/4}$, $\alpha_2 = 3^{1/5} + 2^{1/6}$ and $\alpha_3 = -3^{1/7} + 2^{1/7}$ were considered. Since these are algebraic numbers of degree 16, 30, and 49, there exist integer relations for the vectors $\mathbf{x}_i = [1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{n_i}]^T$ where $n_1 = 16, n_2 = 30$ and $n_3 = 49$. Both the time and number of digits required to recover these relations are presented in Table 2.1. These were found by first incrementing the number of digits by 50 until the correct relation was found, and then successively reducing the number of digits in decrements of 5 until an incorrect relation was returned. The time and number of digits used in the last correct run appear in the table. All algorithms were implemented in Maple 12 and run on a machine with an AMD Athlon 64 3200+ processor (2.0 GHz) running a 32 bit version of windows XP.

	$\alpha_1 = 3^{1/4} - 2^{1/4}$		$\alpha_2 = 3^{1/5} + 2^{1/6}$		$\alpha_3 = -3^{1/7} + 2^{1/7}$	
Algorithm	*Digits*	*Time*	*Digits*	*Time*	*Digits*	*Time*
LLL						
The Basic Algorithm	60	2.3 s	425	118.8 s	430	441.3 s
HJLS						
With full reductions	80	4.3 s	245	156.4 s	585	2586.2 s
PSLQ ($\gamma = \sqrt{2}$)						
The Basic Algorithm	80	2.6 s	245	88.8 s	590	1391.6 s
Periodic Full Reductions	80	1.2 s	245	25.1 s	590	269.3 s
Multi-level scheme	80	0.4 s	240	10.1 s	580	104.3 s
PSLQ ($\gamma = \sqrt{4/3}$)						
The Basic Algorithm	70	3.1 s	205	85.7 s	480	1110.0 s
Periodic Full Reductions	70	1.3 s	210	26.0 s	480	246.0 s
Multi-level scheme	70	0.4 s	200	9.2 s	470	81.6 s

Table 2.1: Selected timings for the various integer relation algorithms

2.5 Simultaneous integer relations

The final implementation of an integer relation algorithm to be considered is the simultaneous relations algorithm. Initially presented in [22] by the authors of HJLS, it is recast here within the framework of PSLQ. This is the procedure that will be used to take advantage of symmetry conditions when extending the method for computing nth integer Chebyshev polynomials on $[0,1]$ to the bivariate case on $[0,1]\times[0,1]$.

Definition 2.6 *The vector* $\mathbf{b}$ *is said to be a simultaneous integer relation for the set of vectors* $\{\mathbf{x}_i\}_{i=1}^t$ *if* $\mathbf{b}\cdot\mathbf{x}_i = 0$ *for* $1 \le i \le t$.

To find a simultaneous integer relation for the vectors $\{\mathbf{x}_i \in \mathbb{R}^n\}_{i=1}^t$, we first construct an orthonormal basis $\{\mathbf{h}_i\}_{i=1}^m$ for the space of vectors that are orthogonal to each $\mathbf{x}_j$. This is done in such a way so that if we form the matrix H with column vectors $\mathbf{h}_i$, then H is an $n\times(n-m)$ lower trapezoidal matrix (a reordering of the components of the $\mathbf{x}_i$ may be required, see Figure 2.7 for details). Setting $A = B = I$ and $H'{=}AH$, we then size reduce H' and run through the main iteration of the PSLQ algorithm to reduce the size of the projections of the column vectors of A^T onto the span of $\{\mathbf{h}_i\}_{i=1}^m$. The only real difference here is that instead of interchanging rows r and $r+1$ of H' when $r = n-m$, we interchange rows r and j of H' where $j > r$ is chosen so that $\left|h'_{j,n-m}\right|$ is maximal. Rows r and j of A and columns r and j of B must be interchanged as well. If after size reducing the entries

of H' we have $\left|h'_{k,n-m}\right| = 0$ for $n-m < k \leq n$, then the $(n-m)$th column vector $\mathbf{b}_{n-m}$ of B is a simultaneous integer relation for the vectors $\{\mathbf{x}_i\}_{i=1}^t$. The justification of this last statement follows in the same fashion as the equivalent statement for the standard PSLQ algorithm. For any $\mathbf{x}_j \in \{\mathbf{x}_i\}_{i=1}^t$, the vector $\mathbf{x}_j$ is orthogonal to each column vector of H and so $\mathbf{0} = \mathbf{x}_j^T H = \mathbf{x}_j^T BAH = \mathbf{x}_j^T BH'$. Since the only nonzero entry in the $(n-m)$th column of H' is $h'_{n-m,n-m}$, we have $\mathbf{x}_j^T \mathbf{b}_{n-m} h'_{n-m,n-m} = 0$ which reduces to $\mathbf{x}_j^T \mathbf{b}_{n-m} = 0$.

To find multiple simultaneous integer relations, when they exist, we can continue the process started above. After the kth relation is found, interchange rows $(n-m-k+1)$ and $(n-k+1)$ of H', rows $(n-m-k+1)$ and $(n-k+1)$ of A, and columns $(n-m-k+1)$ and $(n-k+1)$ of B. Then continue through the main loop of the PSLQ algorithm with the intent of reducing the size of the projections of the first $n-k$ column vectors of A^T onto the span of $\{\mathbf{h}_i\}_{i=1}^m$. During the iteration, when choosing r so that $\gamma^r |h'_{r,r}|$ is maximal we add the restriction $1 \leq r \leq n-m-k$. When $r = n-m-k$, we choose j with $r+1 \leq j \leq n-k$ so $|h'_{j,r}|$ is maximal and interchange rows r and j of H', rows r and j of A, and columns r and j of B. We then size reduce only the first $n-k$ rows of H' so as not to change the final k columns of B which are the previously found simultaneous integer relations for the $\mathbf{x}_i$. In the event that $h'_{j,n-m-k} = 0$ for $n-m-k+1 \leq j \leq n-k$, the column vector $\mathbf{b}_{n-m-k}$ of B is an simultaneous integer relation for the vectors $\{\mathbf{x}_i\}_{i=1}^t$.

The justification of this last statement follows in the same fashion as the above statement for the first simultaneous integer relation. For any $\mathbf{x}_j \in \{\mathbf{x}_i\}_{i=1}^t$, the vector $\mathbf{x}_j$ is orthogonal to each column vector of H and so $\mathbf{0} = \mathbf{x}_j^T H = \mathbf{x}_j^T BAH = \mathbf{x}_j^T BH'$. Since the only possible nonzero entries in the $(n-m-k)$th column of H' are $h'_{n-m-k,n-m-k}$ and $h'_{l,n-m-k}$ for $n-k+1 \leq l \leq n$, we have

$$\mathbf{x}_j^T \mathbf{b}_{n-m-k} h'_{n-m-k,n-m-k} + \sum_{l=n-k+1}^{n} \mathbf{x}_j^T \mathbf{b}_l h'_{l,n-m-k} = 0.$$

For $n-k+1 \leq l \leq n$, the vectors $\mathbf{b}_l$ are the previously found simultaneous integer relations for the vectors $\{\mathbf{x}_i\}_{i=1}^t$ and so the above equation reduces to $\mathbf{x}_j^T \mathbf{b}_{n-m} = 0$ since $h'_{n-m-k,n-m-k} \neq 0$.

For a practical implementation of the algorithm, we run the procedure until $m+k = n$, deeming the values $h'_{i,j}$ to be zero when their magnitudes are less than some ϵ chosen based on the level of precision, and then return the last k column vectors of B as suspected simultaneous integer relations. When used as a subprocedure in the method for finding

The PSLQ Algorithm for Simultaneous Integer Relations
This algorithm takes the vectors $\{\mathbf{x}_i^T = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]\}_{i=1}^t$ as input. It returns suspected simultaneous integer relations for the $\{\mathbf{x}_i\}_{i=1}^t$.

Step 1: Initialization

1. Apply the Gram-Schmidt orthonormalization process to the vectors $\mathbf{x}_1$,...,$\mathbf{x}_t$ to find an orthonormal basis $\mathbf{u}_1$,...,$\mathbf{u}_m$ for the span of $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$. If $m = t$ there are no simultaneous integer relations.
2. Construct an orthonormal basis for the space of vectors orthogonal to $\mathbf{x}_1$,...,$\mathbf{x}_t$. Let n be the number of components in $\mathbf{x}_1$ and let $\{\mathbf{e}_i\}_{i=1}^n$ be the standard basis for $\mathbb{R}^n$ where the jth component of $\mathbf{e}_i$ is 0 if $i \neq j$ and 1 if $i = j$. Set $i = j = 0$ and repeat the following until $j + m = n$.
   Set $i = i + 1$ and remove the component of $\mathbf{e}_i$ lying in the span of $\{\mathbf{u}_1, \ldots, \mathbf{u}_m, \mathbf{h}_1, \ldots, \mathbf{h}_{j-1}\}$. If the resulting vector $\mathbf{w}$ is zero, return to the top of the loop. Otherwise, set $\mathbf{h}_j = \mathbf{w}$. By construction, the first $j - 1$ components of $\mathbf{h}_j$ will equal zero. If the jth component of $\mathbf{h}_j$ is zero, choose k so that the kth component of $\mathbf{h}_j$ is nonzero and interchange jth and kth components of all current vectors (keep track of the required interchanges). Set $j = j + 1$.
3. Set $B = I$ and let H' be the $n \times (n - m)$ lower trapezoidal matrix with column vectors $\mathbf{h}_i$. Initially, $A = B = I$ and $H' = AH = H$. Let $\mathbf{h}'_i$ be the ith row vector of H' and size reduce H' as in step 2 of the basic PSLQ algorithm (but change the phrase "for j from $i - 1$" to "for j from $\min(i - 1, n - m)$". Finally, set $N = m$ and let $\gamma = 2/\sqrt{3}$.

Step 2: The Main Iteration

While $N < n$ do
  While $\max(|h_{n-N+1,n-N}|, \ldots, |h_{n-N+m,n-N}|) > 0$ do
    Select r with $1 \leq r \leq n - N$ so that $\gamma^i |h'_{i,i}|$ is maximal when $i = r$.
    If $r < n - N$, interchange columns $\mathbf{b}_r$ and $\mathbf{b}_{r+1}$ of B and rows $\mathbf{h}'_r$ and $\mathbf{h}'_{r+1}$ of H'. Then update H' as in step 3, part 2 of the basic PSLQ algorithm.
    If $r = n - N$, choose j with $r + 1 \leq j \leq r + m$ so $|h'_{j,r}|$ is maximal.
      Then interchange columns $\mathbf{b}_r$ and $\mathbf{b}_j$ of B and rows $\mathbf{h}'_r$ and $\mathbf{h}'_j$ of H' and size reduce the first $n-(N-m)$ rows of H' as in step 3, part 3 of the basic PSLQ algorithm but let i run from $r+1$ to $n-N+m$ and let j run from l downto 1 where $l = r + 1$ unless $i = r + 1$ or $r = n - N$ in which case $l = r$.
  end do.
  Column $\mathbf{b}_{n-N}$ is an integer relation for all input vectors $\mathbf{x}_i$. Interchange columns $\mathbf{b}_{n-N}$ and $\mathbf{b}_{n-N+m}$ of B and rows $\mathbf{h}'_{n-N}$ and $\mathbf{h}'_{n-N+m}$ of H'. Set $N = N + 1$.
end do.

Step 3: Return the last $n - m$ column vectors of B, resorting the components to account for any interchanges made in step 2.

Figure 2.7: Pseudocode implementation of the PSLQ algorithm for simultaneous integer relations

bivariate polynomials with integer coefficients of minimal supremum norm on $[0,1]\times[0,1]$, the input vectors $\{\mathbf{x}_i\}_{i=1}^{t}$ will have rational components. This allows for exact arithmetic to be used to verify that the returned vectors $\{\mathbf{b}_i\}_{i=m+1}^{n}$ are in fact simultaneous integer relations for the vectors $\{\mathbf{x}_i\}_{i=1}^{t}$ and that the vectors $\{\mathbf{b}_i\}_{i=m+1}^{n} \bigcup \{\mathbf{x}_i\}_{i=1}^{t}$ span $\mathbb{R}^n$. Once we have verified that these vectors span $\mathbb{R}^n$, we can claim that any simultaneous integer relation $\mathbf{r}$ for the vectors $\{\mathbf{x}_i\}_{i=1}^{t}$ can be expressed in the form $\mathbf{r} = \sum_{i=m+1}^{n} e_i \mathbf{b}_i$. Now by Theorem 2.1, since the column vectors of B form a basis for the lattice $\mathbb{Z}^n$, we can also express $\mathbf{r}$ in the form $\mathbf{r} = \sum_{i=1}^{n} d_i \mathbf{b}_i$ where the d_i are integers. Since the vectors $\{\mathbf{b}_i\}_{i=m+1}^{n}$ are linearly independent, if both of these expressions hold we must have $d_i = 0$ for $i \leq m$. If we verify the above two properties, we can claim that the vectors $\{\mathbf{b}_i\}_{i=m+1}^{n}$ form a basis for the lattice of all simultaneous integer relations to the vectors $\{\mathbf{x}_i\}_{i=1}^{t}$.

Chapter 3

The Bivariate Case on the Unit Square

The extension of the integer Chebyshev problem to the bivariate case is a relatively unexplored problem. Just as there are many ways to extend the study of monic polynomials of minimal deviation from zero on an interval to a region in the plane [7], the extension of the integer Chebyshev problem can be done in many ways. Here the focus is on the extension to rectangular regions in the plane with special attention given to the unit square. Two such extensions, which differ in the way one measures the degree of a bivariate polynomial, are considered.

Definition 3.1 *The total degree of the monomial $x^i y^j$ is defined as $i+j$ and the total degree of the polynomial $p(x,y)$ is the maximum of the total degrees of its monomial parts with nonzero coefficients. Let $\mathcal{Z}_{n,total}[x,y]$ be the set of polynomials of total degree at most n with integer coefficients. For $n>0$, define $\mathrm{T}_n[a,b]\times[c,d]$ as*

$$\mathrm{T}_n[a,b]\times[c,d] = \left(\min_{p\in\mathcal{Z}_{n,total}[x,y]\setminus\{0\}} \|p(x,y)\|_{[a,b]\times[c,d]} \right)^{1/n}$$

and let

$$\mathrm{T}[a,b]\times[c,d] = \lim_{n\to\infty} \mathrm{T}_n[a,b]\times[c,d].$$

We call $\mathrm{T}[a,b]\times[c,d]$ the total degree integer Chebyshev constant for the region $[a,b]\times[c,d]$. Any polynomial $p(x,y)\in\mathcal{Z}_{n,total}[x,y]$ that satisfies $(\|p(x,y)\|_{[a,b]\times[c,d]})^{1/n} = \mathrm{T}_n[a,b]\times[c,d]$ is called a total degree nth integer Chebyshev polynomial on $[a,b]\times[c,d]$.

Definition 3.2 *The maximum degree of the monomial $x^i y^j$ is defined as the maximum of i and j and the maximum degree of the polynomial $p(x,y)$ is the largest maximum degree of its monomial parts with nonzero coefficients. Let $\mathcal{Z}_{n,max}[x,y]$ be the set of polynomials of maximum degree at most n with integer coefficients. For $n > 0$, define $\mathrm{M}_n[a,b]\times[c,d]$ as*

$$\mathrm{M}_n[a,b]\times[c,d] = \left(\min_{p\in\mathcal{Z}_{n,max}[x,y]\setminus\{0\}} \|p(x,y)\|_{[a,b]\times[c,d]} \right)^{1/n}$$

and let

$$\mathrm{M}[a,b]\times[c,d] = \lim_{n\to\infty} \mathrm{M}_n[a,b]\times[c,d].$$

We call $\mathrm{M}[a,b]\times[c,d]$ the maximum degree integer Chebyshev constant for the region $[a,b]\times[c,d]$. Any polynomial $p(x,y) \in \mathcal{Z}_{n,max}[x,y]$ that satisfies $(\|p(x,y)\|_{[a,b]\times[c,d]})^{1/n} = \mathrm{M}_n[a,b]\times[c,d]$ is called a maximum degree nth integer Chebyshev polynomial on $[a,b]\times[c,d]$.

The use of the minimum function in place of the infimum in the definitions of $\mathrm{T}_n[a,b]\times[c,d]$ and $\mathrm{M}_n[a,b]\times[c,d]$ can be justified using bivariate polynomial interpolation. Although the question as to whether or not the bivariate polynomial interpolation problem has a unique solution for an arbitrarily distributed set of nodes is nontrivial, a unique solution can be ensured by an appropriate choice of nodes [30]. One such configuration of points when interpolating with a bivariate polynomial of total degree at most n is the triangular array left when removing the grid points (x_i, y_j) with $i + j \geq n + 3$ from the rectangular grid $\{(x_i, y_j)\}_{i,j=1}^{n+1}$ with $x_1 < x_2 < \ldots < x_{n+1}$ and $y_1 < y_2 < \ldots < y_{n+1}$. Since a bivariate polynomial of total degree at most n can be recovered from its values at such a set of points, this can be used to show there are only finitely many $p(x,y) \in \mathcal{Z}_{n,total}[x,y]$ with $\|p(x,y)\|_{[a,b]\times[c,d]} \leq k$ for any constant k. If $(x_i, y_j) = (q/r, s/t) \in [a,b]\times[c,d]$ is a rational point, $u = \mathrm{lcm}(r,t)$, and $p(x,y)$ satisfies the desired conditions, then $u^n p(x_i, y_j)$ is an integer of norm less than $u^n k$ and so the set of allowable values of $p(x_i, y_j)$ is finite. Choosing rational values for all the x_i and y_j gives the result since each polynomial $p(x,y)$ satisfying the required conditions corresponds to one of the finitely many allowable combinations of values at the (x_i, y_j). Noting that a bivariate polynomial of maximum degree n is a bivariate polynomial of total degree at most $2n$ gives the desired result for the maximum degree case.

3.1 Existence of $\mathrm{T}[a,b]\times[c,d]$ and $\mathrm{M}[a,b]\times[c,d]$

The existence of the limits in the definitions of $\mathrm{T}[a,b]\times[c,d]$ and $\mathrm{M}[a,b]\times[c,d]$ are shown in the same way as the existence of the limit in the definition of $\Omega[a,b]$ from Section 1.2. The total degree case is considered first.

Given any positive integers n and N, let $p_n(x,y)$ be a total degree nth integer Chebyshev polynomial for the region $[a,b]\times[c,d]$ and write $N = nq+r$ with $0 \le r < n$ using the division algorithm. Then $p(x,y)^q x^r$ is a polynomial in $\mathcal{Z}_{N,total}[x,y]$ and so

$$\begin{aligned}(\mathrm{T}_N[a,b]\times[c,d])^N &\le \|p(x,y)^q x^r\|_{[a,b]\times[c,d]} \le \|p(x,y)\|^q_{[a,b]\times[c,d]}\,\|x\|^r_{[a,b]\times[c,d]} \\ &\le (\mathrm{T}_n[a,b]\times[c,d])^{qn}\,t^r = (\mathrm{T}_n[a,b]\times[c,d])^{N-r}\,t^r\end{aligned}$$

where $t = \max\left(\|x\|_{[a,b]\times[c,d]}, \mathrm{T}_n[a,b]\times[c,d]\right)$. Since $t/\mathrm{T}_n[a,b]\times[c,d] \ge 1$, this gives

$$\begin{aligned}\mathrm{T}_N[a,b]\times[c,d] \le (\mathrm{T}_n[a,b]\times[c,d])^{1-r/N}\,t^{r/N} &= \mathrm{T}_n[a,b]\times[c,d]\left(\frac{t}{\mathrm{T}_n[a,b]\times[c,d]}\right)^{r/N} \\ &\le \mathrm{T}_n[a,b]\times[c,d]\left(\frac{t}{\mathrm{T}_n[a,b]\times[c,d]}\right)^{n/N}.\end{aligned}$$

It follows that $\limsup_{N\to\infty} \mathrm{T}_N[a,b]\times[c,d] \le \mathrm{T}_n[a,b]\times[c,d]$. On the other hand, as n can be chosen so that $\mathrm{T}_n[a,b]\times[c,d]$ is arbitrarily close to the limit inferior of the sequence $\{\mathrm{T}_N[a,b]\times[c,d]\}$, we must have that $\limsup\ \mathrm{T}_N[a,b]\times[c,d] \le \liminf\ \mathrm{T}_N[a,b]\times[c,d]$ and so $\mathrm{T}[a,b]\times[c,d] = \lim_{N\to\infty} \mathrm{T}_N[a,b]\times[c,d]$ exists.

For the maximum degree case, only superficial changes are required. Given any positive integers n and N, let $p_n(x,y)$ be a maximum degree nth integer Chebyshev polynomial for the region $[a,b]\times[c,d]$ and write $N = nq+r$ using the division algorithm. Then $p(x,y)^q(xy)^r$ is a polynomial in $\mathcal{Z}_{N,max}[x,y]$ and so the above argument gives

$$\mathrm{M}_N[a,b]\times[c,d] \le \mathrm{M}_n[a,b]\times[c,d]\left(\frac{t}{\mathrm{M}_n[a,b]\times[c,d]}\right)^{n/N}$$

where $t = \max\left(\|xy\|_{[a,b]\times[c,d]}, \mathrm{M}_n[a,b]\times[c,d]\right)$. As in the total degree case, this leads to $\limsup_{N\to\infty} \mathrm{M}_N[a,b]\times[c,d] \le \mathrm{M}_n[a,b]\times[c,d]$ and the existence of $\lim_{N\to\infty} \mathrm{M}_N[a,b]\times[c,d]$.

3.2 Initial bounds on $\mathrm{T}[0,1]\times[0,1]$ and $\mathrm{M}[0,1]\times[0,1]$

The method of Gelfond and Schnirelman which led to the initial bound on $\Omega[0,1]$ can be extended to the bivariate case. For the total degree case, let

$$\mathrm{I} = \int_0^1 \int_0^1 p^2(x,y)dxdy \text{ where } p(x,y) \in \mathcal{Z}_{n,total}[x,y] \setminus \{0\}$$

and define d_n as $d_n = \mathrm{lcm}\{1,2,\ldots,n\}$. For a monomial $x^i y^j$ of total degree at most $2n$, one of i and j must be less than or equal to n and so $d_{n+1}d_{2n+1}\int_0^1\int_0^1 x^i y^j dxdy \in \mathbb{Z}$. Applying this result to I shows $\mathrm{I}\, d_{n+1}d_{2n+1}$ is a positive integer. Additionally, for any prime q dividing d_n, q must be less than n and the power of q dividing d_n is $\lfloor \log_q(n) \rfloor \leq \ln(n)/\ln(q)$. This gives us the following inequality:

$$1 \;\leq\; \mathrm{I}\, d_{n+1}d_{2n+1} \;\leq\; \mathrm{I} \prod_{\substack{q\leq n+1 \\ q \text{ prime}}} q^{\frac{\ln(n+1)}{\ln q}} \prod_{\substack{q\leq 2n+1 \\ q \text{ prime}}} q^{\frac{\ln(2n+1)}{\ln q}}.$$

Taking the natural logarithm of the outermost quantities yields

$$0 \leq \ln(\mathrm{I}) + \sum_{\substack{q\leq n+1 \\ q \text{ prime}}} \ln(n+1) + \sum_{\substack{q\leq 2n+1 \\ q \text{ prime}}} \ln(2n+1) = \ln(\mathrm{I}) + \ln(n+1)\pi(n+1) + \ln(2n+1)\pi(2n+1).$$

If we now note that $\mathrm{I} = \int_0^1\int_0^1 p^2(x,y)dxdy \leq \left\|p^2(x,y)\right\|_{[0,1]\times[0,1]}$, we see

$$\ln(\mathrm{I}) \leq 2n \ln\left(\left\|\left(p^2(x,y)\right)\right\|_{[0,1]\times[0,1]}^{1/2n}\right)$$

or

$$-\left(\frac{1}{2}\frac{\ln(n+1)\pi(n+1)}{n} + \frac{\ln(2n+1)\pi(2n+1)}{2n}\right) \leq \frac{\ln(I)}{2n} \leq \ln\left(\|p(x,y)\|_{[0,1]\times[0,1]}^{1/n}\right).$$

Choosing $p(x,y)$ so that $\|p(x,y)\|_{[0,1]\times[0,1]}^{1/n} = \mathrm{T}_n[0,1]\times[0,1]$ and exponentiating gives

$$\exp\left(-\frac{1}{2}\frac{\ln(n+1)\pi(n+1)}{n} - \frac{\ln(2n+1)\pi(2n+1)}{2n}\right) \leq \mathrm{T}_n[0,1]\times[0,1].$$

Finally, letting n tend to infinity and applying the prime number theorem gives the result we are after,

$$e^{-3/2} \;\leq\; \mathrm{T}[0,1]\times[0,1].$$

For the maximum degree case, applying the same argument to

$$d_{2n+1}d_{2n+1}\int_0^1\int_0^1 p^2(x,y)dxdy \text{ where } p(x,y)\in\mathcal{Z}_{n,max}[x,y]\setminus\{0\}$$

leads to

$$e^{-2} \;\leq\; \mathrm{M}[0,1]\times[0,1].$$

Easy upper bounds on $\mathrm{T}[0,1]\times[0,1]$ and $\mathrm{M}[0,1]\times[0,1]$ follow from the work of Section 3.1 where it is shown these limits exist. For any polynomials $p(x,y)\in\mathcal{Z}_{n,total}[x,y]\setminus\{0\}$ and $q(x,y)\in\mathcal{Z}_{n,max}[x,y]\setminus\{0\}$,

$$\mathrm{T}[0,1]\times[0,1]=\limsup_{N\to\infty}\mathrm{T}_N[a,b]\times[c,d]\leq\mathrm{T}_n[a,b]\times[c,d]\leq\|p(x,y)\|^{1/n}_{[0,1]\times[0,1]}$$

and

$$\mathrm{M}[0,1]\times[0,1]=\limsup_{N\to\infty}\mathrm{M}_N[a,b]\times[c,d]\leq\mathrm{M}_n[a,b]\times[c,d]\leq\|q(x,y)\|^{1/n}_{[0,1]\times[0,1]}.$$

These results are summed up in the following theorem.

Theorem 3.1 *For any nonzero polynomials $p(x,y)\in\mathcal{Z}_{n,total}[x,y]$ and $q(x,y)\in\mathcal{Z}_{n,max}[x,y]$,*

$$e^{-3/2}\leq\mathrm{T}[0,1]\times[0,1]\leq\|p(x,y)\|^{1/n}_{[0,1]\times[0,1]}$$

and

$$e^{-2}\leq\mathrm{M}[0,1]\times[0,1]\leq\|q(x,y)\|^{1/n}_{[0,1]\times[0,1]}.$$

An initial upper bound on $\mathrm{M}[0,1]\times[0,1]$ follows from the work in the single variable case. For any polynomial $p(x)$ of degree n, $p(x)p(y)$ is a polynomial of maximum degree n and so if we choose $p(x)$ to be an nth integer Chebyshev polynomial on $[0,1]$,

$$\mathrm{M}_n[0,1]\times[0,1]\leq\|p(x)p(y)\|_{[0,1]\times[0,1]}=\|p(x)\|_{[0,1]}\|p(y)\|_{[0,1]}=\Omega_n[0,1]^2.$$

Taking the limit as n tends to infinity gives

$$\mathrm{M}[0,1]\times[0,1]\leq\Omega[0,1]^2\leq(1/2.36482727)^2.$$

It would be nice if one could show that one of $x-y$ and $1-x-y$ does not divide a maximum degree nth integer Chebyshev polynomial on $[0,1]\times[0,1]$ for sufficiently large n. In this case, if $p_n(x,y)$ were such a polynomial then one of the polynomials $p_n(x,x)$ and $p_n(x,1-x)$ would be a nonzero polynomial in $\mathcal{Z}_{2n}[x]$ with supremum norm on the interval $[0,1]$ at most $(\mathrm{M}_n[0,1]\times[0,1])^n$. This would imply that $\Omega_{2n}[0,1]^{2n}\leq(\mathrm{M}_n[0,1]\times[0,1])^n$ for sufficiently large n, or $\Omega[0,1]^2\leq\mathrm{M}[0,1]\times[0,1]$.

3.3 Symmetry conditions on $[0,1]\times[0,1]$

The following theorems on symmetry are very useful when computing bivariate integer Chebyshev polynomials. The first, in conjunction with the PSLQ algorithm, allows for a reduction in the dimension of the search space in the total degree case and the second allows us to work on the region $[0,1/4]\times[0,1/4]$ when considering the maximum degree case.

Theorem 3.2 *For any nonnegative integer n, there is a total degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ satisfying the condition $Q(x,y) = (-1)^n Q(1-x,1-y)$.*

Proof: Let $p(x,y)$ be a total degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ and define the polynomials $Q_1(x,y)$ and $Q_2(x,y)$ as follows.

$$Q_1(x,y) = xp(x,y) + (-1)^n(1-x)p(1-x,1-y)$$

$$Q_2(x,y) = (1-x)p(x,y) + (-1)^n xp(1-x,1-y)$$

By considering what happens to a monomial $x^i y^j$ of total degree n, one sees that $Q_1(x,y)$ and $Q_2(x,y)$ are polynomials of total degree at most n and since

$$xQ_1(x,y) - (1-x)Q_2(x,y) = (2x-1)p(x,y) \neq 0,$$

at least one of $Q_1(x,y)$ and $Q_2(x,y)$ is not the zero polynomial. If we let $Q_0(x,y)$ be a nonzero $Q_i(x,y)$, then for any $(x_0,y_0)\in[0,1]\times[0,1]$,

$$|Q_0(x_0,y_0)| \leq |x_0| \left\|p(x,y)\right\|_{[0,1]\times[0,1]} + |1-x_0| \left\|p(x,y)\right\|_{[0,1]\times[0,1]} = \left\|p(x,y)\right\|_{[0,1]\times[0,1]}.$$

As $p(x,y)$ is a total degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$, we must have equality for at least one point $(x_0,y_0)\in[0,1]\times[0,1]$ which gives

$$\left\|Q_0(x,y)\right\|_{[0,1]\times[0,1]} = \left\|p(x,y)\right\|_{[0,1]\times[0,1]}.$$

Finally, by construction the polynomial $Q_0(x,y)$ satisfies $Q_0(x,y) = (-1)^n Q_0(1-x,1-y)$ and so satisfies all the required conditions.

□

Theorem 3.3 *For any nonnegative integer n, there is a maximum degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ satisfying the conditions $Q(x,y) = (-1)^n Q(1-x,y)$ and $Q(x,y) = (-1)^n Q(x,1-y)$.*

Proof: Let $p(x, y)$ be a maximum degree nth integer Chebyshev polynomial for the region $[0, 1]\times[0, 1]$ and define the polynomials $Q_1(x, y)$ and $Q_2(x, y)$ as follows.

$$Q_1(x, y) = xp(x, y) + (-1)^n(1 - x)p(1 - x, y)$$

$$Q_2(x, y) = (1 - x)p(x, y) + (-1)^n xp(1 - x, y)$$

By considering what happens to the monomial $x^n y^j$ of maximum degree n, one sees that $Q_1(x, y)$ and $Q_2(x, y)$ are polynomials of maximum degree at most n. Repeating the argument in the proof of Theorem 3.2 gives a nonzero polynomials $Q_0(x, y)$ satisfying the properties

$$\|Q_0(x, y)\|_{[0,1]\times[0,1]} = \|p(x, y)\|_{[0,1]\times[0,1]} \quad \text{and} \quad Q_0(x, y) = (-1)^n Q_0(1 - x, y).$$

Defining the polynomials $\tilde{Q}_1(x, y)$ and $\tilde{Q}_2(x, y)$ as

$$\tilde{Q}_1(x, y) = yQ_0(x, y) + (-1)^n(1 - y)Q_0(x, 1 - y)$$

and

$$\tilde{Q}_2(x, y) = (1 - y)Q_0(x, y) + (-1)^n yQ_0(x, 1 - y),$$

and repeating the argument a final time gives a maximum degree nth integer Chebyshev polynomial for the region $[0, 1]\times[0, 1]$ satisfying the conditions $\tilde{Q}_0(x, y) = (-1)^n\tilde{Q}_0(1-x, y)$ and $\tilde{Q}_0(x, y) = (-1)^n\tilde{Q}_0(x, 1 - y)$.

□

3.4 Computing bivariate nth integer Chebyshev polynomials

The method used to compute nth integer Chebyshev polynomials on the interval $[0, 1]$ carries forward to the bivariate case with relatively few modifications. The largest challenges are the loss of the condition used in the single variable case to test for necessary divisors and the inability to reduce the problem to one on $[0, 1/4]\times[0, 1/4]$ in the total degree case.

3.4.1 The total degree case

The process of finding total degree nth integer Chebyshev polynomials on $[0, 1]\times[0, 1]$ begins in the same fashion as the process on the interval $[0, 1]$. Starting with the upper bound on

$\|p_n(x,y)\|_{[0,1]\times[0,1]}$ of

$$\|p_n(x,y)\|_{[0,1]\times[0,1]} \leq c_n = \min_{0<k<n} \|p_k(x,y)p_{n-k}(x,y)\|_{[0,1]\times[0,1]}$$

where $p_i(x,y)$ is a total degree ith integer Chebyshev polynomial on the region $[0,1]\times[0,1]$, we move on to the step of finding necessary factors. Although the process used in the single variable case does not carry forward, if $c_n < \Omega_n[0,1]$ then

$$xy(1-x)(1-y)(x-y)(1-x-y) \text{ divides } p_n(x,y).$$

For if $c_n < \Omega_n[0,1]$, then $p_n(0,y) = 0$, $p_n(1,y) = 0$, $p_n(x,0) = 0$, $p_n(x,1) = 0$, $p_n(x,x) = 0$, and $p_n(x,1-x) = 0$ since they are all single variable polynomials of degree at most n with supremum norm less than $\Omega_n[0,1]$ on the interval $[0,1]$. This gives an infinite number of points of intersection between the curve $p_n(x,y) = 0$ and each of the curves $x = 0$, $1-x = 0$, $y = 0$, $1-y = 0$, $x-y = 0$, and $1-x-y = 0$. Since the linear polynomials x, $1-x$, y, $1-y$, $x-y$, and $1-x-y$ are all irreducible, they must all divide $p_n(x,y)$. We are just applying a weak form of Bezout's theorem [11], if the polynomials $f(x,y)$ and $g(x,y)$ have no common factors, then number of points of intersection of $\mathrm{V}(f(x,y)) = \{(x,y) : f(x,y) = 0\}$ and $\mathrm{V}(g(x,y)) = \{(x,y) : g(x,y) = 0\}$ is finite.

To take advantage of the fact that there must be a total degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ satisfying the condition $p_n(x,y) = (-1)^n p_n(1-x,1-y)$, the version of PSLQ given in Section 2.5 is used. If $p_n(x,y) = F(x,y)G(x,y)$ where $F(x,y)$ represents the known divisors of $p_n(x,y)$ and $G(x,y) = \sum a_{i,j}x^i y^j$ represents the unknown factor of total degree g, the symmetry condition gives

$$F(x,y)\sum a_{i,j}x^i y^j \;=\; (-1)^n F(1-x,1-y)\sum a_{i,j}(1-x)^i(1-y)^j$$

or

$$\sum a_{i,j}d_{i,j}(x,y) = \sum a_{i,j}\left(F(x,y)x^i y^j - (-1)^n F(1-x,1-y)(1-x)^i(1-y)^j\right) = 0.$$

For any point (x,y), the coefficients $a_{i,j}$ of $G(x,y)$ give an integer relation between the values $d_{i,j}(x,y)$. Letting $\mathbf{a} = [a_{0,0}, a_{1,0}, a_{0,1}, \ldots, a_{1,g-1}, a_{0,g}]$ be the vector with components equal to the coefficients of $G(x,y)$, we can select rational points (x_k, y_k) and construct vectors

$$\mathbf{v}_k = [d_{0,0}(x_k,y_k), d_{1,0}(x_k,y_k), d_{0,1}(x_k,y_k), \ldots, d_{1,g-1}(x_k,y_k), d_{0,g}(x_k,y_k)]$$

with the property that $\mathbf{a} \cdot \mathbf{v}_k = 0$. This allows us to use the version of the PSLQ algorithm for finding simultaneous integer relations. After verifying that the returned vectors $\{\mathbf{b}_l\}_{l=1}^m$ form a basis for the lattice of all simultaneous integer relations to the vectors $\{\mathbf{v}_k\}$, we can express $\mathbf{a}$ in the form $\mathbf{a} = \sum_{l=1}^m e_l \mathbf{b}_l$ where the e_l are integers. Using the basis vectors $\mathbf{b}_l = \langle b_{l,0,0}, b_{l,1,0}, b_{l,0,1}, \ldots, b_{l,1,g-1}, b_{l,0,g} \rangle$ to construct the polynomials $b_l(x,y) = \sum b_{l,i,j} x^i y^j$, we can then express the polynomial $G(x,y)$ in the form

$$G(x,y) = \sum_{l=1}^{m} e_l b_l(x,y) \quad \text{for some integers} \quad e_l.$$

If each polynomial $b_l(x,y)$ satisfies the condition $b_l(x,y) = (-1)^n b_l(1-x, 1-y)$, then the polynomials $\{b_l(x,y)\}_{l=1}^m$ form a basis for the lattice of bivariate polynomials with integer coefficients of total degree at most n satisfying the condition $f(x,y) = (-1)^n f(1-x, 1-y)$. If one of the polynomials $b_l(x,y)$ does not satisfy this symmetry condition, then the process should be redone with a larger selection of points (x_k, y_k) in order to further reduce the size of m.

Having expressed $G(x,y)$ as an integer linear combination of the polynomials $b_l(x,y)$, the next step is to select m rational points $(s_i/t_i, u_i/v_i) \in [0,1] \times [0,1]$ and let

$$r_i = \operatorname{lcm}(t_i, v_i)^g \, G\left(\frac{s_i}{t_i}, \frac{u_i}{v_i}\right) \in \mathbb{Z}.$$

The points $(s_i/t_i, u_i/v_i)$ must be chosen so the linear system of equations

$$\left\{ r_i = \operatorname{lcm}(t_i, v_i)^g \sum_{l=1}^{m} e_l \, b_l\left(\frac{s_i}{t_i}, \frac{u_i}{v_i}\right) \right\}_{i=1}^{m}$$

can be used to express the integers e_l as rational linear combinations of the integers r_i. Although we do not know the values of the e_l or r_i that give rise to $G(x,y)$, we can reduce the number of possibilities to a finite set. For $n \leq 12$, this resulted in a set of manageable size from which the desired combination could be selected. To reduce the possible choices for combinations of the r_i, we first bound the r_i using the inequality

$$|r_i| = \left| \operatorname{lcm}(t_i, v_i)^g \, G\left(\frac{s_i}{t_i}, \frac{u_i}{v_i}\right) \right| \leq \left| \frac{\operatorname{lcm}(t_i, v_i)^g \, c_n}{F\left(\frac{s_i}{t_i}, \frac{u_i}{v_i}\right)} \right|$$

when $F(s_i/t_i, u_i/v_i) \neq 0$, and by the use of the simplex algorithm as in Section 1.3.2 when $F(s_i/t_i, u_i/v_i) = 0$. As in the single variable case, the expressions for the e_l as rational

linear combinations of the r_i let us set up a system of congruences that the r_i must satisfy. Stepping through the permissable combinations of the r_i and constructing the corresponding polynomials $\sum_{l=1}^{m} e_l b_l(x,y)$ then allows us to find a nonzero polynomial $G(x,y)$ for which $\|F(x,y)G(x,y)\|_{[0,1]\times[0,1]}$ is minimal. The results are presented in Table 3.1. It is worth noting that the symmetry conditions were only used for degrees 11 and 12. For degrees 1 through 10 the table gives a complete list, up to simple transformations, of total degree nth integer Chebyshev polyonomials for the region $[0,1]\times[0,1]$.

From examining the polynomials in Table 3.1, one might be tempted to conjecture that for odd $n > 3$, there must exist a total degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ satisfying the conditions $p_n(x,y) = -p_n(1-x,1-y)$ and $p_n(x,y) = -p_n(y,x)$, and that for even n, there must exist one satisfying the conditions $p_n(x,y) = p_n(1-x,y)$ and $p_n(x,y) = p_n(x,1-y)$. This is not the case. Within the class of bivariate polynomials with integer coefficients of degree at most 13 satisfying the conditions $q(x,y) = -q(1-x,1-y)$ and $q(x,y) = -q(y,x)$, the polynomial

$$\begin{aligned} q_{13}(x,y) &= xy(1-x)(1-y)(x-y)^3(1-x-y)^2 \\ &\quad \cdot(2x^2+2xy+y^2-3x-2y+1)(x^2+2xy+2y^2-2x-3y+1) \end{aligned}$$

has minimal supremum norm on the region $[0,1]\times[0,1]$. However,

$$\|q_{13}(x,y)\|_{[0,1]\times[0,1]} = 0.00000555534\ldots > 0.00000504648\ldots = \|\tilde{p}_{13}(x,y)\|_{[0,1]\times[0,1]}$$

where $\tilde{p}_{13}(x,y)$ is the polynomial

$$\begin{aligned} \tilde{p}_{13}(x,y) &= xy(1-x)(1-y)(x-y)^2(1-x-y)(x^2-2xy+2y^2-y) \\ &\quad \cdot(x^2+2xy+2y^2-2x-3y+1)(2x^2+2xy+y^2-3x-2y+1). \end{aligned}$$

Within the class of bivariate polynomials with integer coefficients of degree at most 14 satisfying the conditions $w(x,y) = w(1-x,y)$ and $w(x,y) = w(x,1-y)$, the polynomial

$$\begin{aligned} w_{14}(x,y) &= xy(1-x)(1-y)(x-y)^2(1-x-y)^2 \\ &\quad \cdot(4x^6-2x^4y^2-2x^2y^4+4y^6-12x^5+2x^4y+4x^3y^2+4x^2y^3+2xy^4-12y^5 \\ &\quad +13x^4-4x^3y-5x^2y^2-4xy^3+13y^4-6x^3+3x^2y+3xy^2-6y^3+x^2-xy+y^2) \end{aligned}$$

has minimal supremum norm on the region $[0,1]\times[0,1]$. However, in this case we have

$$\|w_{14}(x,y)\|_{[0,1]\times[0,1]} = 0.00000160932\ldots > 0.00000153842\ldots = \|\tilde{p}_{14}(x,y)\|_{[0,1]\times[0,1]}$$

Table 3.1: Total degree bivariate integer Chebyshev polynomials for the region $[0,1]\times[0,1]$

n	$\mathrm{T}_n[0,1]\times[0,1]^{-1}$	Polynomial(s)
1	1	$1,\ x,\ 2x-1,\ x-y$
2	2	$x(1-x),\ (1-x-y)(x-y)$
3	2.182247	$x(1-x)(2x-1)$
4	2	$x(1-x)(2x-1)^2,\ x^2(1-x)^2,\ x(1-x)(5x^2-5x+1),$ $xy(1-y)(1-x),$ $(x-1+y)^2(x-y)^2,$ $x(1-x)(1-x-y)(x-y),$ $x(1-x)(x^2-x-2y^2+2y),$ $x(1-x)(4x^2-4x+y^2-y+1),$ $(x-y)(1-x-y)(x^2+y^2-x-y),$ $(x-y)(1-x-y)(4x^2+4y^2-4x-4y+1),$ $(x^2-2xy+2y^2-y)(x^2+2xy+2y^2-2x-3y+1),$ $x^4-x^2y^2+y^4-2x^3+x^2y+xy^2-2y^3+x^2-xy+y^2,$ $x^4-3x^2y^2+y^4-2x^3+3x^2y+3xy^2-2y^3+x^2-3xy+y^2,$ $5x^4-x^2y^2+y^4-10x^3+x^2y+xy^2-2y^3+6x^2-xy+y^2-x,$ $5x^4-2x^2y^2+y^4-10x^3+2x^2y+2xy^2-2y^3+6x^2-2xy+y^2-x$
5	2.236067	$x^2(2x-1)(1-x)^2,$ $xy(1-x)(1-y)(x-y),$ $x(1-x)(2x-1)(x-y)(1-x-y)$
6	2.519842	$xy(1-x)(1-y)(x-y)(1-x-y)$
7	2.499906	$xy(1-x)(1-y)(x-y)(1-x-y)^2$
8	2.539176	$xy(1-x)(1-y)(x-y)^2(1-x-y)^2$
9	2.502077	$xy(1-x)(1-y)(x-y)^3(1-x-y)^2$
10	2.584427	$xy(1-x)(1-y)(x-y)^2(1-x-y)^2(3x^2-3x+1-3y+3y^2)$
11	2.547609	$xy(1-x)(1-y)(x-y)(1-x-y)^2$ $\cdot(x^2-2xy-y+2y^2)(2x^2-x-2xy+y^2)$
12	2.557357	$xy(1-x)(1-y)(x-y)^3(1-x-y)^3(3x^2-3x+1-3y+3y^2)$

where

$$\tilde{p}_{14}(x,y) \;=\; xy(1-x)(1-y)(x-y)^2(1-x-y)^4(x^2-2xy+2y^2-y)(2x^2-2xy+y^2-x)$$

is a polynomial of minimal supremum norm on $[0,1]\times[0,1]$ within the class of bivariate polynomials of degree at most 14 with integer coefficients satisfying the symmetry conditions $z(x,y) = z(1-x,1-y)$ and $z(x,y) = z(y,x)$.

3.4.2 The maximum degree case

Although the process used in the total degree case extends to the maximum degree case, it is more advantageous to use Theorem 3.3 directly and reduce the problem to one on the region $[0,1/4]\times[0,1/4]$. To do this, we first consider the case when n is even. Let $p_n(x,y)$ be a maximum degree nth integer Chebyshev polynomial for the region $[0,1]\times[0,1]$ satisfying the symmetry conditions $p_n(x,y) = p_n(1-x,y)$ and $p_n(x,y) = p_n(x,1-y)$. Viewing $p_n(x,y)$ as a polynomial in $\mathbb{Z}[y][x]$ and writing $p_n(x,y) = \sum_{k=0}^{n} g_k(y)x^k$ where each $g_k(y)$ is a polynomial of degree at most n, we can isolate the polynomials $g_k(y)$ by evaluating the kth partial derivative of $p_n(x,y)$ with respect to x at $x = 0$. Since $p_n(x,y) = p_n(x,1-y)$,

$$g_k(y) = \frac{1}{k!}\left[\frac{\partial^k}{\partial x^k}p_n(x,y)\right]_{x=0} = \frac{1}{k!}\left[\frac{\partial^k}{\partial x^k}p_n(x,1-y)\right]_{x=0} = g_k(1-y).$$

By Lemma 1.5, the coefficients $g_k(y)$ of x^k must be of the form $g_k(y) = \tilde{g}_k\left(y(1-y)\right)$ for some polynomial $\tilde{g}(v) \in \mathcal{Z}_{n/2}[v]$ and so we may rewrite $p_n(x,y)$ in the form $p_n(x,y) = \tilde{p}_n(x,v) = \sum_{k=0}^{n} \tilde{g}_k(v)x^k$ where $v = y(1-y)$. Repeating the argument, if we view $\tilde{p}_n(x,v)$ as a polynomial in $\mathbb{Z}[x][v]$, we can write $\tilde{p}_n(x,v)$ in the form $\tilde{p}_n(x,v) = \sum_{k=0}^{n/2} h_k(x)v^k$ where each polynomial $h_k(x)$ is of degree at most n and isolate $h_k(x)$ by evaluating the partial derivatives of $\tilde{p}_n(x,v)$ with respect to v at $v = 0$. Now, since $p_n(x,y)$ satisfies the condition $p_n(x,y) = p_n(1-x,y)$, we also have $\tilde{p}_n(x,v) = \tilde{p}_n(1-x,v)$ and so

$$h_k(x) = \frac{1}{k!}\left[\frac{\partial^k}{\partial v^k}\tilde{p}_n(x,v)\right]_{v=0} = \frac{1}{k!}\left[\frac{\partial^k}{\partial v^k}p_n(1-x,v)\right]_{v=0} = h_k(1-x).$$

By Lemma 1.5, $h_k(x) = \tilde{h}_k\left(x(1-x)\right)$ for some $\tilde{h}_k(u) \in \mathcal{Z}_{n/2}[u]$ which gives the final form we are after,

$$p_n(x,y) = \sum_{k=0}^{n/2} \tilde{h}_k\left(x(1-x)\right)\left(y(1-y)\right)^k = q_{n/2}\left(x(1-x), y(1-y)\right)$$

for some polynomial $q_{n/2}(u,v)$ with integer coefficients of maximum degree at most $n/2$.

When n is odd, the first half of the argument gives $p_n(x,y) = (1-2y)\sum_{k=0}^{n} \tilde{g}_k(y(1-y))x^k$ where each polynomial $\tilde{g}_k(v)$ is of maximum degree at most $(n-1)/2$. Applying the second half of the argument to $\sum_{k=0}^{n} \tilde{g}_k(y(1-y))x^k$ then gives

$$p_n(x,y) = (1-2y)\left((1-2x)\sum_{k=0}^{(n-1)/2} \tilde{h}_k(x(1-x))(y(1-y))^k\right)$$

for some $\tilde{h}_k(u) \in \mathcal{Z}_{(n-1)/2}[u]$. We have the following result.

Theorem 3.4 *There exists a maximum degree nth integer Chebyshev polynomial for the region* $[0,1]\times[0,1]$ *of the form*

$$p_n(x,y) = q_{n/2}\left(x(1-x), y(1-y)\right)$$

where $q_{n/2}(u,v) \in \mathbb{Z}[u,v]$ *is of maximum degree at most* $n/2$ *when* n *is even, and of the form*

$$p_n(x,y) = (1-2y)(1-2x)q_{(n-1)/2}\left(x(1-x), y(1-y)\right)$$

where $q_{(n-1)/2}(u,v) \in \mathbb{Z}[u,v]$ *of maximum degree at most* $(n-1)/2$ *when* n *is odd.*

The method used to find a maximum degree nth integer Chebyshev polynomial $p_n(x,y)$ for the region $[0,1]\times[0,1]$ follows the usual three step process and begins with choosing an upper bound c_n for $\|p_n(x,y)\|_{[0,1]\times[0,1]}$. The bound

$$\|p_n(x,y)\|_{[0,1]\times[0,1]} \leq c_n = \left(\Omega_n[0,1]\right)^2$$

is a reasonable choice since it is the best we can do by taking polynomials of the form $f(x)f(y)$ where $f(x)$ is a polynomial with integer coefficients of degree at most n.

The second step of finding necessary factors begins in a similar fashion to the total degree case. For $n \geq 2$ the polynomials $p_n(x,0)$, $p_n(x,1)$, $p_n(0,y)$, and $p_n(1,y)$ are all single variable polynomials of degree at most n with supremum norm on $[0,1]$ less than $\Omega_n[0,1]$ and so must all be the zero polynomial. Since this gives an infinite number of points of intersection between the curve $p_n(x,y) = 0$ and each of the curves $x = 0$, $x = 1$, $y = 0$ and $y = 1$, it must be the case that $x(1-x)y(1-y)$ divides $p_n(x,y)$ for $n \geq 2$. However, unlike the total degree case, it is sometimes possible to show there are additional factors of $x(1-x)$ and $y(1-y)$. Using Theorem 3.4 to write $p_n(x,y)$ in the form $F(x,y)G\left(x(1-x), y(1-y)\right)$

where $F(x,y)$ represents the known factors of $p_n(x,y)$ and $G\left(x(1-x),y(1-y)\right)$ represents the unknown portion, we can attempt to show the polynomial $G(x,y)$ is divisible by xy. If we let g be the maximum degree of $G(x,y)$, let $\tilde{G}(x) = G(x,0) = \sum_{i=0}^{g} a_i x^i$, and set $w(x) = \left(1-\sqrt{1-4x}\right)/2$, then $t^g\tilde{G}(s/t) \in \mathbb{Z}$ for any $s/t \in \mathbb{Q}$ and

$$\left|F\left(w(x),w(y)\right)G(x,y)\right| \leq c_n \quad \text{for any} \quad (x,y) \in [0,1/4]\times[0,1/4].$$

This allows for the simplex algorithm to be applied using exact arithmetic as in Section 1.3.2. If we can show that $t^g\tilde{G}(s/t) \in (-1,1)$, then $\tilde{G}(s/t)$ must be zero. The goal is to show the degree g polynomial $\tilde{G}(x)$ has $g+1$ zeros in the interval $[0,1/4]$ and so must be the zero polynomial. If successful, then the curves $G(x,y) = 0$ and $y = 0$ have an infinite number of points of intersection and so we can claim that y divides $G(x,y)$. With no additional work, this also shows that x divides $G(x,y)$ since at each stage we maintain the equality $F(x,y) = F(y,x)$. Each time we find an additional factor $x(1-x)y(1-y)$ of $G\left(x(1-x),y(1-y)\right)$, it is transferred to $F(x,y)$ and the process is restarted. When no additional factors are found, we move to the third and final step of the procedure.

In order to perform an exhaustive search for the unknown factor $G(x,y)$, we select $(g+1)^2$ rational points $(s_i/t_i, u_i/v_i) \in [0,1/4]\times[0,1/4]$ and let

$$r_i = (t_iv_i)^g\, G\left(\frac{s_i}{t_i},\frac{u_i}{v_i}\right) \in \mathbb{Z}.$$

The points $(s_i/t_i, u_i/v_i)$ must be chosen so the linear system of equations

$$\left\{ r_i = (t_iv_i)^g \sum_{\substack{i+j\leq g\\ i\geq 0, j\geq 0}} a_{i,j}\left(\frac{s_i}{t_i}\right)^i\left(\frac{u_i}{v_i}\right)^j \right\}_{i=1}^{(g+1)^2}$$

can be used to express the integers $a_{i,j}$ as rational linear combinations of the integers r_i. Although we do not know the values of the r_i, we can reduce the number of possible combinations to a finite set and then search through this set in order to find the combination giving rise to the coefficients of $G(x,y)$. To reduce the possible choices for combinations of the r_i, we first bound the r_i using the inequality

$$|r_i| = \left|\operatorname{lcm}\left(t_i,v_i\right)^g G\left(\frac{s_i}{t_i},\frac{u_i}{v_i}\right)\right| \leq \left|\frac{(t_iv_i)^g\, c_n}{F\left(w\left(\frac{s_i}{t_i}\right),w\left(\frac{u_i}{v_i}\right)\right)}\right|$$

when $F\left(w(s_i/t_i), w(u_i/v_i)\right) \neq 0$, and by the use of the simplex algorithm as in Section 1.3.2 when $F\left(w(s_i/t_i), w(u_i/v_i)\right) = 0$. As in the single variable case, the expressions for the $a_{i,j}$ as rational linear combinations of the r_i let us set up a system of congruences that the r_i must satisfy. Stepping through the permissable combinations of the r_i and constructing the corresponding polynomials $\sum a_{i,j}x^iy^j$ then allows us to find a nonzero polynomial $G(x, y)$ for which $\|F(x, y)G(x(1-x), y(1-y))\|_{[0,1]\times[0,1]}$ is minimal.

This method allowed for the discovery of maximum degree nth integer Chebyshev polynomials on the region $[0, 1]\times[0, 1]$ for all $n \leq 19$. In all cases, $\mathrm{M}_n[0, 1]\times[0, 1] = (\Omega_n[0, 1])^2$ and the polynomials $f_n(x)f_n(y)$ where $f_n(x)$ is an nth integer Chebyshev polynomial for the interval $[0, 1]$ were found to be maximum degree nth integer Chebyshev polynomials of the region $[0, 1]\times[0, 1]$. It is interesting to note that for degree 4, there were additional maximum degree nth integer Chebyshev polynomials for the unit square that were not of the form $f(x)f(y)$ for any polynomial $f(x)$.

Appendix A

Integer Chebyshev Polynomials (Maple Code)

A.1 The single variable case on $[0, 1]$

The following is a Maple implementation of the code outlined in Section 1.3.2 for finding symmetric nth integer Chebyshev polynomials on the interval $[0, 1]$. It is intended to be run with the command `intCheb(startpoly,n,c);` where `startpoly` is either 1 or a known symmetric factor, `n` is the degree, and `c` is an upper bound on $\Omega_n[0, 1]$.

```
intCheb := proc(startpoly, n, c)
local F, LCM, Lower, N, SimpTable, Upper, approxresbound, b1, b2, b3, changed, congs,
      eps, i, ind, innerTable, linpolyinfo,linpolys, newlowup, pass, polyroots, polys,
      r, rbs, resIs0, resbound, sols, steps, u;
    Digits := max(20, n);
    polys := [x*(1 - x), (2*x - 1)^2, 5*x^2 - 5*x + 1,
              6*x^2 - 6*x + 1, 29*x^4 - 58*x^3 + 40*x^2 - 11*x + 1,
              (13*x^3 - 20*x^2 + 9*x - 1)*(13*x^3 - 19*x^2 + 8*x - 1),
              (31*x^4 - 63*x^3 + 44*x^2 - 12*x + 1)*(31*x^4 - 61*x^3 + 41*x^2 - 11*x + 1),
              941*x^8-3764*x^7+6349*x^6-5873*x^5+3243*x^4-1089*x^3+216*x^2-23*x+1];
    polyroots := [[0, 1], [0.5],seq([fsolve(polys[i])], i = 3 .. nops(polys))];
    eps := 1/1000000*c;
    if startpoly = 1 then
        r := 0;
        while evalf(c*sqrt(2*r+1)*(n+r+1)!/((n-r)!*(2*r+1)!))< 1 do r:=r+1 end do;
        F := (x*(1 - x))^r;
        if n mod 2 = 1 then F := F*(2*x - 1) end if
    else F := startpoly
    end if;
    if nargs = 4 then F := F*args[4] end if;
    N := 1/2*n - 1/2*degree(F) + 1;
    print(F, N - 1);
    u := unapply(1/2 - 1/2*sqrt(1 - 4*x), x);
    changed := true;
    while changed and 1 < N do
        if not has(F, 2*x - 1) and
        abs(2^(n - degree(F))*c*subs(x = 1/2, 1/F)) < 1 then
            print(‘adding the following factor‘, (2*x - 1)^2);
            F := F*(2*x - 1)^2;
```

```
    N := N - 1
end if;
for i from 3 to nops(polys) do
    if (type(polys[i], `+`) and not has(F, polys[i])
    or type(polys[i], `*`) and not has(F, op(1, polys[i]))) and
      abs(lcoeff(polys[i])^(n - degree(F))*c^degree(polys[i])
      *`*`(seq(subs(x=polyroots[i][j], 1/F),j=1..nops(polyroots[i])))<0.999 then
        print(`adding the following factor`, polys[i]);
        F := F*polys[i];
        N := N - 1/2*degree(polys[i])
    end if
end do;
changed := false;
innerTable := table();
SimpTable := table();
linpolyinfo := getlinpolyinfo(F, N, c, eps,[0, 1/4, 1/5, 1/6],
                [infinity, infinity, infinity, infinity], false);
congs := linpolyinfo[1];  Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
LCM := linpolyinfo[4];    steps := linpolyinfo[5];  sols := linpolyinfo[6];
linpolys := linpolyinfo[7];
for pass to 2 do
    print(`checking factor x*(1-x)`);
    newlowup := applySimp(1, 1, LCM, indices, sols, F,N, c, eps, Lower, Upper,
                    steps, addressof(SimpTable), addressof(innerTable), true);
    if newlowup[2] = 0 then
        F := F*x*(1 - x); N := N - 1; changed := true
    else
        Lower[1] := -newlowup[2]; Upper[1] := newlowup[2]
    end if;
    if not changed and 1 < N then
        print(`checking factor (2x-1)`);
        newlowup := applySimp(1, 2, LCM, indices, sols, F, N, c, eps, Lower, Upper,
                        steps, addressof(SimpTable), addressof(innerTable), true);
        if newlowup[2] = 0 then
            F := F*(2*x - 1)^2; N := N - 1; changed := true
        else
            Lower[2] := -newlowup[2]; Upper[2] := newlowup[2]
        end if
    end if;
    if not changed and 2 < N then
        print(`checking factor (5*x^2-5*x+1)`);
        newlowup := applySimp(1, 3, LCM, indices, sols, F, N, c, eps, Lower, Upper,
                        steps, addressof(SimpTable), addressof(innerTable), true);
        if newlowup[2] = 0 then
            F := F*(5*x^2 - 5*x + 1); N := N - 1; changed := true
        else
            Lower[3] := -newlowup[2]; Upper[3] := newlowup[2]
        end if
    end if;
    if not changed and 3 < N and pass = 1 then
        print(`checking factor (6*x^2-6*x+1)`);
        newlowup := applySimp(1, 4, LCM, indices, sols, F, N, c, eps, Lower, Upper,
                        steps, addressof(SimpTable), addressof(innerTable), true);
        if newlowup[2] = 0 then
            F := F*(6*x^2 - 6*x + 1); N := N - 1; changed := true
        else
            Lower[4] := -newlowup[2]; Upper[4] := newlowup[2]
        end if
    end if;
    if changed then pass := 2 end if
end do;
if not changed and 13 < N then
    rbs := [Upper[1], Upper[2], Upper[3], Upper[4], infinity];
    print(`checking factor 29*x^4-58*x^3+40*x^2-11*x+1`);
    innerTable := table();
    SimpTable := table();
    linpolyinfo := getlinpolyinfo(F, N, c, eps,[0, 1/4, 1/5, 1/6, 987/4325],
                          [rbs[1], rbs[2], rbs[3], rbs[4], infinity], false);
    congs := linpolyinfo[1];  Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
    LCM := linpolyinfo[4];    steps := linpolyinfo[5];  sols := linpolyinfo[6];
    linpolys := linpolyinfo[7];
```

```
b1 := applySimp(1, 5, LCM, indices, sols, F, N, c, eps, Lower, Upper, steps,
                        addressof(SimpTable), addressof(innerTable), true);
innerTable := table();
SimpTable := table();
linpolyinfo := getlinpolyinfo(F, N, c, eps, [0, 1/4, 1/5, 1/6, 610/4037],
                    [rbs[1], rbs[2], rbs[3], rbs[4], infinity], false);
congs := linpolyinfo[1];  Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
LCM := linpolyinfo[4];    steps := linpolyinfo[5];  sols := linpolyinfo[6];
linpolys := linpolyinfo[7];
b2 := applySimp(1, 5, LCM, indices, sols, F, N, c, eps, Lower, Upper, steps,
                        addressof(SimpTable), addressof(innerTable), true);
approxresbound := b1[2]*b2[2]*29^(N - 1)/(4325^(N - 1)*4037^(N - 1));
if approxresbound < 0.999 then
    print(‘approximate bound on resultant: ‘,evalf(approxresbound, 10));
    print(‘Using Lagrange interpolation technique‘);
    resbound := LagrangeStep(F, N - 1, c,29*x^4 - 58*x^3 + 40*x^2 - 11*x + 1,
        [[987/4325, b1[2]/4325^(N - 1)],[610/4037, b2[2]/4037^(N - 1)]], eps);
    print(‘bound on resultant is :‘,evalf(resbound, 10));
    if resbound < 0.999 then
        F := F*(29*x^4 - 58*x^3 + 40*x^2 - 11*x + 1); N := N-2; changed:=true
    end if
end if;
if not changed then
    print(‘checking factor (13*x^3-20*x^2+9*x-1)*(13*x^3-19*x^2+8*x-1)‘);
    innerTable := table();
    SimpTable := table();
    linpolyinfo := getlinpolyinfo(F, N, c, eps, [0, 1/4, 1/5, 1/6,
        325741/1367481], [rbs[1], rbs[2], rbs[3], rbs[4], infinity], false);
    congs := linpolyinfo[1]; Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
    LCM := linpolyinfo[4];   steps := linpolyinfo[5];  sols := linpolyinfo[6];
    linpolys := linpolyinfo[7];
    b1 := applySimp(1, 5, LCM, indices, sols, F, N, c, eps, Lower, Upper,
              steps, addressof(SimpTable), addressof(innerTable), true);
    innerTable := table();
    SimpTable := table();
    linpolyinfo := getlinpolyinfo(F, N, c, eps, [0, 1/4, 1/5, 1/6, 11345/63021],
                          [rbs[1], rbs[2], rbs[3], rbs[4], infinity], false);
    congs := linpolyinfo[1]; Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
    LCM := linpolyinfo[4];   steps := linpolyinfo[5];  sols := linpolyinfo[6];
    linpolys := linpolyinfo[7];
    b2 := applySimp(1, 5, LCM, indices, sols, F, N, c, eps, Lower, Upper,
              steps, addressof(SimpTable), addressof(innerTable), true);
    innerTable := table();
    SimpTable := table();
    linpolyinfo := getlinpolyinfo(F, N, c, eps, [0, 1/4, 1/5, 1/6,
           261224/1893085], [rbs[1], rbs[2], rbs[3], rbs[4], infinity], false);
    congs := linpolyinfo[1]; Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
    LCM := linpolyinfo[4];   steps := linpolyinfo[5];  sols := linpolyinfo[6];
    linpolys := linpolyinfo[7];
    b3 := applySimp(1, 5, LCM, indices, sols, F, N, c, eps, Lower, Upper,
              steps, addressof(SimpTable), addressof(innerTable), true);
    approxresbound := b1[2]*b2[2]*b3[2]*169^(N - 1)/(1367481^(N - 1)
                                    *63021^(N - 1)*1893085^(N - 1));
    if approxresbound < 0.999 then
        print(‘approximate bound on resultant: ‘,evalf(approxresbound, 10));
        print(‘Using Lagrange interpolation technique‘);
        resbound := LagrangeStep(F, N - 1, c, (13*x^3 - 20*x^2 + 9*x - 1)
            *(13*x^3 - 19*x^2 + 8*x - 1), [[325741/1367481,
            b1[2]/1367481^(N - 1)], [11345/63021, b2[2]/63021^(N - 1)],
            [261224/1893085, b3[2]/1893085^(N - 1)]], eps);
        print(‘bound on resultant is :‘, evalf(resbound, 10));
        if resbound < 0.999 then
            F := F*(13*x^3 - 20*x^2 + 9*x - 1)*(13*x^3 - 19*x^2 + 8*x - 1);
            N := N - 3; changed := true
        end if
    end if
end if;
if not changed and 11 < N then
    innerTable := table();
    SimpTable := table();
    linpolyinfo := getlinpolyinfo(F, N, c, eps, [0, 1/4, 1/5, 1/6],
```

```
                                    [rbs[1], rbs[2], rbs[3], rbs[4]], false);
                congs := linpolyinfo[1]; Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
                LCM := linpolyinfo[4];   steps := linpolyinfo[5];  sols := linpolyinfo[6];
                linpolys := linpolyinfo[7]
            end if
        end if;
        if not changed and 11 < N then
            print(`final check for x*(1-x)`);
            resIs0 := true;
            for ind[2] from Lower[2] to 0 do for ind[3] from Lower[3] to Upper[3] do
                    print(`---  ?`, ind[2], ind[3], `  ---`);
                    newlowup := applySimp(1, 1, LCM, indices, sols, F, N, c, eps,
                         [Lower[1], ind[2], ind[3], Lower[4], seq(Lower[k], k = 5 .. N)],
                         [Upper[1], ind[2], ind[3], Upper[4], seq(Upper[k], k = 5 .. N)],
                         steps, addressof(SimpTable), addressof(innerTable), false);
                    if newlowup <> {} and (newlowup[1] <> 0 or newlowup[2] <> 0)
                    then
                        resIs0 := false; ind[3] := Upper[3]; ind[2] := Upper[2]
                    end if
            end do end do;
            if resIs0 then F := F*x*(1 - x); N := N - 1; changed := true end if;
            if not changed then
                print(`final check for 2*x-1`);
                resIs0 := true;
                for ind[1] from Lower[1] to 0 do for ind[3] from Lower[3] to Upper[3] do
                        print(`---  `, ind[1], `?`, ind[3],`  ---`);
                        newlowup := applySimp(1, 2, LCM, indices, sols, F, N, c, eps,
                          [ind[1], Lower[2], ind[3], Lower[4], seq(Lower[k], k = 5 .. N)],
                          [ind[1], Upper[2], ind[3], Upper[4], seq(Upper[k], k = 5 .. N)],
                          steps, addressof(SimpTable), addressof(innerTable), false);
                        if newlowup <> {} and (newlowup[1] <> 0 or newlowup[2] <> 0)
                        then
                            resIs0 := false; ind[3] := Upper[3]; ind[1] := Upper[1]
                        end if
                end do end do;
                if resIs0 then F := F*(2*x - 1)^2; N := N - 1; changed := true end if
            end if;
            if not changed then
                print(`final check for 5*x^2-5*x+1`);
                resIs0 := true;
                for ind[1] from Lower[1] to 0 do for ind[2] from Lower[2] to Upper[2] do
                        print(`---  `, ind[1], ind[2], `?  ---`);
                        newlowup := applySimp(1, 3, LCM, indices, sols, F, N, c, eps,
                          [ind[1], ind[2], Lower[3], Lower[4], seq(Lower[k], k = 5 .. N)],
                          [ind[1], ind[2], Upper[3],Upper[4], seq(Upper[k], k = 5 .. N)],
                          steps, addressof(SimpTable), addressof(innerTable), false);
                        if newlowup <> {} and (newlowup[1] <> 0 or newlowup[2] <> 0)
                        then
                            resIs0 := false; ind[2] := Upper[2]; ind[1] := Upper[1]
                        end if
                end do end do;
                if resIs0 then F := F*(5*x^2 - 5*x + 1); N := N - 1; changed := true end if
            end if
        end if;
        print(F, N - 1)
    end do;
    if degree(F) < n then
        print([seq(max(abs(Lower[k]), abs(Upper[k])), k = 1 .. min(N, 4))]);
        print(`================================`);
        print();
        F := searchpolys(F, N - 1, c, seq(`if`(N < k,infinity, max(abs(Lower[k]),
                                                  abs(Upper[k]))), k = 1 .. 4))
    end if;
    return F
end proc;


LagrangeStep := proc(F, g, c, critpoly, bounds, eps)
local G, G2, G3, P, RootsOfF, S, a, bound, counter, dG3, fixedx, i, iteration, iterations,
      j, left, middle, minpoints, minpos, minval, points, right, u, val, xvals, xvar;
    iterations := 8;
```

```
    Digits := max(degree(F, x) + 2*g, 10);
    u := unapply(1/2 - 1/2*sqrt(1 - 4*x), x);
    if type(F, `+`) then RootsOfF := [fsolve(F, x = 0 .. 1/2)]
    elif type(F, `^`) then RootsOfF := [fsolve(op(1, F), x = 0 .. 1/2)]
    else RootsOfF := [];
         for i to nops(F) do if type(op(i, F), `+`)
             then RootsOfF := [op(RootsOfF), fsolve(op(i, F), x = 0 .. 1/2)]
             else RootsOfF := [op(RootsOfF),fsolve(op(1, op(i, F)), x = 0 .. 1/2)]
         end if end do
    end if;
    RootsOfF := sort([seq(RootsOfF[i]*(1 - RootsOfF[i]), i = 1 .. nops(RootsOfF))]);
    G := 0;
    for i to nops(bounds) do
        P := 1;
        for j to g + 1 do if i <> j then P := P*(x - xv[j])/(xv[i] - xv[j]) end if end do;
        G := G + abs(bounds[i][2]*P)
    end do;
    for i from nops(bounds) + 1 to g + 1 do
        P := 1;
        for j to g + 1 do if i <> j then P := P*(x - xv[j])/(xv[i] - xv[j]) end if end do;
        G := G + abs(c*P/subs(x = u(xv[i]), F))
    end do;
    S := [op(map(x -> x*(1 - x), [fsolve(critpoly, x = 0 .. 1/2)]))];
    a := abs(lcoeff(critpoly, x));
    bound := a^g;
    for j to nops(S) do
        G2 := subs(x = S[j], G);
        xvals := [seq(bounds[i][1], i = 1 .. nops(bounds)),
                  seq(evalf(1/4*k*sqrt(5/6)/(g + 1)), k = nops(bounds) + 1 .. g + 1)];
        for iteration to iterations do for xvar from nops(bounds) + 1 to g + 1 do
                G3 := G2;
                points := {0., op(RootsOfF), evalf(1/4)};
                for fixedx to g + 1 do
                    if fixedx <> xvar then
                        G3 := subs(xv[fixedx] = xvals[fixedx], G3);
                        points := {xvals[fixedx], op(points)}
                    end if
                end do;
                points := sort([op(points)]);
                dG3 := diff(G3, xv[xvar]);
                minpoints := [];
                for counter to nops(points) - 1 do
                    left := points[counter];
                    right := points[counter + 1];
                    while 1/10000 < right - left do
                        middle := 1/2*right + 1/2*left;
                        if 0 < evalf(subs(xv[xvar] = middle, dG3))
                            then right := middle else left := middle
                        end if
                    end do;
                    minpoints := [op(minpoints), middle]
                end do;
                minval := evalf(subs(xv[xvar] = xvals[xvar], G3));
                minpos := xvals[xvar];
                for counter to nops(minpoints) do
                    val := evalf(subs(xv[xvar] = minpoints[counter], G3));
                    if val < minval then
                        minval := val;  minpos := minpoints[counter]
                    end if
                end do;
                xvals[xvar] := minpos;
                if j = nops(S) and bound*minval < 1 - eps then
                    iteration := iterations; xvar := g + 1
                end if
            end do
        end do;
        bound := bound*minval
    end do;
    return bound
end proc;
```

```
searchpolys := proc(F, n, c, r1b, r2b, r3b, r4b)
local Blower, Bupper, D, LCM, Lower, N, SimpTable, Upper, a, b, checkPoints, congs,
      direction, dosubnum, eps, est, flag, i, indices, innerTable, j, linpolyinfo,
      linpolys, lower, minpoly, minsup, newlowup, poly, simpblockend, simpblockstart,
      simpcutoff, sol, sols, steps, subproblem, thesup, upper;
    SimpTable := table();
    innerTable := table();
    eps := 1/1000000*c;
    N := n + 1;
    Digits := max(degree(F, x) + 2*n, 10);
    checkPoints := getcheckPoints(F,N,c,eps);
    linpolyinfo := getlinpolyinfo(F,N,c,eps, [0, 1/4, 1/5, 1/6], [r1b, r2b, r3b, r4b], true);
    congs := linpolyinfo[1];  Upper := linpolyinfo[2];  Lower := linpolyinfo[3];
    LCM := linpolyinfo[4];    steps := linpolyinfo[5];  sols := linpolyinfo[6];
    linpolys := linpolyinfo[7];
    if nargs = 8 then for j to nops(args[8]) do
            Lower[j] := args[8][j]; Upper[j] := args[8][j]
        end do
    end if;
    if nargs < 8 or args[8] = [] then Upper[1] := 0 end if;
    print(`Upper `, Upper);
    print(`Lower `, Lower);
    print(`steps `, steps);
    if N < 11 then simpblockstart := 0; simpblockend := 0
    elif N = 11 then simpblockstart := 2; simpblockend := 3; simpcutoff := 50000
    elif N = 12 then simpblockstart := 4; simpblockend := 5; simpcutoff := 100000
    elif N = 13 then simpblockstart := 5; simpblockend := 7; simpcutoff := 200000
    else
        simpblockstart := 6;  simpblockend := N;
        while (Upper[simpblockend] - Lower[simpblockend] + 1)/steps[simpblockend] < 1 do
            simpblockend := simpblockend - 1
        end do;
        simpblockend := min(simpblockend + 2, N - 2);  simpcutoff := 500000*N - 6500000
    end if;
    print(blockstart, simpblockstart, blockend, simpblockend);
    dosubnum := 11;
    minsup := 1;
    minpoly := {1};
    indices := [seq(0, k = 1 .. N)];
    direction := 1;
    i := 1;
    while 0 < i do
        if direction = 1 then while direction = 1 do
                if i < simpblockstart and Upper[i] <> Lower[i] then
                    if n < dosubnum and 1 < i and indices[i - 1] = 0
                    and `+`(seq(abs(indices[k]), k = 1 .. i - 1)) = 0 then
                        newlowup := applySimp(i, i, LCM, indices, sols, F, N, c, eps, Lower,
                                [seq(Upper[k],k=1..i-1), 0, seq(Upper[k], k=i+1..N)], steps,
                                        addressof(SimpTable),addressof(innerTable), false)
                    else newlowup := applySimp(i, i, LCM, indices, sols, F, N, c, eps, Lower,
                            Upper, steps, addressof(SimpTable), addressof(innerTable), false)
                    end if;
                    if newlowup = {} then direction := -1
                                     else lower[i] := newlowup[1]; upper[i] := newlowup[2]
                    end if
                elif simpblockstart <= i and i <= simpblockend then
                    if i = simpblockstart then
                        print(doingblock);
                        Bupper := Upper;  Blower := Lower;
                        for j from simpblockstart to simpblockend do
                            if Upper[j] <> Lower[j] then
                                est := `*`(seq(ceil((Bupper[k]-Blower[k]+1)/steps[k]),
                                                    k = simpblockstart .. N));
                                print(`estimated size: `, est);
                                if est <= simpcutoff then
                                    j := simpblockend;
                                    print(`less than `, simpcutoff,` so jumping out`)
                                else
                                    newlowup := applySimp(i,j, LCM, indices, sols,F, N, c, eps,
                                                    Blower, Bupper, steps, addressof(SimpTable),
```

```
                                                    addressof(innerTable), false);
                        if newlowup = {} then
                            direction := -1;  j := simpblockend
                        else
                            Blower[j] := newlowup[1];
                            Bupper[j] := newlowup[2];
                            if dosubnum < N and j < 5 and newlowup[1] = 0
                            and newlowup[2] = 0 then j := simpblockend
                            end if
                        end if
                    end if
                end if
            end do;
            print(doneblock)
        end if;
        if i = simpblockstart + 1 and 0 < Bupper[i - 1] - Blower[i - 1] and
        1 < Bupper[i] - Blower[i] and 2*simpcutoff
          <= est/ceil((Bupper[i - 1] - Blower[i - 1] + 1)/steps[i - 1]) then
            newlowup := applySimp(i, i, LCM, indices, sols, F, N, c, eps, Blower,
              Bupper, steps, addressof(SimpTable), addressof(innerTable), false);
            if newlowup = {} then direction := -1
            else
                if n < dosubnum and 1 < i and indices[i - 1] = 0
                and `+`(seq(abs(indices[k]), k = 1 .. i - 1)) = 0
                    then upper[i] := 0
                    else upper[i] := newlowup[2]
                end if;
                lower[i] := newlowup[1]
            end if
        else
            if n < dosubnum and 1 < i and indices[i - 1] = 0
            and `+`(seq(abs(indices[k]), k = 1 .. i - 1)) = 0
                then upper[i] := 0
                else upper[i] := Bupper[i]
            end if;
            lower[i] := Blower[i]
        end if
    elif Upper[i] = infinity or Lower[i] = -infinity then
        if n < dosubnum and 1 < i and indices[i - 1] = 0
        and `+`(seq(abs(indices[k]), k = 1 .. i - 1)) = 0
        then newlowup := applySimp(i, i, LCM, indices, sols, F, N, c, eps, Lower,
               [seq(Upper[k], k = 1 .. i - 1), 0, seq(Upper[k], k = i + 1 .. N)],
                       steps, addressof(SimpTable), addressof(innerTable), false)
        else newlowup := applySimp(i, i, LCM, indices, sols, F, N, c, eps, Lower,
                Upper, steps, addressof(SimpTable), addressof(innerTable), false)
        end if;
        if newlowup = {} then direction := -1
                         else lower[i] := newlowup[1]; upper[i] := newlowup[2]
        end if
    else
        if n < dosubnum and 1 < i and indices[i - 1] = 0
        and `+`(seq(abs(indices[k]), k = 1 .. i - 1)) = 0
            then upper[i] := 0 else upper[i] := Upper[i]
        end if;
        lower[i] := Lower[i]
    end if;
    if direction = 1 then
        if congs[i] = 0 then
            indices[i] := lower[i];
            if i <= 4 and indices[i] = 0 and dosubnum <= n then
                subproblem:=dosubproblem(linpolys, indices, Upper, Lower, F, n, c, i);
                if subproblem[2] <> 1 then
                    if subproblem[2] < minsup - eps then
                        minsup := subproblem[2];
                        minpoly := subproblem[1]
                    elif subproblem[2] < minsup + eps then
                        minpoly := minpoly union subproblem[1];
                        minsup := min(minsup, subproblem[2])
                    end if
                end if;
                i := i + 1;
```

```
                    direction := -1
                else
                    i := i + 1;
                    if i = N + 1 then direction := 0 end if
                end if
            else
                b := inner([seq(indices[k], k = 1 .. i - 1)], congs[i, 3]);
                D := iquo(b, congs[i, 1], 'rmndr');
                if rmndr = 0 then
                    a := congs[i, 2]*D mod steps[i];
                    a := lower[i] + ((a - lower[i]) mod steps[i]);
                    if a <= upper[i] then
                        indices[i] := a;
                        if i <= 4 and a = 0 and dosubnum <= n then
                           subproblem:=dosubproblem(linpolys,indices,Upper,Lower,F,n,c,i);
                           if subproblem[2] <> 1 then
                                if subproblem[2] < minsup - eps then
                                    minsup := subproblem[2];
                                    minpoly := subproblem[1]
                                elif subproblem[2] < minsup + eps then
                                    minpoly := minpoly union subproblem[1];
                                    minsup := min(minsup, subproblem[2])
                                end if
                           end if;
                           i := i + 1;
                           direction := -1
                        else
                           i := i + 1;
                           if i = N + 1 then direction := 0 end if
                        end if
                    else direction := -1
                    end if
                else direction := -1
                end if
            end if
        end if
    end do
elif direction = -1 then while direction = -1 and 0 < i do
        i := i - 1;
        if 0 < i then
            indices[i] := indices[i] + steps[i];
            if indices[i] <= upper[i] then
                if i <= 4 and indices[i] = 0 and dosubnum <= n then
                    subproblem:=dosubproblem(linpolys,indices,Upper,Lower,F,n,c,i);
                    if subproblem[2] <> 1 then
                        if subproblem[2] < minsup - eps then
                            minsup := subproblem[2];
                            minpoly := subproblem[1]
                        elif subproblem[2] < minsup + eps then
                            minpoly := minpoly union subproblem[1];
                            minsup := min(minsup, subproblem[2])
                        end if
                    end if;
                    i := i + 1;
                    direction := -1
                else
                    i := i + 1;
                    direction := `if`(0 < i - N, 0, 1)
                end if
            end if
        end if
    end do
else
    direction := -1;
    sol := subs({seq(r[k] = indices[k], k = 1 .. N)}, sols);
    j := 0;
    flag := true;
    while j < nops(checkPoints) and flag do
        j := j + 1;
        if checkPoints[j, 2] < abs(inner(checkPoints[j, 1], sol)) then
            flag := false
```

```
            end if
        end do;
        if flag then
            poly := factor(inner(sol, [seq((x*(1 - x))^k, k = 0 .. N - 1)])*F);
            if lcoeff(poly, x) < 0 then poly := -poly end if;
            thesup := maxi(poly);
            print(poly);
            print(evalf(thesup, 10));
            print(indices);
            print(`----------------------------`);
            if thesup <> 0 and thesup < minsup - eps then
                minsup := thesup; minpoly := {poly}
            elif thesup <> 0 and thesup < minsup + eps
            then
                minpoly := minpoly union {poly};
                minsup := min(minsup, thesup)
            end if
        end if
    end if
end do;
if nargs = 8 then return [minpoly, minsup] end if;
print(`min sup norm is `, minsup);
return minpoly
end proc;


maxi := proc(p)
local i, m, p2, p3, poly, v, v2, vals;
    p2 := diff(p, x);
    if type(p, `+`) then p3 := p2
    else poly := 1;
         for i to nops(p) do
             if type(op(i, p), `^`) then
                 poly := poly*op(1, op(i, p))^(op(2, op(i, p)) - 1)
             end if
         end do;
         p3 := simplify(p2/poly)
    end if;
    vals := {0, 0.5, fsolve(p3, x = 0 .. 0.5)};
    m := 0;
    for v in vals do
        v2 := abs(subs(x = v, p));
        if m < v2 then m := v2 end if
    end do;
    return m
end proc;


dosubproblem := proc(linpolys, indices, Upper, Lower, F, n, c, i)
local coeff0, coeffx, intflag, j, newRes, subproblem;
    coeffx := coeff(linpolys[i][1], x, 1);
    coeff0 := coeff(linpolys[i][1], x, 0);
    newRes := [seq(indices[k]/(coeff(linpolys[k][1], x, 1)*coeff0
               -coeff(linpolys[k][1], x, 0)*coeffx), k = 1 .. i - 1), infinity,
               seq(floor(abs(Upper[k]/(coeff(linpolys[k][1], x, 1)*coeff0
               -coeff(linpolys[k][1], x, 0)*coeffx))), k = i + 1 .. 4)];
    intflag := true;
    for j to 4 do
        if i <> j and not type(newRes[j], integer) then intflag := false end if
    end do;
    if intflag then subproblem := searchpolys(subs(x = x*(1 - x), linpolys[i][1])*F,
             n - 1, c, seq(newRes[k], k = 1 .. 4), [seq(newRes[k], k = 1 .. i - 1)])
    else subproblem := [1, 1]
    end if;
    return subproblem
end proc;


getcheckPoints := proc(F, N, c, eps)
local n, checkPoints, meshsize, vals, v;
    n := N - 1;
```

```
    checkPoints := [];
    meshsize := 0.5/(5*N);
    vals := [`if`(F = 1, NULL, fsolve(diff(F, x), x=0..1/2)), seq(meshsize*k, k=1..5*N-1)];
    for v in vals do
        if c < abs(subs(x = v, F)) then
            checkPoints:=[op(checkPoints),[[seq((v*(1-v))^k,k=0..n)],(c+eps)/abs(subs(x=v,F))]]
        end if
    end do;
    return evalf(checkPoints)
end proc;


getlinpolyinfo := proc(F, N, c, eps, Rats, rbounds, addextrapoints)
local n, rats, numextra, u, Rflg, i, j, k, rbound, linpolys, linpolystest, numToVary,
      firstlinpolys, lastlinpolys, G, minsize, testnum, R, sols, thelcms, LCM, E, Mtrx,
      conginfo, minlinpolys, minconginfo, minsols, minLCM, Lower, Upper, congs, steps, gcdi;
    n := N - 1;
    rats := Rats;
    if addextrapoints then numextra := 3 else numextra := 0 end if;
    u := unapply(1/2 - 1/2*sqrt(1 - 4*x), x);
    Rflg := false;
    for i to nops(Rats) do if Rats[i] = 0 then Rflg := true end if end do;
    if not Rflg then rats := [op(rats), 0] end if;
    for i from 4 to 2*N + 2*numextra do for j to floor(1/4*i) do
            if igcd(i, j) = 1 then
                Rflg := false;
                for k to nops(Rats) do
                    if Rats[k] = j/i then Rflg := true end if
                end do;
                if not Rflg then rats := [op(rats), j/i]
                end if
            end if
        end do
    end do;
    for i to nops(rats) do
        try rbound[i] := floor(evalf(denom(rats[i])^n*(c + eps)/abs(subs(x = u(rats[i]), F))))
        catch: rbound[i] := infinity
        end try
    end do;
    linpolys:=[seq([denom(rats[k])*x-numer(rats[k]), rbound[k]], k=nops(Rats)+1..nops(rats))];
    linpolys := sort(linpolys, (a, b) -> a[2] < b[2]);
    linpolys := [seq([denom(rats[k])*x - numer(rats[k]), rbounds[k]],
                 k = 1 .. nops(Rats)),seq(linpolys[k], k = 1 .. nops(linpolys))];
    linpolys := [seq(linpolys[k], k = 1 .. N + numextra)];
    if n < 4 or not addextrapoints then
        linpolystest := [[seq(linpolys[k], k = 1 .. N)]]
    else
        numToVary := min(N - 4, 4, floor(1/2*N));
        firstlinpolys := [seq(linpolys[k], k = 1 .. N - numToVary)];
        lastlinpolys:=combinat[choose]([seq(linpolys[k], k=N-numToVary+1..N+2)], numToVary);
        linpolystest:=[seq([op(firstlinpolys), op(lastlinpolys[k])], k=1..nops(lastlinpolys))]
    end if;
    G := sum(A[kk]*x^kk, kk = 0 .. N - 1);
    minsize := [infinity, infinity];
    for testnum to nops(linpolystest) do
        if n < 4 or not addextrapoints then
            linpolys := linpolystest[1]
        else linpolys := [seq(linpolystest[testnum][k], k = 1 .. 4),
                          op(sort([seq(linpolystest[testnum][k], k = 5 .. N)], (a, b)
                          -> a[2] < b[2] or a[2] = b[2] and coeff(a[1], x) < coeff(b[1], x)))]
        end if;
        for i to N do R[i]:=coeff(linpolys[i,1],x,1)^n
                            *subs(x=-coeff(linpolys[i,1],x,0)/coeff(linpolys[i,1],x,1),G)
        end do;
        sols := solve({seq(R[k] - r[k], k = 1 .. N)},{seq(A[k], k = 0 .. N - 1)});
        for i to N do
            thelcms[i] := lcm(seq(denom(op(k, rhs(sols[i]))), k = 1 .. nops(rhs(sols[i]))))
        end do;
        LCM := lcm(seq(thelcms[k], k = 1 .. N));
        E := [];
        for i to N do
```

```
            if thelcms[i] <> 1 then E := [op(E), rhs(sols[i])*LCM mod LCM] end if
        end do;
        Mtrx := Matrix(nops(E), N);
        for i to nops(E) do for j to N do
                Mtrx[i, j] := coeff(E[i], r[j], 1)
        end do end do;
        conginfo := getCongInfo(Mtrx, nops(E), N, LCM, [seq(linpolys[k, 2], k = 1 .. N)]);
        if conginfo[1][1] < minsize[1]
        or conginfo[1][1] = minsize[1] and conginfo[1][2] < minsize[2] then
            minlinpolys := linpolys;
            minconginfo := conginfo;
            minsols := sols;
            minLCM := LCM;
            minsize := conginfo[1]
        end if
    end do;
    sols := subs(minsols, [seq(A[k], k = 0 .. n)]);
    Lower := minconginfo[2];
    Upper := minconginfo[3];
    congs := minconginfo[4];
    LCM := minLCM;
    steps := [seq(`if`(congs[k] = 0, 1, LCM/congs[k, 1]), k = 1 .. N)];
    for i to N do
        if congs[i] <> 0 then
            gcdi := igcd(congs[i][1], op(congs[i][3]));
            congs[i][1] := congs[i][1]/gcdi;
            congs[i][3] := congs[i][3]/gcdi
        end if
    end do;
    return [congs, Upper, Lower, LCM, steps, sols, minlinpolys]
end proc;


getCongInfo := proc(M, m, n, modulus, bounds)
local D, Mtrx, a, b, col, congs, est, est2, extendsols, gcd, i, j, k, lower,
      maxnum, maxone, p, piv, q, row, s, sols, t, temp, upper, v, val;
    Mtrx := Matrix(m, n);
    for i to m do for j to n do Mtrx[i, j] := M[i, n + 1 - j] end do end do;
    row := 1;
    i := 1;
    while row <= m and i <= n do
        j := row;
        while j <= m and Mtrx[j, i] = 0 do j := j + 1 end do;
        if row < j and j <= m then for col from i to n do
            temp := Mtrx[row, col]; Mtrx[row, col] := Mtrx[j, col]; Mtrx[j, col] := temp
        end do end if;
        if j <= m then
            piv[row] := i;
            j := j + 1;
            while j <= m do
                if Mtrx[j, i] = Mtrx[row, i] then
                    for col from i to n do
                        Mtrx[j, col] := (Mtrx[j, col] - Mtrx[row, col]) mod modulus
                    end do
                elif Mtrx[j, i] <> 0 then
                    gcd := igcdex(Mtrx[row, i], Mtrx[j, i],'s', 't');
                    p := Mtrx[row, i]/gcd;
                    q := Mtrx[j, i]/gcd;
                    for col from i to n do
                        temp := Mtrx[row, col];
                        Mtrx[row, col] := (s*Mtrx[row, col] + t*Mtrx[j, col]) mod modulus;
                        Mtrx[j, col] := (p*Mtrx[j, col] - q*temp) mod modulus
                    end do
                end if;
                j := j + 1
            end do;
            row := row + 1
        end if;
        i := i + 1
    end do;
    for i from row to m do piv[i] := 0 end do;
```

```
    for i to m do
        gcd := igcd(seq(Mtrx[i, k], k = 1 .. n));
        if gcd <> 0 then
            s := gcd/igcd(gcd, modulus);
            if s <> 1 then for col to n do
                    Mtrx[i, col] := Mtrx[i, col]/s
                end do
            end if
        end if
    end do;
    for i from 2 to m do
        if piv[i] <> 0 then for j to i - 1 do
                s := round(Mtrx[j, piv[i]]/Mtrx[i, piv[i]]);
                for col from piv[i] to n do
                    Mtrx[j, col] := mods(Mtrx[j, col] - s*Mtrx[i, col], modulus)
                end do
            end do
        end if
    end do;
    congs := [seq(0, i = 1 .. n)];
    for i to m do
        if piv[i] <> 0 then
            a := Mtrx[i, piv[i]];
            gcd := igcdex(a, modulus, 's', 't');
            congs[n + 1 - piv[i]] := [gcd, s, [seq(-Mtrx[i, n + 1 - k], k = 1 .. n - piv[i])]]
        end if
    end do;
    upper := [seq(bounds[k], k = 1 .. n)];
    lower := -upper;
    v := 1;
    est := 0;
    for i to n do
        if upper[i] <> infinity then
            if congs[i] = 0 then
                v := v*(1 + upper[i] - lower[i])
            else v := v*ceil((1 + upper[i] - lower[i])*congs[i][1]/modulus)
            end if
        end if;
        est := est + v
    end do;
    est2 := 1;
    for i to min(4, floor(1/2*n)) do
        val := (1 + upper[n - i + 1] - lower[n - i + 1])*congs[n - i + 1][1]/modulus;
        est2 := est2*(val - floor(val))^i
    end do;
    return [[est, est2], lower, upper, congs]
end proc;


applySimp := proc(h, i, LCM, indices, sols, F, N, c, eps, Lower, Upper,
                  steps, addressSimpTable, addressinnerTable, justmax)
local FpowerOftwomN, Fv, Maxri, Maxri2sub, Minri, Minri2sub, STval, SimpTable, cnsts,
      cnstsmax, cnstsmin, count, countmax, denomxval, domax, domin, dothismax, dothismin,
      fixedRes, iTsval, iTval, iTwithsub, innerTable, j, jmaxmax, jmaxmin, jminmax, jminmin,
      lastratmaxri, lastratminri, leftside, leftsidemaxmax, leftsidemaxmin, leftsideminmax,
      leftsideminmin, maxri, minri, multiplier, numerxval, numpass, pass, ratmaxri, ratminri,
      subMaxriIniTval, subMinriIniTval, theBound, theBoundval, theBoundval2use, twomN,
      xpowerOftwomN, xv, xval, xvmaxmax, xvmaxmin, xvminmax, xvminmin;
    SimpTable := pointto(addressSimpTable);
    innerTable := pointto(addressinnerTable);
    iTwithsub := table();
    count := 1;
    if Upper[i] = infinity then
        if justmax then countmax := 6 else countmax := infinity end if
    elif N < 10 then countmax := 4
    elif N = 10 then countmax := max(5 - i, 3)
    elif N = 11 then countmax := max(5 - i, 3)
    elif N = 12 then countmax := max(7 - i, 3)
    elif N = 13 then countmax := max(8 - i, 4)
    elif N = 14 then countmax := max(9 - i, 4)
    else countmax := max(N - 4 - i, 5)
```

```
end if;
fixedRes := {seq(r[k] = indices[k], k = 1 .. h - 1)};
multiplier := `if`(N < 6, `if`(N < 4, 3, 2), 1);
twomN := 2*multiplier*N;
FpowerOftwomN := twomN^degree(F);
xpowerOftwomN := twomN^(2*N - 2);
if Digits < 10 then theBound := (c + eps)*LCM*FpowerOftwomN*xpowerOftwomN;
                    theBound := op(1, theBound)*10^op(2, theBound)
else
    theBound := (c + eps)*LCM*FpowerOftwomN*xpowerOftwomN;
    theBound := evalf(theBound, 5);
    theBound := (op(1, theBound) + 1)*10^op(2, theBound)
end if;
for j to multiplier*N do
    xval := j/twomN;
    numerxval := numer(xval);  denomxval := denom(xval);
    STval := SimpTable[numerxval, denomxval];
    if type(STval, name) then
        Fv[j] := subs(x = xval, F)*FpowerOftwomN;
        xv[j] := [seq((xval*(1 - xval))^k*xpowerOftwomN, k = 0 .. N - 1)];
        SimpTable[numerxval, denomxval] := [xv[j], Fv[j], theBound]
    else xv[j] := STval[1]; Fv[j] := STval[2]
    end if;
    iTval := innerTable[numerxval, denomxval];
    if type(iTval, name) then
        iTval := inner(LCM*sols, xv[j])*Fv[j];
        innerTable[numerxval, denomxval] := iTval
    end if;
    iTwithsub[numerxval, denomxval] := subs(fixedRes, iTval)
end do;
leftside := [seq(iTwithsub[numer(k/twomN), denom(k/twomN)], k = 1 .. multiplier*N)];
cnsts := [seq(`if`(k = i or Upper[k] = infinity, NULL, r[k] <= Upper[k]), k = h .. N),
          seq(`if`(k = i or Lower[k] = -infinity, NULL,-r[k] <= -Lower[k]), k = h .. N),
          seq(leftside[k]/theBound <= 1, k = 1 .. multiplier*N),
          seq(-leftside[k]/theBound <= 1, k = 1 .. multiplier*N)];
cnstsmax := cnsts;
cnstsmin := cnsts;
print(N, `  ---  `, time(), `~~~~~~~~~~~~~~~~~~~`, Lower[i], Upper[i]);
domax := true;
dothismax := true;
if justmax then domin:=false; dothismin:=false else domin:=true; dothismin:=true end if;
maxri := infinity;
ratmaxri := infinity;
minri := -infinity;
ratminri := -infinity;
numpass := 3;
pass := numpass;
while domax or domin do
    if pass = 1 then lastratmaxri := ratmaxri end if;
    if dothismax then
        Maxri := simplex[maximize](r[i], cnstsmax);
        if Maxri = {} then print(infeasible); RETURN({}) end if;
        ratmaxri := rhs(op(select(has, Maxri, r[i])));
        maxri := floor(ratmaxri)
    end if;
    if pass = numpass and 2 < count
    and lastratmaxri - maxri < 2*ratmaxri - 2*maxri then domax := false
    end if;
    if pass = 1 then lastratminri := ratminri end if;
    if dothismin then
        Minri := simplex[minimize](r[i], cnstsmin);
        if Minri = {} then print(infeasible); RETURN({}) end if;
        ratminri := rhs(op(select(has, Minri, r[i])));
        minri := ceil(ratminri)
    end if;
    if pass = numpass and 2 < count and minri - lastratminri < 2*minri - 2*ratminri then
        domin := false
    end if;
    if maxri < minri then
        print(`no feasible integer points`); RETURN({})
    end if;
```

```
if pass = numpass and countmax <= count then
    if countmax < count or 2*lastratmaxri - 2*ratmaxri < steps[i] or N < 15 and 5 < i then
        domax := false
    end if;
    if countmax < count or 2*ratminri - 2*lastratminri < steps[i] or N < 15 and 5 < i then
        domin := false
    end if
end if;
if 2 < count and pass = numpass and 4 < i then
    if 4*ratminri - 4*lastratminri < steps[i] then domin := false end if;
    if 4*lastratmaxri - 4*ratmaxri < steps[i] then domax := false end if
end if;
if 13 < N and i < 5 and count < 4 then
    domax := true;
    if not justmax then domin := true end if
end if;
if justmax then
    if maxri = 0 then domax := false elif count < 3 then domax := true end if
end if;
if pass = numpass or justmax and domax = false then
    if justmax then print([evalf(ratmaxri, 8)], time())
               else print(evalf([ratminri, ratmaxri], 8), time())
    end if
end if;
if domax or domin then
    if pass = numpass then
        multiplier := 2*multiplier;
        twomN := 2*twomN;
        FpowerOftwomN := FpowerOftwomN*2^degree(F);
        xpowerOftwomN := xpowerOftwomN*2^(2*N - 2);
        theBound := theBound*2^(degree(F) + 2*N - 2);
        if domax then dothismax := true end if;
        if domin then dothismin := true end if
    end if;
    jmaxmax := 0;  jmaxmin := 0;  jminmin := 0;  jminmax := 0;
    Maxri2sub := evalf(Maxri);
    Minri2sub := evalf(Minri);
    for j to multiplier*N do
        xval := j/twomN;
        numerxval := numer(xval);
        denomxval := denom(xval);
        STval := SimpTable[numerxval, denomxval];
        if type(STval, name) then
            Fv := subs(x = xval, F)*FpowerOftwomN;
            xv := [seq((xval*(1 - xval))^k*xpowerOftwomN, k = 0 .. N - 1)];
            theBoundval := theBound;
            SimpTable[numerxval, denomxval] := [xv, Fv, theBoundval]
        else
            xv := STval[1];
            Fv := STval[2];
            theBoundval := STval[3]
        end if;
        iTsval := iTwithsub[numerxval, denomxval];
        if type(iTsval, name) then
            iTval := innerTable[numerxval, denomxval];
            if type(iTval, name) then
                iTval := inner(LCM*sols, xv)*Fv;
                innerTable[numerxval, denomxval] := iTval
            end if;
            iTsval := subs(fixedRes, iTval);
            iTwithsub[numerxval, denomxval] := iTsval
        end if;
        theBoundval2use := theBoundval*(1 - 0.1^floor(Digits/2));
        if dothismax then
            subMaxriIniTval := subs(Maxri2sub, iTsval);
            if theBoundval2use <= subMaxriIniTval then
                jmaxmax := jmaxmax + 1;
                xvmaxmax[jmaxmax] := [xval, theBoundval]
            elif subMaxriIniTval <= -theBoundval2use then
                jmaxmin := jmaxmin + 1;
                xvmaxmin[jmaxmin] := [xval, theBoundval]
```

```
                end if
            end if;
            if dothismin then
                subMinriIniTval := subs(Minri2sub, iTsval);
                if theBoundval2use <= subMinriIniTval then
                    jminmax := jminmax + 1;
                    xvminmax[jminmax] := [xval, theBoundval]
                elif subMinriIniTval <= -theBoundval2use then
                    jminmin := jminmin + 1;
                    xvminmin[jminmin] := [xval, theBoundval]
                end if
            end if
        end do;
        if dothismax or domax and pass = numpass then
            leftsidemaxmax := [seq(iTwithsub[numer(xvmaxmax[k][1]),
                denom(xvmaxmax[k][1])], k = 1 .. jmaxmax)];
            leftsidemaxmin := [seq(iTwithsub[numer(xvmaxmin[k][1]),
                denom(xvmaxmin[k][1])], k = 1 .. jmaxmin)];
            if ratmaxri = lastratmaxri then
                cnstsmax := [op(cnstsmax), seq(leftsidemaxmax[k]/xvmaxmax[k][2] <= 1,
                k=1..jmaxmax), seq(-1 <= leftsidemaxmin[k]/xvmaxmin[k][2], k=1..jmaxmin)]
            else cnstsmax := [seq(`if`(k = i or Upper[k] = infinity or
                 rhs(op(select(has, Maxri, r[k]))) < Upper[k], NULL, r[k] <= Upper[k]),
                 k=h..N), seq(`if`(k = i or Lower[k] = -infinity or Lower[k]
                 < rhs(op(select(has, Maxri, r[k]))), NULL, -r[k] <= -Lower[k]), k=h..N),
                 seq(leftsidemaxmax[k]/xvmaxmax[k][2] <= 1, k = 1 .. jmaxmax),
                 seq(-1 <= leftsidemaxmin[k]/xvmaxmin[k][2], k = 1 .. jmaxmin)]
            end if
        end if;
        if dothismin or domin and pass = numpass then
            leftsideminmax := [seq(iTwithsub[numer(xvminmax[k][1]),
                denom(xvminmax[k][1])], k = 1 .. jminmax)];
            leftsideminmin := [seq(iTwithsub[numer(xvminmin[k][1]),
                denom(xvminmin[k][1])], k = 1 .. jminmin)];
            if ratminri = lastratminri then cnstsmin := [op(cnstsmin),
                seq(-1 <= leftsideminmin[k]/xvminmin[k][2], k = 1 .. jminmin),
                seq(leftsideminmax[k]/xvminmax[k][2] <= 1, k = 1 .. jminmax)]
            else cnstsmin := [seq(`if`(k = i or Upper[k] = infinity or
                rhs(op(select(has, Minri, r[k]))) < Upper[k], NULL, r[k] <= Upper[k]),
                k=h..N), seq(`if`(k = i or Lower[k] = -infinity or Lower[k]
                < rhs(op(select(has, Minri, r[k]))), NULL, -r[k] <= -Lower[k]), k=h..N),
                seq(-1 <= leftsideminmin[k]/xvminmin[k][2], k = 1 .. jminmin),
                seq(leftsideminmax[k]/xvminmax[k][2] <= 1, k = 1 .. jminmax)]
            end if
        end if;
        if nops(cnstsmax) < N - h + 1 then domax := false end if;
        if nops(cnstsmin) < N - h + 1 then domin := false end if;
        if nops(cnstsmax) <= N - h + 1 then dothismax := false end if;
        if nops(cnstsmin) <= N - h + 1 then dothismin := false end if
    end if;
    pass := pass + 1;
    if numpass < pass then pass := 1 end if;
    if pass = 1 then count := count + 1 end if
end do;
maxri := min(maxri, Upper[i]);
minri := max(minri, Lower[i]);
if justmax then print([maxri], time()) else print([minri, maxri], time()) end if;
if maxri < minri then RETURN({}) else RETURN([minri, maxri]) end if
end proc;
```

A.2 The total degree case on $[0,1]\times[0,1]$

The following is a Maple implementation of the code outlined in Section 3.4.1. It is intended to be run with the command `searchpolys(n,c,[[x1,y1,b1],...,[xk,yk,bk]],`

`1,[poly1,poly2,...,polym]);` where `n` is the total degree and `c` is an upper bound on $\mathrm{T}_n[0,1]\times[0,1]$. The values `xi` and `yi` are the coordinates of the rational point (x_i, y_i) on the boundary of the unit square and `bi` is a bound on the magnitude of r_i as described in Section 3.4.1. The optional fourth argument can be used to force additional factors and the optional fifth argument must be a list of polynomials forming a basis for the lattice of bivariate polynomials with integer coefficients of total degree at most n satisfying a given symmetry condition. If no fifth argument is given, the standard basis of monomials is used. The procedure `ApplySimp` is used to find the bounds on the r_i. The procedure `searchpolys` calls `getCongInfo` from Appendix A.1.

```
searchpolys := proc(n, c, passedRatBounds)
local D, E, F, G, LCM, Mtrx, N, R, a, b, checkPoints, conginfo, congs, cp, direction,
      eps, firstpoints, flag, gcdi, i, indices, j, k, lastpoints, lower, meshdenom,
      meshsize, minLCM, minconginfo, minpoints, minpoly, minsize, minsols, minsup,
      numToVary, numextra, points, pointstest, poly, rats, size, sol, sols, steps,
      testnum, thelcms, thepoints, thesup, thevars, upper;
    if 6 <= n then F := x*y*(1 - x)*(1 - y)*(x - y)*(1 - x - y) else F := 1 end if;
    if 4 <= nargs then F := F*args[4] end if;
    Digits := max(4*n, 50);
    eps := 1/1000000*c;
    if nargs = 5 then thevars := args[5]
    else
        thevars := [];
        for i from 0 to n - degree(F) do
            thevars :=[op(thevars), seq(x^(i - j)*y^j, j = 0 .. i)]
        end do
    end if;
    N := nops(thevars);
    G := sum(A[k]*thevars[k], k = 1 .. N);
    meshdenom := floor(evalf(N*sqrt(N)));
    meshsize := evalf(1/meshdenom);
    for i to meshdenom - 1 do
        cp[i] := [seq([i*meshsize, j*meshsize], j = 1 .. meshdenom - 1)]
    end do;
    checkPoints := [seq(op(cp[j]), j = 1 .. meshdenom - 1)];
    checkPoints := [seq([[seq(subs({x = checkPoints[j][1],
                          y = checkPoints[j][2]},thevars[k]), k = 1 .. N)],
                    c + eps, abs(subs({x = checkPoints[j][1], y = checkPoints[j][2]}, F))],
                    j = 1 .. nops(checkPoints))];
    if n < 6 then rats := [0]; i := 1; j := 1 else rats := []; i := 2; j := 1 end if;
    while nops(rats) < N + 3 do
        if igcd(i, j) = 1 then rats := [op(rats), j/i] end if;
        j := j + 1;
        if i <= j then i := i + 1; j := 1 end if
    end do;
    thepoints := passedRatBounds;
    for i to nops(rats) do for j to nops(rats) do
            if subs({x = rats[i], y = rats[j]}, F) <> 0 and
               0 < `+`(op(map( z -> abs(subs({x = rats[i], y = rats[j]}, z)), thevars)))
            then thepoints := [op(thepoints), [ rats[i], rats[j],
                floor(lcm(denom(rats[i]), denom(rats[j]))^(n - degree(F))
                      *c/abs(subs({x = rats[i], y = rats[j]}, F)))]]
            end if
        end do
    end do;
    thepoints := sort(thepoints, (a, b) -> a[3] < b[3]);
    points := [thepoints[1]];
    size := 1;
    i := 1;
    R[1] := lcm(denom(thepoints[i][1]), denom(thepoints[i][2]))^(n - degree(F))
            *subs({x = thepoints[i][1], y = thepoints[i][2]}, G);
    while size < N do
```

```
    i := i + 1;
    R[size + 1] := lcm(denom(thepoints[i][1]), denom(thepoints[i][2]))^(n - degree(F))
                   *subs({x = thepoints[i][1], y = thepoints[i][2]}, G);
    sols := solve({seq(R[k] - r[k], k = 1 .. size + 1)}, {seq(A[k], k = 1 .. N)});
    if sols <> NULL then
        points := [op(points), thepoints[i]];
        size := size + 1
    end if
end do;
numextra := 2;
thepoints := [op(points), seq(thepoints[k], k = i + 1 .. i + numextra)];
numToVary := min(floor(1/2*N), 2);
firstpoints := [seq(thepoints[k], k = 1 .. N - numToVary)];
lastpoints := combinat[choose]([seq(thepoints[k],
              k = N - numToVary + 1 .. N + numextra)], numToVary);
pointstest := [seq(sort([op(firstpoints), op(lastpoints[k])],
               (a, b) -> a[3] < b[3]), k = 1 .. nops(lastpoints))];
minsize := [infinity, infinity];
for testnum to nops(pointstest) do
    points := pointstest[testnum];
    for i to N do
        R[i] := lcm(denom(points[i][1]), denom(points[i][2]))^(n - degree(F))
                *subs({x = points[i][1], y = points[i][2]}, G)
    end do;
    sols := solve({seq(R[k] - r[k], k = 1 .. N)},{seq(A[k], k = 1 .. N)});
    if sols <> NULL then
        for i to N do
            thelcms[i] := lcm(seq(denom(op(k, rhs(sols[i]))),
                                  k = 1 .. nops(rhs(sols[i]))))
        end do;
        LCM := lcm(seq(thelcms[k], k = 1 .. N));
        E := [];
        for i to N do
            if thelcms[i] <> 1 then
                E := [op(E), rhs(sols[i])*LCM mod LCM]
            end if
        end do;
        Mtrx := Matrix(nops(E), N);
        for i to nops(E) do for j to N do
                Mtrx[i, j] := coeff(E[i], r[j], 1)
            end do
        end do;
        conginfo := getCongInfo(Mtrx, nops(E), N, LCM, [seq(points[k][3], k = 1 .. N)]);
        if conginfo[1][1] < minsize[1] or
        conginfo[1][1] = minsize[1] and
        conginfo[1][2] < minsize[2] then
            minpoints := points;
            minconginfo := conginfo;
            minsols := sols;
            minLCM := LCM;
            minsize := conginfo[1]
        end if
    end if
end do;
sols := subs(minsols, [seq(A[k], k = 1 .. N)]);
lower := minconginfo[2];
Upper := minconginfo[3];
k := 1;
while Upper[k] = 0 do k := k + 1 end do;
Upper[k] := 0;
congs := minconginfo[4];
LCM := minLCM;
steps := [seq(`if`(congs[k] = 0, 1, LCM/congs[k, 1]), k = 1 .. N)];
for i to N do
    if congs[i] <> 0 then
        gcdi := igcd(congs[i][1], op(congs[i][3]));
        congs[i][1] := congs[i][1]/gcdi;
        congs[i][3] := congs[i][3]/gcdi
    end if
end do;
print(`Upper `, Upper);
```

```
print(‘Lower ‘, lower);
print(‘steps ‘, steps);
print(minsize);
minsup := 1;
minpoly := {1};
indices := [seq(0, k = 1 .. N)];
direction := 1;
i := 1;
while 0 < i do
    if direction = 1 then while direction = 1 do
            if i>1 and indices[i-1]=0 and ‘+‘(seq(abs(indices[k]),k=1..i-1))=0 then
               upper[i]:=0
            else
               upper[i]:=Upper[i];
            fi;
            if direction = 1 then
                if congs[i] = 0 then
                    indices[i] := lower[i];
                    i := i + 1;
                    if i = N + 1 then direction := 0 end if
                else
                    b := inner([seq(indices[k], k = 1 .. i - 1)], congs[i, 3]);
                    D := iquo(b, congs[i, 1], ’rmndr’);
                    if rmndr = 0 then
                        a := congs[i, 2]*D mod steps[i];
                        a := lower[i] + ((a - lower[i]) mod steps[i]);
                        if a <= upper[i] then
                            indices[i] := a;
                            i := i + 1;
                            if i = N + 1 then direction := 0 end if
                        else direction := -1
                        end if
                    else direction := -1
                    end if
                end if
            end if
        end do
    elif direction = -1 then while
        direction = -1 and 0 < i do
            i := i - 1;
            if 0 < i then
                indices[i] := indices[i] + steps[i];
                if indices[i] <= upper[i] then
                    i := i + 1;
                    direction := ‘if‘(0 < i - N, 0, 1)
                end if
            end if
        end do
    else
        direction := -1;
        sol := subs({seq(r[k] = indices[k], k = 1 .. N)}, sols);
        j := 0;
        flag := true;
        while j < nops(checkPoints) and flag do
            j := j + 1;
            if checkPoints[j, 2] < abs(inner(checkPoints[j, 1], sol)*checkPoints[j, 3])
            then flag := false
            end if
        end do;
        if flag then
            poly := factor(inner(sol, thevars)*F);
            thesup := maxi(poly);
            print(poly);
            print(evalf(thesup, 10));
            print(indices);
            print(‘---------------------------‘);
            if thesup <> 0 and thesup < minsup - eps then
                minsup := thesup; minpoly := {poly}
            elif thesup <> 0 and thesup < minsup + eps
            then
                minpoly := minpoly union {poly};
```

```
                minsup := min(minsup, thesup)
            end if
        end if
    end if
end do;
print(`min sup norm is `, minsup);
RETURN(minpoly)
end proc;


maxi := proc (p)
local m1, m2;
    m1 := maximize(evalf(p),x = 0 .. 1,y = 0 .. 1);
    m2 := maximize(evalf(-p),x = 0 .. 1,y = 0 .. 1);
    return evalf(max(m1,m2))
end proc;


ApplySimp := proc(thePoint, F, c, bounds, polys)
local Fval, Maxri, boundConstraints, count, countmax, denomVal, i, j, leftside,
      maxTable, maxconstraints, maxval, n, numpointsMax, objectiveFunc, poly;
    n := max(op(map(p -> degree(p, x), polys)), op(map(p -> degree(p, y), polys)));
    poly := `+`(seq(polys[i]*a[i], i = 1 .. nops(polys)));
    boundConstraints := [seq(subs({x = bounds[i][1], y = bounds[i][2]}, poly)*
                            lcm(denom(bounds[i][1]), denom(bounds[i][2]))^n
                            <= bounds[i][3], i = 1 .. nops(bounds)),
                        seq(-subs({x = bounds[i][1], y = bounds[i][2]}, poly)*
                            lcm(denom(bounds[i][1]), denom(bounds[i][2]))^n
                            <=bounds[i][3], i = 1 .. nops(bounds))];
    maxTable := table();
    if n < 6 then denomVal := 8 else denomVal := 16 end if;
    numpointsMax := 0;
    for i from 0 to denomVal do for j from 0 to denomVal do
            Fval := subs({x = i/denomVal, y = j/denomVal}, F);
            if Fval <> 0 then
                leftside := Fval*subs({x = i/denomVal, y = j/denomVal}, poly)/c;
                numpointsMax := numpointsMax + 1;
                maxTable[numpointsMax] := leftside <= 1;
                numpointsMax := numpointsMax + 1;
                maxTable[numpointsMax] := -leftside <= 1
            end if
        end do
    end do;
    objectiveFunc := subs({x = thePoint[1], y = thePoint[2]}, poly)*
                         lcm(denom(thePoint[1]), denom(thePoint[2]))^n;
    maxconstraints := [op(boundConstraints), seq(maxTable[i], i = 1 .. numpointsMax)];
    count := 1;
    countmax := 6;
    while count < countmax do
        count := count + 1;
        Maxri := simplex[maximize](objectiveFunc, maxconstraints);
        maxval := subs(Maxri, objectiveFunc);
        print(`maxval  `, evalf([maxval], 5));
        if count < countmax then
            denomVal := 2*denomVal;
            maxTable := table();
            numpointsMax := 0;
            for i from 0 to denomVal do
                if gcd(i, denomVal) = 1 then for j from 0 to
                    denomVal do
                        if gcd(j, denomVal) = 1 then
                            Fval := subs({x = i/denomVal, y = j/denomVal}, F);
                            if Fval <> 0 then
                                leftside := Fval
                                    *subs({x = i/denomVal, y = j/denomVal}, poly)/c;
                                if 1 < subs(Maxri, leftside)
                                then
                                    numpointsMax := numpointsMax + 1;
                                    maxTable[numpointsMax]:= leftside <= 1
                                end if;
                                if 1 < subs(Maxri, -leftside)
```

```
                                then
                                    numpointsMax := numpointsMax + 1;
                                    maxTable[numpointsMax] := -leftside <= 1
                                end if
                            end if
                        end if
                    end do
                end if
            end do;
            maxconstraints := [op(maxconstraints), seq(maxTable[i], i = 1 .. numpointsMax)]
        end if;
        print(`points added  `, numpointsMax)
    end do;
    return maxval
end proc;
```

A.3 The maximum degree case on $[0,1]\times[0,1]$

The following is a Maple implementation of the code outlined in Section 3.4.2. It is intended to be run with the command `searchpolys(n,c,[[x1,y1,b1],...,[xk,yk,bk]],poly);` where `n` is the maximum degree and `c` is an upper bound on $\mathrm{M}_n[0,1]\times[0,1]$. The values `xi` and `yi` are the coordinates of the rational point (x_i, y_i) on the boundary of the quarter square and `bi` is a bound on the magnitude of r_i as described in Section 3.4.2. The optional fourth argument `poly` can be used to force additional factors beyond those that will be automatically added. The procedure `ApplySimp` is used to find the bounds on the r_i. The procedure `searchpolys` calls `getCongInfo` from Appendix A.1.

```
searchpolys := proc(n, c, passedRatBounds)
local D, E, F, G, LCM, Lower, Mtrx, N, R, Upper, a, b, checkPoints, conginfo, congs, cp,
      direction, eps, firstpoints, flag, gcdi, i, indices, j, lastpoints, lower, meshsize,
      minLCM, minconginfo, minpoints, minpoly, minsize, minsols, minsup, numToVary,
      numextra, points, pointstest, poly, rats, size, sol, sols, steps, testnum, thelcms,
      thepoints, thesup, thevars, u, upper;
    if n = 0 then return {1} end if;
    if 2 <= n then F := x*(1 - x)*y*(1 - y) else F := 1 end if;
    if n mod 2 = 1 then F := F*(1 - 2*x)*(1 - 2*y) end if;
    if nargs = 4 then F := F*args[4] end if;
    Digits := max(4*n, 20);
    eps := 1/1000000*c;
    thevars := [];
    if degree(F, x) = n then return F end if;
    for i from 0 to 1/2*n - 1/2*degree(F, x) do
        thevars := [op(thevars), seq(x^i*y^j, j = 0 .. 1/2*n - 1/2*degree(F, x))]
    end do;
    N := nops(thevars);
    G := sum(A[k]*thevars[k], k = 1 .. N);
    u := unapply(1/2 - 1/2*sqrt(1 - 4*x), x);
    meshsize := evalf(1/16*1/N);
    for i to 4*N do cp[i] := [seq([i*meshsize, j*meshsize], j = 1 .. 4*N)] end do;
    checkPoints := [seq(op(cp[j]), j = 1 .. 4*N)];
    checkPoints := [seq([[seq(subs({x = checkPoints[j][1],
                                   y = checkPoints[j][2]}, thevars[k]), k = 1 .. N)],
                    c + eps, abs(subs({x = u(checkPoints[j][1]), y = u(checkPoints[j][2])},
                                                F))], j = 1 .. nops(checkPoints))];
    if n < 3 then rats := [0]; i := 4; j := 1 else rats := []; i := 4; j := 1 end if;
    while nops(rats) < ceil(sqrt(N)) + 2*N do
        if igcd(i, j) = 1 then rats := [op(rats), j/i] end if;
        j := j + 1;
```

```
    if i < 4*j then i := i + 1; j := 1 end if
end do;
thepoints := passedRatBounds;
for i to nops(rats) do for j to nops(rats) do
        if subs({x = u(rats[i]), y = u(rats[j])}, F) <> 0
        then thepoints := [op(thepoints), [rats[i], rats[j], floor((denom(rats[i])
                           *denom(rats[j]))^(1/2*n - 1/2*degree(F, x))
                           *c/abs(subs({x = u(rats[i]), y = u(rats[j])}, F)))]]
        end if
    end do
end do;
thepoints := sort(thepoints, (a, b) -> a[3] < b[3]);
points := [thepoints[1]];
size := 1;
i := 1;
R[1] := (denom(thepoints[i][1])*denom(thepoints[i][2]))^(1/2*n - 1/2*degree(F, x))
       *subs({x = thepoints[i][1], y = thepoints[i][2]}, G);
while size < N do
    i := i + 1;
    R[size + 1] := (denom(thepoints[i][1])
                   *denom(thepoints[i][2]))^(1/2*n - 1/2*degree(F, x))
                   *subs({x = thepoints[i][1], y = thepoints[i][2]}, G);
    sols := solve({seq(R[k] - r[k], k = 1 .. size + 1)}, {seq(A[k], k = 1 .. N)});
    if sols <> NULL then
        points := [op(points), thepoints[i]];
        size := size + 1
    end if
end do;
numextra := 5;
thepoints := [op(points), seq(thepoints[k], k = i + 1 .. i + numextra)];
numToVary := 3;
firstpoints := [seq(thepoints[k], k = 1 .. N - numToVary)];
lastpoints := combinat[choose]([seq(thepoints[k],
              k = N - numToVary + 1 .. N + numextra)], numToVary);
pointstest := [seq(sort([op(firstpoints), op(lastpoints[k])],
               (a, b) -> a[3] < b[3]), k = 1 .. nops(lastpoints))];
minsize := [infinity, infinity];
for testnum to nops(pointstest) do
    points := pointstest[testnum];
    for i to N do R[i] :=
        (denom(points[i][1])*denom(points[i][2]))^(1/2*n - 1/2*degree(F, x))
        *subs({x = points[i][1], y = points[i][2]}, G)
    end do;
    sols := solve({seq(R[k] - r[k], k = 1 .. N)}, {seq(A[k], k = 1 .. N)});
    if sols <> NULL then
        for i to N do thelcms[i] := lcm(seq(denom(op(k, rhs(sols[i]))),
                                           k = 1 .. nops(rhs(sols[i]))))
        end do;
        LCM := lcm(seq(thelcms[k], k = 1 .. N));
        E := [];
        for i to N do
            if thelcms[i] <> 1 then E := [op(E), rhs(sols[i])*LCM mod LCM] end if
        end do;
        Mtrx := Matrix(nops(E), N);
        for i to nops(E) do for j to N do
            Mtrx[i, j] := coeff(E[i], r[j], 1)
        end do end do;
        conginfo := getCongInfo(Mtrx, nops(E), N, LCM, [seq(points[k][3], k = 1 .. N)]);
        if conginfo[1][1] < minsize[1]
        or conginfo[1][1] = minsize[1] and conginfo[1][2] < minsize[2] then
            minpoints := points;
            minconginfo := conginfo;
            minsols := sols;
            minLCM := LCM;
            minsize := conginfo[1]
        end if
    end if
end do;
sols := subs(minsols, [seq(A[k], k = 1 .. N)]);
Lower := minconginfo[2];
Upper := minconginfo[3];
```

```
Upper[1] := 0;
congs := minconginfo[4];
LCM := minLCM;
steps := [seq(`if`(congs[k] = 0, 1, LCM/congs[k, 1]), k = 1 .. N)];
for i to N do
    if congs[i] <> 0 then
        gcdi := igcd(congs[i][1], op(congs[i][3]));
        congs[i][1] := congs[i][1]/gcdi;
        congs[i][3] := congs[i][3]/gcdi
    end if
end do;
print(`Upper `, Upper);
print(`Lower `, Lower);
print(`steps `, steps);
print(minsize);
minsup := 1;
minpoly := {1};
indices := [seq(0, k = 1 .. N)];
direction := 1;
i := 1;
while 0 < i do
    if direction = 1 then while direction = 1 do
            if 1<i and indices[i-1]=0 and `+`(seq(abs(indices[k]), k=1..i-1))=0 then
                upper[i] := 0
            else
                upper[i] := Upper[i]
            end if;
            lower[i] := Lower[i];
            if direction = 1 then
                if congs[i] = 0 then
                    indices[i] := lower[i];
                    i := i + 1;
                    if i = N + 1 then direction := 0 end if
                else
                    b := inner([seq(indices[k], k = 1 .. i - 1)], congs[i, 3]);
                    D := iquo(b, congs[i, 1], 'rmndr');
                    if rmndr = 0 then
                        a := congs[i, 2]*D mod steps[i];
                        a := lower[i] + ((a - lower[i]) mod steps[i]);
                        if a <= upper[i] then
                            indices[i] := a;
                            i := i + 1;
                            if i = N + 1 then direction := 0 end if
                        else direction := -1
                        end if
                    else direction := -1
                    end if
                end if
            end if
        end do
    elif direction = -1 then while
        direction = -1 and 0 < i do
            i := i - 1;
            if 0 < i then
                indices[i] := indices[i] + steps[i];
                if indices[i] <= upper[i] then
                    i := i + 1;
                    direction := `if`(0 < i - N, 0, 1)
                end if
            end if
        end do
    else
        direction := -1;
        sol := subs({seq(r[k] = indices[k], k = 1 .. N)}, sols);
        j := 0;
        flag := true;
        while j < nops(checkPoints) and flag do
            j := j + 1;
            if checkPoints[j, 2] < abs(inner(checkPoints[j, 1], sol)*checkPoints[j, 3])
            then flag := false
            end if
```

```
            end do;
            if flag then
                poly := inner(sol, thevars);
                poly := factor(subs({x = x*(1 - x), y = y*(1 - y)}, poly)*F);
                thesup := maxi(poly);
                print(poly);
                print(evalf(thesup, 10));
                print(indices);
                print(`----------------------------`);
                if thesup <> 0 and thesup < minsup - eps then
                    minsup := thesup; minpoly := {poly}
                elif thesup <> 0 and thesup < minsup + eps
                then
                    minpoly := minpoly union {poly};
                    minsup := min(minsup, thesup)
                end if
            end if
        end if
    end do;
    print(`min sup norm is `, minsup);
    RETURN(minpoly)
end proc;


maxi := proc (p)
local m1, m2;
    m1 := maximize(evalf(p),x = 0 .. 1,y = 0 .. 1);
    m2 := maximize(evalf(-p),x = 0 .. 1,y = 0 .. 1);
    return evalf(max(m1,m2))
end proc;


ApplySimp := proc(thePoint, N, c, F, bounds)
local Fval, Maxri, boundConstraints, count, countmax, denomVal, i, j, leftside, maxTable,
      maxconstraints, maxval, n, numpointsMax, objectiveFunc, poly, thevars;
    if N <= degree(F, x) then return 1 end if;
    n := N - degree(F, x);
    thevars := [];
    for i from 0 to 1/2*n do thevars := [op(thevars), seq(x^i*y^j, j = 0 .. 1/2*n)] end do;
    poly := `+`(seq(thevars[i]*a[i], i = 1 .. nops(thevars)));
    boundConstraints := [seq(subs({x = bounds[i][1], y = bounds[i][2]}, poly)
                        *(denom(bounds[i][1])*denom(bounds[i][2]))^(1/2*n)
                        <= bounds[i][3], i = 1 .. nops(bounds)),
                        seq(-subs({x = bounds[i][1], y = bounds[i][2]}, poly)
                        *(denom(bounds[i][1])*denom(bounds[i][2]))^(1/2*n)
                        <= bounds[i][3], i = 1 .. nops(bounds))];
    maxTable := table();
    denomVal := max(16, N);
    numpointsMax := 0;
    for i from 0 to denomVal do for j from 0 to denomVal do
            Fval := subs({x = i/denomVal, y = j/denomVal}, F);
            if Fval <> 0 then
                leftside := Fval*subs({x = i*(1 - i/denomVal)/denomVal,
                                       y = j*(1 - j/denomVal)/denomVal}, poly)/c;
                numpointsMax := numpointsMax + 1;
                maxTable[numpointsMax] := leftside <= 1;
                numpointsMax := numpointsMax + 1;
                maxTable[numpointsMax] := -leftside <= 1
            end if
        end do
    end do;
    objectiveFunc := subs({x = thePoint[1], y = thePoint[2]}, poly)
                     *(denom(thePoint[1])*denom(thePoint[2]))^(1/2*n);
    maxconstraints := [op(boundConstraints), seq(maxTable[i], i = 1 .. numpointsMax)];
    count := 1;
    countmax := 6;
    while count < countmax do
        count := count + 1;
        Maxri := simplex[maximize](objectiveFunc, maxconstraints);
        maxval := subs(Maxri, objectiveFunc);
        print(`maxval  `, evalf([maxval], 5));
```

```
        if count < countmax then
            denomVal := 2*denomVal;
            maxTable := table();
            numpointsMax := 0;
            for i from 0 to denomVal do
                if gcd(i, denomVal) = 1 then for j from 0 to
                    denomVal do
                        if gcd(j, denomVal) = 1 then
                            Fval := subs({x = i/denomVal, y = j/denomVal}, F);
                            if Fval <> 0 then
                                leftside := Fval*subs({x = i*(1 - i/denomVal)/denomVal,
                                                       y = j*(1 - j/denomVal)/denomVal}, poly)/c;
                                if 1 < subs(Maxri, leftside) then
                                    numpointsMax := numpointsMax + 1;
                                    maxTable[numpointsMax] := leftside <= 1
                                end if;
                                if 1 < subs(Maxri, -leftside) then
                                    numpointsMax := numpointsMax + 1;
                                    maxTable[numpointsMax] := -leftside <= 1
                                end if
                            end if
                        end if
                    end do
                end if
            end do;
            maxconstraints := [op(maxconstraints), seq(maxTable[i], i = 1 .. numpointsMax)]
        end if;
        print(‘points added  ‘, numpointsMax)
    end do;
    return maxval
end proc;
```

Appendix B

Integer Relation Algorithms (Maple Code)

B.1 The LLL algorithm

The following is a Maple implementation of the LLL algorithm and corresponding integer relation algorithm as outlined in Section 2.1. For a selection of improvements to the basic algorithm given here, one is referred to [10].

```
LLL := proc(L)
local n, N, i, j, k, Mu, q, B, B2, nB2, v1, v2, nv1, nv2, m, tmp;
    n := nops(L);
    N := nops(L[1]);
    B := array(1 .. n, 1 .. N);
    B2 := array(1 .. n, 1 .. N);
    nB2 := array(1 .. n);
    Mu := array(1 .. n, 1 .. N);
    for i to n do for j to N do
            B[i, j] := L[i][j];
            if i = j then Mu[i, j] := 1 else Mu[i, j] := 0 fi
    od od;
    for i to n do
        v1 := [seq(B[i, k], k = 1 .. N)];
        for j to i - 1 do
            v2 := [seq(B2[j, k], k = 1 .. N)];
            Mu[i, j] := inner(v1, v2)/inner(v2, v2);
            for k to N do v1[k] := v1[k] - Mu[i, j]*v2[k] od
        od;
        for k to N do B2[i, k] := v1[k] od;
        nB2[i] := inner(v1, v1)
    od;
    k := 2;
    while k < n + 1 do
        for j from k - 1 by -1 to 1 do
            q := round(Mu[k, j]);
            if q <> 0 then for i to N do
                    B[k, i] := B[k, i] - q*B[j, i];
                    Mu[k, i] := Mu[k, i] - q*Mu[j, i]
                od
            fi
```

```
        od;
        if (3/4 - Mu[k, k - 1]^2)*nB2[k - 1] <= nB2[k] then k := k + 1
        else
            for i to N do tmp := B[k, i]; B[k, i] := B[k - 1, i]; B[k - 1, i] := tmp od;
            for i to N do v1[i] := B2[k, i] + Mu[k, k - 1]*B2[k - 1, i] od;
            nv1 := nB2[k] + Mu[k, k - 1]^2*nB2[k - 1];
            m := Mu[k, k - 1]*nB2[k - 1]/nv1;
            for i to N do v2[i] := B2[k - 1, i] - m*v1[i] od;
            nv2 := nB2[k]*nB2[k - 1]/nv1;
            for i to k - 2 do
                tmp := Mu[k, i]; Mu[k, i] := Mu[k - 1, i]; Mu[k - 1, i] := tmp
            od;
            for i from k + 1 to n do
                tmp := Mu[i, k];
                Mu[i, k] := Mu[i, k - 1] - Mu[k, k - 1]*Mu[i, k];
                Mu[i, k - 1] := tmp + m*Mu[i, k]
            od;
            Mu[k, k - 1] := m;
            for i to N do B2[k - 1, i] := v1[i]; B2[k, i] := v2[i] od;
            nB2[k - 1] := nv1;
            nB2[k] := nv2;
            k := max(2, k - 1)
        fi
    od;
    RETURN([seq([seq(B[i, j], j = 1 .. N)], i = 1 .. n)])
end;


intRel := proc(x)
local i, j, B, B2, n;
    n := nops(x);
    B := array(1 .. n + 1, 1 .. n);
    for i to n do for j to n do if i = j then B[i, j] := 1 else B[i, j] := 0 fi od od;
    for i to n do B[n + 1, i] := 10^Digits*x[i] od;
    B2 := LLL( [seq([seq(B[i, j], i = 1 .. n + 1)], j = 1 .. n)], n);
    RETURN([seq(B2[1][i], i = 1 .. n)])
end;


Examples:

> LLL([[1,2,3],[1,2,4],[0,2,1]]);

                  [[0, 0, 1], [-1, 0, 0], [0, 2, 0]]

> Digits:=10:
> intRel([11,27,31]);

                              [1, -5, 4]

> Digits:=60:
> alpha:=3^(1/4)-2^(1/4):
> intRel([seq(evalf(alpha^i),i=0..16)]);

     [1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1]
```

B.2 The HJLS algorithm with full reductions

The following is a Maple implementation of the HJLS algorithm with full reductions as outlined in Section 2.3.1.

```
HJLS := proc(x)
local A, B, M, r2, n, i, j, k, ii, t, r, maxm, maxmr2, anorm,
      alpha, beta, lambda, delta, ld, bd, v1, v2, h0, tol;
    n := nops(x);
```

```
A := array(1 .. n, 1 .. n);
B := array(1 .. n, 1 .. n);
M := array(1 .. n, 1 .. n);
v1 := array(1 .. n);
v2 := array(1 .. n);
h0 := x/evalf(sqrt(inner(x, x)));
r2 := [seq(evalf(sqrt(2)^i), i = 1 .. n - 1)];
tol := max(floor(9/10*Digits), Digits - 4);
for i to n do for j to n do
        if i = j then A[i, j] := 1; B[i, j] := 1; M[i, j] := 1
                 else A[i, j] := 0; B[i, j] := 0; M[i, j] := 0
        fi
od od;
M[n, n] := 0;
for j to n - 1 do
    v1 := [seq(M[i, j], i = 1 .. n)];
    for k from 0 to j - 1 do
        if k = 0 then v2 := h0 else v2 := [seq(M[i, k], i = 1 .. n)] fi;
        v1 := v1 - inner(v1, v2)*v2
    od;
    v1 := v1/sqrt(inner(v1, v1));
    for i from j to n do M[i, j] := v1[i] od
od;
for i from 2 to n do for j from i - 1 by -1 to 1 do
        if M[j, j] < 2*abs(M[i, j]) then
            t := round(M[i, j]/M[j, j]);
            for k to n do
                M[i, k] := M[i, k] - t*M[j, k];
                A[i, k] := A[i, k] - t*A[j, k];
                B[k, j] := B[k, j] + t*B[k, i]
            od
        fi
od od;
r := 1;
maxm := M[1, 1];
maxmr2 := r2[1]*M[1, 1];
for i from 2 to n - 1 do
    if maxm < M[i, i] then maxm := M[i, i] fi;
    t := r2[i]*M[i, i];
    if maxmr2 <= t then maxmr2 = t; r := i fi
od;
anorm := 1;
while 10^(-tol) < abs(M[n, n - 1]/anorm) or r <> n - 1 do
    alpha := M[r, r];
    beta := M[r + 1, r];
    lambda := M[r + 1, r + 1];
    delta := sqrt(beta^2 + lambda^2);
    ld := lambda/delta;
    bd := beta/delta;
    for k to n do
        t := A[r, k]; A[r, k] := A[r + 1, k]; A[r + 1, k] := t;
        t := B[k, r]; B[k, r] := B[k, r + 1]; B[k, r + 1] := t;
        t := M[r, k]; M[r, k] := M[r + 1, k]; M[r + 1, k] := t
    od;
    M[r, r] := delta;
    M[r, r + 1] := 0;
    M[r + 1, r] := alpha*bd;
    M[r + 1, r + 1] := alpha*ld;
    for k from r + 2 to n do
        t := M[k, r];
        M[k, r] := bd*t + ld*M[k, r + 1];
        M[k, r + 1] := ld*t - bd*M[k, r + 1]
    od;
    for i from r + 1 to n do
        if i = r + 1 then ii := r else ii := r + 1 fi;
        for j from ii by -1 to 1 do
            if M[j, j] < 2*abs(M[i, j]) then
                t := round(M[i, j]/M[j, j]);
                for k to n do
                    M[i, k] := M[i, k] - t*M[j, k];
                    A[i, k] := A[i, k] - t*A[j, k];
```

```
                    B[k, j] := B[k, j] + t*B[k, i]
                od
            fi
        od od;
        v1 := [seq(A[n - 1, i], i = 1 .. n)];
        anorm := evalf(sqrt(inner(v1, v1)));
        r := 1;
        maxm := M[1, 1];
        maxmr2 := r2[1]*M[1, 1];
        for i from 2 to n - 1 do
            if maxm < M[i, i] then maxm := M[i, i] fi;
            t := r2[i]*M[i, i];
            if maxmr2 <= t then maxmr2 := t; r := i fi
        od
    od;
    RETURN([seq(B[i, n - 1], i = 1 .. n)])
end;


Examples:

> HJLS([11,27,31]);

                            [-1, 5, -4]

> Digits:=80:
  a:=evalf(3^(1/4)-2^(1/4)):
  HJLS([seq(a^i,i=0..16)]);

     [1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1]
```

B.3 The basic PSLQ algorithm

The following is a Maple implementation of the basic PSLQ algorithm as outlined in Section 2.4.1. Improved practical algorithms are presented in the next two sections.

```
pslq := proc(x, T)
local n, B, H, y, i, j, k, gamma, G, GH, miny, minyp, maxHii, s,
      t, r, temp, alpha, beta, lambda, delta, bd, ld, a, b, eps, v;
    eps := evalf(10^min(-floor(9/10*Digits), -Digits + 4));
    n := nops(x);
    gamma := evalf(sqrt(4/3)) + 10^(-Digits + 1);
    B := array(1 .. n, 1 .. n);
    for j to n do for k to n do
            if j = k then B[k, k] := 1 else B[j, k] := 0 fi
        od
    od;
    H := array(1 .. n, 1 .. n - 1);
    s := array(1 .. n);
    for k to n do s[k] := evalf(sqrt(sum(x[jj]^2, jj = k .. n))) od;
    for i to n do
        for j from i + 1 to n - 1 do H[i, j] := 0 od;
        if i <= n - 1 then H[i, i] := s[i + 1]/s[i] fi;
        for j to i - 1 do H[i, j] := evalf(- x[i]*x[j]/(s[j]*s[j + 1])) od
    od;
    y := evalf(x/s[1]);
    for i from 2 to n do for j from i - 1 by -1 to 1 do
            if H[j, j] < 2*abs(H[i, j]) then
                t := round(H[i, j]/H[j, j]);
                y[j] := y[j] + t*y[i];
                for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                for k to n do B[k, j] := B[k, j] + t*B[k, i] od
            fi
    od od;
    G := array(1 .. n - 1);
```

```
    G[1] := gamma;
    GH := array(1 .. n - 1);
    GH[1] := H[1, 1];
    for i from 2 to n - 1 do G[i] := G[i - 1]*gamma; GH[i] := G[i]*H[i, i] od;
    miny := 1;
    maxHii := 1;
    while eps < miny and 1/maxHii < T do
        r := 1;
        for i from 2 to n - 1 do if GH[r] < GH[i] then r := i fi od;
        temp := y[r];  y[r] := y[r + 1];  y[r + 1] := temp;
        for i to n - 1 do
            temp := B[i, r];  B[i, r] := B[i, r + 1];  B[i, r + 1] := temp;
            temp := H[r, i];  H[r, i] := H[r + 1, i];  H[r + 1, i] := temp
        od;
        temp := B[n, r];  B[n, r] := B[n, r + 1];  B[n, r + 1] := temp;
        if r < n - 1 then
            alpha := H[r + 1, r];    beta := H[r, r];
            lambda := H[r, r + 1];   delta := sqrt(beta^2 + lambda^2);
            bd := beta/delta;        ld := lambda/delta;
            H[r, r] := delta;        H[r, r + 1] := 0;
            H[r + 1, r] := alpha*bd; H[r + 1, r + 1] := - alpha*ld;
            for i from r + 2 to n do
                a := H[i, r];
                b := H[i, r + 1];
                H[i, r] := a*bd + b*ld;
                H[i, r + 1] := b*bd - a*ld
            od;
            GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
        fi;
        GH[r] := G[r]*abs(H[r, r]);
        for i from r + 1 to n do for j from min(i - 1, r + 1)
            by -1 to 1 do
                if abs(H[j, j]) < 2*abs(H[i, j]) then
                    t := round(H[i, j]/H[j, j]);
                    for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                    for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
                    y[j] := y[j] + t*y[i]
                fi
        od od;
        miny := abs(y[1]);
        minyp := 1;
        for i from 2 to n do if abs(y[i]) < miny then miny := abs(y[i]); minyp := i fi od;
        v := [seq(B[i, minyp], i = 1 .. n)];
        miny := evalf(miny/sqrt(inner(v, v)));
        maxHii := abs(H[1, 1]);
        for i from 2 to n - 1 do if maxHii < abs(H[i, i]) then  maxHii = abs(H[i, i]) fi od
    od;
    if eps < miny then RETURN(‘Lower Bound:‘,
        evalf(1/maxHii, min(Digits, 6)))
    else RETURN([seq(B[i, minyp], i = 1 .. n)])
    fi
end;


Examples:

> pslq([11,27,31],100);

                              [-1, 5, -4]

> pslq([113,343,31,112],100);

                             [-7, 1, 0, 4]

> Digits:=70:
> alpha:=evalf(3^(1/4)-2^(1/4)):
> pslq([seq(alpha^i,i=0..16)],10000);
>

     [-1, 0, 0, 0, 3860, 0, 0, 0, 666, 0, 0, 0, 20, 0, 0, 0, -1]
```

```
> Digits:=32:
> alpha:=evalf( 3/4*Pi - 7/8*exp(1) + 1/9 ):
> pslq([-alpha,Pi,exp(1),sqrt(2),1],100);

                          [72, 54, -63, 0, 8]

> %/(72);

                        [1, 3/4, -7/8, 0, 1/9]
```

B.4 The PSLQ algorithm with periodic full reductions

The following is a Maple implementation of the PSLQ algorithm with periodic full reductions as outlined in Section 2.4.2.

```
pslq := proc(x, T)
local n, B, H, y, i, j, k, gamma, G, GH, miny, minyp, maxHii, s, t, r,
      temp, alpha, beta, lambda, delta, bd, ld, a, b, eps, v, doFullRed;
    eps := evalf(10^min(-floor(9/10*Digits), -Digits + 4));
    n := nops(x);
    gamma := evalf(sqrt(4/3)) + 10^(-Digits + 1);
    B := array(1 .. n, 1 .. n);
    for j to n do for k to n do if j = k then B[k, k] := 1 else B[j, k] := 0 fi od od;
    H := array(1 .. n, 1 .. n - 1);
    s := array(1 .. n);
    for k to n do s[k] := evalf(sqrt(sum(x[jj]^2, jj = k .. n))) od;
    for i to n do
        for j from i + 1 to n - 1 do H[i, j] := 0 od;
        if i <= n - 1 then H[i, i] := s[i + 1]/s[i] fi;
        for j to i - 1 do H[i, j] := evalf(- x[i]*x[j]/(s[j]*s[j + 1])) od
    od;
    y := evalf(x/s[1]);
    for i from 2 to n do for j from i - 1 by -1 to 1 do
            if H[j, j] < 2*abs(H[i, j]) then
                t := round(H[i, j]/H[j, j]);
                y[j] := y[j] + t*y[i];
                for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                for k to n do B[k, j] := B[k, j] + t*B[k, i] od
            fi
    od od;
    G := array(1 .. n - 1);
    G[1] := gamma;
    GH := array(1 .. n - 1);
    GH[1] := H[1, 1];
    for i from 2 to n - 1 do G[i] := G[i - 1]*gamma; GH[i] := G[i]*H[i, i] od;
    miny := 1;
    maxHii := 1;
    while eps < miny and 1/maxHii < T do
        doFullRed := false;
        while not doFullRed do
            r := 1;
            for i from 2 to n - 1 do if GH[r] < GH[i] then r := i fi od;
            if abs(H[r, r]) < 2*abs(H[r + 1, r]) then
                t := round(H[r + 1, r]/H[r, r]);
                y[r] := y[r] + t*y[r + 1];
                for j to r do H[r + 1, j] := H[r + 1, j] - t*H[r, j] od;
                for j to n do B[j, r] := B[j, r] + t*B[j, r + 1] od
            fi;
            temp := y[r];  y[r] := y[r + 1];  y[r + 1] := temp;
            for i to n - 1 do
                temp := B[i, r];  B[i, r] := B[i, r + 1];  B[i, r + 1] := temp;
                temp := H[r, i];  H[r, i] := H[r + 1, i];  H[r + 1, i] := temp
            od;
            temp := B[n, r];  B[n, r] := B[n, r + 1];  B[n, r + 1] := temp;
```

```
            if r < n - 1 then
                alpha := H[r + 1, r];    beta := H[r, r];
                lambda := H[r, r + 1];   delta := sqrt(beta^2 + lambda^2);
                bd := beta/delta;        ld := lambda/delta;
                H[r, r] := delta;        H[r, r + 1] := 0;
                H[r + 1, r] := alpha*bd; H[r + 1, r + 1] := - alpha*ld;
                for i from r + 2 to n do
                    a := H[i, r];
                    b := H[i, r + 1];
                    H[i, r] := a*bd + b*ld;
                    H[i, r + 1] := b*bd - a*ld
                od;
                GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
            else doFullRed := true
            fi;
            GH[r] := G[r]*abs(H[r, r])
        od;
        for i from 2 to n do for j from i - 1 by -1 to 1 do
                if H[j, j] < 2*abs(H[i, j]) then
                    t := round(H[i, j]/H[j, j]);
                    y[j] := y[j] + t*y[i];
                    for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                    for k to n do B[k, j] := B[k, j] + t*B[k, i] od
                fi
        od od;
        miny := abs(y[1]);
        minyp := 1;
        for i from 2 to n do if abs(y[i]) < miny then miny := abs(y[i]); minyp := i fi od;
        v := [seq(B[i, minyp], i = 1 .. n)];
        miny := evalf(miny/sqrt(inner(v, v)));
        maxHii := abs(H[1, 1]);
        for i from 2 to n - 1 do if maxHii < abs(H[i, i]) then maxHii = abs(H[i, i]) fi od
    od;
    if eps < miny then RETURN(‘Lower Bound:‘, evalf(1/maxHii, min(Digits, 6)))
    else RETURN([seq(B[i, minyp], i = 1 .. n)])
    fi
end;
```

B.5 The PSLQ algorithm for simultaneous integer relations

The following is a Maple implementation of the PSLQ algorithm for simultaneous integer relations as outlined in Section 2.5.

```
pslq_simultaneous := proc(vects)
local B, G, GH, H, Mat, N, a, allzero, alpha, b, bd, beta, bsize, delta, det, eps,
      gamma, h, i, indepset, j, k, lambda, ld, maxHval, maxHvalpos, n, numindep,
      r, rels, t, temp, theorder, upperj, v, v1, v2;
    N := nops(vects);
    n := nops(vects[1]);
    eps := evalf(10^(-floor(7*Digits/10)));
    gamma := evalf(sqrt(4/3)) + 10^(-Digits + 1);
    B := array(1 .. n, 1 .. n);
    for j to n do for k to n do
        if j = k then B[k, k] := 1 else B[j, k] := 0 end if
    end do end do;
    h := array(1 .. N);
    h[1] := Vector(vects[1])/evalf(sqrt(inner(vects[1], vects[1])));
    indepset := array(1 .. N);
    indepset[1] := 1;
    i := 2;
    k := 1;
    while k < N do
        k := k + 1;
        h[i] := Vector(vects[k]);
        for j to i - 1 do h[i] := h[i] - LinearAlgebra[DotProduct](h[i], h[j])*h[j] end do;
```

```
    if n*eps < `+`(seq(abs(h[i][ii]), ii = 1 .. n)) then
        h[i] := h[i]/evalf(sqrt(LinearAlgebra[DotProduct](h[i], h[i])));
        indepset[i] := k;
        i := i + 1
    end if
end do;
N := i - 1;
numindep := N;
if numindep = n then
    Mat := Matrix(1 .. n, 1 .. n);
    for i to n do for j to n do
            Mat[i, j] := vects[indepset[i]][j]
    end do end do;
    det := LinearAlgebra[Determinant](Mat);
    if det = 0 then print(`try again with more digits (or decrease eps)`)
    else print(`Input spans R^n`)
    end if;
    return [seq(0, i = 1 .. n)]
end if;
H := array(1 .. n, 1 .. n - N);
bsize := 0;
j := 0;
theorder := array(1 .. n);
for i to n do theorder[i] := i end do;
while bsize < n - N do
    j := j + 1;
    v1 := Vector([seq(B[i, j], i = 1 .. n)]);
    for k from -N + 1 to bsize do
       if k < 1 then v2 := h[k + N] else v2 := Vector([seq(H[i, k], i = 1 .. n)]) end if;
       v1 := v1 - LinearAlgebra[DotProduct](v1, v2)*v2
    end do;
    for i to bsize do v1[i] := 0 end do;
    if n*eps < `+`(seq(abs(v1[ii]), ii = 1 .. n)) then
        for i to n do if abs(v1[i]) < eps then v1[i] := 0 end if end do;
        bsize := bsize + 1;
        i := bsize;
        while v1[i] = 0 do i := i + 1 end do;
        if bsize < i then
            temp := v1[bsize]; v1[bsize] := v1[i]; v1[i] := temp;
            for k to bsize - 1 do
                temp := H[bsize, k]; H[bsize, k] := H[i, k]; H[i, k] := temp
            end do;
            for k to N do
                temp := h[k][bsize]; h[k][bsize] := h[k][i]; h[k][i] := temp
            end do;
            temp := theorder[bsize]; theorder[bsize] := theorder[i]; theorder[i] := temp
        end if;
        v1 := v1/sqrt(LinearAlgebra[DotProduct](v1, v1));
        for i to n do H[i, bsize] := v1[i] end do
    end if
end do;
for i from 2 to n do for j from min(i - 1, n - N) by -1 to 1 do
        if H[j, j] < 2*abs(H[i, j]) then
            t := round(H[i, j]/H[j, j]);
            for k to j do H[i, k] := H[i, k] - t*H[j, k] end do;
            for k to n do B[k, j] := B[k, j] + t*B[k, i] end do
        end if
end do end do;
G := array(1 .. n - 1);
G[1] := gamma;
GH := array(1 .. n - 1);
GH[1] := H[1, 1];
for i from 2 to n - N do G[i] := G[i - 1]*gamma; GH[i] := G[i]*H[i, i] end do;
while N < n do
    maxHval := abs(H[n - N + 1, n - N]);
    for i from n - N + 2 to n - N + numindep do
        if maxHval < abs(H[i, n - N]) then maxHval := abs(H[i, n - N]) end if
    end do;
    r := 1;
    for i from 2 to n - N do if GH[r] < GH[i] then r := i end if end do;
    while eps < maxHval or r <> n - N do
```

```
        if r < n - N then
            for i to n - numindep do
                temp := H[r, i]; H[r, i] := H[r + 1, i]; H[r + 1, i] := temp
            end do;
            for i to n do
                temp := B[i, r]; B[i, r] := B[i, r + 1]; B[i, r + 1] := temp
            end do;
            alpha := H[r + 1, r];     beta := H[r, r];
            lambda := H[r, r + 1];    delta := sqrt(beta^2 + lambda^2);
            bd := beta/delta;         ld := lambda/delta;
            H[r, r] := delta;         H[r, r + 1] := 0;
            H[r + 1, r] := alpha*bd;  H[r + 1, r + 1] := -alpha*ld;
            for i from r + 2 to n do
                a := H[i, r];
                b := H[i, r + 1];
                H[i, r] := a*bd + b*ld;
                H[i, r + 1] := b*bd - a*ld
            end do;
            GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
        else
            maxHval := abs(H[n - N + 1, r]);
            maxHvalpos := n - N + 1;
            for i from n - N + 2 to n - N + numindep do
                if maxHval < abs(H[i, r]) then
                    maxHval := abs(H[i, r]);
                    maxHvalpos := i
                end if
            end do;
            for i to n - numindep do
                temp := H[r, i];  H[r, i] := H[maxHvalpos, i]; H[maxHvalpos, i] := temp
            end do;
            for i to n do
                temp := B[i, r]; B[i, r] := B[i, maxHvalpos]; B[i, maxHvalpos] := temp
            end do
        end if;
        GH[r] := G[r]*abs(H[r, r]);
        for i from r + 1 to n - N + numindep do
            if r = n - N or i = r + 1 then upperj := r
            else upperj := r + 1
            end if;
            for j from upperj by -1 to 1 do
                if abs(H[j, j]) < 2*abs(H[i, j]) then
                    t := round(H[i, j]/H[j, j]);
                    for k to j do H[i, k] := H[i, k] - t*H[j, k] end do;
                    for k to n do B[k, j] := B[k, j] + t*B[k, i] end do
                end if
            end do
        end do;
        maxHval := abs(H[n - N + 1, n - N]);
        for i from n - N + 2 to n - N + numindep do
            if maxHval < abs(H[i, n - N]) then  maxHval := abs(H[i, n - N]) end if
        end do;
        r := 1;
        for i from 2 to n - N do if GH[r] < GH[i] then r := i end if end do
    end do;
    for i from n - N + 1 to n - N + numindep do H[i, n - N] := 0 end do;
    for i to n - numindep do
        temp := H[n - N, i];
        H[n - N, i] := H[n - N + numindep, i]; H[n - N + numindep, i] := temp
    end do;
    for i to n do
        temp := B[i, n - N];
        B[i, n - N] := B[i, n - N + numindep]; B[i, n - N + numindep] := temp
    end do;
    N := N + 1
end do;
rels := array(1 .. n - numindep, 1 .. n);
for i to n - numindep do for j to n do
    rels[i, theorder[j]] := B[j, i + numindep]
end do end do;
allzero := true;
```

```
    for i to n - numindep do
        v := [seq(rels[i, j], j = 1 .. n)];
        for j to nops(vects) do
            if abs(inner(vects[j], v)) <> 0 then allzero := false end if
        end do
    end do;
    if allzero then print(‘They are all integer relations‘)
               else print(‘They are not all integer relations‘)
    end if;
    Mat := Matrix(1 .. n, 1 .. n);
    for i to numindep do for j to n do Mat[i, j] := vects[indepset[i]][j] end do end do;
    for i to n - numindep do for j to n do Mat[i + numindep, j] := rels[i, j] end do end do;
    det := LinearAlgebra[Determinant](Mat);
    if det = 0 then print(‘ZERO DETERMINANT, NOT A BASIS!!!!‘)
               else print(‘They form a basis for the set of simultaneous integer relations‘)
    end if;
    RETURN(seq([seq(rels[i, j], j = 1 .. n)],i = 1 .. n - numindep))
end proc;
```

B.6 A multi-level implementation of PSLQ

```
##############
##############  PSLQ
##############
##############  procedures:
###############    PSLQ, _PSLQ, _PSLQmain, _PSLQfunc1c,
##############    _PSLQfunc1r, _PSLQfunc2,_PSLQfunc3
##############
##############  Author: Alan Meichsner
###############    Summer 2006 (multilevel version)
##############    Summer 2007 (complex version)
##############
####
#### Input: a list of quantities that evaluate to (complex)floats
####
#### Output: A SUSPECTED integer relation for the input.
#### (A list of integers with the property that if the lists are thought
#### of as vectors, then  inner(Input,Output)/(|input|*|output|) < epsilon
#### where the value of epsilon is determined by the value of Digits.)
####

PSLQ:=proc(vect)
    local D, X, XiI, XiR, Xninv, allIm, allRe, complexinput, eps, i, n;
    if not type(vect,list) then
        error"argument must be a list of quantities that evaluate to floating point numbers"
    fi;
    D:=Digits;
    n := nops(vect);
    if n<2 then
      error"argument must be a list with at least two elements"
    fi;
    Digits:=D+5;
    eps := 10^(-D+log[10](2.*n));
    X := evalf(vect);
    Xninv:=1/sqrt(‘+‘(seq(Re(X[i])^2+Im(X[i])^2, i = 1 .. n)));
    X := X*Xninv;
    allRe:=true;
    allIm:=true;
    for i to n do
      if X[i]=0 then return([seq(‘if‘(i=j,1,0),j=1..n)]) fi;
      XiR:=Re(X[i]);
      XiI:=Im(X[i]);
      if not type(XiR,float) and not type(XiI,float) then
        error"argument must be a list of quantities that evaluate to (complex)floating point numbers"
      fi;
      if XiR<>0. then allIm:=false fi;
      if XiI<>0. then allRe:=false fi;
    od;
```

```
    if allRe or allIm then
        complexinput:=false;
        if allIm then X:=map(x->-I*x,X) fi;
      else
        complexinput:=true;
    fi;
    _PSLQ(X,n,eps,D,complexinput);
end:


_PSLQ:=proc(X,n,eps,D,complexinput)
    local A, ABlim, B, Diglist, G, H, basenum, gamma, hdig, i, itype, j, s, sftype, y;
    options system, remember;
    if complexinput then
        sftype:='complex(sfloat)';
        itype:='complex(integer)';
      else
        sftype:='sfloat';
        itype:='integer';
    fi;
    s := Array(1 .. n);
    H := rtable(1 .. n, 1 .. n - 1, 'datatype'=sftype,subtype=Matrix);
    A := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);
    B := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);
    y := Array(1 .. n);
    G := hfarray(1 .. n - 1);
    hdig := 14;
    if D<=hdig then Diglist:=[D]
    else Diglist:=[hdig];
        basenum:=16;
        if 80*basenum<D then
            while 64*basenum<D do
                basenum:=8*basenum;
                Diglist:=[basenum,op(Diglist)]
            od;
        fi;
    fi;
    Diglist:=[D,op(Diglist)];
    if complexinput then
      gamma := evalf(sqrt(2.),hdig)+10^(-hdig+1);
    else
      gamma := evalf(sqrt(4./3),hdig) + 10^(-hdig+1);
    fi;
    G[1] := gamma;
    for i from 2 to n - 1 do G[i] := G[i - 1]*gamma od;

    for i to n do y[i] := X[i] od;
    s[n] := Re(y[n])^2+Im(y[n])^2;
    for i from n - 1 by -1 to 1 do s[i] := s[i + 1] + Re(y[i])^2+Im(y[i])^2 od;
    s := evalf(map(sqrt, s));
    for i to n do for j to n - 1 do
            if j < i then
                H[i, j] := - conjugate(y[i])*y[j]/(s[j]*s[j + 1])
            elif j = i then H[i, j] := s[i + 1]/s[i]
            else H[i, j] := 0
            fi
        od
    od;
    for i to n do for j to n do
            if i=j
                 then A[i,j]:=1; B[i,j]:=1
                 else A[i,j]:=0; B[i,j]:=0
            fi;
    od od;
    ABlim:=2;
    _PSLQmain([H,A,B,y,G],hdig,n,eps,Diglist,complexinput,true);
end:


_PSLQmain := proc()
    local A, A2, A3, AB2, ABlim, B, B2, B3, D, Diglist, G, H, H2, H3, Min,
```

```
      X, bottomlevel, calchfA, complexinput, eps, eps2, evalDigs, hdig,
      hfABlim, hftype, i, initH, itype, j, k, maxAB, maxHii, minBv,
      minBvIm, minBvRe, minHii, miny, minypos, n, nextDigs, oldMin,
      scalefactor, sftype, singlestep, t, toplevel, xlim, y, y2,
      IR, rcount, icount, rpart, ipart, c;
H:=args[1][1];
A:=args[1][2];
B:=args[1][3];
n:=args[3];
X:=args[1][4];
X:=[seq(X[i],i=1..n)];
G:=args[1][5];
hdig:=args[2];
eps:=args[4];
Diglist:=args[5];
complexinput:=args[6];
toplevel:=args[7];
D:=Diglist[1];
Diglist:=[seq(Diglist[k],k=2..nops(Diglist))];
nextDigs:=Diglist[1];
Digits:=D;
y:=Array(1..n);
for i to n do y[i]:=evalf(X[i]) od;
if toplevel then
   ABlim:=2;
   if D<=hdig then calchfA:=false else calchfA:=true fi;
else
   ABlim:=10^(D-nextDigs);
   calchfA:=true;
fi;
if complexinput then
    sftype:='complex(sfloat)';
    hftype:='complex[8]';
    itype:='complex(integer)';
  else
    sftype:='sfloat';
    hftype:='float[8]';
    itype:='integer';
fi;

if nops(Diglist)=1 then
    bottomlevel:=true;
    hfABlim:=10^hdig;
    evalDigs:=nextDigs+4;
  else
    bottomlevel:=false;
    evalDigs:=nextDigs;
fi;
if not toplevel then
    initH:=rtable(1..n,1..n-1,'datatype'=sftype,subtype=Matrix);
    for i to n do for j to n-1 do initH[i,j]:=evalf(H[i,j]) od od;
fi;
if hdig < D then
    for i from 2 to n do
        for j from i - 1 by -1 to 1 do
            if abs(H[j, j]) < 2*abs(H[i, j]) then
                t := H[i,j]/H[j,j];
                if complexinput then
                  t:=`if`(Re(t)<0,trunc(Re(t)-.5),trunc(Re(t)+.5))
                    +I*`if`(Im(t)<0,trunc(Im(t)-.5),trunc(Im(t)+.5))
                else
                  t:=`if`(t<0,trunc(t-.5),trunc(t+.5));
                fi;
                for k to j do
                    H[i, k] := H[i, k] - t*H[j, k]
                od;
                for k to n do
                    A[i, k] := A[i, k] - t*A[j, k];
                    B[k, j] := B[k, j] + t*B[k, i]
                od;
                y[j] := y[j] + t*y[i]
```

```
            fi
        od
    od
fi;

if bottomlevel then
    A2 := rtable(1 .. n, 1 .. n, 'datatype'=hftype,subtype=Matrix);
    B2 := rtable(1 .. n, 1 .. n, 'datatype'=hftype,subtype=Matrix);
    H2 := rtable(1 .. n, 1 .. n - 1, 'datatype'=hftype,subtype=Matrix);
    y2 := rtable(1 .. n, 'datatype'=hftype,subtype=Array);
    AB2:=Array(1..n,1..2*n,'datatype'=hftype);
else
    A2 := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);
    B2 := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);
    H2 := rtable(1 .. n, 1 .. n - 1, 'datatype'=sftype,subtype=Matrix);
    y2 := Array(1 .. n);
fi;
A3 := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);
B3 := rtable(1 .. n, 1 .. n, 'datatype'=itype,subtype=Matrix);

for i to n do for j to n - 1 do H2[i, j] := evalf(H[i, j],evalDigs) od od;
miny := abs(y[1]);
minypos:=1;
for i from 2 to n do
    if abs(y[i]) < miny then miny := abs(y[i]); minypos := i fi
od;
if complexinput then
    minBvRe := [seq(Re(B[i, minypos]), i = 1 .. n)];
    minBvIm := [seq(Im(B[i, minypos]), i = 1 .. n)];
    Min := miny/sqrt(evalf(inner(minBvRe,minBvRe)
                      +inner(minBvIm,minBvIm))   );
else
    minBv := [seq(B[i, minypos], i = 1 .. n)];
    Min := miny/sqrt(evalf(inner(minBv,minBv)));
fi;

maxAB:=1;
eps2:=10^(-nextDigs+log[10](evalf(n)));

while eps<Min and maxAB<ABlim do
    for i to n do
        for j to n do
            if i = j
               then B2[i, j] := 1; A2[i, j] := 1
               else B2[i, j] := 0; A2[i, j] := 0
            fi
        od;
    od;
    for i to n do y2[i] := evalf(y[i]/miny,evalDigs) od;
    xlim:=max(evalf(eps/Min),eps2);

    singlestep:=false;
    if bottomlevel then
       if complexinput then
         if evalhf(_PSLQfunc1c(H2, A2, B2, y2, hfABlim, xlim, n, G,calchfA,AB2))=0 then
           singlestep:=true
         fi;
       else
         if evalhf(_PSLQfunc1r(H2, A2, B2, y2, hfABlim, xlim, n, G,calchfA,AB2))=0 then
           singlestep:=true
         fi;
       fi;
    elif _PSLQmain([H2,A2,B2,y2,G],hdig,n,xlim,Diglist,complexinput,false)=0 then
       singlestep:=true
    fi;

    if hdig<D then
         if bottomlevel then
           if complexinput then
             B3:=mvMultiply(B,
                      map(x->`if`(Re(x)<0,trunc(Re(x)-.5),trunc(Re(x)+.5))
```

```
                    +I*`if`(Im(x)<0,trunc(Im(x)-.5),trunc(Im(x)+.5)),B2));
      else
        B3:=mvMultiply(B,map(x->`if`(x<0,trunc(x-.5),trunc(x+.5)),B2));
      fi
    else
        B3:=mvMultiply(B,B2);
    fi;
    for i to n do for j to n do B[i,j]:=B3[i,j]; od od;

    if not toplevel then
         if bottomlevel then
           if complexinput then
             A3:=mvMultiply(map(x->`if`(Re(x)<0,trunc(Re(x)-.5),trunc(Re(x)+.5))
                        +I*`if`(Im(x)<0,trunc(Im(x)-.5),trunc(Im(x)+.5)),A2),A);
           else
             A3:=mvMultiply(map(x->`if`(x<0,trunc(x-.5),trunc(x+.5)),A2),A);
           fi;
         else
             A3:=mvMultiply(A2,A);
         fi;
         for i to n do for j to n do A[i,j]:=A3[i,j] od od;
         maxAB:=0;
         if complexinput then
           for i to n do for j to n do
               if not toplevel then
                 if abs(Re(A[i,j]))>maxAB then maxAB:=abs(Re(A[i,j])) fi;
                 if abs(Im(A[i,j]))>maxAB then maxAB:=abs(Im(A[i,j])) fi;
               fi;
               if abs(Re(B[i,j]))>maxAB then maxAB:=abs(Re(B[i,j])) fi;
               if abs(Im(B[i,j]))>maxAB then maxAB:=abs(Im(B[i,j])) fi;
           od od
         else
           for i to n do for j to n do
               if not toplevel and abs(A[i,j])>maxAB then maxAB:=abs(A[i,j]) fi;
               if abs(B[i,j])>maxAB then maxAB:=abs(B[i,j]) fi;
           od od
         fi;
    fi;

    for i to n do
         y[i] := inner(X, [seq(B[k, i], k = 1 .. n)])
    od;

    oldMin:=Min;
    miny := abs(y[1]);
    minypos := 1;
    for i from 2 to n do
        if abs(y[i])<miny then miny:=abs(y[i]); minypos:=i fi;
    od;
    if complexinput then
        minBvRe := [seq(Re(B[i, minypos]), i = 1 .. n)];
        minBvIm := [seq(Im(B[i, minypos]), i = 1 .. n)];
        Min := miny/sqrt(evalf(inner(minBvRe,minBvRe)
                          +inner(minBvIm,minBvIm))   );
    else
        minBv := [seq(B[i, minypos], i = 1 .. n)];
        Min := miny/sqrt(evalf(inner(minBv,minBv)));
    fi;
    userinfo(1, 'PSLQ', "Digits = " || (convert(D,string)) ||
             ",  time = " || (convert(time(),string)) ||
             ",  current minimum = " || (convert(evalf(Min,6),string)) );
    if (not toplevel) and 2*length(op(1,miny))<nextDigs then maxAB:=ABlim fi;
    if oldMin<10*Min then
        if toplevel then singlestep:=true;
        elif not singlestep then maxAB:=ABlim;
        fi;
    elif not bottomlevel then singlestep:=false
    fi;
else
    if complexinput then
     for i to n do
```

```
            y[i] := y2[i];
            for j to n do
              B[i,j]:=‘if‘(Re(B2[i,j])<0,trunc(Re(B2[i,j])-.5),
                                        trunc(Re(B2[i,j])+.5))
                      +I*‘if‘(Im(B2[i,j])<0,trunc(Im(B2[i,j])-.5),
                                          trunc(Im(B2[i,j])+.5))
            od
          od;
        else
          for i to n do
            y[i] := y2[i];
            for j to n do
              B[i,j]:=‘if‘(B2[i,j]<0,trunc(B2[i,j]-.5),trunc(B2[i,j]+.5))
            od
          od;
        fi;
        Min := 0
    fi;
    if eps<Min and maxAB<ABlim then
         if toplevel then
            if bottomlevel then
               if complexinput then
                 H:=mvMultiply(map(x->‘if‘(Re(x)<0,trunc(Re(x)-.5),trunc(Re(x)+.5))
                             +I*‘if‘(Im(x)<0,trunc(Im(x)-.5),trunc(Im(x)+.5)),A2),H);
               else
                 H:=mvMultiply(map(x->‘if‘(x<0,trunc(x-.5),trunc(x+.5)),A2),H);
               fi;
            else
               H:=mvMultiply(A2,H)
            fi
         else
            H:=mvMultiply(A,initH);
         fi;
         if abs(H[n,n-1])=0. or abs(H[n-1,n-1])=0. then Min:=0 fi;
         if singlestep and eps<Min then
            H3:=LinearAlgebra:-QRDecomposition(
                LinearAlgebra:-Transpose(H),’conjugate’=complexinput,’output’=’R’);
            for i to n do for j to n-1 do H[i,j]:=H3[j,i] od od;
            if not toplevel then
               minHii:=max(abs(H[n-1,n-1]),abs(H[n,n-1]));
               maxHii:=minHii;
               for i to n-2 do
                   if abs(H[i,i])<minHii then minHii:=abs(H[i,i])
                   elif abs(H[i,i])>maxHii then maxHii:=abs(H[i,i])
                   fi;
               od;
               if minHii<10^floor(-D/2)
                   or evalf(maxHii*sqrt(n)*10^(D/n),hdig)<1
                   then RETURN(0)
               fi;
            fi;
            for i to n do for j to n do
                A3[i,j]:=A[i,j];
                B3[i,j]:=B[i,j]
            od od;
            userinfo(1, ’PSLQ’, "single stepping at higher precision required");
            if _PSLQfunc2([H, A3, B3, y],n,G,ABlim,toplevel,complexinput)=1
              then
                 miny := abs(y[1]);
                 minypos := 1;
                 for i from 2 to n do
                     if abs(y[i]) < miny then
                         miny := abs(y[i]);
                         minypos := i
                     fi;
                 od;
                 if complexinput then
                     minBvRe := [seq(Re(B3[i, minypos]), i = 1 .. n)];
                     minBvIm := [seq(Im(B3[i, minypos]), i = 1 .. n)];
                     Min := miny/sqrt(evalf(inner(minBvRe,minBvRe)
                                    +inner(minBvIm,minBvIm))   );
```

```
                    else
                        minBv := [seq(B3[i, minypos], i = 1 .. n)];
                        Min := miny/sqrt(evalf(inner(minBv,minBv)));
                    fi;
                    userinfo(1, 'PSLQ', "Digits = " || (convert(D,string)) ||
                      ",  time = " || (convert(time(),string)) ||
                      ",  current minimum = " || (convert(evalf(Min,6),string)) );
                    for i to n do for j to n do
                        A[i,j]:=A3[i,j];
                        B[i,j]:=B3[i,j]
                    od od;
                else RETURN(0);
                fi;
            fi;
            if eps<Min then
                Digits:=evalDigs;
                maxHii:=0;
                for i to n-1 do
                    if abs(H[i,i])>maxHii then maxHii:=abs(H[i,i]) fi
                od;
                if bottomlevel
                then for i to n do for j to n-1 do
                        H2[i,j]:=H[i,j]/maxHii
                    od od
                else scalefactor:=evalf(1/maxHii,evalDigs+4);
                    for i to n do for j to n-1 do
                        H2[i,j]:=H[i,j]*scalefactor
                    od od
                fi;
                if not singlestep then
                   if bottomlevel and not complexinput
                     then evalhf(_PSLQfunc3(H2, n));
                     else
                       H3:=LinearAlgebra:-QRDecomposition(
                       LinearAlgebra:-Transpose(H2),'conjugate'=complexinput,'output'='R');
                       for i to n do for j to n-1 do H2[i,j]:=H3[j,i] od od;
                   fi;
                fi;
                Digits:=D;
            fi;
        fi;
    od;
    if not toplevel then RETURN(1) fi;

    Min := abs(y[1]);
    minypos := 1;
    for j from 2 to n do
        if abs(y[j]) < Min then Min := abs(y[j]); minypos := j fi
    od;
    IR:=[seq(B[i, minypos], i = 1 .. n)];
    rcount:=[0,0];
    icount:=[0,0];
    for i to n do
      rpart,ipart:=Re(IR[i]),Im(IR[i]);
      if rpart>0 then rcount:=rcount+[1,1]
        elif rpart<0 then rcount:=rcount+[1,0]
      fi;
      if ipart>0 then icount:=icount+[1,1]
        elif ipart<0 then icount:=icount+[1,0]
      fi;
    od;
    if rcount[1]<icount[1] then
       c:=`if`(rcount[1]-icount[1]-2*(rcount[2]-icount[2])>0,-I,I);
    else
       c:=`if`(rcount[1]+icount[1]-2*(rcount[2]+icount[2])>0,-1,1);
    fi;
    RETURN(map(z->c*z,IR))
end:


_PSLQfunc1c := proc(H, A, B, y, ABlim, xlim, n, G, calcA, AB2)
```

```
local GH, Hrrbound, Min, a, alpha, b, bcd, bd, beta, count, delta,
      doFullRed, dropback, eps, i, j, k, lambda, lcd, ld, maxAB,
      miny, minyp, r, sf, skiplast, ss, stepcount, t, temp, upperJ;
eps := xlim;
maxAB:=1;
Hrrbound:=1e-140;
for i to n-2 do if abs(H[i,i])<Hrrbound then RETURN(0) fi od;
if abs(H[n-1,n-1])<Hrrbound
    then if abs(H[n-1,n-2])<Hrrbound
            then RETURN(0)
            else skiplast:=true; sf:=1;
         fi
    else skiplast:=false; sf:=3;
fi;
Hrrbound:=1/(G[n-1]*ABlim);
if calcA then
  for i to n do for j to n do AB2[i,j]:=A[i,j]; AB2[i,n+j]:=B[i,j] od od;
fi;
for i from 2 to n do
   if skiplast and i=n then upperJ:=i-2 else upperJ:=i-1 fi;
   for j from upperJ by -1 to 1 do
        if abs(H[j, j]) < 2*abs(H[i, j]) then
            t := round(H[i,j]/H[j,j]);
            y[j] := y[j] + t*y[i];
            for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
            for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
            if calcA then for k to n do
                A[i, k] := A[i, k] - t*A[j, k];
                if abs(Re(A[i,k]))>maxAB then maxAB:=abs(Re(A[i,k])) fi;
                if abs(Im(A[i,k]))>maxAB then maxAB:=abs(Im(A[i,k])) fi;
                if abs(Re(B[k,j]))>maxAB then maxAB:=abs(Re(B[k,j])) fi;
                if abs(Im(B[k,j]))>maxAB then maxAB:=abs(Im(B[k,j])) fi;
            od fi
        fi
od od;
if maxAB>=ABlim then dropback:=true else dropback:=false fi;

GH := array(1 .. n - 1);
for i to n - 1 do GH[i] := G[i]*abs(H[i, i]) od;
miny := abs(y[1]);
minyp := 1;
for i from 2 to n do
    if abs(y[i]) < miny then
        miny := abs(y[i]); minyp := i
    fi
od;
ss:=0;
for i to n do ss:=ss+abs(B[i,minyp])^2 od;
Min := evalf(miny/sqrt(ss));
count:=0;
while eps<Min and maxAB<ABlim and not dropback do
    count:=count+1;
    doFullRed := false;
    if calcA then for i to n do for j to n do
      AB2[i,j]:=A[i,j]; AB2[i,n+j]:=B[i,j]
    od od fi;
    stepcount:=0;
    while not doFullRed and eps<miny and maxAB<ABlim do
        stepcount:=stepcount+1;
        if stepcount=sf*(n-1) then doFullRed:=true fi;
        r := 1;
        for i from 2 to n - 1 do
            if GH[r] < GH[i] then r := i fi
        od;
        if abs(H[r, r])<2*abs(H[r + 1, r]) then
            t := round(H[r + 1, r]/H[r, r]);
            y[r] := y[r] + t*y[r + 1];
            if abs(y[r])<miny then miny:=abs(y[r]) fi;
            for j to r do H[r + 1, j] := H[r + 1, j] - t*H[r, j] od;
            for j to n do B[j, r] := B[j, r] + t*B[j, r + 1] od;
            if calcA then for j to n do
```

```
            A[r + 1, j] := A[r + 1, j] - t*A[r, j];
            if abs(Re(A[r+1,j]))>maxAB then maxAB:=abs(Re(A[r+1,j])) fi;
            if abs(Im(A[r+1,j]))>maxAB then maxAB:=abs(Im(A[r+1,j])) fi;
            if abs(Re(B[j,r]))>maxAB then maxAB:=abs(Re(B[j,r])) fi;
            if abs(Im(B[j,r]))>maxAB then maxAB:=abs(Im(B[j,r])) fi;
        od fi
    fi;
    temp:=y[r]; y[r]:=y[r+1]; y[r+1]:=temp;
    for i to n - 1 do
        temp:=B[i,r]; B[i,r]:=B[i,r+1]; B[i,r+1]:=temp;
        temp:=H[r,i]; H[r,i]:=H[r+1,i]; H[r+1,i]:=temp
    od;
    temp:=B[n,r]; B[n,r]:=B[n,r+1]; B[n,r+1]:=temp;
    if calcA then for i to n do
        temp:=A[r,i]; A[r,i]:=A[r+1,i]; A[r+1,i]:=temp;
    od fi;
    if r < n - 1 then
        alpha := H[r + 1, r];
        beta := H[r, r];
        lambda := H[r, r + 1];
        delta := sqrt(Re(beta)^2+Im(beta)^2+Re(lambda)^2+Im(lambda)^2);
        bd := beta/delta;
        bcd:=(Re(beta)-Im(beta)*I)/delta;
        ld := lambda/delta;
        lcd:=(Re(lambda)-Im(lambda)*I)/delta;
        H[r, r] := delta;
        H[r, r + 1] := 0;
        H[r + 1, r] := alpha*bcd;
        H[r + 1, r + 1] := - alpha*ld;
        for i from r + 2 to n do
            a := H[i, r];
            b := H[i, r + 1];
            H[i, r] := a*bcd + b*lcd;
            H[i, r + 1] := b*bd - a*ld
        od;
        GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
    else
        doFullRed := true;
        if abs(H[r,r])<Hrrbound and calcA then dropback:=true fi;
    fi;
    GH[r] := G[r]*abs(H[r, r])
od;

if not dropback then
    for i from 2 to n do
        if skiplast and i=n then upperJ:=i-2 else upperJ:=i-1 fi;
        for j from upperJ by -1 to 1 do
            if abs(H[j, j]) < 2*abs(H[i, j]) then
                t := round(H[i, j]/H[j, j]);
                y[j] := y[j] + t*y[i];
                for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
                if calcA then for k to n do
                    A[i, k] := A[i, k] - t*A[j, k];
            if abs(Re(A[i,k]))>maxAB then maxAB:=abs(Re(A[i,k])) fi;
            if abs(Im(A[i,k]))>maxAB then maxAB:=abs(Im(A[i,k])) fi;
            if abs(Re(B[k,j]))>maxAB then maxAB:=abs(Re(B[k,j])) fi;
            if abs(Im(B[k,j]))>maxAB then maxAB:=abs(Im(B[k,j])) fi;
                od fi
            fi
        od
    od;
fi;
miny := abs(y[1]);
minyp := 1;
for i from 2 to n do
    if abs(y[i]) < miny then
        miny := abs(y[i]); minyp := i
    fi
od;
ss:=0;
```

```
        for i to n do ss:=ss+Re(B[i,minyp])^2+Im(B[i,minyp])^2 od;
        Min := miny/sqrt(ss);
    od;
    if maxAB>=ABlim or dropback then
      for i to n do for j to n do A[i,j]:=AB2[i,j]; B[i,j]:=AB2[i,n+j] od od;
    fi;
    if dropback or count<2
      then RETURN(0)
      else RETURN(1)
    fi;
end:


_PSLQfunc1r := proc(H, A, B, y, ABlim, xlim, n, G, calcA, AB2)
    local GH, Hrrbound, Min, a, alpha, b, bd, beta, count, delta,
          doFullRed, dropback, eps, i, j, k, lambda, ld, maxAB, miny,
          minyp, r, sf, skiplast, ss, stepcount, t, temp, upperJ;
    eps := xlim;
    maxAB:=1;
    Hrrbound:=1e-140;
    for i to n-2 do if abs(H[i,i])<Hrrbound then RETURN(0) fi od;
    if abs(H[n-1,n-1])<Hrrbound
        then if abs(H[n-1,n-2])<Hrrbound
                then RETURN(0)
                else skiplast:=true; sf:=1;
             fi
        else skiplast:=false; sf:=3;
    fi;
    Hrrbound:=1/(G[n-1]*ABlim);
    if calcA then
      for i to n do for j to n do AB2[i,j]:=A[i,j]; AB2[i,n+j]:=B[i,j] od od;
    fi;
    for i from 2 to n do
       if skiplast and i=n then upperJ:=i-2 else upperJ:=i-1 fi;
       for j from upperJ by -1 to 1 do
            if abs(H[j, j]) < 2*abs(H[i, j]) then
                t := round(H[i,j]/H[j,j]);
                y[j] := y[j] + t*y[i];
                for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
                if calcA then for k to n do
                    A[i, k] := A[i, k] - t*A[j, k];
                    if abs(A[i,k])>maxAB then maxAB:=abs(A[i,k]) fi;
                    if abs(B[k,j])>maxAB then maxAB:=abs(B[k,j]) fi;
                od fi
            fi
    od od;
    if maxAB>=ABlim then dropback:=true else dropback:=false fi;

    GH := array(1 .. n - 1);
    for i to n - 1 do GH[i] := G[i]*abs(H[i, i]) od;
    miny := abs(y[1]);
    minyp := 1;
    for i from 2 to n do
        if abs(y[i]) < miny then
            miny := abs(y[i]); minyp := i
        fi
    od;
    ss:=0;
    for i to n do ss:=ss+B[i,minyp]^2 od;
    Min := evalf(miny/sqrt(ss));
    count:=0;
    while eps<Min and maxAB<ABlim and not dropback do
        count:=count+1;
        doFullRed := false;
        if calcA then for i to n do for j to n do
          AB2[i,j]:=A[i,j]; AB2[i,n+j]:=B[i,j]
        od od fi;
        stepcount:=0;
        while not doFullRed and eps<miny and maxAB<ABlim do
            stepcount:=stepcount+1;
```

```
    if stepcount=sf*(n-1) then doFullRed:=true fi;
    r := 1;
    for i from 2 to n - 1 do
        if GH[r] < GH[i] then r := i fi
    od;
    if abs(H[r, r])<2*abs(H[r + 1, r]) then
        t := round(H[r + 1, r]/H[r, r]);
        y[r] := y[r] + t*y[r + 1];
        if abs(y[r])<miny then miny:=abs(y[r]) fi;
        for j to r do H[r + 1, j] := H[r + 1, j] - t*H[r, j] od;
        for j to n do B[j, r] := B[j, r] + t*B[j, r + 1] od;
        if calcA then for j to n do
            A[r + 1, j] := A[r + 1, j] - t*A[r, j];
            if abs(A[r+1,j])>maxAB then maxAB:=abs(A[r+1,j]) fi;
            if abs(B[j,r])>maxAB then maxAB:=abs(B[j,r]) fi;
        od fi
    fi;
    temp:=y[r]; y[r]:=y[r+1]; y[r+1]:=temp;
    for i to n - 1 do
        temp:=B[i,r]; B[i,r]:=B[i,r+1]; B[i,r+1]:=temp;
        temp:=H[r,i]; H[r,i]:=H[r+1,i]; H[r+1,i]:=temp
    od;
    temp:=B[n,r]; B[n,r]:=B[n,r+1]; B[n,r+1]:=temp;
    if calcA then for i to n do
        temp:=A[r,i]; A[r,i]:=A[r+1,i]; A[r+1,i]:=temp;
    od fi;
    if r < n - 1 then
        alpha := H[r + 1, r];
        beta := H[r, r];
        lambda := H[r, r + 1];
        delta := sqrt(beta^2 + lambda^2);
        bd := beta/delta;
        ld := lambda/delta;
        H[r, r] := delta;
        H[r, r + 1] := 0;
        H[r + 1, r] := alpha*bd;
        H[r + 1, r + 1] := - alpha*ld;
        for i from r + 2 to n do
            a := H[i, r];
            b := H[i, r + 1];
            H[i, r] := a*bd + b*ld;
            H[i, r + 1] := b*bd - a*ld
        od;
        GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
    else
        doFullRed := true;
        if abs(H[r,r])<Hrrbound and calcA then dropback:=true fi;
    fi;
    GH[r] := G[r]*abs(H[r, r])
od;

if not dropback then
    for i from 2 to n do
        if skiplast and i=n then upperJ:=i-2 else upperJ:=i-1 fi;
        for j from upperJ by -1 to 1 do
            if abs(H[j, j]) < 2*abs(H[i, j]) then
                t := round(H[i, j]/H[j, j]);
                y[j] := y[j] + t*y[i];
                for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
                if calcA then for k to n do
                    A[i, k] := A[i, k] - t*A[j, k];
                    if abs(A[i,k])>maxAB then maxAB:=abs(A[i,k]) fi;
                    if abs(B[k,j])>maxAB then maxAB:=abs(B[k,j]) fi;
                od fi
            fi
        od
    od;
fi;

miny := abs(y[1]);
```

```
        minyp := 1;
        for i from 2 to n do
            if abs(y[i]) < miny then
                miny := abs(y[i]); minyp := i
            fi
        od;
        ss:=0;
        for i to n do ss:=ss+B[i,minyp]^2 od;
        Min := evalf(miny/sqrt(ss));
    od;
    if maxAB>=ABlim or dropback then
      for i to n do for j to n do A[i,j]:=AB2[i,j]; B[i,j]:=AB2[i,n+j] od od;
    fi;
    if dropback or count<2
      then RETURN(0)
      else RETURN(1)
    fi;
end:


_PSLQfunc2 := proc(arg1, n, G, ABlim, toplevel, complexinput)
  local A, B, GH, H, a, alpha, b, bcd, bd, beta, count, delta,
        i, j, k, lambda, lcd, ld, maxAB, r, stepcount, t, temp, y;
  H:=arg1[1];
  A:=arg1[2];
  B:=arg1[3];
  y:=arg1[4];
  maxAB:=1;
  for count to 2 do
      for i from 2 to n do for j from i - 1 by -1 to 1 do
              if abs(H[j, j]) < 2*abs(H[i, j]) then
                  t := H[i,j]/H[j,j];
                  if complexinput then
                      t:=`if`(Re(t)<0,trunc(Re(t)-.5),trunc(Re(t)+.5))
                        +I*`if`(Im(t)<0,trunc(Im(t)-.5),trunc(Im(t)+.5))
                  else
                      t:=`if`(t<0,trunc(t-.5),trunc(t+.5));
                  fi;
                  y[j] := y[j] + t*y[i];
                  for k to j do H[i, k] := H[i, k] - t*H[j, k] od;
                  for k to n do B[k, j] := B[k, j] + t*B[k, i] od;
                  if not toplevel then
                    for k to n do
                      A[i, k] := A[i, k] - t*A[j, k];
                      if complexinput then
                        if abs(Re(A[i,k]))>maxAB then maxAB:=abs(Re(A[i,k])) fi;
                        if abs(Im(A[i,k]))>maxAB then maxAB:=abs(Im(A[i,k])) fi;
                        if abs(Re(B[k,j]))>maxAB then maxAB:=abs(Re(B[k,j])) fi;
                        if abs(Im(B[k,j]))>maxAB then maxAB:=abs(Im(B[k,j])) fi;
                      else
                        if abs(A[i,k])>maxAB then maxAB:=abs(A[i,k]) fi;
                        if abs(B[k,j])>maxAB then maxAB:=abs(B[k,j]) fi;
                      fi
                    od
                  fi;
              fi
          od
      od;
      if abs(H[n,n-1])=0. then count:=2 fi;
      if count<2 then
          GH := array(1 .. n - 1);
          for i to n - 1 do GH[i] := G[i]*abs(H[i, i]) od;
          r:=0;
          stepcount:=0;
          while r<n-1 and stepcount<n-1 and maxAB<ABlim do
              stepcount:=stepcount+1;
              r := 1;
              for i from 2 to n - 1 do
                  if GH[r] < GH[i] then r := i fi
              od;
              if abs(H[r, r]) < 2*abs(H[r + 1, r]) then
```

```
    t := H[r+1,r]/H[r,r];
    if complexinput then
        t:=`if`(Re(t)<0,trunc(Re(t)-.5),trunc(Re(t)+.5))
          +I*`if`(Im(t)<0,trunc(Im(t)-.5),trunc(Im(t)+.5))
    else
        t:=`if`(t<0,trunc(t-.5),trunc(t+.5));
    fi;
    y[r] := y[r] + t*y[r + 1];
    for j to r do H[r + 1, j] := H[r + 1, j] - t*H[r, j] od;
    for j to n do B[j, r] := B[j, r] + t*B[j, r + 1] od;
    if not toplevel then
        for j to n do
          A[r + 1, j] := A[r + 1, j] - t*A[r, j];
          if complexinput then
            if abs(Re(A[r+1,j]))>maxAB then maxAB:=abs(Re(A[r+1,j])) fi;
            if abs(Im(A[r+1,j]))>maxAB then maxAB:=abs(Im(A[r+1,j])) fi;
            if abs(Re(B[j,r]))>maxAB then maxAB:=abs(Re(B[j,r])) fi;
            if abs(Im(B[j,r]))>maxAB then maxAB:=abs(Im(B[j,r])) fi;
          else
            if abs(A[r+1,j])>maxAB then maxAB:=abs(A[r+1,j]) fi;
            if abs(B[j,r])>maxAB then maxAB:=abs(B[j,r]) fi;
          fi
        od
    fi;
fi;
temp:=y[r]; y[r]:=y[r+1]; y[r+1]:=temp;
for i to n - 1 do
    temp:=B[i,r]; B[i,r]:=B[i,r+1]; B[i,r+1]:=temp;
    temp:=H[r,i]; H[r,i]:=H[r+1,i]; H[r+1,i]:=temp
od;
temp:=B[n,r]; B[n,r]:=B[n,r+1]; B[n,r+1]:=temp;
if not toplevel then for i to n do
    temp:=A[r,i]; A[r,i]:=A[r+1,i]; A[r+1,i]:=temp;
od fi;
if r < n - 1 then
    alpha := H[r + 1, r];
    beta := H[r, r];
    lambda := H[r, r + 1];
    if complexinput then
  delta := sqrt(Re(beta)^2+Im(beta)^2+Re(lambda)^2+Im(lambda)^2);
  bd := beta/delta;
  bcd:=(Re(beta)-Im(beta)*I)/delta;
  ld := lambda/delta;
  lcd:=(Re(lambda)-Im(lambda)*I)/delta;
  H[r, r] := delta;
  H[r, r + 1] := 0;
  H[r + 1, r] := alpha*bcd;
  H[r + 1, r + 1] := - alpha*ld;
  for i from r + 2 to n do
      a := H[i, r];
      b := H[i, r + 1];
      H[i, r] := a*bcd + b*lcd;
      H[i, r + 1] := b*bd - a*ld
  od;
    else
      delta := sqrt(beta^2 + lambda^2);
      bd := beta/delta;
      ld := lambda/delta;
      H[r, r] := delta;
      H[r, r + 1] := 0;
      H[r + 1, r] := alpha*bd;
      H[r + 1, r + 1] := - alpha*ld;
      for i from r + 2 to n do
          a := H[i, r];
          b := H[i, r + 1];
          H[i, r] := a*bd + b*ld;
          H[i, r + 1] := b*bd - a*ld
      od;
    fi;
    GH[r + 1] := G[r + 1]*abs(H[r + 1, r + 1])
fi;
```

```
            GH[r] := G[r]*abs(H[r, r])
        od;
    fi;
  od;
  if maxAB<ABlim then RETURN(1) else RETURN(0) fi;
end:


_PSLQfunc3 := proc(B, n)
local a, aa, b, c, i, j, k, t;
    for i to n - 1 do
        aa := B[i, i]^2;
        for j from i + 1 to n - 1 do
            if B[i, j] <> 0 then
                b := B[i, j];
                aa := aa + b^2;
                c := sqrt(aa);
                a := B[i, i]/c;
                b := b/c;
                B[i, i] := c;
                B[i, j] := 0;
                for k from i + 1 to n do
                    t := B[k, i];
                    B[k, i] := a*t + b*B[k, j];
                    B[k, j] := -b*t + a*B[k, j]
                od
            fi
        od
    od
end:
```

Bibliography

[1] J. Aguirre and J. C. Peral. The trace problem for totally positive algebraic integers. In *Number Theory and Polynomials*, Conference proceedings, University of Bristol, April 2006. Appendix by J. P. Serre, LMS Lecture Notes.

[2] David H. Bailey and David J. Broadhurst. Parallel integer relation detection: Techniques and applications. *Mathematics of Computation*, 70:1719–1736, 2000.

[3] David H. Bailey and Helaman R. P. Ferguson. Numerical results on relations between fundamental constants using a new algorithm. *Mathematics of Computation*, 53(188):649–656, 1989.

[4] David H. Bailey and Simon Plouffe. Recognizing numerical constants. In *The Organic Mathematics Project Proceedings*, pages 73–88. Canadian Mathematical Society, 1997.

[5] Emiliano Aparicio Bernardo. On the asymptotic structure of the polynomials of minimal diophantic deviation from zero. *Journal of Approximation Theory*, 55:270–278, 1988.

[6] Emiliano Aparicio Bernardo. Generalization of a theorem of Fekete to the case of generalized complex polynomials. *Litovskii Matematicheskii Sbornik (Lietuvos Matematikos Rinkinys)*, 30(4):645–650, October–December 1990. translated version, 1991, Plenum Publishing Corporation.

[7] Borislav D. Bojanov, Werner Haussmann, and Geno P. Nikolov. Bivariate polynomials of least deviation from zero. *Canadian Journal of Mathematics*, 53(3):489–505, 2001.

[8] Peter Borewin and Tamas Erdelyi. The integer Chebyshev problem. *Mathematics of Computation*, 65(214):661–681, April 1996.

[9] G. V. Chudnovsky. Number theoretic applications of polynomials with rational coefficients defined by extremality conditions. In M. Artin and J. Tate, editors, *Arithmetic and Geometry: Papers Dedicated to I.R. Shafarevich on the Occasion of his Sixtieth Birthday*, volume 1 Arithmetic of *Progress in mathematics 35-36*, pages 87–105. Birkhuser, 1983. ISBN 3-376433131-1.

[10] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Number 138 in Graduate Texts in Mathematics. Springer, 1993.

[11] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate texts in mathematics. Springer-Verlag, New York, 2nd edition, 1997.

[12] James Callahan et al. *Calculus in Context: The Five College Calculus Project*. W. H. Freeman and Company, New York, 1995.

[13] H. R. P. Ferguson and R. W. Forcade. Generalization of the Euclidean algorithm for real numbers to all dimensions higher than two. *Bulletin (New Series) of the American Mathematical Society*, 1(6):912–914, November 1979.

[14] Helaman R. P. Ferguson. A short proof of the existence of vector Euclidean algorithms. *Proceedings of the American Mathematical Society*, 97(1):8–10, May 1986.

[15] Helaman R. P. Ferguson. A noninductive GL(n,$\mathbb{Z}$) algorithm that constructs integral linear relations for n $\mathbb{Z}$-linearly dependent real numbers. *Journal of Algorithms*, 8:131–145, 1987.

[16] Helaman R. P. Ferguson, David H. Bailey, and Steve Arno. Analysis of PSLQ, an integer relation finding algorithm. *Mathematics of Computation*, 68(225):351–369, January 1999.

[17] Le Baron O. Ferguson. *Approximation by Polynomials with Integral Coefficients*. Number 17 in Mathematical Surveys. American Mathematical Society, Providence, Rhode Island, 1980. ISBN: 0-8218-1517-2.

[18] V. Flamming, G. Rhin, and C. J. Smyth. The integer transfinite diameter of intervals and totally real algebraic integers. *Journal de Theorie des Nombres de Bordeaux*, 9:137–168, 1997.

[19] John B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, 5th edition, 1994.

[20] Keith Geddes, Stephen Czapor, and George Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[21] Laurent Habsieger and Bruno Salvy. On integer Chebyshev polynomials. *Mathematics of Computation*, 66(218):763–770, April 1997.

[22] J. Hastad, B. Just, J. C. Lagarias, and C. P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM Journal of Computing*, 18(5):859–881, October 1989.

[23] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[24] A. Meichsner. Integer relation algorithms and the recognition of numerical constants. Master's thesis, Simon Fraser University, 2001.

[25] H. L. Montgomery. Small polynomials with integral coefficients. In *Ten Lectures on the Interface Between Analytic Number Theory and Harmonic Analysis*, volume 84 of *CBMS*. Amer. Math. Soc., Providence, R. I., 1994.

[26] I. P. Natanson. *Constructive Function Theory*, volumes I and II. Ungar, New York, 1964-1965.

[27] Igor E. Pritsker. Small polynomials with integer coefficients. *Journal D'Analyse Mathematique*, 96(1):151–190, December 2005.

[28] E. B. Saff and R. S. Varga. On lacunary incomplete polynomials. *Math. Z.*, 177:297–314, 1981.

[29] C. J. Smyth. Totally positive algebraic integers of small trace. *Ann. Inst. Fourrier, Grenoble*, 34(3):1–28, 1984.

[30] Helmuth Späth. *Two Dimensional Spline Interpolation Algorithms*. A. K. Peters Ltd., 1995.

[31] B. Vallée. *Une Approche Géométrique des Algorithmes de Réduction en Petite Dimension*. PhD thesis, Univ. of Caen, 1986.

[32] Qiang Wu. On the linear independence measure of logarithms of rational numbers. *Mathematics of computation*, 72(242):901–911, 2002.

[33] Qiang Wu. A new exceptional polynomial for the integer transfinite diameter of $[0, 1]$. *Journal de Theorie des Nombres de Bordeaux*, 15:847–861, 2003.