Some Iterative Algorithms in Experimental Mathematics

Matthew P. Skerritt
BCompSci BMath(Hons) MPhil

October 2020

A document submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
at
University of Newcastle, Australia

This research was supported by an Australian Government Research Training Program (RTP) Scholarship.

Dedicated to the late Jon Borwein, who started me down this path originally yet never got to see it come to fruition.

Statement of Originality

Matthew P. Skerritt

Acknowledgment of Authorship

I hereby certify that the work embodied in this thesis contains published papers of which I am a joint author. I have included as part of the thesis a written declaration endorsed in writing by my supervisor, attesting to my contribution to the joint publications.

Part I on page 6 in its entirety extends the work reported in the paper "Extending the PSLQ Algorithm to Algebraic Integer Relations" by Skerritt and Vrbik [67], of which I was the majority contributor. The extended results in this thesis are entirely my own work.

Part II on page 99 consists of two published papers. Chapter 4 on page 99 is based on the paper "Dynamics of the Douglas Rachford Method for Ellipses and p-Spheres" by Borwein, Lindstrom, Sims, Schneider, and Skerritt [28]. Chapter 5 on page 121 is based on the paper "Computing intersections of implicitly specified plane curves" by Lindstrom, Sims, and Skerritt [53].

In each of the papers all of the authors contributed equally to the underlying research and final preparation, through extensive group discussions. Each author contributed to all aspects of the research while making their major contribution in their principal areas of expertise, mine being in computational algorithmics and computer implementation. The publications have been appropriately edited and some new material has been included.

Matthew P. Skerritt

By signing below I confirm that Matthew P. Skerritt contributed as outlined above to the papers in question.

Principal Supervisor
Michael Coons

Abstract

We examine certain search problems and the methodology of experimental mathematics as a paradigm for their investigation. The problems in question fall on either side of a natural divide between those with a discrete search space, and those with a continuous one. In the first instance we consider finding integer relations, and in the second, finding an intersection of non-convex constraints (expressed as subsets of a Hilbert space). In both cases we examine, experimentally, an iterative algorithm or algorithms to search for solutions to the problem. In addition, we modify or extend the algorithms in some way and compare the extension with the original algorithm.

We examine PSLQ and LLL algorithms for solving real and complex integer relation problems. We also extend the notion of integer relations further to include relations consisting of algebraic integers, although we restrict our attention to quadratic integers. We modify the PSLQ algorithm to compute certain quadratic integer cases that correspond to norm Euclidean quadratic fields and examine the use of PSLQ, LLL, and the modified PSLQ algorithms for finding quadratic integer relations in general.

In order to gain further insights into the behaviour of the Douglas–Rachford algorithm in the case of non-convex constraint sets we consider two generalisations of a line and sphere (circle) in 2 dimensional Euclidean space, namely: that of a line together with an ellipse and that of a line together with a p-sphere.

We further apply the Douglas–Rachford algorithm to the problem of computing a point in the intersection of two analytic plane curves in $\mathbb{R}^2$ which we often identify with the complex plane $\mathbb{C}$. The graphs of these curves act as the constraint sets for the (non-convex) feasibility problem. We also extend the Douglas–Rachford algorithm by replacing Euclidean reflection with Schwarzian reflection, and compare original and modified algorithms.

Contents

1 Introduction

We present herein a thesis focussing on certain search algorithms and the methodology of experimental mathematics as a paradigm for their investigation. There is a natural divide between algorithms for which the search space is discrete and those for which it is continuous; each requiring very distinct methods of analysis. In the first case we consider the problem of finding integer relations, while in the second we consider the problem of finding an intersection of non-convex constraints (expressed as subsets of a Hilbert space) In both cases we examine, experimentally, an iterative algorithm or algorithms to search for solutions to the problem. In addition, we modify or extend the algorithms in some way and compare against the base algorithm. The algorithms perform surprisingly well in some cases for which we have no right to expect them to.

We work in the tradition of the late Jonathan M. Borwein who—before his death in 2016—was a supervisor for this work. Prof. Borwein long championed computed assisted mathematical discovery incorporating both calculation and visualisation, under the moniker of *experimental mathematics*.

> Had the ancient Greeks (and other early civilizations who started the mathematics bandwagon) had access to computers, it is likely that the word "experimental" in the phrase "experimental mathematics" would be superfluous; the kinds of activities or processes that make a particular mathematical activity "experimental" would be viewed simply as *mathematics*. We say this with some confidence because if you remove from our initial definition [of experimental mathematics] the requirement that a computer be used, what would be left accurately describes what most, if not all, professional mathematicians spend much of their time doing, and always have done!
>
> — Borwein and Devlin [21]

1.1 Integer Relations

The extended Euclidean algorithm is perhaps the simplest example of an integer relation algorithm. For integers a and b the algorithm computes integers m and n with the property that $am + bn = \gcd(a, b)$ or equivalently $am + bn - \gcd(a, b) = 0$. Thus the Euclidean algorithm relates a, b, and $\gcd(a, b)$ by integers.

A perhaps less well known variant of the Euclidean algorithm can be performed on real numbers and is found in Proposition 3 of Book X of *Euclid's Elements* (see Heath [50] for an English translation). Given $x, y, \in \mathbb{R}$ the algorithm computes $w \in \mathbb{R}$ such that $x = mw$ and $y = nw$ for some $m, n \in \mathbb{Z}$, provided that $x/y \in \mathbb{Q}$ (i.e., x and y are commensurate). If we let $s = n$ and $t = -m$ then we have found the relation $xs + yt = 0$. If x and y are not commensurate then the Euclidean algorithm produces an infinite sequence of increasingly good rational approximations to x/y.

A general case for more than two real numbers is desirable. Euler, Jacobi, Poincaré, Minkowski, Perron, Brun, Mahler, and others have all been involved in generalisation efforts. These historic efforts are summarised in Ferguson and Forcade [43]. None of the resulting iterative algorithms have been proved to work for more than three real numbers, and numerous counterexamples have been found. Furthermore none of the algorithms adequately generalised the increasing approximation property when no relation exists.

The first generalisation satisfying both properties was published by Ferguson and Forcade [43] in 1979. Bergman [20] provided extensive, although unpublished, notes. The resulting effort eventually—after Ferguson and Forcade [44] in 1982, and Ferguson [38, 39] in 1986 and 1987—led to the PSOS algorithm by Ferguson [40] in 1988 and ultimately to the PSLQ algorithm by Ferguson and Bailey [41] in 1992.

Applications of integer relations are many. One may determine that a number α is algebraic if one can find an integer relation for $\left(\alpha^0, \alpha^1, \dots, \alpha^n\right)$ for some $n \in \mathbb{N}$. Bailey and Ferguson [10] used this technique with the PSOS algorithm to show that γ, $\log \gamma$, $\log \pi$, $\zeta(3)$ and other constants do not satisfy a simple low-degree polynomial. The discovery of the Bailey-Borwein-Plouffe (BBP) formula [8] for π came as a result of "inspired guessing and extensive searching using the PSLQ integer relation algorithm". A procedure by Bailey and Plouffe [11] uses PSLQ to search for (or "recognise") potential symbolic representations of a floating point number; it was able to recognise $\int_0^\infty t^{7/4} \mathrm{e}^{-t}\, \mathrm{d}t = 21\pi\sqrt{2}/(16\,\Gamma(1/4))$ for example. Meichsner's Masters thesis [58] is dedicated to similarly recognising numerical constants using integer relation algorithms, and his PhD thesis [59] uses an extension of PSLQ to compute bivariate n^{th} integer Chebyshev polynomials. Chamberland [31] uses PSLQ to find relations between functions. Borwein and Lisoněk [22] give a survey of other applications.

A further extension of the integer relation problem is from real numbers and integers to complex numbers and Gaussian integers respectively. This extension was shown to be handled by the PSLQ algorithm in the 1999 paper by Ferguson, Bailey and Arno [42] in which they analysed the algorithm and proved bounds on the number of iterations required to find a relation. The complex case is rarely mentioned in the literature, although we note that it is handled by *Maple*'s implementation of the algorithm.

In addition to PSLQ, both the LLL algorithm (introduced by Lenstra, Lenstra Jr, and Lovász [52] in 1982) and the HJLS algorithm (introduced by Hastad, Just, Lagarias, and Schnorr [48] in 1989 with an erratum in 2014 [49]) can compute integer relations.

The LLL algorithm is not an integer relation finding algorithm in and of itself, although it may be leveraged as such. Instead its purpose is to find a reduced basis for a given lattice. We look at this algorithm and its use in finding integer relations in Section 2.3 on page 10.

The HJLS algorithm on the other hand is specifically an integer relation finding algorithm. It is, however, numerically unstable as noted by Borwein [29]. Borwein further notes that when modified to become stable HJLS is just PSLQ with a choice of parameter $\gamma = \sqrt{2}$ (we discuss these parameters in Section 2.2 on page 7). So we simply mention HJLS in passing here, and do not consider it further.

We examine PSLQ and LLL for the real and complex cases described above, which we call the *classical* cases. We then extend the notion of integer relations further to include relations consisting of algebraic integers, although we restrict our attention to quadratic integers for this thesis. We modify the PSLQ algorithm to compute certain quadratic integer cases that correspond to norm Euclidean quadratic fields and examine the use of PSLQ, LLL, and the modified PSLQ algorithms for finding quadratic integer relations in general. The results presented are an extension of those reported in Skerritt and Vrbik [67].

1.2 Douglas-Rachford

One of the most significant applications of metric fixed point theory, and the theory of nonexpansive mappings in particular, has been establishing convergence of various projection algorithms for solving the convex feasibility problem:

> *Find a point satisfying two (or more) convex constraints; that is, in the intersection of two (or more) closed convex subsets of a Hilbert space.*

Of special interest to us is the algorithm first proposed by Douglas and Rachford [36] in 1956, which we call the *Douglas–Rachford* algorithm. Convergence was shown by Lions and Mercier [54] in 1979. The efficacy of this algorithm is well known (see Bauschke, Combettes, and Luke [14, 15, 16] and Bauschke and Borwein [12]).

It has been observed (and exploited) that, despite a lack of theoretical underpinning, the algorithm continues to work well in many situations where at least one of the sets is no longer convex. Elser, Rankenburg, and Thibault [37] in 2007 applied the algorithm experimentally to several problems—including Graph Colouring, 3-SAT, Sudoku, and Protein Folding—and found the algorithm to be surprisingly effective (and sometimes even superior) to dedicated algorithms for each problem. Gravel and Elser [46] in 2008 presented a construction to reduce a problem with more than two constraints to an equivalent problem with only two constraints. This reduction technique, which they called “divide and concur” and which grew out of work by Pierra [62, 63], was applied to 3-SAT and sphere packing problems and was similarly found to be effective (and occasionally superior to the dedicated algorithm). A further treatise can be found in Aragón Artacho, Borwein, and Tam [4].

Borwein and Sims [25] gave a first step toward a theoretical basis for the non-convex case in 2011. They presented a prototypical non-convex two-set scenario in which one of the sets is a line and the other (non-convex) set is the Euclidean sphere, and proved local convergence of the algorithm. This result for such a seemingly simple case proved surprisingly difficult. Global convergence was eventually proved by Benoist [19] in 2015.

Investigation into the algorithm has continued. See, for example, Aragón Artacho, Borwein, and Tam [3], Bauschke and Moursi [17], Bauschke et al. [18], and Borwein and Tam [23, 26].

In order to gain further insights into the behaviour of the Douglas–Rachford algorithm in the case of non-convex constraint sets we consider, in Chapter 4, two generalisations of a line and sphere (circle) in 2 dimensional Euclidean space: that of a line together with an ellipse and that of a line together with a p-sphere. As we shall see, even such simple generalisations from a circle yield very substantial increases in the complexity of the algorithm's behaviour.

In Chapter 5 we apply the Douglas–Rachford algorithm to the problem of computing a point in the intersection of two analytic plane curves in $\mathbb{R}^2$ which we often identify with the complex plane $\mathbb{C}$. The graphs of these curves act as the constraint sets for the (non-convex) feasibility problem. We also extend the Douglas–Rachford algorithm by replacing Euclidean reflection with Schwarzian reflection, and compare original and modified algorithms.

The results on the Douglas–Rachford algorithm presented in this thesis appear in two published papers to which the author contributed. Chapter 4 presents the results from Borwein, Lindstrom, Sims, Schneider, and Skerritt [28] (2018) and Chapter 5 presents the results from Lindstrom, Sims, and Skerritt [53] (2017).

Part I

Integer Relations

2 Classical Integer Relations

2.1 Preliminaries

The classical integer relation cases are covered by the following definition.

Definition 2.1.1 (Classical Integer Relation). *Let $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}\}$, $n \in \mathbb{N}$ s.t. $n > 1$ and*

$$\mathcal{O} = \begin{cases} \mathbb{Z} & \text{if } \mathbb{F} = \mathbb{R} \\ \mathbb{Z}[\sqrt{-1}\,] & \text{if } \mathbb{F} = \mathbb{C} \end{cases}$$

For $x \in \mathbb{F}^n$ an integer relation of x is a vector $a \in \mathcal{O}^n$, $a \neq 0$, such that $a_1x_1 + \cdots + a_nx_n = 0$.

If $x_i = 0$ for any $1 \leq i \leq n$ then for any $k \in \mathcal{O}$ the vector $a = (0, \ldots, 0, k, 0, \ldots, 0)$ (where k is in the i^{th} position of a) is a trivial integer relation for x. As such, without loss of generality, we may assume that $x_i \neq 0$ for all $1 \leq i \leq n$ when considering integer relations.

We will see later on (in Chapter 3) that the ring $\mathcal{O}$ may be generalised to more general sets of integers (in a number theoretic sense), but for now we consider only the cases as described above. With this in mind, in order to avoid confusion when speaking of integers, we shall refer to the set $\mathbb{Z}$ specifically as the *rational integers*. The reason for this follows from Definitions 3.1.6 and 3.1.7 on page 47, however we introduce the term here in the interests of consistency.

Definition 2.1.2 (Rational integer). *The set $\mathbb{Z}$ is referred to as the set of rational integers, and any element of the set as a rational integer.*

Observe that for the linear combination property of an integer relation to be well defined, it must be the case that $\mathcal{O} \subset \mathbb{F}$ (which is clearly the case for the cases encompassed by Definition 2.1.1). We consider, then, the notion of a nearest integer to a given element of the field. This is important for the PSLQ algorithm.

Definition 2.1.3 (Nearest Integer, $\lceil\cdot\rfloor$, $\lceil\cdot\rfloor_{\mathcal{O}}$). *Let $x \in \mathbb{F}$. An integer $a \in \mathcal{O}$ is a nearest integer to x if $|x - a|$ is minimal. We consider a function $\lceil\cdot\rfloor : \mathbb{F} \to \mathcal{O}$ to be a nearest integer function if it maps each $x \in \mathbb{F}$ to one of its nearest integers. When the ring of integers needs to be specified, we will denote a nearest integer function by $\lceil\cdot\rfloor_{\mathcal{O}}$.*

The LLL algorithm relies heavily on the notion of a lattice, defined as follows.

Definition 2.1.4 (Lattice, Lattice Basis). *Let $B = \{b_1, \dots, b_n\} \subset \mathbb{F}^n$ be a linearly independent set of vectors. The set $L = \{\lambda_1 b_1 + \cdots + \lambda_n b_n \mid \lambda_i \in \mathcal{O}\}$ is the lattice spanned by B. We call the set B the basis of the lattice L. When the ring of integers needs to be specified we will refer to the lattice as a $\mathcal{O}$-lattice, or otherwise prepend the term "lattice" with a description of the integers in question (e.g., a Gaussian integer lattice and a $\mathbb{Z}[\sqrt{-1}]$-lattice are one and the same).*

In general, there may be many bases for any given lattice. Any linearly independent set of vectors which produce the same lattice is considered a basis for that lattice. We sometimes represent a lattice by a matrix consisting of basis vectors as follows:

Definition 2.1.5 (Matrix representation of a lattice). *Let L be a lattice, and $B = \{b_1, \dots, b_n\}$ be a basis for that lattice. The matrix representation of L using basis B is the matrix whose row vectors are the basis vectors:*

$$\begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Be aware that much (perhaps even most) of the literature represents a lattice using column vectors instead of row vectors. We have instead used row vectors because that is the convention that *Maple*'s LLL implementation uses, and so in using it for this thesis we ensure that the experimental results reported herein are consistent with the code used to produce those results. To avoid confusion we will avoid matrix representations of lattices, when it is practical to do so.

2.2 PSLQ

We provide a high level description of the PSLQ algorithm. We show the mathematical details of the algorithm, but omit many technical considerations needed for a practical and effective implementation. An alternative (but still high-level) introduction is given by Straub [69].

Details more suitable for a practical implementation are presented in Appendix A on page 140. For a more thorough treatment the interested reader should consult the literature; in particular: Bailey and Broadhurst [9], and Borwein [29] are good sources. Be aware, however, that much of the literature presents only the real case of PSLQ, whereas our treatment allows for the more general complex case. The modifications required for the complex case are very minor and are easily applied to the real case; we point them out where they occur, below.

The PSLQ algorithm has positive parameters τ, γ, and ρ that must satisfy

$$\frac{1}{\rho} \geq |x - \lceil x \rfloor| \quad \forall x \in \mathbb{F} \tag{2.1}$$

$$1 < \tau \leq \rho \tag{2.2}$$

$$\frac{1}{\tau^2} = \frac{1}{\gamma^2} + \frac{1}{\rho^2} \tag{2.3}$$

in order to establish runtime bounds on the algorithm [42].

For each $\mathbb{F}$ there exists ρ such that condition (2.1) is sharp. Using this value for ρ gives the most flexibility with the other parameters. From condition (2.3) we see that $\tau \to \rho$ as $\gamma \to \infty$ and that for fixed ρ there will be a greatest lower bound for γ such that $\tau > 1$.

Definition 2.2.1 (γ_1). *Let ρ be such that condition (2.1) is sharp. Then γ_1 is the positive value of γ that satisfies $1 = 1/\gamma^2 + 1/\rho^2$.*

We use the value of ρ such that condition (2.1) is sharp, and choose any $\gamma > \gamma_1$. So long as $\rho > 1$ (i.e., $1/\rho < 1$) then all three conditions will be satisfied.

Note that when $\mathbb{F} = \mathbb{R}$ and $\mathcal{O} = \mathbb{Z}$ then the above strategy gives $\rho = 2$ and $\gamma_1 = \sqrt{4/3}$. This value of γ_1 is precisely the lower bound on γ given in the literature.

Similarly when $\mathbb{F} = \mathbb{C}$ and $\mathcal{O} = \mathbb{Z}[\sqrt{-1}]$ (i.e., Gaussian integers) then $\rho = \sqrt{2}$ and $\gamma_1 = \sqrt{2}$. This is precisely the bound on γ given in the literature for the complex case.

The PSLQ algorithm is presented in Algorithm 2.2.7 on page 10. In order to make sense of it, we need the following definitions.

Definition 2.2.2 (Lower Trapezoidal). *Let $H = (h_{i,j})$ be an $m \times n$ matrix. If $h_{i,j} = 0$ whenever $i < j$ then H is lower trapezoidal.*

Note that a lower trapezoidal square matrix is exactly a lower triangular matrix.

Definition 2.2.3 (H_x). *Let $x \in \mathbb{F}^n$. Then the $n \times (n-1)$ matrix $H_x = (h_{i,j})$ is defined by*

$$h_{i,j} = \begin{cases} 0 & \text{if } i < j \\ s_{i+1}/s_i & \text{if } i = j \\ -\overline{x_i} x_j/(s_j s_{j+1}) & \text{if } i > j \end{cases} \quad \text{where} \quad s_i = \sqrt{\sum_{k=i}^{n} x_k \overline{x_k}}$$

Note that the complex conjugates are needed for full generality to cope with the complex case. Often the literature will present only the real case of PSLQ in which case $x_k \overline{x_k} = x_k^2$ and is reported as such. Similarly for the conjugates in Definition 2.2.5 on the next page.

Definition 2.2.4 (Hermite Reduction, D_H). *Let $H = (h_{i,j})$ be a lower trapezoidal $m \times n$ matrix with $h_{j,j} \neq 0$ for all j. Then the $m \times m$ matrix $D_H = (d_{i,j})$ where*

$$d_{i,j} = \begin{cases} 0 & \text{if } i < j \\ 1 & \text{if } i = j \\ \left\lceil \dfrac{-1}{h_{j,j}} \displaystyle\sum_{k=j+1}^{i} d_{i,k} h_{k,j} \right\rfloor & \text{if } i > j \end{cases}$$

is the reducing matrix of H. The matrix $D_H H$ is the Hermite reduction of H.

Observe that D_H is a lower triangular matrix containing invertible integers on its diagonal. It is therefore an invertible matrix whose inverse is also integer valued.

Definition 2.2.5 ($Q_{[A,k]}$). *Let $A = (a_{i,j})$ be an $m \times n$ matrix with $m > n$, and let $1 \leq k \leq n$. Let $\beta = a_{k,k}$, $\lambda = a_{k,k+1}$, and $\delta = \sqrt{\beta\bar{\beta} + \lambda\bar{\lambda}}$. Then the $n \times n$ block diagonal matrix*

$$Q_{[A,k]} := \begin{cases} I_n & \text{if } k = n \\ (q_{i,j}) & \text{otherwise} \end{cases}$$

where $(q_{i,j})$ is the block diagonal matrix with submatrix

$$\begin{pmatrix} q_{k,k} & q_{k,k+1} \\ q_{k+1,k} & q_{k+1,k+1} \end{pmatrix} = \frac{1}{\delta} \begin{pmatrix} \bar{\beta} & -\lambda \\ \bar{\lambda} & \beta \end{pmatrix}$$

and 1's for all other diagonal entries.

Observe that multiplication on the right by $Q_{[A,k]}$ changes only columns k and $k+1$ in a way that is effectively multiplying those columns as a submatrix by the submatrix explicitly stated in the definition.

$Q_{[H',r]}$, when used in Line 8 of Algorithm 2.2.7 on the following page, is an orthogonal matrix. The swapping of rows that occurs in the prior steps will usually cause H' to cease to be lower trapezoidal. The post-multiplication with $Q_{[H',r]}$ ensures that H' is once again lower trapezoidal [29, 42]. The only case where the row swap does not remove the lower-trapezoidal property of H' is when $r = n - 1$ in which case $Q_{[H',r]}$ is the identity matrix and so H' is unaffected.

Finally, we use the following notation to refer to rows and columns of matrices, when needed.

Notation 2.2.6 ($\mathrm{col}_k, \mathrm{row}_k$). *For a matrix M we denote by $\mathrm{col}_k(M)$ the k^{th} column of M and by $\mathrm{row}_k(M)$ the k^{th} row of M.*

Algorithm 2.2.7: PSLQ

```
Input  : $x \in \mathbb{F}^n, \gamma > \gamma_1$
Output: $a \in \mathcal{O}^n$
   /* ———————— Initialisation ———————— */
1  $H' \leftarrow H_{x/\|x\|} \quad A \leftarrow I_n$
   /* ———————— Main Calculation ———————— */
2  repeat
3      $H' \leftarrow D_{H'} H'$                                   /* Hermite reduce $H'$ */
4      $A \leftarrow D_{H'} A$                                     /* Update $A$ */
5      $r \leftarrow \arg\max_{1 \le r \le n-1}(\gamma^r |H'_{r,r}|)$   /* Find $r$ such that $\gamma^r |H'_{r,r}|$ is maximal */
6      $\mathrm{row}_r(H') \leftrightarrow \mathrm{row}_{r+1}(H')$       /* Exchange rows $r$ and $r+1$ in $H'$ */
7      $\mathrm{row}_r(A) \leftrightarrow \mathrm{row}_{r+1}(A)$         /* Exchange rows $r$ and $r+1$ in $A$ */
8      $H' \leftarrow H' Q_{[H',r]}$                               /* Make sure $H'$ is lower trapezoidal */
9  until $r = n - 1$ and $H'_{n-1,n-1} = 0$
10 return $\mathrm{col}_n(A^{-1})$
```

After each iteration the value $1/\max|H'_{r,r}|$ is a lower bound for the norm of *any* integer relation of x. Furthermore if a is the integer relation found by the algorithm, then $\|a\| \le \gamma^{n-2} M$ where M is the norm of the smallest possible integer relation [42, Theorem 3].

Note that the algorithm as presented does not terminate if there is no integer relation for the input x. This can be remedied either by specifying termination after a maximum number of iterations are performed, or after the lower bound for the norm of an integer relation exceeds some value.

The algorithm is exact if the individual steps can be performed exactly. That is to say, if we could compute with all real numbers exactly then the algorithm would always calculate an integer relation if there is one to be found. Furthermore, it will find an integer relation in a polynomially bounded number of iterations [29, 42]. In practice, however, an implementation of the PSLQ algorithm must use floating point arithmetic and so numerical error may prevent the detection of a valid integer relation. Nonetheless PSLQ has shown remarkable numerically stability.

Finally, we reiterate that the algorithm as presented here lacks the details needed for practical numeric application. There are many optimisations that can, and should, be employed in order for an implementation to be effective. The interested reader should consult the literature [9, 29].

2.3 LLL

The LLL algorithm introduced by Lenstra, Lenstra Jr, and Lovász in 1982 [52], as stated above, is not an integer relation algorithm per se. The algorithm takes a linearly independent set of vectors describing a lattice, and computes a reduced basis for that lattice. Lenstra, Lenstra Jr, and Lovász used this algorithm as a means to factor polynomials with rational coefficients. The algorithm may, however, also be utilised to find integer relations (see Section 2.3.2).

2.3.1 Reduced Lattice Bases

Lenstra, Lenstra Jr, and Lovász define a reduced basis according to Definition 2.3.1. Borwein [29] notes that a desirable definition of a reduced basis (for a lattice L, say) would be a linearly independent set of vectors $b_1, \dots, b_n$ with the property that b_i is the shortest vector in the lattice independent of all the vectors $b_1, \dots, b_{i-1}$, but that no algorithm is known to calculate such a basis in reasonable (presumably polynomial) time. Observe that Definition 2.3.1 is a weaker condition.

Definition 2.3.1 (LLL Reduced Basis). *Let $b_1, \dots, b_n$ be a basis for a lattice, L We call these vectors LLL reduced if*

$$|\mu_{i,j}| \le \tfrac{1}{2} \quad \textit{for} \quad 1 \le j < i \le n \tag{2.4}$$

$$\|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2 \ge \tfrac{3}{4}\, \|b_{i-1}^*\|^2 \; \textit{for}\; 1 < i \le n \tag{2.5}$$

where

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \quad \textit{and} \quad \mu_{i,j} = \frac{b_i \cdot b_j^*}{\|b_j^*\|^2}$$

The vectors b_i^* are precisely the vectors one constructs when using the Gram-Schmidt orthogonalisation process on the vectors b_i. Condition (2.4) ensures that the vectors are “close” to orthogonal (and gives a quantitative measurement of such). Conditions (2.4) and (2.5) together bound the values of $\|b_i\|$ in terms of the norms of the shortest vectors in the lattice.

The constant $3/4$ in condition (2.5) is arbitrary, and may freely be chosen in the real range $(1/4, 1)$ [52]. If a different number is chosen then some of the values in the following results will change, so we will adhere to the value $3/4$ for simplicity.

The LLL algorithm takes a basis for a lattice L and produces an LLL reduced basis for that lattice. We will omit the algorithm details in this section because the algorithm is not an integer relation algorithm in and of itself. The algorithm is presented, instead, in Appendix A.2 on page 143.

The following theorem is needed to show that LLL can find integer relations. It is proved in Borwein [29, Appendix B].

Theorem 2.3.2. *Let $b_1, \dots, b_n$ be an LLL reduced basis for a lattice, L. Then for every nonzero vector $x \in L$ we have $\|b_1\| \leq 2^{(n-1)/2}\|x\|$* □

2.3.2 Integer Relations with LLL

In order to use LLL to find an integer relation, we must express the problem of finding an integer relation in terms of lattice reduction. In more precise terms we reduce the problem of finding an integer relation for a vector, x, to a particular lattice reduction case. The process we use is described (and proved correct) by Borwein [29].

To perform the reduction, simply speaking, we construct the lattice $\Lambda_x(N)$, defined below, and then perform LLL on that lattice. If there is an integer relation to be found, it will be encoded in the reduced lattice given by LLL. The details are given presently.

Definition 2.3.3 (Real $\Lambda_x(N)$). *Given a vector $x \in \mathbb{R}^n$, we denote by $\Lambda_x(N)$ the lattice spanned by the $(n+1)$-dimensional vectors:*

$$\{(1, 0, \dots, 0, Nx_1), (0, 1, 0, \dots, 0, Nx_2), \dots, (0, 0, \dots, 1, Nx_n)\}$$

where $N \in \mathbb{R}$.

We will introduce an analogous $\Lambda_x(N)$ for complex vectors in Section 2.3.3 (see Definition 2.3.6 on page 14). The essential approach is the same for both lattices, and it is expected that the intended lattice will be clear in context. For now, we consider only the $\Lambda_x(N)$ for the real case.

Proposition 2.3.4. *Let $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ and suppose that an integer relation of x exists. Suppose $\{b_1, \dots, b_n\}$ is an LLL reduced basis for $\Lambda_x(N)$. Then so long as N was sufficiently large, b_1 must be of the form $(b_{11}, \dots, b_{1n}, 0)$ and $(b_{11}, \dots, b_{1n})$ is an integer relation of x.*

Proof. Let $a = (a_1, \dots, a_n)$ be a smallest integer relation for x. Consider the set of vectors $\left\{y \in \mathbb{Z}^n \ : \ \|y\| < 2^{n/2}\|a\| \text{ and } \sum_{i=1}^{n} y_i x_i \neq 0\right\}$, and observe that the set is finite and non-empty. We choose from this set a vector y such that $|\sum_{k=1}^{n} y_k x_k|$ is minimal, and choose N such that $N|\sum_{i=1}^{n} y_i x_i| > 2^{n/2}\|a\|$.

Let $\lambda \in \Lambda_x(N)$; it must be of the form $\left(l_1, \dots, l_n, N\sum_{k=1}^{n} l_k x_k\right)$ for $l_i \in \mathbb{Z}$. Now suppose that $\sum_{k=1}^{n} l_k x_k \neq 0$ (i.e., $(l_1, \dots, l_n)$ is not an integer relation for x). Then

$$\begin{aligned}\|\lambda\| &= \sqrt{l_1^2 + \cdots + l_n^2 + \left(N \sum_{k=1}^{n} l_k x_k\right)^2} \\ &> \sqrt{N^2 \left(\sum_{k=1}^{n} l_k x_k\right)^2} \\ &= N\left|\sum_{k=1}^{n} l_k x_k\right| \geq N\left|\sum_{k=1}^{n} y_k\, x_k\right| > 2^{n/2}\|a\| > 2^{(n-1)/2}\|a\|\end{aligned}$$

We know that a is an integer relation of x, so $\sum_{k=1}^{n} a_k\, x_k = 0$, and hence $\lambda_a := (a_1, \dots, a_n, 0) \in \Lambda_x(N)$. It is easy to see that $\|a\| = \|\lambda_a\|$. So from the calculation above we know that $\|\lambda\|$ is greater than $2^{(n-1)/2}$ times the norm of a vector in $\Lambda_x(N)$ (specifically, the vector λ_a). Consequently λ cannot be the reduced basis vector b_1 by Theorem 2.3.2. The result follows. □

Note that the integer relation a from the beginning of the proof is not necessarily the integer relation embedded in the reduced basis vector b_1.

The value N effectively acts as a penalty, raising the norm of any lattice point which does not have an integer relation embedded in it above the value for which it can be a candidate for the reduced basis vector b_1. The required magnitude of N, however, is not given by the theorem and depends on the norm of any smallest integer relation for x. As such, since the goal is to find an integer relation in the first place, this value is not known in advance.

Although the proposition guarantees that the reduced lattice vector b_1 will contain an integer relation embedded within it, it is not difficult to see that any element of $\Lambda_x(N)$ which is of the form $(m_1, \dots, m_n, 0)$ contains an embedded integer relation, $m = (m_1, \dots, m_n)$, of x. So, after running LLL on $\Lambda_x(N)$, the resulting reduced lattice might contain multiple integer relations. We may identify them simply by looking to see which vectors have a 0 as the final element. We will give this element a name.

Definition 2.3.5 (discriminant). *Let L be a lattice used to find integer relations. For all k for which the k^{th} entry of any lattice vector is used in the identification of an embedded integer relations, we call a_k a discriminant of a for all $a \in L$.*

Note that the above definition allows for the possibility of multiple discriminants. This will be needed when we explore methods of finding Gaussian integer relations with LLL in the next subsection, and for more abstracted integer notions discussed in Chapter 3.

2.3.3 Complex Integer Relations with LLL

The LLL algorithm as written specifically computes rational integer (i.e., $\mathbb{Z}$) lattices, and not Gaussian integer lattices. It is possible, however, to reduce the problem of finding a reduced Gaussian integer lattice to the problem of finding a reduced rational integer lattice, but at the expense of doubling both the size of the vectors, and the number of vectors in the lattice basis.

The technique we use is a special case of a more general technique described by Gan, Ling, and Mow [45] for using LLL to compute reduced bases of Gaussian integer lattices. Given a matrix representation[1] M for a lattice we construct a new lattice with matrix representation M' by

$$M' = \begin{bmatrix} \Re M & \Im M \\ -\Im M & \Re M \end{bmatrix}$$

and then the LLL algorithm is performed on the new lattice.

Gan, Ling, and Mow note that in general the reduced lattice returned by LLL is not in the same block format as the constructed lattice represented by M'. As such, it is difficult or impossible to extract the reduced Gaussian integer lattice basis. Consequently, the output must be specially processed in such a way as to cope with the obfuscated information

We note here that Gan, Ling, and Mow have created a modification of the algorithm to directly compute LLL reduced Gaussian integer lattices. This modification avoids the problems described above. In principle, this modified LLL could directly compute Gaussian integer relations using the technique described in the previous section. In practice, however, *Maple*'s implementation of the LLL algorithm does not implement this modification and the author discovered the modification sufficiently late as to lack the time to implement it.

In lieu of the direct modification to LLL, we instead apply the above reduction technique directly to the $\Lambda_x(N)$ lattice (from Definition 2.3.3) and perform LLL on the resulting lattice. In contrast to the general reduction technique, we are always able to easily extract the Gaussian integer relation from the output of the LLL algorithm. The modified $\Lambda_x(N)$ is given in the following definition.

[1]Recall that we are using the unusual convention whereby the row vectors of the matrix are the lattice basis vectors.

Definition 2.3.6 (Complex $\Lambda_x(N)$). *Given a vector $x \in \mathbb{C}^n$, we denote by $\Lambda_x(N)$ the lattice spanned by the $(2n+2)$-dimensional vectors:*

$$\{(1,0,\dots,0,N\Re x_1,0,\dots,0,N\Im x_n),\dots,(0,\dots,0,1,N\Re x_n,0,\cdots,0,N\Im x_n)\}$$
$$\cup$$
$$\{(0,\dots,0,-N\Im x_1,1,0,\dots,0,N\Re x_1),\dots,(0,\dots,0,-N\Im x_n,0,\cdots,0,1,N\Re x_n)\}$$

where $N \in \mathbb{R}$.

A vector in this lattice will have the form $(l_1,\dots,l_n,d_1,m_1,\dots,m_n,d_2)$. The elements d_1 and d_2 are discriminants. If *both* of discriminants are 0 then the lattice vector encodes a Gaussian integer relation, a say, of x which we can extract as $a = (m_1 + l_1\mathrm{i},\dots,m_n + l_n\mathrm{i})$. It is not immediately obvious that this is the case, so we take some pains to prove it here.

Lemma 2.3.7. *Let $x = (x_1,\dots,x_n) \in \mathbb{C}^n$ and consider the lattice $\Lambda_x(N)$. For every Gaussian integer relation, a, of x there is a vector*

$$(\Re a_1,\dots,\Re a_n,0,\Im a_1,\dots,\Im a_n,0) \in \Lambda_x(N)$$

Moreover, for every lattice element of that form there is a corresponding Gaussian integer relation of x.

Proof. An arbitrary element of the lattice $\Lambda_x(N)$ is of the form $(l_1,\dots,l_n,d_1,m_1,\dots,m_n,d_2)$ where

$$d_1 = N\sum_{k=1}^{n} l_k\,\Re x_k - N\sum_{k=1}^{n} m_k\,\Im x_k = N\sum_{k=1}^{n}(l_k\,\Re x_k - m_k\,\Im x_k)$$
$$d_2 = N\sum_{k=1}^{n} m_k\,\Im x_k + N\sum_{k=1}^{n} m_k\,\Re x_k = N\sum_{k=1}^{n}(l_k\,\Im x_k + m_k\,\Re x_k)$$

For any vector $a = (a_1,\dots,a_n) \in \mathbb{Z}[\mathrm{i}]^n$ we have

$$\begin{aligned}\sum_{k=1}^{n} a_k x_k &= \sum_{k=1}^{n}(\Re a_k + \Im a_k\mathrm{i})(\Re x_k + \Im x_k\mathrm{i})\\ &= \sum_{k=1}^{n}(\Re a_k\Re x_k + \Re a_k\Im x_k\mathrm{i} + \Im a_k\Re x_k\mathrm{i} - \Im a_k\Im x_k)\\ &= \sum_{k=1}^{n}(\Re a_k\Re x_k - \Im a_k\Im x_k) + \mathrm{i}\sum_{k=1}^{n}(\Re a_k\Im x_k + \Im a_k\Re x_k)\end{aligned}$$

Since $\Re a_k$ and $\Im a_k$ are all integers, the vector $(\Re a_1, \dots, \Re a_n, d_1, \Im a_1, \dots, \Im a_n, d_2) \in \Lambda_x(N)$. In this case d_1 and d_2 are precisely the real and imaginary parts, respectively, of $N \sum_{k=1}^{n} a_k x_k$.

If a is a Gaussian integer relation for x then $\sum_{k=1}^{n} a_k x_k = 0$ and so $d_1 = d_2 = 0$ giving us $(\Re a_1, \dots, \Re a_n, 0, \Im a_1, \dots, \Im a_n, 0) \in \Lambda_x(N)$. Conversely, if $(l_1, \dots, l_n, 0, m_1, \dots, m_n, 0) \in \Lambda_x(N)$, then

$$N \sum_{k=1}^{n} (l_k \Re x_k - m_k \Im x_k) = N \sum_{k=1}^{n} (l_k \Im x_k + m_k \Re x_k) = 0$$

and $\sum_{k=1}^{n} (l_k + m_k \mathrm{i}) x_k = d_1 + d_2 \mathrm{i} = 0$, hence $a = (l_1 + m_1 \mathrm{i}, \dots, l_n + m_n \mathrm{i})$ is a Gaussian integer relation for x. □

In essence, this lemma establishes a bijective correspondence between Gaussian integer relations, and vectors in the lattice with both discriminants equal to 0. We further observe that the norm of a Gaussian integer relation and the norm of its corresponding lattice vector coincide.

$$\begin{aligned}\|a\| &= \sqrt{|a_1|^2 + \cdots + |a_n|^2} \\ &= \sqrt{(\Re a_1)^2 + (\Im a_1)^2 + \cdots + (\Re a_n)^2 + (\Im a_n)^2} = \|(\Re a_1, \dots, \Re a_n, 0, \Im a_1, \dots, \Im a_n, 0)\|\end{aligned}$$

Armed with these facts, we can apply the same essential argument used to prove Proposition 2.3.4 to conclude that performing LLL on the lattice above will find a Gaussian integer relation so long as one exists and N is sufficiently large. We state the analogous lemma, but omit the proof due to its similarity.

Proposition 2.3.8. *Let $z = (z_1, \dots, z_n) \in \mathbb{C}^n$ and suppose that a Gaussian integer relation of z exists. Suppose $\{b_1, \dots, b_{2n}\}$ is an LLL reduced basis for $\Lambda_x(N)$. Then so long as N was sufficiently large, b_1 must be of the form $(b_{11}, \dots, b_{1n}, 0, b_{1(n+2)}, \dots, b_{1(2n+1)}, 0)$ and $(b_{11} + b_{1(n+2)}\,\mathrm{i}, \dots, b_{1n} + b_{1(2n+1)}\,\mathrm{i})$ is a Gaussian integer relation of x.* □

2.4 Numeric Considerations

Both the PSLQ and LLL algorithms as presented assume perfect real arithmetic. In particular, any guarantee of finding relations is contingent on this assumption. In practice these algorithms are run on computers using floating point arithmetic, and the limited precision of such computations (even if that precision is large in computational terms) can yield incorrect results in some cases; particularly if the integers in a relation are large compared to the precision of the calculation.

Furthermore, the nature of floating point arithmetic requires extra considerations in the implementation and use of these algorithms over and above that needed for the pure mathematics of the algorithms. We discuss these herein.

2.4.1 Minimum Theoretical Required Precision

Bailey and Broadhurst [9] make mention of an information theoretic argument stating that in order to find an integer relation of n numbers where the integers of that relation are of no more than d digits, then at least nd precision must be used for the computation. The argument is not given, nor is a reference for it. Furthermore, they state that PSLQ "can reliably recover relations with only a few percent more digits of precision than the information theory bound". Borwein [29] clarifies that PSLQ "usually recovers a given relation using only about 15% more digits than this bound suggests are necessary."

The literature in question considers only the real PSLQ case, so it is not clear how this argument applies to Gaussian integers for the complex case of PSLQ. In particular, it is not clear how to measure the number of digits of a Gaussian integer with regard to this information bound (e.g., is $12 + 34\mathrm{i}$ a 2-digit or 4-digit Gaussian integer?).

We have explored these precision questions experimentally and present this exploration below (see Sections 2.5 and 2.6). For the purposes of this discussion we can say that we have found that for the purposes of this information theoretic bound the number of digits of a Gaussian integer is equal to the number of digits of the real part or of imaginary part, whichever is largest. As such, our example $12 + 34\mathrm{i}$ is a 2-digit Gaussian integer, whereas $12 + 345\mathrm{i}$ would be a 3-digit Gaussian integer.

In practice, we tend not to know ahead of time how many digits the elements of an integer relation will have, yet we may still make use of this relation. If we suppose that we are attempting to find an integer relation for a vector $x \in \mathbb{F}^n$ using a precision of p decimal digits, then we can infer that any integer relation whose maximum element has more than p/n digits should be considered highly suspect (since we ought to have required more than p decimal digits of precision to find such an integer relation).

2.4.2 Candidate Integer Relations

In the absence of the certainty of perfect real arithmetic, some extra care must be taken with the output of both PSLQ and LLL (and, indeed, any integer relation algorithm).

A common occurrence is that the algorithm terminates yielding a "false" integer relation. Such a relation gives $\sum a_k x_k \approx 0$ when computed to the precision of the computation, but if checked at higher precision the same calculation is no longer approximately 0.

For example, attempting to compute an integer relation for $x = (\pi, \mathrm{e}, \gamma)$ in *Maple* using PSLQ with 50 decimal digits of precision gives the result

$$a = (68770636450125990, -105820702627301623, 124045778945321093)$$

however, when we check the relation (also to 50 decimal digits of precision) we find that $\sum_{k=1}^{3} a_k x_k \approx -1.8 \times 10^{-32}$. Checking to twice as much precision (100 decimal digits) gives $\sum_{k=1}^{3} a_k x_k \approx -1.18 \times 10^{-32}$ If a were an integer relation we would have expected a value closer to 10^{-50} for the first check, and closer to 10^{-100} for the second check.

Applying the analysis from the previous section here we have $p = 50$ and $n = 3$. We observe that the largest integer contains 18 decimal digits, which is larger than the bound of $p/n = 50/3 \approx 16.6$. Equivalently, to find such an integer relation the theoretical bound stipulates that should have needed at least $18n = 54$ decimal digits of precision, which is more than we used. So this relation should have been immediately suspect.[2]

If instead we use 100 decimal digits of precision then PSLQ yields the result

$$\begin{aligned} a = (&-58008562265955389466919139441407, 48900683858791174575736827116846, \\ &85433289021827429309535576346023) \end{aligned}$$

Checking at the same precision gives $\sum_{k=1}^{3} a_k x_k \approx 1.4 \times 10^{-67}$, and checking to twice the precision gives $\sum_{k=1}^{3} a_k x_k \approx 1.82 \times 10^{-67}$. If a were an integer relation we would have expected a value closer to 10^{-100} for the first check, and closer to 10^{-200} for the second check.

Again applying the analysis from the previous section here we have $p = 100$ and $n = 3$. We observe that the largest integer contains 32 decimal digits, which is actually within the bound of $p/n = 100/3 \approx 33.3$. Equivalently, to find such an integer relation the theoretical bound stipulates that should have needed at least $32n = 96$ decimal digits of precision, which is lower than we used (although not within the 15% margin specified by Borwein). It is worth noting, then, that a candidate relation being within the theoretical bounds is not a guarantee of its correctness.

Although neither of these are an integer relation for $(\pi, \mathrm{e}, \gamma)$, they are sufficiently close numerically for *Maple*'s implementation of PSLQ to give them as output when computing at the indicated numeric precision. It is important to note that no error is indicated; *Maple*'s PSLQ function simply gives the output as the result of its computation.

We should consider the output of any integer relation finding algorithm (whether it be PSLQ, LLL, or any other) with some suspicion. As such, we consider any such integer

[2]The author notes that, as a coarse heuristic, candidate relations consisting entirely of large integers of approximately the same magnitude are usually not actual integer relations. Moreover these are often recognisable “by eye”, although one should not fail to check them anyway.

relations as candidates (see Definition 2.4.1). It is a good habit to verify the veracity of candidate relations by computing $\sum_{k=1}^{n} a_k x_k$. Ideally such verification should be done at higher precision than was used to compute the integer relation to begin with (notwithstanding that extra precision was unnecessary in the examples above).

Definition 2.4.1 (Candidate Integer Relation). *The output of any numerically inexact (e.g., floating point) implementation of an integer relation finding algorithm is considered to be a candidate integer relation, or simply a candidate relation.*

2.4.3 Identifying Candidate Integer Relations

The above discussion raises the question of how, exactly, the algorithms decide when they have a candidate relation in practical numeric implementations. The answer differs somewhat between the PSLQ and LLL, and we will discuss each one separately.

PSLQ

In principle, PSLQ as presented in Algorithm 2.2.7 on page 10 could simply be modified to terminate when $r = n - 1$ and $|H'_{n-1,n-1}| < \epsilon$ where ϵ is some threshold (we discuss the choice of such thresholds below). In practice, however, more significant modifications are made to the algorithm. The modified algorithm is presented in Algorithm 2.4.2.

Algorithm 2.4.2: PSLQ

```
Input  : $x \in \mathbb{F}^n, T > 0, \gamma \geq \gamma_1, \epsilon > 0$
Output: A vector in $\mathcal{O}^n$ or FAIL

   /* ———————————— Initialisation ———————————— */
1  $y \leftarrow x/\|x\|$                                          /* Normalise input vector */
2  $H' \leftarrow D_{H_y} H_y \quad B \leftarrow D_{H_y}^{-1} \quad y \leftarrow y D_{H_y}^{-1}$        /* Initial Hermite reduction */
   /* ———————————— Main Calculation ———————————— */
3  repeat
4      $r \leftarrow \arg\max_{1 \leq r \leq n-1}(\gamma^r |H'_{r,r}|)$            /* Find $r$ s.t. $\gamma^r|H'_{r,r}|$ is maximal */
5      $\mathrm{row}_r(H') \leftrightarrow \mathrm{row}_{r+1}(H')$               /* Swap rows $r$ and $r+1$ in $H'$ */
6      $\mathrm{col}_r(B) \leftrightarrow \mathrm{col}_{r+1}(B)$                 /* Swap columns $r$ and $r+1$ in $B$ */
7      $y_r \leftrightarrow y_{r+1}$                                 /* Swap elements $r$ and $r+1$ in $y$ */
8      $H' \leftarrow H' Q_{[H',r]}$                          /* Make sure $H'$ is lower trapezoidal */
9      $H' \leftarrow D_{H'} H'$                                   /* Hermite reduce $H'$ */
10     $B \leftarrow B D_{H'}^{-1} \quad y \leftarrow y D_{H'}^{-1}$                  /* Update $B$ and $y$ */
11     $k \leftarrow \arg\min_{1 \leq k \leq n}(|y_k|)$                  /* Find $k$ s.t. $|y_k|$ is minimal */
12 until $|y_k|/\|\mathrm{col}_k(B)\| < \epsilon$ or $1/\max_{1 \leq i \leq n}|H'_{i,i}| \geq T$
13 if $|y_k|/\|\mathrm{col}_k(B)\| < \epsilon$ then return $\mathrm{col}_k(B)$ else return $1/\max_{1 \leq i \leq n}|H'_{i,i}|$
```

This is effectively the algorithm as described by Borwein [29, fig. B.5], which in turn is based on the algorithms as described in Bailey and Broadhurst [9], Bailey and Plouffe [11], and Ferguson, Bailey, and Arno [42]. We note that—by virtue of our definitions of H and Q (Definitions 2.2.3 and 2.2.5 on page 8 and on page 9)—our formulation correctly handles the complex case, whereas the algorithm presented in Borwein is specialised to the real case. Additionally, we have kept to the high-level description for better mathematical understanding.

The matrix B is simply the matrix A^{-1} from Algorithm 2.2.7. Each column of B is considered as a possible integer relation of $x/\|x\|$, and the vector y is kept updated so that $y = (x/\|x\|)\,B$. As such, if $y_k = 0$ for some k, then $\mathrm{col}_k(B)$ must be an integer relation for $x/\|x\|$ and thus also for x. Any relation, a say, found this way will not necessarily have the property $\|a\| \leq \gamma^{n-2}M$ that is guaranteed for a relation given by Algorithm 2.2.7, however.

In practice, because we are performing numeric (floating point) computations, the elements of y are unlikely to be exactly 0, even if we have found an integer relation. To detect termination we consider only the smallest $|y_k|$ as the best possibility for a candidate integer relation. We scale the value of $|y_k|$ by $1/\|\mathrm{col}_k(B)\|$ in order to avoid missing a possible relation if the norm the column of B is particularly large. If the scaled value, $|y_k/\|\mathrm{col}_k(B)\||$ is sufficiently close to 0 (i.e., less than some specified threshold ϵ), then the algorithm terminates giving $\mathrm{col}_k(B)$ as the candidate integer relation. Typically the threshold used is the so-called "machine epsilon", $b^{-(p-1)}$, where b is the base of the floating point implementation, and p is the precision of the computation[3].

As a secondary condition the algorithm terminates if the lower bound on the norm of any integer relation exceeds some pre-defined threshold (specified as an input to the algorithm). In this case the lower bound is returned. This only happens if the algorithm has not otherwise detected a candidate relation.

Borwein does not discuss how to choose the threshold T. We suggest that a good approach would be to choose based on the precision of the calculation and the relationship between integer sizes and minimal theoretical precision (as discussed in Section 2.4.1).

Given $x \in \mathbb{R}^n$, and p digits of precision, then we should not see any integer relations with more than p/n digits for any element. The integer relation with the smallest norm which has an element exceeding p/n decimal digits is

$$a_{min} = (10^{\lfloor n/d \rfloor}, 0, \ldots, 0)$$

[3] A value of $b^{-(p-1)}/2$ is sometimes called "machine epsilon", for example by Prof. Demmel or in the software LAPACK, and Scilab, but we do not use this here.

(or any permutation of these elements) and $\|a_{min}\| = 10^{\lfloor n/d \rfloor}$. The integer relation with the largest norm whose elements do not exceed p/n decimal digits is

$$a_{max} = \left(10^{\lfloor n/d \rfloor} - 1, 10^{\lfloor n/d \rfloor} - 1, \dots, 10^{\lfloor n/d \rfloor} - 1\right)$$

and $\|a_{max}\| = \sqrt{n}(10^{\lfloor n/d \rfloor} - 1)$.

In the complex case (when $x \in \mathbb{C}^n$) the vector a_{min} is the same, but a_{max} becomes

$$a_{max} = \left(\left(10^{\lfloor n/d \rfloor} - 1\right)(1 + \mathrm{i}), \left(10^{\lfloor n/d \rfloor} - 1\right)(1 + \mathrm{i}), \dots, \left(10^{\lfloor n/d \rfloor} - 1\right)(1 + \mathrm{i}) \right)$$

and in this case $\|a_{max}\| = \sqrt{2n}(10^{\lfloor n/d \rfloor} - 1)$.

In either case, if the norm of a suspected integer relation is lower than $\|a_{min}\|$ then it cannot have an element greater than p/n decimal digits. Similarly if the norm of a suspected integer relation is higher than $\|a_{max}\|$ then it must have an element greater than p/n decimal digits. Finally if the norm of a suspected integer relation is between $\|a_{min}\|$ and $\|a_{max}\|$ (inclusive), then it may or may not have an element greater than p/n.

The threshold T should be chosen between $\|a_{min}\|$ and $\|a_{max}\|$. If set to $\|a_{min}\|$ then the algorithm will terminate before it could possibly find a candidate relation with integers that are too large, but might fail to find a legitimate relation whose norm is above the threshold. If set to $\|a_{max}\|$ then the algorithm will terminate at the point at which it can no longer find candidate integer relations whose elements are all below the maximum allowed by the theoretical bound, but may nonetheless find a relation with integers that are too large. Since one ought to check any candidate integer relation anyway, and since a candidate relation may not be an actual integer relation even if all its elements are sufficiently small, it seems most sensible to choose $T = \|a_{max}\|$.

We note in passing that this use of the lower bound threshold would be useful in number theoretic uses of PSLQ, such as searching for minimal polynomials of algebraic numbers. An analysis similar to the above allows for a way to relate the (Euclidean) norm lower bound of any integer relation for $(\alpha^n, \dots, \alpha, 1)$ to a bound on the height of a polynomial for which $\sum_{k=0}^{n} \lambda_n \alpha^n = 0$ with $\lambda_k \in \mathbb{Z}$. See Section 3.1.1 on page 45 for an introductory treatment of algebraic numbers.

Returning to the question of numeric termination conditions of the PSLQ algorithm, we note that PSLQ variants as presented in Bailey and Broadhurst [9], Bailey and Plouffe [11], and Ferguson, Bailey, and Arno [42] (the literature on which the algorithm presented by Borwein's is based) do not use norm lower bound as a secondary termination condition, although they do keep track of the increasing norm lower bound. Their primary termination method is essentially the same as the one we present, above, although they do not scale by the norm of the column of B; they simply check whether $|y_k| < \epsilon$. For a secondary termination condition, they instead look to see if precision has been exhausted.

If the integers in the algorithm are stored as floating point numbers, then if any of these numbers are larger than the largest integer that can be exactly represented by the floating point system, numeric precision has been exhausted. This is easily detected. If, however, the integers in the algorithm are stored using a fixed size integer data type this detection may be much harder.

Our experimental exploration in this chapter uses *Maple*'s implementation of PSLQ and we note that it is not entirely clear what approach it uses for termination. *Maple*'s implementation accepts neither a maximum iteration count, nor a threshold for norm lower bound; indeed the only argument that it takes is the vector of numbers to find an integer relation for. Moreover, it does not return any lower bound information whatsoever, only a candidate integer relation.

If we examine the code of *Maple*'s `PSLQ()` function[4] we see the details of an initialisation procedure that performs initial checks and setup before calling the actual PSLQ implementation. Of particular interest: the input vector (of length n, say) is pre-evaluated to a precision of $p + 5$ (where p is the precision in decimal digits of the forthcoming computation), and a value of $\epsilon = 10^{-p+\log_{10} 2n}$ is used. No other parameter information is forthcoming and the details of the actual PSLQ implementation are hidden; we presume that $\gamma = \gamma_1$ is chosen.

It is likely that *Maple* is using an implementation similar to Borwein's; probably one of the more sophisticated (and faster) variants described by Bailey and Broadhurst. Moreover it seems likely that ϵ is being used to determine termination in a manner at least similar to the one described above. It is unclear if there is any other termination condition at play.

LLL

Proposition 2.3.4 and Lemma 2.3.7 on pages 12 and 15, respectively, assure us that the discriminants of a reduced lattice vector that contains an embedded integer relation will be exactly 0 in the real and complex cases respectively. When computing numerically, however, it is unlikely that the discriminants will be exactly 0. This is easily solved by checking to see if the absolute value of each discriminant is less than some threshold (most likely machine epsilon as discussed above).

A more significant challenge to numerically finding integer relations with LLL is the choice of the value of N with which to construct the lattice $\Lambda_x(N)$. Proposition 2.3.4 on page 12 guarantees that LLL will find an integer relation so long as N is sufficiently large, although we do not know ahead of time the minimum value of N. In practice, when

[4]Using `eval(IntegerRelations[PSLQ])`

performing numerical computation, we have found that N may be too large, as well as too small.

In choosing the value of N to use, an immediately obvious choice would be b^p where b is the base of the floating point implementation being used and p is the precision that the constants have been computed to. In many (perhaps even most) cases this will be $b = 2$ for a binary floating point representation, as is the case with modern computer hardware and commonly used extended precision libraries[5] at the time of writing. It is of note, then, that *Maple* uses a base 10 floating point representation, and consequently we use $b = 10$ for our experimental exploration.

We have found that this choice of N causes *Maple*'s implementation of LLL to incorrectly diagnose the lattice vectors as being linearly dependent, prematurely halting the computation with an error. This is in spite of the fact that—by construction—the lattice vectors *must* be linearly independent. Some imprecise experimentation[6] suggests that *Maple* might be coming to this incorrect diagnosis by computing the rank of the matrix representation of the lattice. Regardless of the reason, this choice of N does not work for *Maple*'s LLL implementation.

Moreover, we have found that even in the absence of this strange *Maple* idiosyncrasy (i.e., when the calculation performs as expected), the value of N can still be too large. In these the cases, no candidate relations are embedded in the returned lattice, even if candidate relations can be found with smaller values of N for the same vector. A significantly lower value than $N = b^p$ is typically needed in order to find an integer relation (see Section 2.6 on page 30).

Returning to the question of how to choose the value of N, and motivated by the above, the general tactic we adopt is to repeatedly try computing LLL on $\Lambda_x(N)$ using different values of N until a candidate integer relation is detected in the output reduced lattice vectors. We start with $N = 10^{p-1}$, and decrement the power of 10 after each failed attempt, giving up if we ever get to $N = 10^0$. The details are given below in the LLL Integer Relation procedure

Note that there may be several reduced lattice vectors b_i which contain an embedded candidate integer relation. Conversely, there may also be none (i.e., the empty set may be returned), but only in the case that the loop reached $k = 0$.

Each candidate should, as discussed in Section 2.4.2 above, be checked to higher precision. If all candidates fail such testing the computation could be resumed at the next value of k, although we do not currently do so in our experimental exploration, presented below.

[5] e.g., GMP and MPFR

[6] We performed several computations to diagnose the linear (in)dependence of the row vectors and only `rank` gave an incorrect result.

Procedure LLL Integer Relation

```
Input  : $x \in \mathbb{F}^n, \epsilon > 0$
Output: $a \in \mathscr{P}(\mathscr{O}^n)$
1 $k \leftarrow p - 1$
2 repeat
3     $M \leftarrow \Lambda_x(10^k)$
4     $B \leftarrow LLL(M)$
5     $k \leftarrow k - 1$
6 until $k = 0$ or $|b_{n+1}| < \epsilon$ for some $b \in B$
7 return $\{b \in B \ : \ |b_{n+1}| < \epsilon\}$
```

2.5 Experimental Methodology

We created collections of instances of integer relation problems. Each collection, referred to as a *test set*, consisted of 1000 integer relation problems each of which has a known integer relation.

We note that that the testing reported in both this and the next chapter were performed as part of a single suite of tests. In fact, the testing reported in in this and the next section is simply a particular case of the testing reported in Chapter 3; we simply report only the cases relevant to the discussion in this chapter. The testing reported in this entire thesis extends the testing reported in Skerritt and Vrbik [67].

We tested both PSLQ and LLL to see how many of the problems in each test set could be solved (i.e., the known relation recovered) by the algorithm in question. For those problems that were able to be solved we found and recorded the smallest numeric precision that was able to solve the problem, as well as the time taken to solve the problem at that minimal precision. We also measured the time taken to compute the entire test set.

In the case of LLL we additionally recorded parameters from the LLL Integer Relation procedure. We recorded the starting value of 10^k and the number of LLL computations attempted. In addition, for the problems which could be solved, we recorded the value of 10^k with which a relation was found, and the number of attempted LLL computations needed.

We used *Maple*'s implementation of both PSLQ and LLL for all testing. For PSLQ we used the algorithm directly for all cases. For LLL we used the appropriate $\Lambda_x(N)$ (Definitions 2.3.3 and 2.3.6) depending on the case being tested.

The code and results are available through GitHub [66]. Note that the testing described in this chapter is a special case of a more general testing method encompassing the testing from both here, and the next chapter (see Sections 3.4 and 3.5 on page 66 and on page 70). Additionally, the testing reported in this entire thesis extends the testing reported in Skerritt and Vrbik [67].

We note that *Maple*'s PSLQ implementation does not allow for any choice of the parameters described above for that algorithm. All algorithmic parameters have been decided by the developers; one simply calls the function and passes to it the list of constants (x in Algorithm 2.2.7). As such we are unable to report on the effects of changing them.

As discussed in Section 2.4.3 on page 19, we know that the input vector (of length n, say) is pre-evaluated to a precision of $p + 5$ (where p is the precision in decimal digits of the forthcoming computation), and a value of $\epsilon = 10^{-p+\log_{10} 2n}$ is used for identifying 0. No other parameter information is forthcoming and the details of the actual PSLQ implementation are hidden; we presume that $\gamma = \gamma_1$ is chosen.

When using Algorithm - on the previous page to compute integer relations we use a threshold of $\epsilon = 1$. This is significantly higher than the suggested value of machine epsilon, and was first chosen when experimenting with exact rational computations for LLL.This higher value simply results in potentially more candidate relations to check, although the large value of N will likely cause the discriminant of any lattice vector without an embedded integer relation to be significantly larger than 1. When looking at the results, we see the majority produced only a single candidate relation (7,253 out of 12,000 of cases) or two candidate relations (3,451 out of 12,000 cases) accounting for approximately 88% of all LLL classical integer relation computations. We occasionally see as many as 9 candidates.

In the subsections that follow we describe in greater detail the methods by which we create the test sets, perform the testing, and report the results.

2.5.1 Test Set Generation

The test sets are classified by three parameters: a field ($\mathbb{F}$), a set of constants ($C \subset \mathbb{F}$), and a size ($\iota \in \mathbb{N}$) of the integers generated as part of the individual integer relation problems within the test set. We created a single test set for each valid combination of these parameters. The parameters are described below, along with the values we used in our testing.

Field

The field, $\mathbb{F}$, indicates the type of the test set. Since we are computing classical integer relations, test sets are either *real* (when $\mathbb{F} = \mathbb{R}$) or *complex* (when $\mathbb{F} = \mathbb{C}$)

The type of the test set dictates which integer ring will be used when generating the integer relation problems for that test set. Real test sets use the integers, $\mathbb{Z}$, and complex test sets use the Gaussian integers, $\mathbb{Z}[\sqrt{-1}]$. Note that these are precisely the combinations of $\mathbb{Z}$ and $\mathcal{O}$ from Definition 2.1.1.

Furthermore, the type of the test set indicates which sets of constants (see below) may be used in test set problem generation.

Constants

The set of constants, C, is a finite subset of the field, $\mathbb{F}$, which contains all constants that were used when generating problems for the test set. In practice, the set of constants was chosen first, and the integer relation problems in the set were generated using only that set.

Exactly two sets of constants were used: one containing real constants, the other complex. The real set was

$$\begin{aligned} C_{\mathbb{R}} = &\left\{\pi^k : k \in \mathbb{N}, k \leq 9\right\} \cup \left\{e^k : k \in \mathbb{N}, k \leq 9\right\} \cup \left\{\gamma^k : k \in \mathbb{N}, k \leq 9\right\} \\ &\cup \{\sin k : k \in \mathbb{N}, k \leq 9\} \cup \{\ln 2, \ln 3, \ln 5, \ln 7\} \end{aligned}$$

The complex set consists of complex numbers with integer arguments in the range $[-9, 9]$ such that each argument appears exactly once. For each of these numbers, the modulus was randomly chosen in the range $[1, 9]$.

$$\begin{aligned} C_{\mathbb{C}} = &\left\{5\,e^{-9\,i}, 4\,e^{-8\,i}, 9\,e^{-7\,i}, 5\,e^{-6\,i}, 2\,e^{-5\,i}, 9\,e^{-4\,i}, 8\,e^{-3\,i}, 3\,e^{-2\,i}, 2\,e^{-i}, \right. \\ &\left. 4, 4\,e^{i}, 5\,e^{2\,i}, 2\,e^{3\,i}, 7\,e^{4\,i}, 6\,e^{5\,i}, 3\,e^{6\,i}, 3\,e^{7\,i}, 5\,e^{8\,i}, 5\,e^{9\,i}\right\} \end{aligned}$$

Real test sets may only use the real constants. Complex test sets may use either set of constants.

In order to avoid unexpected results in our testing we ran PSLQ (in *Maple*) on each of these coefficient sets. Computation was performed using 10,000 decimal digits of precision, and verification was performed using 20,000 decimal digits of precision. No integer relations were found.

Integer Size

Integer size, ι, is a positive integer indicating the maximum number of decimal digits of any of the known integer relations for the problems within the set. We used two cases in our testing: $\iota = 1$ and $\iota = 6$. These cases are referred to as *small* and *large* respectively.

In the case of complex test sets then the coefficients (i.e., the real and imaginary parts) of the Gaussian integers are each of the indicated size. In practice, this value is chosen first, and the integer relation problems in the set are generated using integers in the range $[-10^{\iota} + 1, 10^{\iota} - 1]$.

Integer Relation Problem Creation

A single test set was generated for each valid combination of the parameters used in our testing. For a fixed $\mathbb{F}, C$, and ι, the test set corresponding to these parameters was created by generating 1,000 integer relation problems with a known solution. The creation procedure is detailed in the Generate Test Instance procedure.

```
Procedure Generate Test Instance($\mathbb{F}, C, \iota$)
Input  :  $\mathbb{F}$   (a field; either $\mathbb{R}$ or $\mathbb{C}$)
          $C$   (a set of constants)
          $\iota$   (number of decimal digits of the integers to be generated)
Output:   $x$   (an integer relation input vector)
          $\mathfrak{a}$   (known integer relation of $x$)
1 $n \leftarrow$ random integer$(2 \dots 10)$            /* Generate a uniform random integer $2 \le n \le 10$ */
2 $C' \leftarrow$ random permutation$(C)$                /* Randomise the order of the constants */
3 for $k$ from 1 to $n$ do               /* Generate $n$ uniform random integers each with $\iota$ digits */
4     $a_k \leftarrow$ random integer$(-10^\iota + 1 \dots 10^\iota - 1)$
5     if $\mathbb{F} = \mathbb{C}$ then
6         $a_k \leftarrow a_k + \mathrm{i} \cdot$ random integer$(-10^\iota + 1 \dots 10^\iota - 1)$
7 $x \leftarrow \left(\sum_{k=1}^{n}(a_k C'_k), C'_1, \dots, C'_n\right)$     /* Construct the integer relation problem */
8 $\mathfrak{a} \leftarrow (-1, a_1, \dots, a_n)$                /* Construct a known integer relation solution */
9 return $x, \mathfrak{a}$
```

2.5.2 Testing Procedure

For each test set, we attempted to solve the problems within it using PSLQ, and LLL (as appropriate) in *Maple*. Our aim was to see if the algorithm could recover the known integer relation, $\mathfrak{a}$, from the input vector x. Any integer multiple of the known relation was considered to be an equivalent relation for this purpose.

Each test set was used twice. Once with so called *short input* wherein only the coefficients in the known relation $\mathfrak{a}$ were used, and once with so called *long input* wherein twice as many coefficients were used (or all the coefficients from the set if there were not enough). The purpose of this was to verify the robustness of the algorithm in the presence of unnecessary information, and to see the effect on the needed precision and time.

We compute at multiple precisions until we find the known integer relation, or until the precision becomes too large. If the known relation can be found, we define the most favourable computation as the one which finds the known relation using the smallest precision. All recorded measurements are either for the most favourable computation (if it exists), or for the computation performed with the largest precision otherwise.

We measure and record the precision used, and the time taken[7] for all computations. For LLL computations we also measure and record the number of LLL attempts before the relation was recovered, and the number of candidate integer relations.

Result Diagnosis

The result of a computation on an individual test instance is classified as outlined in Table 2.1.

Table 2.1: Result Classifications

GOOD	The known integer relation was recovered.
UNEXPECTED	A different, correct integer relation was found.
BAD	An incorrect ("false") integer relation was found.
FAIL	The algorithm produced no candidate relations.

No UNEXPECTED results were found during our testing. This classification was originally introduced in the testing of an early implementation as a result of an oversight in which $\log 2, \log 3$ and $\log 6$ were together in some problems. This oversight has since been corrected, yet it remains possible (although unlikely) that other unexpected relations may still be computed, so we keep the classification as a possibility.

A FAIL result can occur if an unrecoverable error happens during the computation of the integer relation, although in practice we have never seen this. Alternatively, an LLL computation might produce an output with no candidate integer relations (i.e., none of the discriminants are sufficiently close to 0). We note that we have never seen *Maple*'s PSLQ implementation fail to produce a candidate relation, even when given input which is integer linearly independent.

To assess each result classification, we immediately diagnose a FAIL result if no result is produced by the algorithm. Otherwise, we inspect the output of the integer relation computation and look to see if it is an integer multiple of the known integer relation. Observe that if $a = (a_1, \dots, a_n)$ is the output of the integer relation computation and $\mathfrak{a} = (-1, z_1, \dots, z_k)$ is the known relation, then if $a = \lambda\mathfrak{a}$ it must be the case that $\lambda = -a_1$. Finally, if we did not see a multiple of the known relation, we check to see if the output of the integer relation computation nonetheless appears to be a valid integer relation to a precision of 1,000 decimal digits (which is almost double the maximum precision that our integer relation calculations ever use; see below).

In the case that LLL is the integer relation computation function, then we may have multiple columns vectors whose last element is less than 1 in absolute value (i.e., multiple

[7] measured in CPU seconds using *Maple*'s `time()` function

candidates for an integer relation). If this happens, we diagnose each individually, and take the best result using the order GOOD > UNEXPECTED > BAD > FAIL.

Procedure Diagnose($x, A, \mathfrak{a}$)

```
Input :  $x$  (integer relation problem)
         $A$  (set of candidate integer relations for $x$)
         $\mathfrak{a}$  (known integer relation for $x$)
Output:  $result \in \{$GOOD, UNEXPECTED, BAD, FAIL$\}$
         $relation \in A \cup \{$NONE$\}$

1  $result \leftarrow$ FAIL, $relation \leftarrow$ NONE
2  for each $a \in A$ do
3      if $a = (-a_1)\mathfrak{a}$ then
4          if $result <$ GOOD then $result \leftarrow$ GOOD, $relation \leftarrow a$ end
5      else
6          $val \leftarrow \sum_{k=1}^{n} a_k x_k$          /* compute using 1000 decimal digit precision */
7          if $|val| \leq 10^{-998}$ then
8              if $result <$ UNEXPECTED then $result \leftarrow$ UNEXPECTED, $relation \leftarrow a$ end
9          else
10             if $result <$ BAD then $result \leftarrow$ BAD, $relation \leftarrow a$ end
11 return $(result, relation)$
```

Numeric Precision

In order to find the minimum required numeric precision, each integer relation problem was computed multiple times using different precisions. We start at a precision of 1 decimal digit, and double the precision until we produce a GOOD result (or until we exceed some threshold; at which point we stop computing that problem). At this point (provided we are still computing) we have bounded the minimum precision inside a range of $(2^k, 2^{k+1}]$ and use a binary search to find the minimum precision within this bound that gives a GOOD result.

We note that this procedure assumes that if a GOOD result is found for some precision p, then it will be found for all $p' > p$. We do not know a priori that this is the case, but it is infeasible to test each precision individually. To ameliorate this problem, after we have found our minimal precision, we compute the integer relation again using a precision 1 decimal digit higher, and record this result for later comparison.

```
Procedure Find Minimal Precision($x$, $\mathfrak{a}$)
Input  :  $x$  (an integer relation input vector)
          $\mathfrak{a}$  (known integer relation of $x$)
Output:   $p$  (Minimum precision required for a GOOD result)
1  $p \leftarrow 1/2$
2  repeat        /* Double precision until a good result is produced, or p is too large */
3      $p \leftarrow 2p$
4      $candidates \leftarrow Integer\_Relation(x)$        /* Use $p$ decimal digit precision */
5      $result \leftarrow Diagnose(x, candidates, \mathfrak{a})$
6  until $result$ = GOOD or $p > 500$
7  if $result$ = GOOD then        /* Perform a binary search to find where the result changes */
8      $lower \leftarrow \max\{1, p/2\}$, $upper \leftarrow p$
9      repeat
10         $m \leftarrow \lfloor(\text{lower} + (\text{upper} - \text{lower}))/2\rfloor$
11         $candidates \leftarrow Integer\_Relation(x)$        /* Use $m$ decimal digit precision */
12         $result \leftarrow Diagnose(x, candidates, \mathfrak{a})$
13         if $result$ = GOOD then $upper \leftarrow m$ else $lower \leftarrow m$ end
14     until $upper - lower \leq 1$
15     $p \leftarrow upper$        /* The upper bound must be the minimal precision we seek */
16 return $p$
```

2.6 Experimental Results

In all cases, the known integer relation was able to be successfully computed by the integer relation finding algorithm (whether it be PSLQ, or LLL). This is summarised in Table 2.2.

Table 2.2: Experimental results for classical integer relations

Test Set Parameters			Short Input		Long Input	
$\mathbb{F}$	C	ι	PSLQ	LLL	PSLQ	LLL
$\mathbb{R}$	$C_\mathbb{R}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{R}$	$C_\mathbb{R}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{C}$	$C_\mathbb{R}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{C}$	$C_\mathbb{R}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{C}$	$C_\mathbb{C}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{C}$	$C_\mathbb{C}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F

We explore the more detailed results exploring minimum required precision, and comparison between LLL and PSLQ below. In all plots in the following subsections, the horizontal axis represents the individual problems within the test set, numbered from 1 to 1000 after being ordered in ascending order first by theoretical minimum required precision, then by the number of digits in the known relation. Note that this ordering is different to the ordering of the problems in the test set (keyed by the index "ID" in the input file).

2.6.1 Real Test Sets

Precision

The precisions required for the real test sets are shown in Fig. 2.1. We depict, for each test problem, the theoretical minimum precision for that problem along with the smallest precision needed for both PSLQ, and LLL.

We see that required precision tends to grow with minimum theoretical precision, as ought to be expected. We also see that the required LLL precision is more varied than that needed for PSLQ, and is typically significantly larger.

We observe that some computations needed lower precision than the theoretical minimum. These are predominantly LLL computations, although there are also some such PSLQ computations. We are uncertain why this happens, although we do note that it is rare. We suspect that these computations simply got lucky.

We look at the distribution of the required precisions (as a multiple of the theoretical minimum precision) in Fig. 2.2. This is with an eye to verifying Borwein's claim that PSLQ frequently requires less than 15% more than the theoretcial minimum. As such we have marked on each graph the line representing 115% as a horizontal dashed line. For each combination of coefficient size (value of ι) and input length we have grouped the test problems by minimum theoretical precision, and plot the each required precision we found for any test instance with that theoretical minimum. We indicate number of test instances with that required precision by the width of the line.

We see that for small coefficient (i.e., single digit) integer relations we are frequently over the 115% mark (sometimes as much as four times higher). We do not consider this to be particularly significant, as most of these precisions are ridiculously low for practical computation where hardware floating point sizes are approximately 7 or 16 decimal digits of precision (for single or double precision IEEE floating point numbers). We also note that for the low theoretical precisions a difference of 1 or 2 decimal digits in precision is a significant percentage of the theoretical minimum, so a larger variance in percentages is expected. This observation does not, however, explain the difference in behaviour (in this regard) of 18 and 19 decimal digits of theoretical minimum precision. It is worth considering that, for similar sized theoretical minimums, when comparing small and large coefficient test sets the former will have an input vector length approximately 6 times larger than the latter.

For the large coefficient test sets we see as theoretical minimum rises that the required precisions usually do fall at or below 115%, although not always. Based on the data seen here, we suggest that 120% is a better bound.

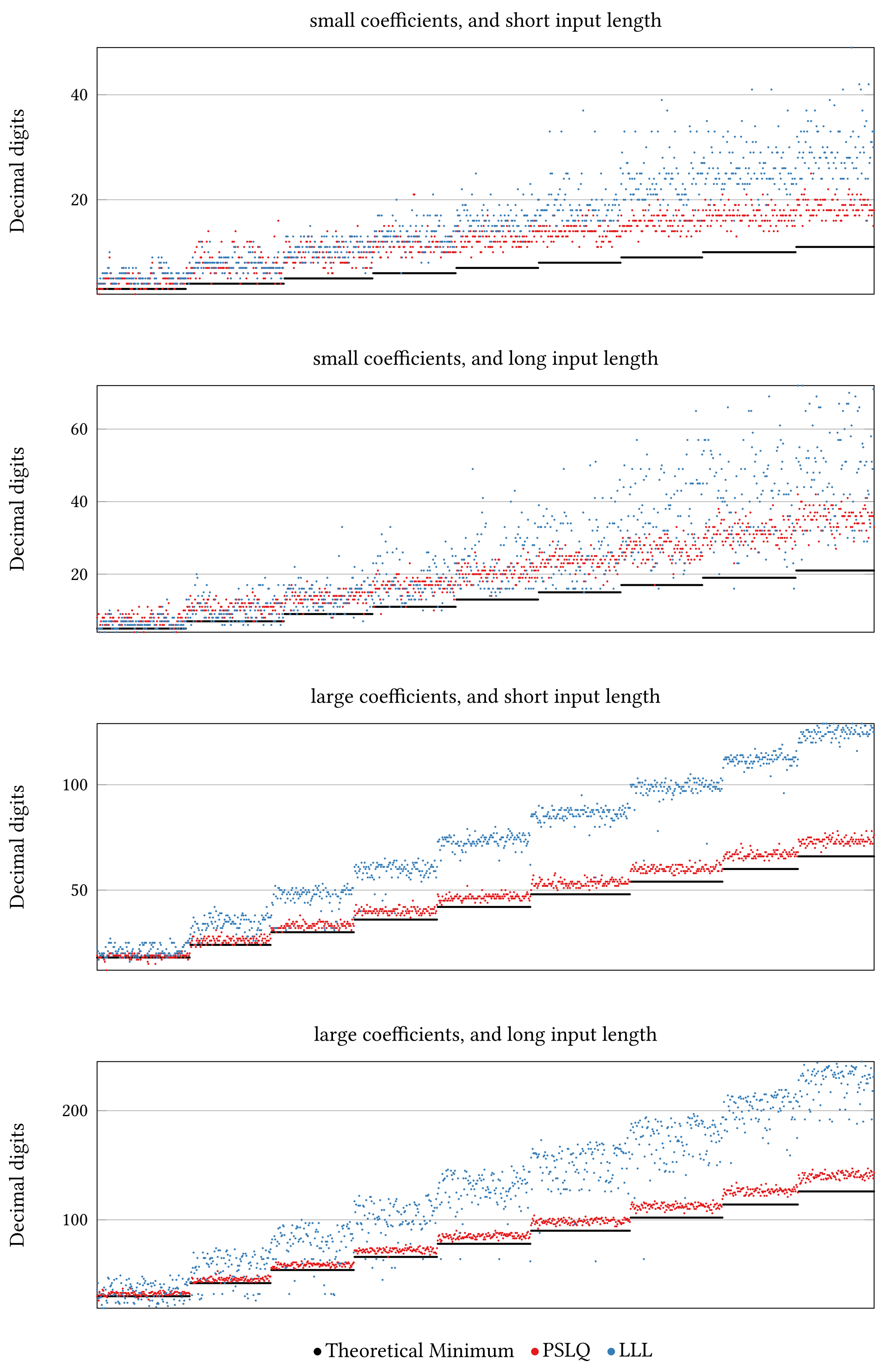


Figure 2.1: Precision required for real test sets using real constants.

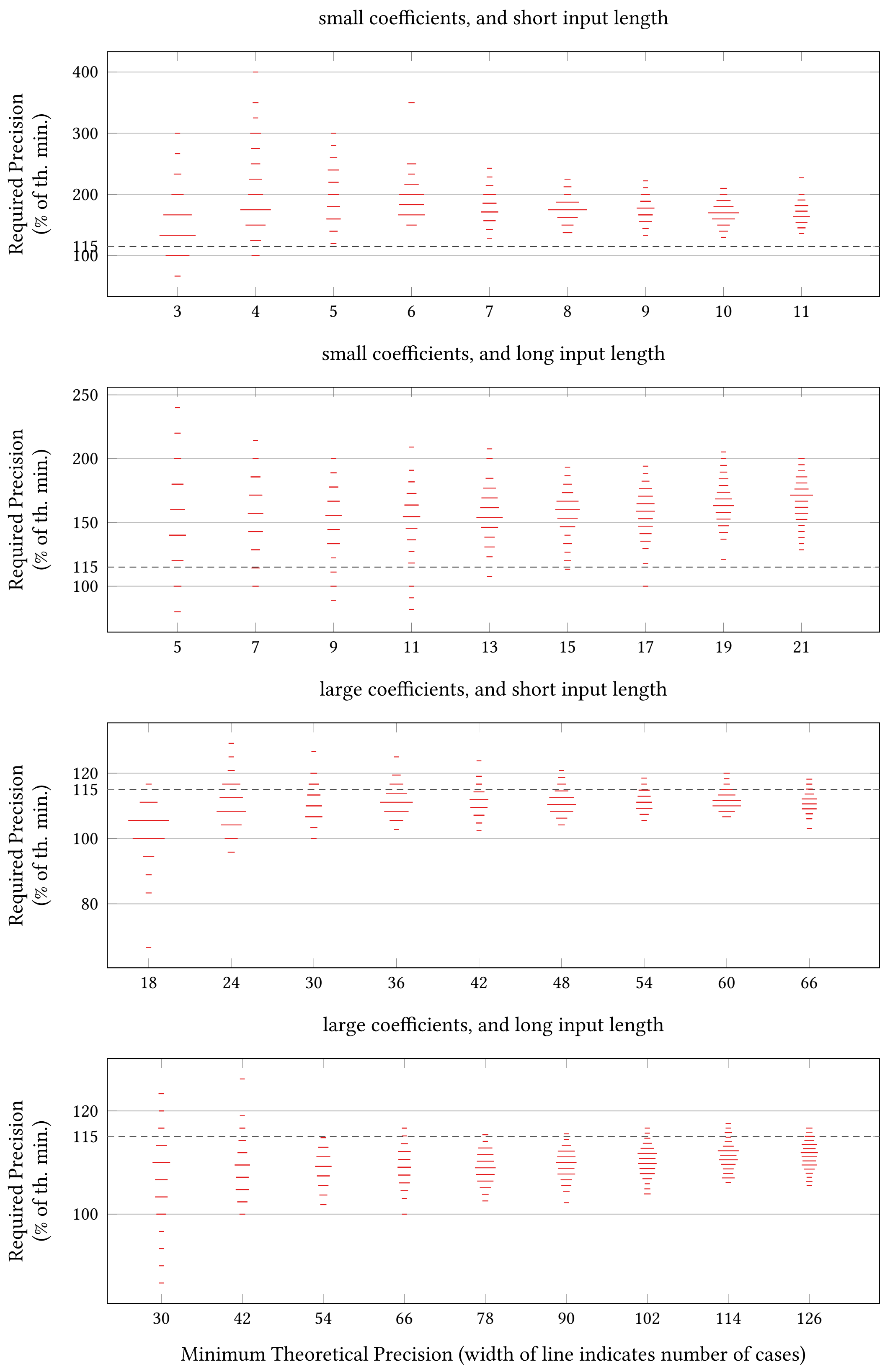


Figure 2.2: PSLQ precision needed for real test sets (as a percentage of theoretical precision).

Timing

We compare the time required for PSLQ and LLL computations on the same test sets. The results are shown in Fig. 2.3. A logarithmic vertical scale is used to better show the small times compared to the larger ones. Recall that the times reported are, as stated above, for the most favourable computation (the one using the lowest precision that nonetheless produces a GOOD result).

We see that LLL uses significantly more time than PSLQ, often two orders of magnitude greater. This is partly explained by the fact that to compute a single integer relation with LLL (a single run of the LLL Integer Relation procedure) requires multiple executions of the LLL algorithm itself, whereas PSLQ requires only a single execution. As the LLL computation times are frequently in the order of 100 times greater than the PSLQ computation times it seems that individual LLL computations are predominantly slower than individual PSLQ computations. The relative speed of a single LLL computation to a single PSLQ computation is moot, however. Using LLL to find an integer relation requires multiple LLL computations and so for the purposes of finding integer relations LLL is demonstrably considerably slower.

We note in passing that the slowness of the LLL algoritihm is compounded when computing an entire test set. This is because each test problem requires multiple executions of the algorithm in question at different precisions in order to find the optimal required precision. In the worst case of real classical test sets ($\iota = 6$ and long input length) we saw a total time of approximately half a minute for PSLQ compared to approximately three and a half days for LLL. This situation was even more pronounced for other cases described in the next chapter; the longest time we recorded was over two months (around 77 days). These total execution times are not particularly relevant to the performance of the algorithms, so we do not discuss them further.

LLL Attempts

If we look at the number of LLL attempts reported in the output files, we see that in the case of small coefficients and short input length that the number of attempts varies from 1 to 28. Similarly, if we look at the case of small coefficients and long input we see that the number of attempts varies from 3 to 40.

In the case of large coefficients, we see a similar effect. For short input we see the number of attempts ranges between 5 and 69, and for long input we see a range of 5 to 126 attempts.

If we look at all the real test set LLL computations, and compare the number of attempts against the precision (in decimal digits) of the computation, we see the results given

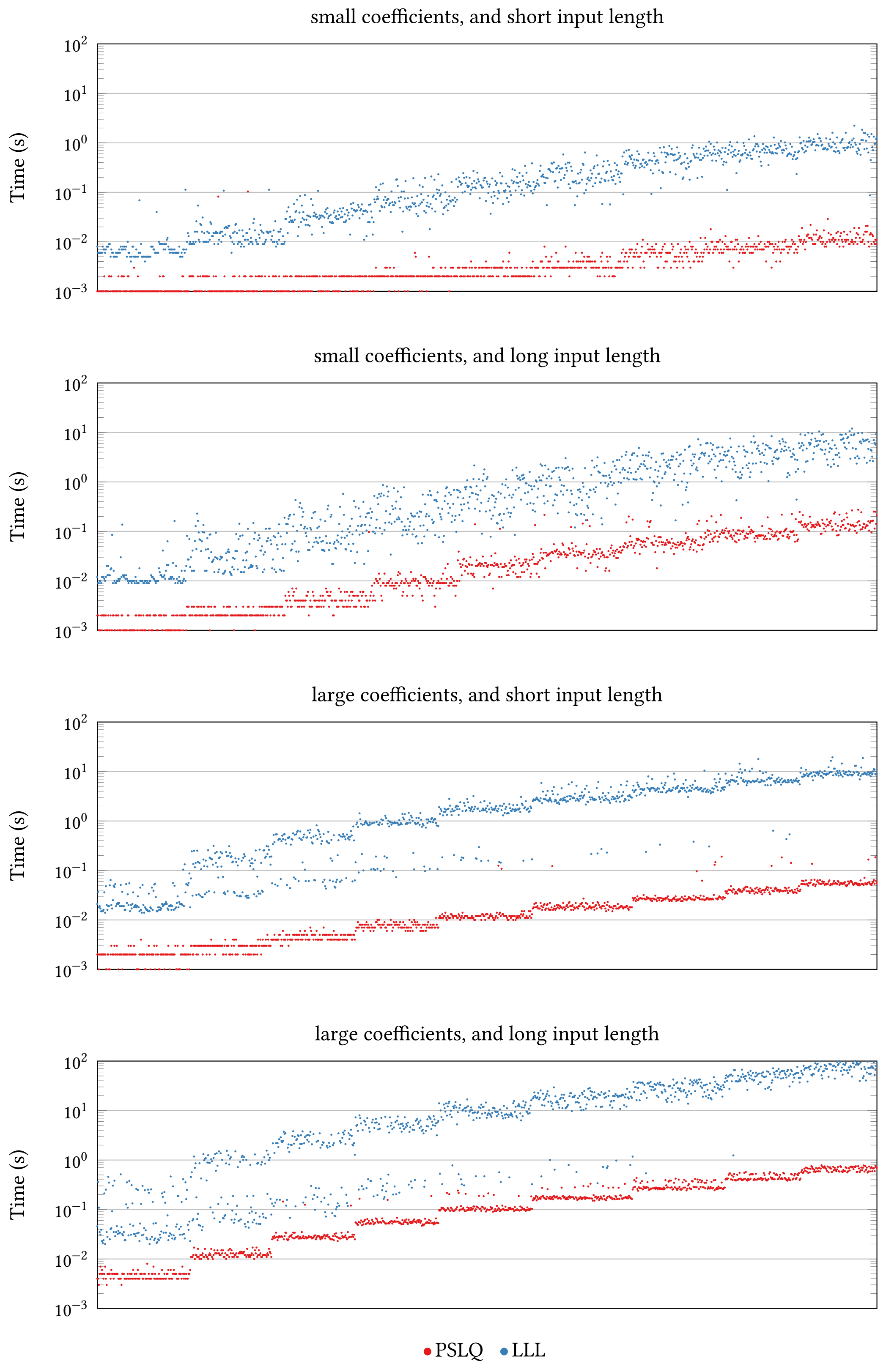


Figure 2.3: Computation time taken for real test sets using real constants.

in Fig. 2.4. We see that the number of attempts is usually in the vicinity of half the computation precision. Moreover, the number of attempts is frequently over 50. We also see that occasionally we need very few attempts, and that this phenomenon occurs for all precision.

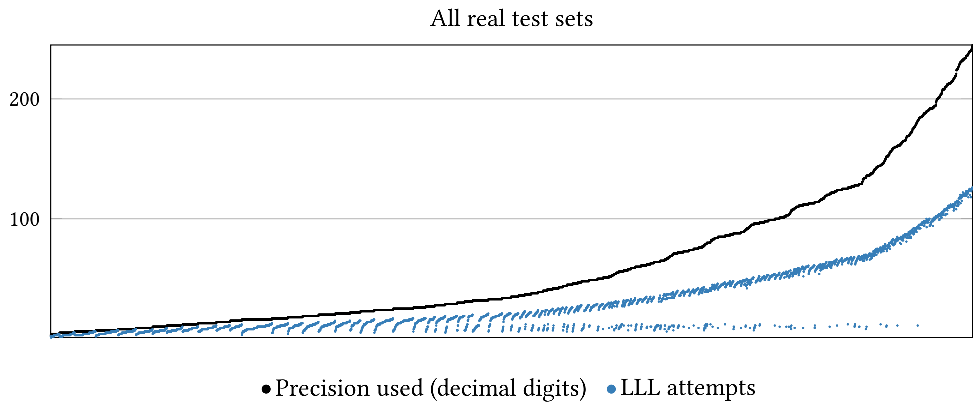


Figure 2.4: LLL computation precision compared against number of LLL attempts (real test sets)

The number of attempts is related to the value of k for which computing LLL on $\Lambda_x(10^k)$ was successful by $k = p - attempts$. If we plot the value of k as a multiple of computation precision, with an eye to choosing a better starting value of k in the LLL Integer Relation procedure on page 24, we see no clear relation which may be useful. The results can be seen in Fig. 2.5.

The number of attempts also speaks to the phenomenon whereby the value of N may, numerically, be too large. In extreme cases, we needed to reduce the value of N by a factor of more than 10^{-100} before we were able to compute a known integer relation. This particular case happened 161 times out of the 4000 total tests performed for the real cases, and we note that in 882 cases the number of attempts was 50 or above. Most cases were not this extreme, however, with the median number of attempts being 16.

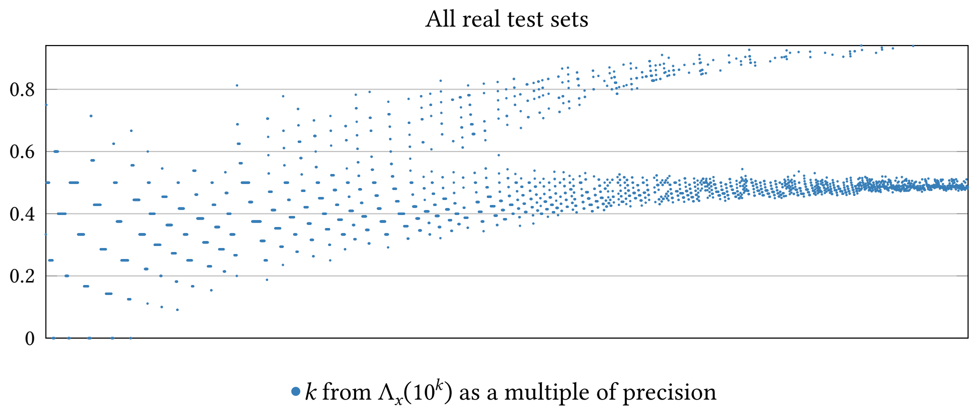


Figure 2.5: k from first successful $\Lambda_x(10^k)$ as a multiple of precision for LLL (real test sets)

2.6.2 Complex Test Sets

Precision

The precisions required for the complex test sets are shown in Figs. 2.6 and 2.7. We depict, for each test problem, theoretical minimum precision and the smallest precision needed for PSLQ. We show two bounds for the theoretical minimum precision: the smaller assumes that the number of digits of a Gaussian integer is the number of digits of the largest coefficient (i.e., real or imaginary part), and the larger assumes that the number of digits of a Gaussian integer is the total number of digits in both the real and imaginary parts. In our case, the larger is twice the smaller.

We answer the question alluded to in Section 2.4.1. Although for small coefficients (both with real and complex constants) we see the successful precision clusters around the larger of the two minimum precision options, we see that for large coefficients the needed precision hovers slightly above the lower theoretical minimum (and significantly below the larger).

We conclude that for the complex case of PSLQ, the argument referenced by Bailey and Broadhurst should be as follows. In order to find an integer relation of n numbers where the integers of that relations have real and imaginary parts of no more than d digits, then at least nd precision must be used for the computation.

The distribution of the required precisions (as a multiple of the theoretical minimum precision) is shown in Figs. 2.8 and 2.9.

We see that for small coefficient (i.e., single digit) integer relations we are almost always over the 115% mark (sometimes as much as four times higher). The extremes we see are not as high as those of the real case, however, for the most part the precisions are larger for the classical cases.

For the large coefficient test sets we see, contrary to the real cases, that the as theoretical minimum rises, the required precision also rises. Furthermore, also in contrast to the real case, se see that as the theoretical minimum increases, more of progressively more of the required precisions are above the 115% bound. We do note that, for these large coefficient cases, the requried precision is almost always bounded by 130%.

Timing

The timing required for PSLQ and LLL computations on the same test sets are shown in Fig. 2.10 for the cases with real constants, and Fig. 2.11 for the cases with complex constants. We see the same behaviour we saw in the real case, but with a more pronounced difference between LLL and PSLQ.

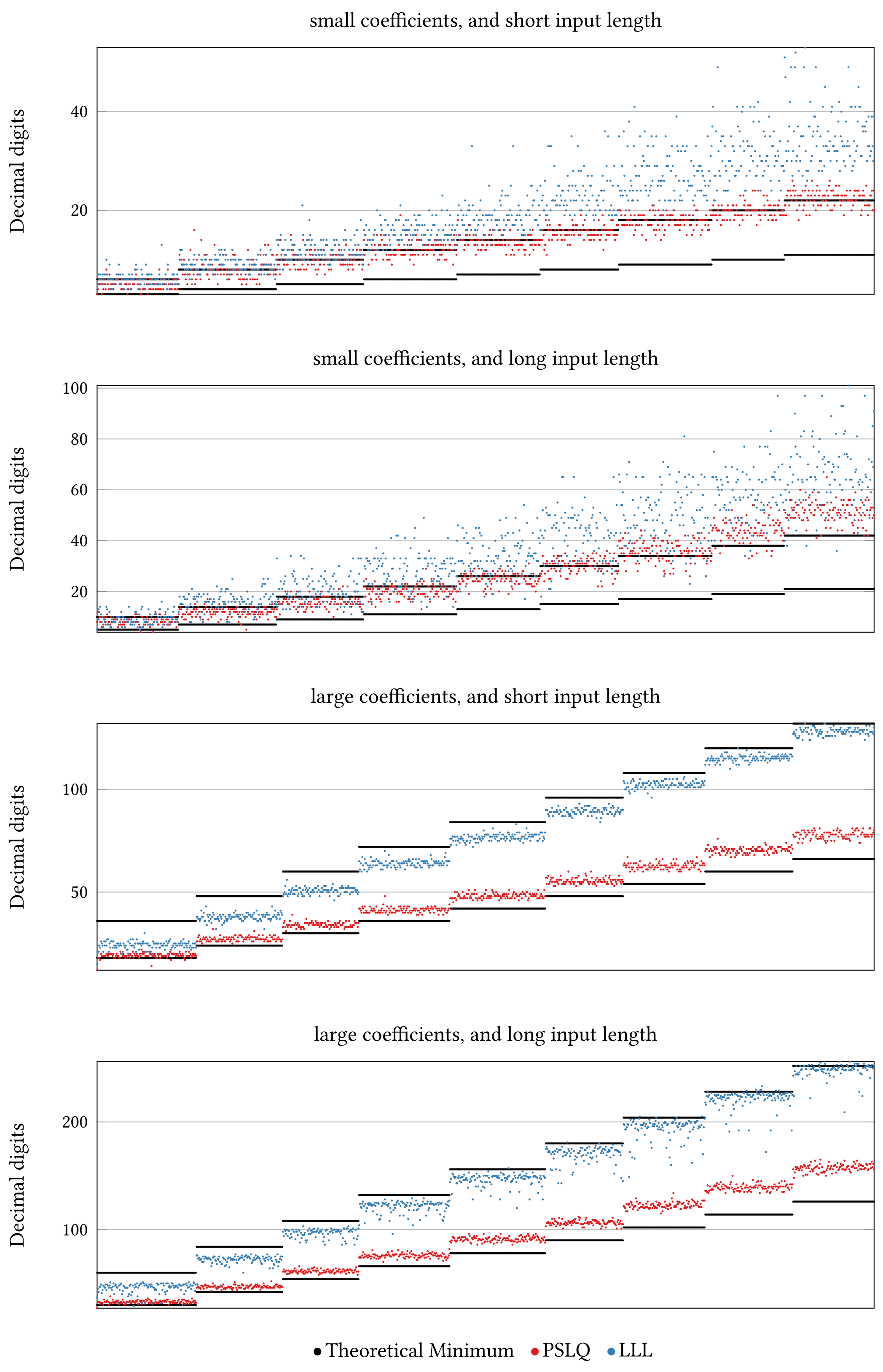


Figure 2.6: Precision required for complex test sets using real constants.

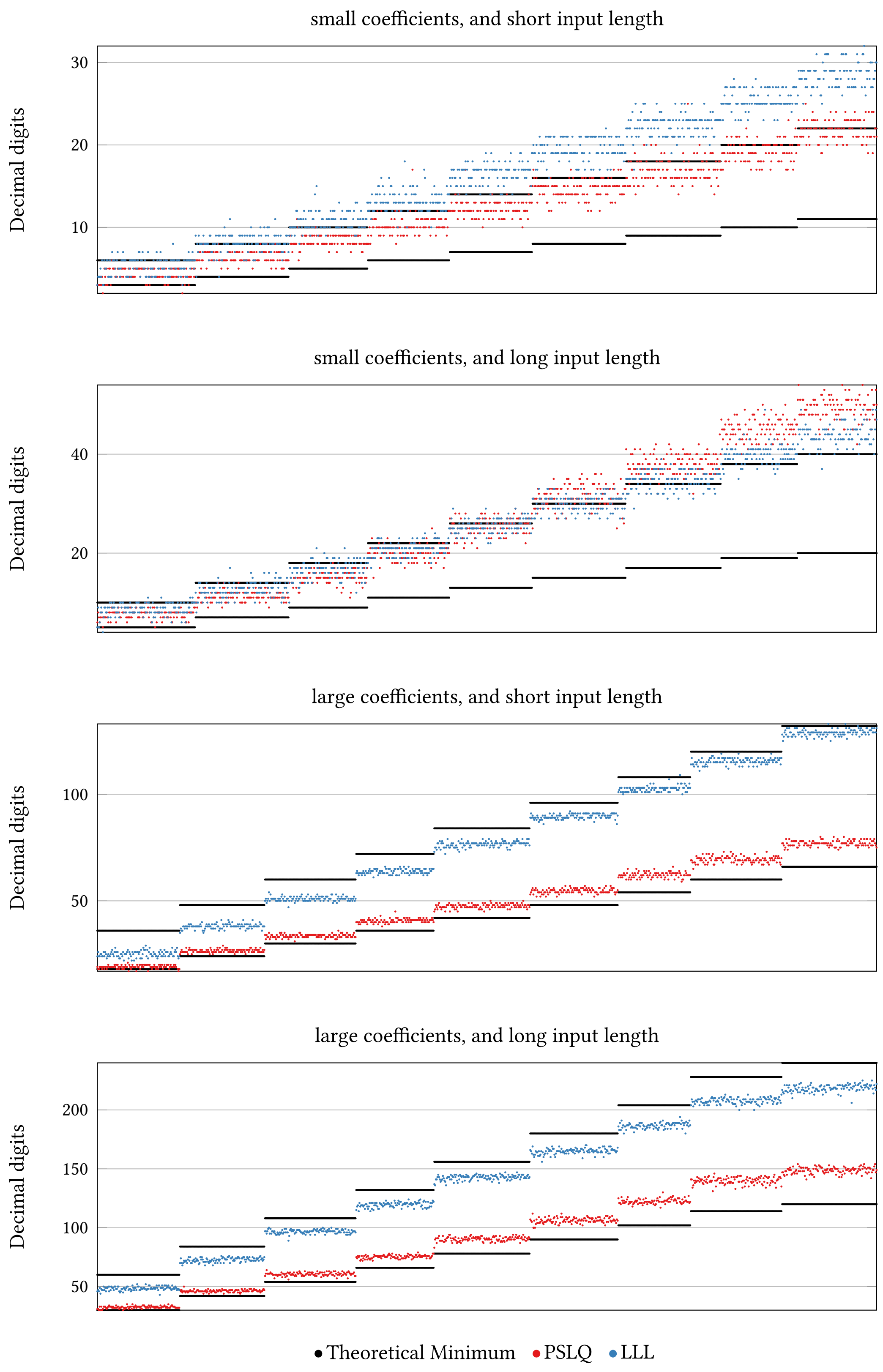


Figure 2.7: Precision required for complex test sets using complex constants.

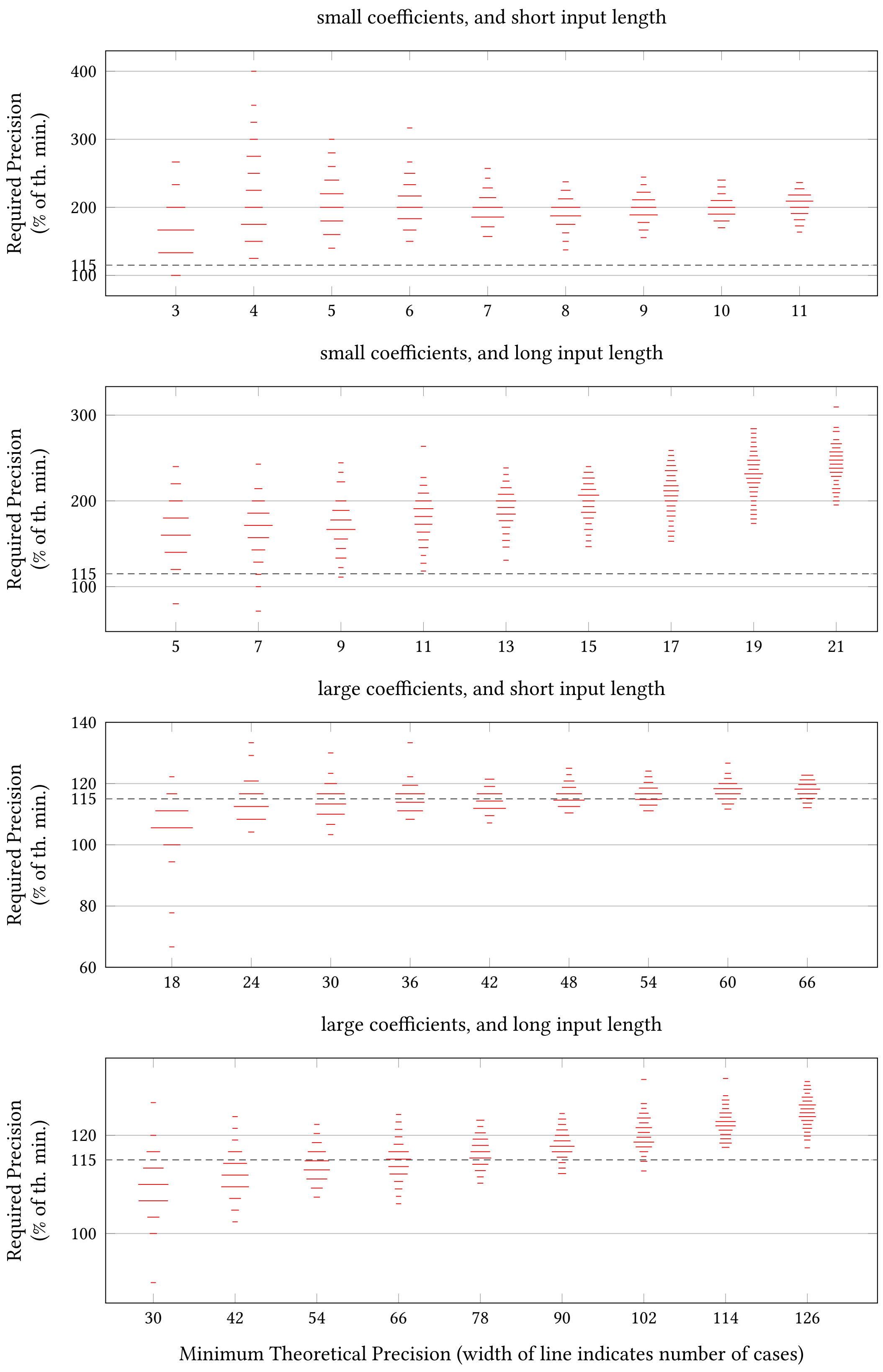


Figure 2.8: PSLQ precision needed for complex test sets using real constants (as a percentage of theoretical precision).

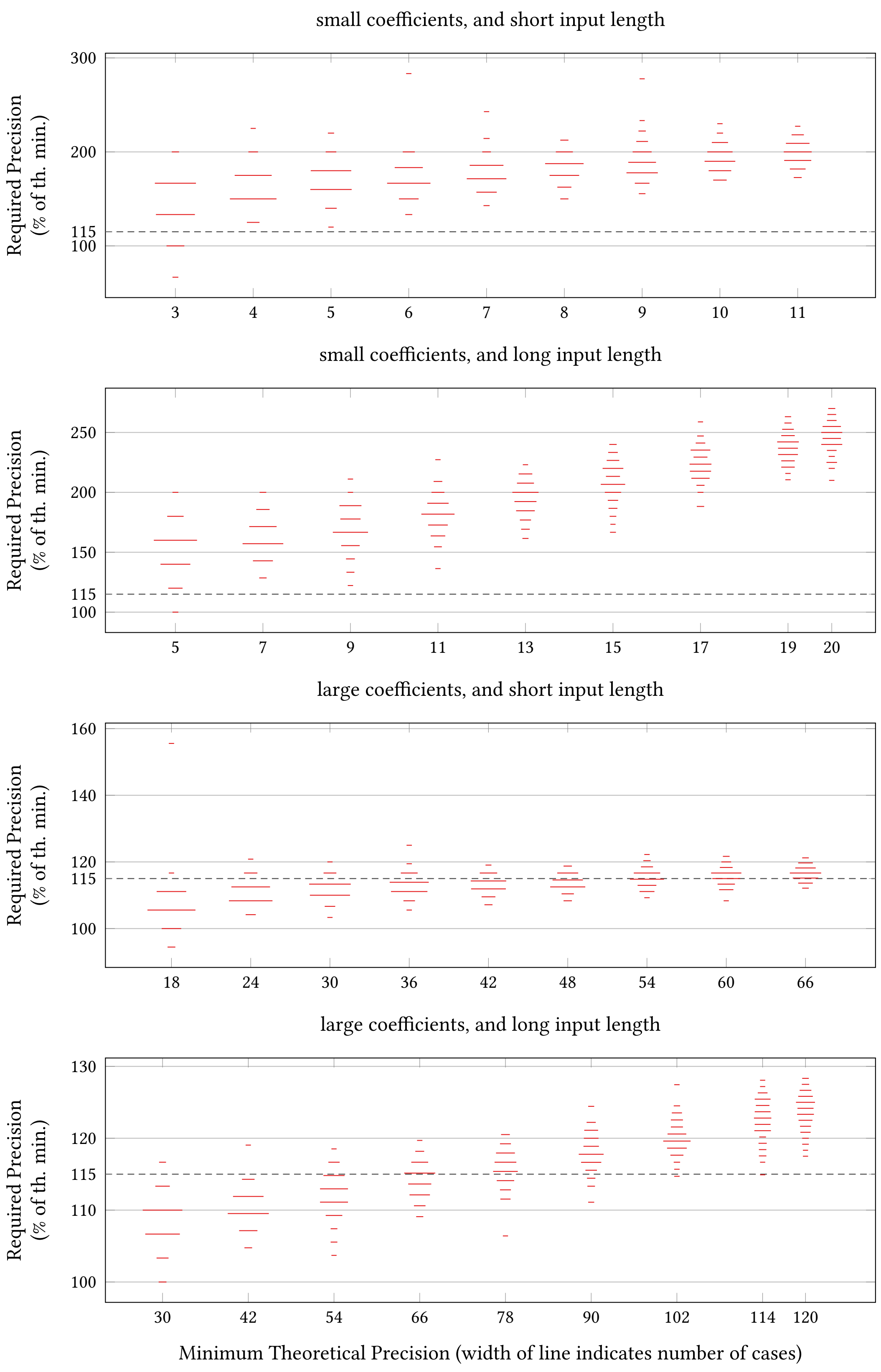


Figure 2.9: PSLQ precision needed for complex test sets using complex constants (as a percentage of theoretical precision).

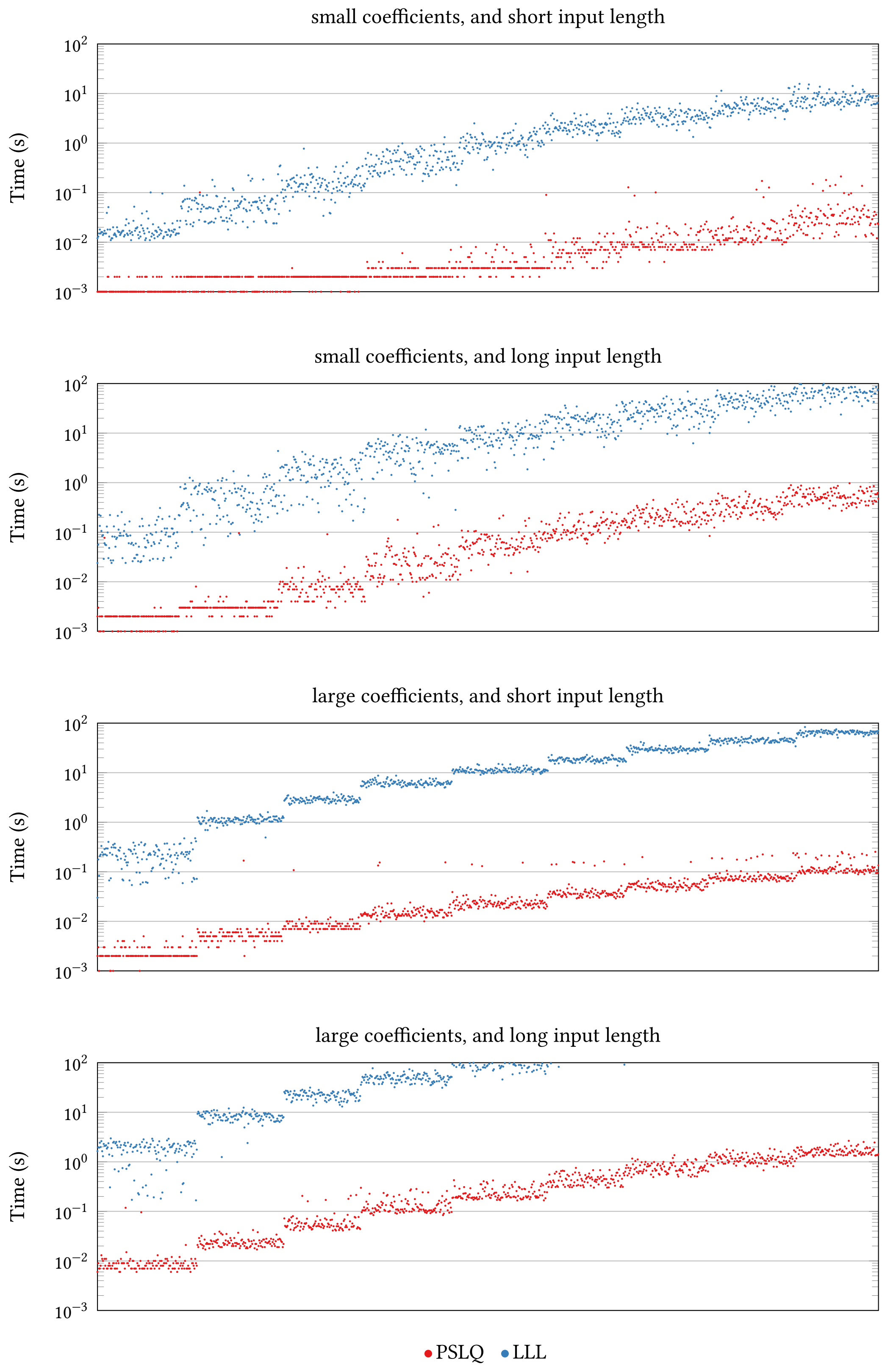


Figure 2.10: Computation time taken for complex test sets using real constants.

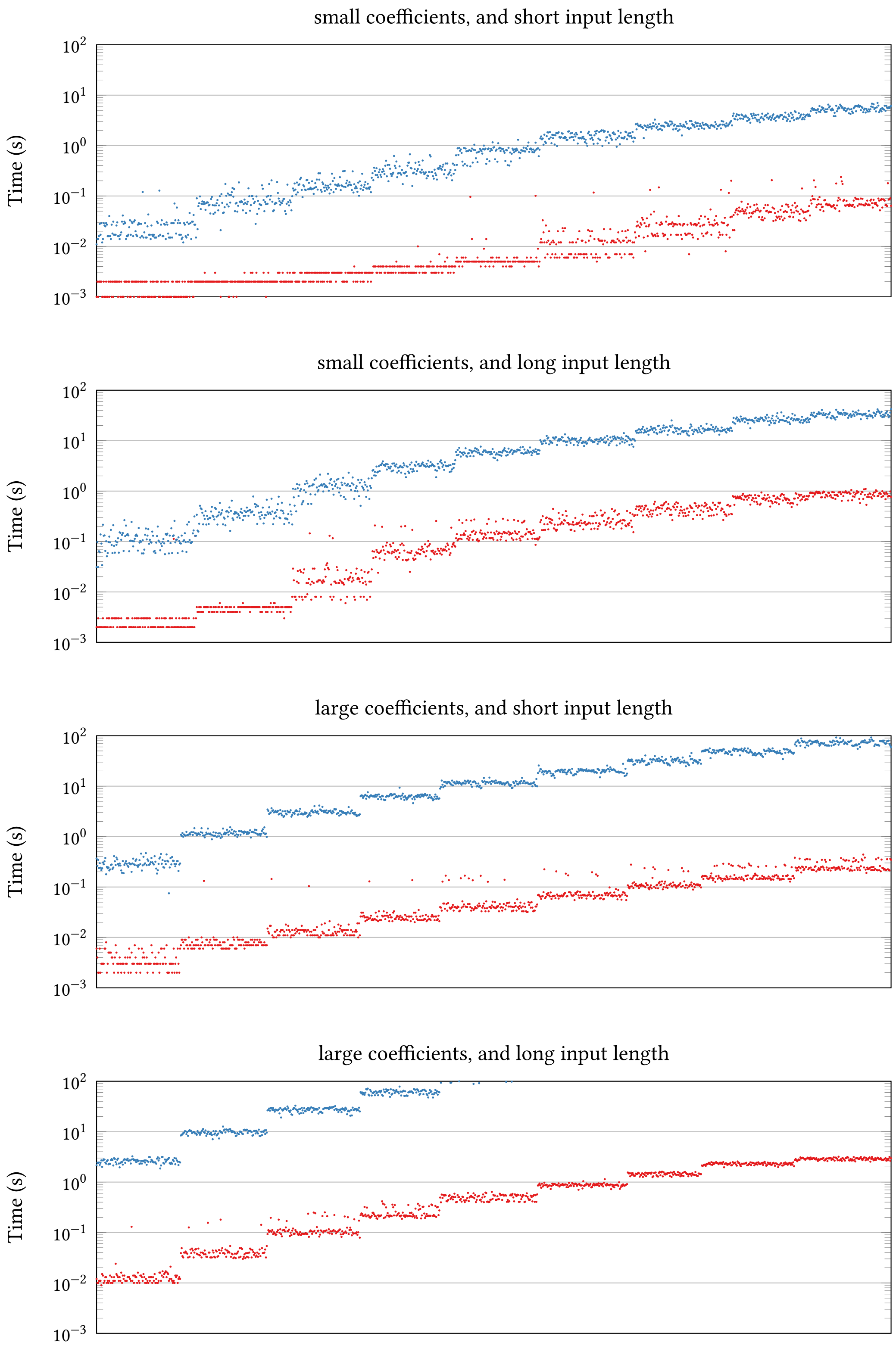


Figure 2.11: Computation time taken for complex test sets using complex constants.

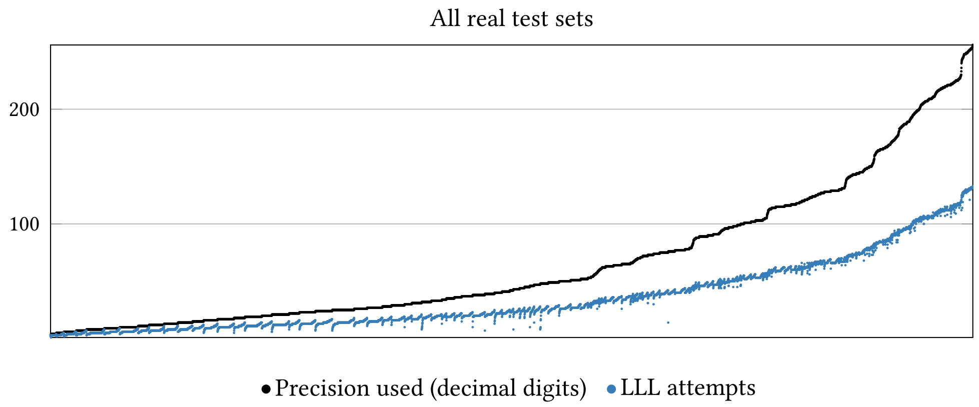


Figure 2.12: LLL computation precision compared against number of LLL attempts (real test sets)

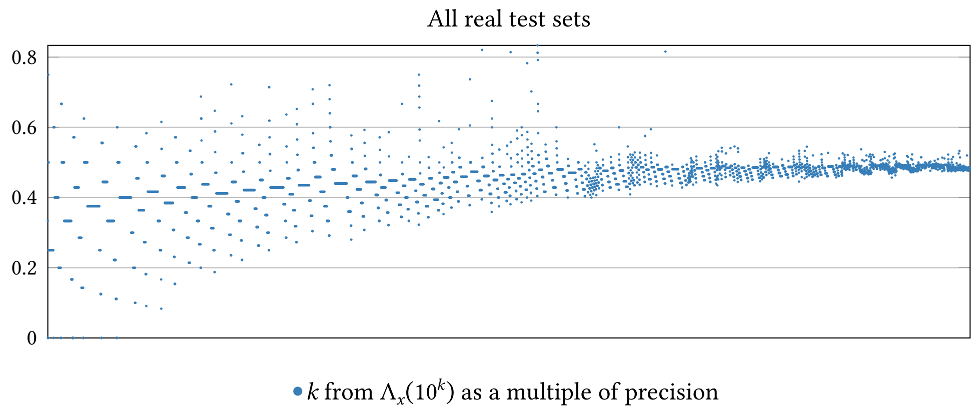


Figure 2.13: k from first successful $\Lambda_x(10^k)$ as a multiple of precision for LLL (real test sets)

LLL Attempts

Looking at the comparison of the number of attempts against the precision (in decimal digits) of the computation for all complex test sets we see the results given in Fig. 2.12. A plot of the value of k as a multiple of computation precision is given in Fig. 2.13. We see the same behaviour as in the real cases.

For test sets using real constants we count 275 out of 4000 test problems for which the number of attempts was above 100 and 1,071 where the number of attempts was 50 or above; the median number of attempts was 26. For test sets using complex constants we count 239 out of 4000 test problems for which the number of attempts was above 100 and 1,067 where the number of attempts was 50 or above; the median number of attempts was 21. In aggregate we have 514 test problems out of the total 8000 for which the number of attempts was above 100 and 2,138 cases where the number of attempts was 50 or above; the median number of attempts was 23.

3 Algebraic Integer Relations

We extend the idea of integer relations to include integers from algebraic extension fields. As a first step toward this we will focus on quadratic extension fields.

3.1 Preliminaries

We give a brief overview of algebraic number theory; enough to introduce algebraic integers and to facilitate the necessary modifications to the algorithm. We begin with an overview of the general theory, and then provide details for the specific case of quadratic number fields which we concentrate on for this section.

For a more thorough study, the reader is referred to the literature. For example, Hardy and Wright [47], Milne [60], and Stewart and Tall [68].

3.1.1 Algebraic Number Theory

We begin with the notion of a field extension.

Definition 3.1.1 (Field Extension, $\mathbb{K} : \mathbb{F}$). *Let $\mathbb{F}$ be a field. A field, $\mathbb{K}$, for which $\mathbb{F} \subseteq \mathbb{K}$ (i.e., $\mathbb{F}$ is a subfield of $\mathbb{K}$) is called a field extension (or equivalently an extension field) of $\mathbb{F}$. We denote this by $\mathbb{K} : \mathbb{F}$ when we need to unambiguously identify the field that is being extended.*

This definition allows us to consider a field as a trivial extension of itself (the case when $\mathbb{K} = \mathbb{F}$). Doing so will be useful when extending PSLQ.

We note in passing that a common notation in the literature for field extensions is $\mathbb{K}/\mathbb{F}$. The notation we have presented is the one used by Stewart and Tall [68], and is a little more consistent with Definition 3.1.2, below, so we adhere to it.

An extension field $\mathbb{K} : \mathbb{F}$ forms a vector space over the base field $\mathbb{F}$. This is not difficult to see; $\mathbb{K}$ is a field so there is already a well defined addition operation on it that satisfies the required vector space axioms. Moreover $\mathbb{F} \subset \mathbb{K}$ so scalar multiplication is similarly well defined, and the relevant axioms satisfied.

The dimension of this implicit vector space structure is of interest to us.

Definition 3.1.2 (Order of a Field Extension, $[\mathbb{K}:\mathbb{F}]$). *Let $\mathbb{K}:\mathbb{F}$ be a field extension. The order of $\mathbb{K}$ over $\mathbb{F}$, denoted $[\mathbb{K}:\mathbb{F}]$, is the dimension of $\mathbb{K}$ as a vector space over $\mathbb{F}$.*

We make use of the notion of algebraic elements in a field extension. That is, those that are a zero of a (non-trivial) polynomial with coefficients in the base field.

Definition 3.1.3 (Algebraic over a field). *Let $\mathbb{F}$ be a field, and consider an extension field $\mathbb{K}:\mathbb{F}$. An element $k \in \mathbb{K}$ is said to be algebraic over $\mathbb{F}$ if k is a zero of a polynomial with coefficients in $\mathbb{F}$. That is, if it satisfies*

$$f_n k^n + \cdots + f_1 k + f_0 = 0$$

for some $n \in \mathbb{N}$ where $f_i \in \mathbb{F}$ and $f_n \neq 0$.

Observe that, because $\mathbb{F}$ is a field, we may assume without loss of generality that the polynomial is monic. That is, the leading coefficient f_n is equal to the identity element of $\mathbb{F}$ (usually denoted as 1).

If, instead of the field $\mathbb{F}$, we consider an integral domain, D say, (i.e., a commutative ring with a multiplicative identity element, and no non-zero elements d_1, d_2 with the property that $d_1 d_2 = 0$), then we have a notion of integral elements. In this case we need to explicitly define the polynomial to be monic (unlike Definition 3.1.3 in which it could be assumed without loss of generality).

Definition 3.1.4 (Integral over a field). *Let D be an integral domain, and consider an extension field $\mathbb{K}:\mathbb{F}$ such that $D \subseteq \mathbb{K}$. An element $k \in \mathbb{K}$ is said to be integral over $\mathbb{F}$ if k is a zero of a monic polynomial with coefficients in D. That is, if it satisfies*

$$k^n + d_{n-1}k^{n-1} + \cdots + d_1 k + d_0 = 0$$

for some $n \in \mathbb{N}$ where $d_i \in D$.

We are primarily interested in field extensions, $\mathbb{K}:\mathbb{Q}$, of the rational numbers. Note that both $\mathbb{R}$ and $\mathbb{C}$ are, by Definition 3.1.3, field extensions of $\mathbb{Q}$, although there are many others. The following notation is useful for describing important classes of rational field extension.

Notation 3.1.5 ($\mathbb{Q}(\cdot)$, $\mathbb{Z}[\cdot]$). *Let $z \in \mathbb{C}$. We denote by $\mathbb{Q}(z)$ the smallest subfield of $\mathbb{C}$ that contains both $\mathbb{Q}$ and z. We denote by $\mathbb{Z}[z]$ the smallest subring of $\mathbb{C}$ that contains both $\mathbb{Z}$ and z.*

Note that it must be the case that $\mathbb{Q} \subseteq \mathbb{Q}(z)$ and $\mathbb{Z} \subseteq \mathbb{Z}[z]$. Moreover $\mathbb{Q}(z)$ is a field extension of $\mathbb{Q}$, and is non-trivial so long as $z \notin \mathbb{Q}$.

We may think of $\mathbb{Q}(z)$ as the field we get when we take $\mathbb{Q} \cup \{z\}$ and then append the minimum amount of new elements required to make a field (in a similar manner to which the real numbers may be extended to the complex numbers after the inclusion of i).

Inasmuch as we are interested in field extensions of the rational numbers, we are particularly interested in field extensions consisting of elements which are algebraic over the rationals. We call these algebraic numbers as a shorthand and note in general that any time we do not specify the base field, we mean it to be the rational numbers. Note that this definition of an algebraic number is typical in the literature, and the general case of being algebraic over a base field is sometimes omitted.

Definition 3.1.6 (Algebraic Number). *A number $z \in \mathbb{C}$ is an algebraic number (or simply algebraic) if it is algebraic over the rational numbers.*

We are further interested numbers which are integral over the rational integers. These numbers behave in relation to algebraic numbers analogously to the way rational integers behave in relation to the rational numbers. The collection of all such numbers forms a subring of the algebraic numbers [68]—and thus also of the complex numbers— and so is closed under addition and multiplication.

Definition 3.1.7 (Algebraic Integer, $\mathscr{A}$). *A number $z \in \mathbb{C}$ is an algebraic integer if it is integral over the rational integers. The ring of all algebraic integers is denoted by $\mathscr{A}$.*

The above definition, as noted by Hardy and Wright[1], is motivated by the fact that any rational number $q = (a/b) \in \mathbb{Q}$ is a zero of the polynomial $bx - a$, and any rational integer $m \in \mathbb{Z}$ is a zero of the monic polynomial $x - m$ (corresponding to the case $b = 1$ in the general rational case). It follows that rational numbers are algebraic, and that if an algebraic integer is a rational number, then it is a rational integer. It is for this reason that we introduced the term "rational integer" in Definition 2.1.2 on page 6

With our notion of rational extension fields, algebraic numbers, and algebraic integers we consider rational extension fields consisting entirely of algebraic numbers.

Definition 3.1.8 (Algebraic Extension). *An extension field, $\mathbb{K} : \mathbb{Q}$, of the rational numbers is an algebraic extension field (or simply an algebraic extension) if every $k \in \mathbb{K}$ has the property that k is an algebraic number.*

Given an algebraic extension we also consider the elements within it that are algebraic integers. The collection of all such elements forms a subring of the extension field.

[1]Although Hardy and Wright specifically say that "the definition is natural".

Definition 3.1.9 (Algebraic Integers of an Algebraic Extension, $\mathcal{O}_{\mathbb{K}}$). *Let $\mathbb{K}$ be an algebraic extension field. The ring of integers of $\mathbb{K}$, denoted $\mathcal{O}_{\mathbb{K}}$, is the intersection $\mathbb{K} \cap \mathscr{A}$ of the extension field with the ring of all algebraic integers.*

We note that all algebraic numbers, by virtue of being zeros of rational-coefficient polynomials, must be contained in the set of complex numbers, $\mathbb{C}$. As such, all algebraic extension fields are subfields of the complex numbers. We sub-divide these fields into *real* and *complex* fields in the obvious way.

Definition 3.1.10 (Real and Imaginary Algebraic Extensions). *Let $\mathbb{K}$ be an algebraic extension field. If $\mathbb{K} \subseteq \mathbb{R}$ then $\mathbb{K}$ is a real algebraic extension field, otherwise it is a complex algebraic extension field.*

For any algebraic extension, $\mathbb{K}$, there will be a number of unique embeddings (one-to-one mappings that preserve the field structure), $\sigma_i : \mathbb{K} \to \mathbb{C}$. The number of these embeddings is precisely the order, $[\mathbb{K} : \mathbb{Q}]$, of $\mathbb{K}$ as a vector field over the rationals.See Stewart and Tall for a proof of this[2].

We use these embeddings to (amongst other things) define a norm on algebraic numbers. This norm is distinct from—and so should not be confused with—the norm in the sense of a normed linear space. In particular, an algebraic norm may be negative.

Definition 3.1.11 (Norm of an Algebraic Number, Algebraic norm). *Let $\mathbb{K}$ be an algebraic extension field of order n. We define a function $N : \mathbb{K} \to \mathbb{C}$ by*

$$N(k) := \prod_{i=1}^{n} \sigma_i(k)$$

where $\sigma_1, \ldots, \sigma_n$ are the n unique embeddings of $\mathbb{K}$ into $\mathbb{C}$. We call this function the algebraic norm of $\mathbb{K}$ (or more generally an algebraic norm). The value $N(k)$ is the norm of the algebraic number k in $\mathbb{K}$ (or, when the context is clear, the norm of k).

Finally, we consider rings in which an analogue of the (rational integer) Euclidean algorithm may be performed. We call such rings *Euclidean domains.*

Definition 3.1.12 (Euclidean domain, Euclidean function). *Let D be an integral domain. If there is a function $\phi : D \setminus \{0\} \to \mathbb{N}$ that satisfies the properties*

$$\phi(d_1 d_2) \geq \phi(d_2) \quad \textit{for all } d_1, d_2 \in D \setminus \{0\}, \tag{3.1}$$

$$\textit{and for each } d_1, d_2 \in D, \textit{ there exists } q, r \in D \textit{ s.t. } d_1 = d_2\, q + r \textit{ and } r = 0 \textit{ or } \phi(r) < \phi(d_2). \tag{3.2}$$

[2]Be aware that Stewart and Tall use the more precise term *monomorphism* for these embeddings.

then D is a Euclidean domain. The function ϕ is called a Euclidean function.

There is some subtlety in the definition of a Euclidean function, and several variations of the conditions exist in the literature. Ağargün and Fletcher [1] give a thorough treatment of the different variations and their implications. For our purposes, these variations and subtleties are not relevant, and we use the given definition.

In particular we are interested in algebraic extensions whose ring of integers is such a ring. Such fields are referred to as *Euclidean*.

Definition 3.1.13 (Euclidean Field). *Let $\mathbb{K}$ be an algebraic extension field. If $\mathcal{O}_{\mathbb{K}}$ is a Euclidean domain, then $\mathbb{K}$ is a Euclidean Field*

The algebraic norm can, in some instances, act as a Euclidean function on the ring of integers of an algebraic extension. Such fields are referred to as *norm Euclidean*.

Definition 3.1.14 (Norm Euclidean Field). *Let $\mathbb{K}$ be a Euclidean field. If the field norm, N, of $\mathbb{K}$ is a euclidean function for the ring of integers, $\mathcal{O}_{\mathbb{K}}$, then $\mathbb{K}$ is a norm Euclidean field.*

3.1.2 The Specific Case of Quadratic Fields

For the purposes of this thesis we focus on quadratic extension fields. That is, fields of the form $\mathbb{Q}(\sqrt{D})$ for $D \in \mathbb{Z}$ where D is square free. The particular manifestations of the above general definitions are discussed in this subsection.

The order of a quadratic extension field is 2. Stewart and Tall [68] show that all algebraic extension fields of order 2 are of this form. Consequently, quadratic extension fields and algebraic extension fields of order 2 coincide.

Real quadratic extension fields are those for which $D \geq 0$. Complex quadratic extension fields are those for which $D < 0$.

Quadratic fields can be written as $\mathbb{Q}(\sqrt{D}) = \{q_1 + q_2\sqrt{D} \mid q_1, q_2 \in \mathbb{Q}\}$, and so consequently are of order 2. The ring of integers of such fields depend on the value of D. Specifically, $\mathcal{O}_{\mathbb{Q}(\sqrt{D})} = \mathbb{Z}[\omega] = \{m_1 + m_2\,\omega \mid m_1, m_2 \in \mathbb{Z}\}$ where

$$\omega = \begin{cases} \sqrt{D} & \text{if } D \equiv 2, 3 \pmod 4 \\ \left(1 + \sqrt{D}\right)/2 & \text{if } D \equiv 1 \pmod 4 \end{cases} \tag{3.3}$$

It should be clear that $\sqrt{D} \in \mathscr{A}$ and, more specifically, is a quadratic integer because it is a zero of the polynomial $x^2 - D$. To see the same for $(1 + \sqrt{D})/2$ when $D \equiv 1 \pmod 4$

requires us to observe that $x^2 - x - (D-1)/4$ is a monic quadratic polynomial with integer coefficients for which $(1+\sqrt{D})/2$ is a zero.

An arbitrary quadratic integer may be written in the form $\alpha + \beta\sqrt{D}$, although the α and β are not necessarily rational integers. In the case that $D \equiv 2, 3 \pmod 4$ the result is immediate because

$$\mathbb{Z}[\omega] = \mathbb{Z}[\sqrt{D}] = \left\{ m_1 + m_2\sqrt{D} \,\middle|\, m_1, m_2 \in \mathbb{Z} \right\} = \left\{ \alpha + \beta\sqrt{D} \,\middle|\, \alpha, \beta \in \mathbb{Z} \right\}$$

However in the case that $D \equiv 1 \pmod 4$, the result is not immediately obvious.

$$\begin{aligned} \mathbb{Z}[\omega] = \mathbb{Z}\left[\left(1+\sqrt{D}\right)/2\right] &= \left\{ m_1 + m_2\left(1+\sqrt{D}\right)/2 \,\middle|\, m_1, m_2 \in \mathbb{Z} \right\} \\ &= \left\{ \left(2m_1 + m_2 + m_2\sqrt{D}\right)/2 \,\middle|\, m_1, m_2 \in \mathbb{Z} \right\} \\ &= \left\{ \left(a + b\sqrt{D}\right)/2 \,\middle|\, a, b \in \mathbb{Z} \text{ and } a \equiv b \pmod 2 \right\} \\ &= \left\{ \alpha + \beta\sqrt{D} \,\middle|\, \alpha, \beta \in \tfrac{1}{2}\mathbb{Z} \text{ and } 2\alpha \equiv 2\beta \pmod 4 \right\} \end{aligned}$$

We will use both forms $\alpha + \beta\sqrt{D}$ and $m_1 + m_2\,\omega$ of quadratic integers as appropriate.

The two distinct embeddings of $\mathbb{Q}(\sqrt{D})$ into the complex numbers are

$$\left(q_1 + q_2\sqrt{D}\right) \mapsto q_1 + q_2\sqrt{D} \quad \text{and} \quad \left(q_1 + q_2\sqrt{D}\right) \mapsto q_1 - q_2\sqrt{D}$$

The algebraic norm, therefore, is $N(q_1 + q_2\sqrt{D}) = q_1^2 - Dq_2^2$. In the particular case of the quadratic integers, using the $m_1 + m_2\,\omega$ form, the norm works out to be

$$N(m_1 + m_2\,\omega) = \begin{cases} m_1^2 - Dm_2^2 & \text{if } D \equiv 2, 3 \pmod 4 \\ \left(m_1 + \frac{1}{2}m_2\right)^2 - \frac{1}{4}Dm_2^2 & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

Note that Hardy and Wright [47] define ω to be $(\sqrt{D}-1)/2$ instead of our definition. Consequently their expression for the norm in the case of $D \equiv 1 \pmod 4$ is correspondingly different than the above. These different definitions are equivalent.

There are only finitely many norm Euclidean quadratic fields [68]. They are $\mathbb{Q}(\sqrt{D})$ for

$$D \in \{-11, -7, -3, -2, -1, 2, 3, 5, 6, 7, 11, 13, 17, 19, 21, 29, 33, 37, 41, 55, 73\}$$

We note in passing that there are Euclidean quadratic fields that are not norm Euclidean fields, for example $\mathbb{Q}(\sqrt{69})$ as discovered by Clark [32]. It is known, however, that there are no other complex Euclidean quadratic fields (Stewart and Tall [68] give a prove of this). As such the complex norm Euclidean fields and the complex Euclidean fields coincide.

3.2 Generalising Integer Relations to Include Algebraic Integers

In order to extend the notion of integer relations to allow for algebraic integers, we first establish the relationship between algebraic integers, algebraic extension fields, and integer relations. We want to generalise, and thus wish to encapsulate the cases already handled by the existing theory.

A naïve strategy would be to replace $\mathbb{F}$ in Definition 2.1.1 on page 6 with an arbitrary extension field, and to replace $\mathcal{O}$ with the ring of integers of that extension field. Such a strategy will not capture the classical cases, however. Observe that neither $\mathbb{R}$ nor $\mathbb{C}$ are algebraic extension fields of $\mathbb{Q}$, although they are general extension fields of $\mathbb{Q}$ in the sense of Definition 3.1.1 As such Definition 3.1.9 does not apply and, strictly speaking, we do not have a ring of integers with which to replace $\mathcal{O}$. If we extend the idea of a ring of integers to $\mathbb{R}$ and $\mathbb{C}$ by computing $\mathbb{R} \cap \mathscr{A}$ and $\mathbb{C} \cap \mathscr{A}$ we see that these rings are not, respectively, $\mathbb{Z}$ and $\mathbb{Z}[\mathrm{i}]$ and so we still fail to capture the classical cases.

We instead generalise by introducing an intermediate extension field, as follows.

Definition 3.2.1 (Algebraic Integer Relation, $\mathbb{K}$ integer relation, $\mathcal{O}$ relation). *Let $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}\}$, and let $\mathbb{K}$ be an algebraic extension field. Denote by $\mathcal{O}$ the ring of integers of $\mathbb{K}$ (i.e., $\mathcal{O} = \mathcal{O}_{\mathbb{K}}$). For $x \in \mathbb{F}^n$ a $\mathbb{K}$-integer relation (or, equivalently, a $\mathcal{O}$ relation) of x is a vector $a \in \mathcal{O}^n$, $a \neq 0$, such that the linear combination $a_1x_1 + \cdots + a_nx_n = 0$. More generally, such relations are referred to as algebraic integer relations.*

Similarly to Definition 2.1.1, trivial integer relations occur when $x_i = 0$ for any i. So we consider only x for which $x_i \neq 0$ for all $1 \leq i \leq n$.

Observe that algebraic integer relations are indeed a generalisation of classical integer relations. When $\mathbb{F} = \mathbb{R}$ and $\mathbb{K} = \mathbb{Q}$ (thinking of $\mathbb{Q} = \mathbb{Q} : \mathbb{Q}$ as a trivial algebraic extension of itself) then an algebraic integer relation is also a classical integer relation satisfying Definition 2.1.1. The same is true for the complex case when $\mathbb{F} = \mathbb{C}$ and $\mathbb{K} = \mathbb{Q}(\sqrt{-1})$.

Definition 3.2.1 allows, in principle, for complex integer relations of real extension fields. In practice none of the techniques we discuss, below, for computing quadratic integer relations allow for finding such cases. We restrict our attention to the cases where $\mathbb{F}$ is the Archimedean norm closure of $\mathbb{K}$ (i.e., $\mathbb{F} = \mathbb{R}$ if $\mathbb{K}$ is a real algebraic extension field, and $\mathbb{F} = \mathbb{C}$ otherwise)[3].

3.3 Computing Quadratic Integer Relations

We consider three methods for computing quadratic integer relations.

[3]Note that this is a slightly stronger condition than $\mathbb{K} \subseteq \mathbb{F}$.

3.3.1 Reduction

One approach to computing algebraic integer relations is to reduce the problem to a one which we already know how to solve. Mostly, this will be reduction to a classical integer relation problem. We may then solve the problem with an existing classical integer relation finding algorithm, such as PSLQ. We will also see that LLL allows a direct reduction to a rational integer lattice problem similar to the one we used for Gaussian integer relations in Section 2.3.3 on page 14.

If we consider an arbitrary quadratic integer relation, a of x, we observe that a single term in the linear combination $\sum a_k x_k$ is of the form $(m_1 + m_2\,\omega)\,x = m_1\,x + m_2(x\,\omega)$. This suggests a method of reduction.

```
Procedure Reduction($x, D$)
Input  :  $x \in \mathbb{F}^n$   (where $\mathbb{F} = \mathbb{R}$ if $D > 1$, $\mathbb{F} = \mathbb{C}$ if $D < -1$)
          $D \in \mathbb{Z}$   ($D \notin \{0, 1, -1\}$)
Output:   $a$        (an integer relation candidate, albeit maybe not from $\mathbb{Q}(\sqrt{D})^n$)
1 $x' \leftarrow (x_1, x_1\omega, \dots, x_n, x_n\omega)$
2 $a' \leftarrow$ integer relation($x'$)        /* Compute $x'$ as a classical integer relation problem. */
3 $a \leftarrow \left(a'_1 + a'_2\,\omega, \dots, a'_{2n-1} + a'_{2n}\,\omega\right)$
4 return a
```

In essence, for a quadratic extension field $\mathbb{Q}(\sqrt{D}) \subset \mathbb{F}$, and input $x \in \mathbb{F}^n$ we exploit the fact that 1 and ω are integrally linearly dependent in $\mathbb{Z}[\omega]$ and attempt to compute the coefficients of each separately in a single double-length classical integer relation problem. The candidate relation we get back might not contain quadratic integers from $\mathbb{Q}(\sqrt{D})$, however.

In the real case (i.e., when $D > 0$ and so $\mathbb{F} = \mathbb{R}$) it is straightforward to see that each $a_k \in \mathcal{O}_{\mathbb{Q}(\sqrt{D})}$, and so the reconstructed candidate relation must always be a candidate $\mathbb{Q}(\sqrt{D})$-integer relation. Moreover, if a' truly is a classical integer relation of x' then it must be the case that a is a $\mathbb{Q}(\sqrt{D})$-integer relation for x.

In contrast, for the complex case (i.e., when $\mathbb{F} = \mathbb{C}$) each a'_k is a Gaussian integer $m_{1,k} + m_{2,k}\,i$ where $m_{1,k}, m_{2,k} \in \mathbb{Z}$. Then

$$\begin{aligned} a_k &= \left(m_{1,2k-1} + m_{2,2k-1}\,\mathrm{i}\right) + \left(m_{1,2k} + m_{2,2k}\,i\right)\omega && (3.4)\\ &= \left(m_{1,2k-1} + m_{1,2k}\,\omega\right) + \left(m_{2,2k-1} + m_{2,2k}\,\omega\right)\mathrm{i} && (3.5)\\ &= \left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right) + \left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)\mathrm{i} && (3.6)\\ &= \alpha_{1,k} + \beta_{1,k}\,\mathrm{i}\sqrt{|D|} + \alpha_{2,k}\,\mathrm{i} - \beta_{2,k}\sqrt{|D|} && (3.7) \end{aligned}$$

which is not a quadratic integer of $\mathbb{Q}(\sqrt{D})$ if $\alpha_{2,k} \neq 0$ or $\beta_{2,k} \neq 0$.

In fact, a_k is an algebraic integer for the (non-quadratic) algebraic extension field $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ (which can be considered as $\mathbb{Q}(\mathrm{i}, \sqrt{|D|})$, the smallest field containing $\mathbb{Q}$, i, and $\sqrt{|D|}$). If a' truly is a classical integer relation of x' then we will have found an algebraic integer relation for x, but not necessarily from the desired quadratic extension field.

We note in passing that this is not a completely reliable method to find $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$-integer relations. The ring of integers of $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ consists of more elements than can be reconstructed from the possible a' vectors.

Ideally, for solving the original problem of a $\mathbb{Q}(\sqrt{D})$-integer relation for x, we want the a'_k to only ever be rational integer valued. There is, however, no way to guarantee this when we are computing a complex classical integer relation problem. So instead we wish to recover a correct relation from a' or from a using some reliable method. We have not yet found a such a method which provably works in general, but we nonetheless consider two methods by which we might extract a quadratic integer relation whose integers are in the desired quadratic extension field.

Decomposition Method

The first method is to simply ignore the parts we aren't interested in. This may seem glib at first, but it is motivated by the observation that in Eq. (3.5) we have effectively decomposed a_k into two embedded quadratic integers. Extending this out we see

$$\begin{aligned}
\sum_{k=1}^{n} a_k x_k &= \sum_{k=1}^{n} \big((m_{1,2k-1} + m_{1,2k}\,\omega) + \mathrm{i}\,(m_{2,2k-1} + m_{2,2k}\,\omega)\big)\, x_k \\
&= \sum_{k=1}^{n} (m_{1,2k-1} + m_{1,2k}\,\omega)\, x_k + \mathrm{i} \sum_{k=1}^{n} (m_{2,2k-1} + m_{2,2k}\,\omega) x_k \\
&= \sum_{k=1}^{n} \xi_{1,k}\, x_k + \mathrm{i} \sum_{k=1}^{n} \xi_{2,k}\, x_k \quad \text{where} \quad \xi_{j,k} := m_{j,2k-1} + m_{j,2k}\,\omega
\end{aligned}$$

Observe that $\xi_{j,k}$ are quadratic integers in $\mathbb{Q}(\sqrt{D})$. If $\xi_1 := (\xi_{1,1}, \ldots, \xi_{1,n})$ is a quadratic integer relation for x then $\sum_{k=1}^{n} \xi_{1,k}\, x_k = 0$ and so it must be the case that $\sum_{k=1}^{n} \xi_{2,k}\, x_k = 0$ hence $\xi_2 := (\xi_{2,1}, \ldots, \xi_{2,n})$ must also be a quadratic integer relation for x.

It is not guaranteed, however, that these embedded quadratic integer vectors (ξ_1 and ξ_2) are actually quadratic integer relations for the original input, x, even if a quadratic integer relation exists. As a simple counterexample, consider the vector

$$x = \big((1 + 2\sqrt{-2})\,\pi, (3 + 5\sqrt{2} + 7\,\mathrm{i} + 11\sqrt{-2})\,\pi, \pi\big)$$

which has a $\mathbb{Q}(\sqrt{-2})$-integer relation $(-1, 0, 1 + 2\sqrt{-2})$, as well as a $\mathbb{Q}(\mathrm{i} + \sqrt{2})$-integer relation $(0, -1, 3 + 5\sqrt{2} + 7\,\mathrm{i} + 11\sqrt{-2})$. Re-writing the latter relation as per the above we get

$$\begin{aligned}
0 &= 0\,x_1 - x_2 + (3 + 5\sqrt{2} + 7\,\mathrm{i} + 11\sqrt{-2})\,x_3 \\
&= 0\,x_1 - x_2 + (3 + 11\sqrt{-3})\,x_3 + \mathrm{i}\,(7 - 5\sqrt{-2})x_3 \\
&= \Big(0\,x_1 - x_2 + (3 + 11\sqrt{-2})\,x_3\Big) + \mathrm{i}\,\Big(0\,x_1 + 0\,x_2 + (7 - 5\sqrt{-2})\,x_3\Big) \\
&= \Big((-3 - 5\sqrt{2} - 7\,\mathrm{i} - 11\sqrt{-2})\,\pi + (3 + 11\sqrt{-2})\,\pi\Big) + \mathrm{i}\,\Big((7 - 5\sqrt{-2})\,\pi\Big) \\
&= \Big((-5\sqrt{2} - 7\,\mathrm{i})\,\pi\Big) + \mathrm{i}\,\Big((7 - 5\sqrt{-2})\,\pi\Big)
\end{aligned}$$

but neither $(-5\sqrt{2}-7\,\mathrm{i})\,\pi$ nor $(7-5\sqrt{-2})\,\pi$ are equal to 0. So neither $0\,x_1 - x_2 + (3+11\sqrt{-2})\,x_3$ nor $0\,x_1 + 0\,x_2 + (7-5\sqrt{-2})\,x_3$ are equal to 0, meaning that the embedded quadratic integers do not form quadratic integer relations. That is, neither of the two embedded quadratic integer vectors $(0, -1, 3 + 11\sqrt{-2})$ nor $(0, 0, 7 - 5\sqrt{-2})$ are quadratic integer relations for x.

The $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$-integer relation in the counterexample is, in a sense, intrinsic to $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$ (in the sense that it could not be decomposed into a $\mathbb{Q}(\sqrt{D})$-integer relations). A direct modification of the above counterexample shows that is possible for a vector $x \in \mathbb{C}^n$ to have a $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$-integer relation and not a $\mathbb{Q}(\sqrt{D})$-integer relations, however the converse is not true because $\mathbb{Q}(\sqrt{D}) \subset \mathbb{Q}(\mathrm{i} + \sqrt{|D|})$. We note that if x has a $\mathbb{Q}(\sqrt{D})$-integer relation, but not any intrinsic $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$-integer relations (i.e., no relations that decompose into $\mathbb{Q}(\sqrt{D})$-integer relations) then the above method must find a quadratic integer relation.

The vectors ξ_1 and ξ_2 can be obtained directly from a' when reconstructing the vector a. If we take only the real part of each a'_k when reconstructing a then we recover ξ_1. Similarly, if we take only the imaginary part then we recover ξ_2. It is in this sense that we introduced this method by stating, somewhat disingenuously, that we ignore the parts we aren't interested in.

We need only check one of the two recovered candidates (ξ_1 or ξ_2), since if one is a quadratic integer relation then the other must be. Some small care must be taken to ensure that neither of them is the zero vector. This might happen if a' consists entirely of complex numbers with no real part, or entirely of complex numbers with no imaginary part. In such an eventuality then, unless a' is the zero vector (which should not happen), only one of ξ_1 or ξ_2 will be the zero vector and so we take the other as the candidate.

Despite the limitations, in the absence of a provably correct method, this one shows promise. Indeed, in our experimental exploration (described later in this chapter) we found this method to be remarkably reliable.

Conjugate Method

Another method we might employ is to see if the vector a is an algebraic integer multiple of a quadratic integer vector. To this end we use one of the embeddings of $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ into $\mathbb{C}$ to transform elements of $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ into elements of $\mathbb{Q}(\sqrt{D})$.

We note that $\mathbb{Q}(\mathrm{i}+\sqrt{|D|}) = \{q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D} \mid q_i \in \mathbb{Q}\}$, and so is of order 4. The four distinct embeddings into the complex numbers are

$$\begin{aligned}
\sigma_1 &: \left(q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\right) \mapsto q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\\
\sigma_2 &: \left(q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\right) \mapsto q_1 - q_2\sqrt{|D|} - q_3\,\mathrm{i} + q_4\sqrt{D}\\
\sigma_3 &: \left(q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\right) \mapsto q_1 - q_2\sqrt{|D|} + q_3\,\mathrm{i} - q_4\sqrt{D}\\
\sigma_4 &: \left(q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\right) \mapsto q_1 + q_2\sqrt{|D|} - q_3\,\mathrm{i} - q_4\sqrt{D}
\end{aligned}$$

We note in passing that, for a fixed $z \in \mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ the values of $\sigma_k(z)$ are the $\mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ conjugates of z. This is a specific case of $\mathbb{K}$ conjugates as described in Stewart and Tall [68] for the general case of an algebraic number. It is from this that the method is named.

For the purposes of extracting a quadratic integer from the Reduction procedure we are interested in the second of these embeddings, σ_2.

If $z \in \mathbb{Q}(\mathrm{i}+\sqrt{|D|})$ then $z = q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}$ for $q_i \in \mathbb{Q}$ and

$$\begin{aligned}
z \cdot \sigma_2(z) &= \left(q_1 + q_2\sqrt{|D|} + q_3\,\mathrm{i} + q_4\sqrt{D}\right) \cdot \left(q_1 - q_2\sqrt{|D|} - q_3\,\mathrm{i} + q_4\sqrt{D}\right)\\
&= \left(\left(q_1 + q_4\sqrt{D}\right) + \mathrm{i}\left(q_3 + q_2\sqrt{D}\right)\right) \cdot \left(\left(q_1 + q_4\sqrt{D}\right) - \mathrm{i}\left(q_3 + q_2\sqrt{D}\right)\right)\\
&= \left(q_1 + q_4\sqrt{D}\right)^2 + \left(q_3 + q_2\sqrt{D}\right)^2
\end{aligned}$$

and so $z \cdot \sigma_2(z) \in \mathbb{Q}(\sqrt{D})$.

We wish to apply this mapping to the elements of the reconstructed candidate quadratic relation, a from our reduction procedure.

Proposition 3.3.1. *Fix a complex quadratic extension field, $\mathbb{Q}(\sqrt{D})$, other than $\mathbb{Q}(\mathrm{i})$. Let $x \in \mathbb{C}^n$ and suppose that $a \in \mathcal{O}^n_{\mathbb{Q}(\mathrm{i}+\sqrt{|D|})}$ is the result of the the Reduction procedure applied to x and D. Then for each element a_k of a (where $1 \le k \le n$) we have $a_k\,\sigma_2(a_k) \in \mathcal{O}_{\mathbb{Q}(\sqrt{|D|})}$.*

Proof. We know from Eqs. (3.5) and (3.6) that

$$a_k = \alpha_{1,k} - \beta_{2,k}\sqrt{|D|} + \alpha_{2,k}\,\mathrm{i} + \beta_{1,k}\sqrt{|D|} = \left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right) + \left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)\mathrm{i}$$

and so

$$\sigma_2(a_k) = \alpha_{1,k} - \beta_{2,k}\sqrt{|D|} - \alpha_{2,k}\,\mathrm{i} - \beta_{1,k}\sqrt{|D|} = \left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right) - \left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)\mathrm{i}$$

and that $\alpha_{j,k} + \beta_{j,k}\sqrt{D}$ is a quadratic integer in $\mathbb{Q}(\sqrt{D})$ for each j. Recall from Section 3.1.2 that if $D \equiv 2, 3 \pmod 4$ then $\alpha_{j,k}$ and $\beta_{j,k}$ are rational integers, but that if $D \equiv 1 \pmod 4$ then $\alpha_{j,k}$ and $\beta_{j,k}$ may be rational half-integers.

In either case the quadratic integers form a ring (and so are closed under addition and multiplication). We compute

$$\begin{aligned} a_k \cdot \sigma_2(a_k) &= \left(\left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right) + \mathrm{i}\left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)\right) \cdot \left(\left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right) - \mathrm{i}\left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)\right) \\ &= \left(\alpha_{1,k} + \beta_{1,k}\sqrt{D}\right)^2 + \left(\alpha_{2,k} + \beta_{2,k}\sqrt{D}\right)^2 \end{aligned}$$

and conclude that $a_k \cdot \sigma_2(a_k) \in \mathcal{O}_{\mathbb{Q}(\sqrt{D})}$. □

We have established that the $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$-integers, a_k, that might be yielded from the reduction procedure (in the complex case that $D < 0$) can be mapped into $\mathbb{Q}(\sqrt{D})$ by multiplication with $\sigma_2(a_k)$. Some consideration must be given to how we use this fact. Simply performing this multiplication independently on each such a_k may very well break the algebraic integer relation property of the vector a. That is, even if $\sum a_k x_k = 0$ it may well be the case that $\sum a_k\,\sigma_2(a_k)\,x_k \neq 0$. It will be the case that $\sigma_2(a_j)\sum a_k x_k = 0$ for any j, however, but we note that if $a_j = 0$ then this equation is trivial.

So the approach we take is to find the smallest j such that $a_j \neq 0$ and consider the vector $\sigma_2(a_j)\,a$ as a possible candidate quadratic integer vector. There is no guarantee that the elements of $\sigma_2(a_j)\,a$ are all quadratic integers even though we know that the j^{th} one is, so we must check to make sure that all the elements are quadratic integers. We may perform this procedure for each non-zero a_j until we find a quadratic integer vector or we have exhausted all possibilities.

Much like the previous method, we cannot guarantee that this method will work. Nonetheless, in the absence of a guaranteed method, it is worth pursuing. We have found this conjugate method to be remarkably reliable in our experimental exploration.

3.3.2 LLL and Complex Quadratic Relations

In Section 2.3.2 on page 12 we discussed computation of Gaussian integer relations using LLL to compute a reduced basis for a particular lattice. We may similarly compute complex quadratic integer relations using LLL to yield a reduced basis for a (different) particular lattice. Both of these are reducing the problem of finding a quadratic integer relation to an instance of the problem of finding a LLL-reduced rational integer lattice. This is distinct from the reduction we discussed in Section 3.3.1, above, only in to what we are reducing the problem of quadratic integer relations.

Definition 3.3.2 (Complex Quadratic $\Lambda_x(N)$, $\Lambda_x(N, D)$). *Fix a complex quadratic extension field, $\mathbb{Q}(\sqrt{D})$. For a given vector $x \in \mathbb{C}^n$ we denote by $\Lambda_x(N, D)$ (or, when the quadratic extension field is clear, simply $\Lambda_x(N)$) the lattice spanned by the $(2n+2)$-dimensional vectors:*

$$\{(1, 0, \dots, 0, N\Re x_1, 0, \dots, 0, N\Im x_n), \dots, (0, \dots, 0, 1, N\Re x_n, 0, \cdots, 0, N\Im x_n)\}$$
$$\cup$$
$$\{(0, \dots, 0, N\mathfrak{W}_1 x_1, 1, 0, \dots, 0, N\mathfrak{W}_2 x_1), \dots, (0, \dots, 0, N\mathfrak{W}_1 x_n, 0, \cdots, 0, 1, N\mathfrak{W}_2 x_n)\}$$

where $N \in \mathbb{R}$ and

$$\mathfrak{W}_1 x_k = \begin{cases} -\sqrt{|D|}\,\Im x_k & \text{if } D \equiv 2, 3 \pmod 4 \\ \left(\Re x_k - \sqrt{|D|}\,\Im x_k\right)/2 & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

$$\mathfrak{W}_2 x_k = \begin{cases} \sqrt{|D|}\,\Re x_k & \text{if } D \equiv 2, 3 \pmod 4 \\ \left(\Im x_k + \sqrt{|D|}\,\Re x_k\right)/2 & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

Observe that $\Lambda_x(N, -1)$ coincides with $\Lambda_x(N)$ from Definition 2.3.6 on page 14. In fact, the entire method discussed in Section 2.3.3 is a special case of the one presented here.

The $\Lambda_x(N, D)$ lattice is motivated by a general technique for using LLL to compute reduced bases of complex quadratic integer lattices as reported by Lyu, Porter, and Ling [57]. Given a matrix representation[4], M, for a lattice we construct a new lattice with matrix representation M' by

$$M' = \begin{bmatrix} \Re M & \Im M \\ \mathfrak{W}_1 M & \mathfrak{W}_2 M \end{bmatrix}$$

Unlike in the complex classical case, we have not obtained $\Lambda_x(N, D)$ by applying this construction to the matrix representation of the real classical $\Lambda_x(N)$ (Definition 2.3.3 on page 12). We have instead applied the operators $\mathfrak{W}_j$ only to the x_k elements of the basis vectors. The reason for this is to simplify the extraction of the embedded quadratic integer relations at the end of the computation. We will see that this technique correctly finds quadratic integer relations.

We note in passing that Lyu, Porter, and Ling also present a modified LLL that directly computes reduced lattice bases of complex quadratic lattices. Their method is only appropriate for the complex quadratic norm Euclidean fields (i.e., for $D \in \{-1, -2, -3, -7, -11\}$). We will see a similar limitation in our direct modification of PSLQ in Section 3.3.3.

[4]Recall that we are using the unusual convention whereby the row vectors of the matrix are the lattice basis vectors.

As was the case with the Gaussian integer LLL discussed in Section 2.3.3, we note that Maple's implementation of the LLL algorithm does not support this modification. Furthermore, the author discovered this modification sufficiently late as to lack the time to implement it.

Returning to the matter of using $\Lambda_x(N, D)$ to find complex quadratic integer relations, we note that a vector in the lattice will have the form $(l_1, \dots, l_n, d_1, m_1, \dots, m_n, d_2)$, and the elements d_1 and d_2 are discriminants. These discriminants, as we show in Lemma 3.3.3, are the real and imaginary parts, respectively, of $N \sum_{k=1}^{n} (l_k + m_k\,\omega)\, x_k)$. Consequently if *both* of discriminants are 0 then the lattice vector encodes a quadratic integer relation, a say, of x where $a = (m_1 + l_1\,\omega, \dots, m_n + l_n\,\omega)$.

We establish a similar bijective correspondence to the one established for the complex classical case in Lemma 2.3.7 on page 15 by stating and proving the analogous lemma.

Lemma 3.3.3. *Fix a complex quadratic extension field, $\mathbb{Q}(\sqrt{D})$. Let $x = (x_1, \dots, x_n) \in \mathbb{C}^n$, and consider the lattice $\Lambda_x(N, D)$. For every quadratic integer relation, $a = (l_1 + m_1\,\omega, \dots, l_n + m_n\,\omega) \in \mathcal{O}^n_{\mathbb{Q}(\sqrt{D})}$, of x there is a vector*

$$(l_1, \dots, l_n, 0, m_1, \dots, m_n, 0) \in \Lambda_x(N)$$

Moreover, for every lattice element of that form there is a corresponding quadratic integer relation of x.

Proof. To simplify the cases we let

$$\lambda_k := \begin{cases} l_k & \text{if } D \equiv 2, 3 \pmod 4 \\ l_k + \frac{m_k}{2} & \text{if } D \equiv 1 \pmod 4 \end{cases} \qquad \mu_k := \begin{cases} m_k & \text{if } D \equiv 2, 3 \pmod 4 \\ \frac{m_k}{2} & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

and observe that

$$\begin{aligned} (l_k + m_k\,\omega)\, x_k &= (l_k + m_k\,\omega)\,(\Re x_k + \Im x_k\,\mathrm{i}) \\ &= \begin{cases} \left(l_k + m_k\,\mathrm{i}\sqrt{|D|}\right)(\Re x_k + \Im x_k\,\mathrm{i}) & \text{if } D \equiv 2, 3 \pmod 4 \\ \left(l_k + m_k\,\frac{1+\mathrm{i}\sqrt{|D|}}{2}\right)(\Re x_k + \Im x_k\,\mathrm{i}) & \text{if } D \equiv 1 \pmod 4 \end{cases} \\ &= \left(\lambda_k + \mu_k\,\mathrm{i}\sqrt{|D|}\right)(\Re x_k + \Im x_k\,\mathrm{i}) \\ &= \left(\lambda_k\,\Re x_k - \mu_k\sqrt{|D|}\,\Im x_k\right) + \mathrm{i}\left(\lambda_k\,\Im x_k + \mu_k\sqrt{|D|}\,\Re x_k\right) \end{aligned}$$

An arbitrary element of $\Lambda_x(N, D)$ is of the form $(l_1, \dots, l_n, d_1, m_1, \dots, m_n, d_2)$ where

$$\begin{aligned}
d_1 &= \sum_{k=1}^{n} l_k\, N\mathfrak{R}x_k \ + \sum_{k=1}^{n} m_k N\mathfrak{W}_1 x_k \\
&= N\sum_{k=1}^{n}(l_k\,\mathfrak{R}x_k + m_k\mathfrak{W}_1 x_k) \\
&= \begin{cases} N\sum_{k=1}^{n}\left(l_k\,\mathfrak{R}x_k - m_k\sqrt{|D|}\,\mathfrak{I}x_k\right) & \text{if } D \equiv 2, 3 \pmod 4 \\ N\sum_{k=1}^{n}\left(l_k\,\mathfrak{R}x_k + m_k(\mathfrak{R}x_k - \sqrt{|D|}\,\mathfrak{I}x_k)/2\right) & \text{if } D \equiv 1 \pmod 4 \end{cases} \\
&= N\sum_{k=1}^{n}\left(\lambda_k\,\mathfrak{R}x_k - \mu_k\sqrt{|D|}\,\mathfrak{I}x_k\right) \\
&= N\sum_{k=1}^{n}\mathfrak{R}\big((l_k + m_k\,\omega)\,x_k\big) = N\mathfrak{R}\left(\sum_{k=1}^{n}(l_k + m_k\,\omega)\,x_k\right)
\end{aligned}$$

and

$$\begin{aligned}
d_2 &= \sum_{k=1}^{n} l_k\, N\mathfrak{R}x_k \ + \sum_{k=1}^{n} m_k N\mathfrak{W}_2 x_k \\
&= N\sum_{k=1}^{n}(l_k\,\mathfrak{R}x_k + m_k\mathfrak{W}_2 x_k) \\
&= \begin{cases} N\sum_{k=1}^{n}\left(l_k\,\mathfrak{R}x_k + m_k\sqrt{|D|}\,\mathfrak{I}x_k\right) & \text{if } D \equiv 2, 3 \pmod 4 \\ N\sum_{k=1}^{n}\left(l_k\,\mathfrak{R}x_k + m_k(\mathfrak{I}x_k + \sqrt{|D|}\,\mathfrak{R}x_k)/2\right) & \text{if } D \equiv 1 \pmod 4 \end{cases} \\
&= N\sum_{k=1}^{n}\left(\lambda_k\,\mathfrak{I}x_k + \mu_k\sqrt{|D|}\,\mathfrak{R}x_k\right) \\
&= N\sum_{k=1}^{n}\mathfrak{I}\big((l_k + m_k\,\omega)\,x_k\big) = N\mathfrak{I}\left(\sum_{k=1}^{n}(l_k + m_k\,\omega)\,x_k\right)
\end{aligned}$$

If a is a $\mathbb{Q}(\sqrt{D})$-integer relation for x, then $a = (l_1 + m_1\,\omega + \cdots + l_n + m_n\,\omega)$ and $\sum_{k=1}^{n} a_k\, x_k = 0$. So $d_1 = d_2 = 0$ which implies $(l_1, \dots, l_n, 0, m_1, \dots, m_n, 0) \in \Lambda_x(N, D)$. Conversely, if $(l_1, \dots, l_n, 0, m_1, \dots, m_n, 0) \in \Lambda_x(N, D)$, then

$$N\mathfrak{R}\left(\sum_{k=1}^{n}(l_k + m_k\,\omega)\,x_k\right) = N\mathfrak{I}\left(\sum_{k=1}^{n}(l_k + m_k\,\omega)\,x_k\right) = 0$$

and $\sum_{k=1}^{n}(l_k + m_k\,\omega) = d_1 + d_2\,\mathrm{i} = 0$, hence $a = (l_1 + m_1\,\omega + \cdots + l_n + m_n\,\omega)$ is a $\mathbb{Q}(\sqrt{D})$-integer relation for x □

Armed with this lemma, we can state and prove the analogous proposition to Propositions 2.3.4 and 2.3.8 on page 12 and on page 16) to conclude that performing LLL on the lattice above will find a quadratic integer relation so long as one exists and N is sufficiently large.

Proposition 3.3.4. *Let $x = (x_1, \dots, x_n) \in \mathbb{C}^n$, $D < 0$ be an integer, and suppose that a $\mathbb{Q}(\sqrt{D})$-integer relation of x exists. Suppose $\{b_1, \dots, b_{2n}\}$ is an LLL reduced basis for $\Lambda_x(N, D)$. Then so long as N was sufficiently large, b_1 must be of the form $(b_{11}, \dots, b_{1n}, 0, b_{1(n+2)}, \dots, b_{1(2n+1)}, 0)$ and $(b_{11} + b_{1(n+2)}\,\omega, \dots, b_{1n} + b_{1(2n+1)}\,\omega)$ is a Gaussian integer relation of x.*

Proof. Let $a = (l_1 + m_1\,\omega, \dots, l_n + m_n\,\omega)$ be a smallest $\mathbb{Q}(\sqrt{D})$-integer relation for x. Consider the set of vectors $\left\{y \in \mathcal{O}^n_{\mathbb{Q}(\sqrt{D})} \ : \ \|y\| < 2^{n/2}\|a\| \text{ and } \sum_{k=1}^n y_k\,x_k \neq 0\right\}$, and observe that the set is finite and non-empty. We choose from this set a vector y such that $|\sum_{i=1}^n y_i x_i|$ is minimal, and choose N such that $N|\sum_{i=1}^n y_i x_i| > 2^{n/2}\|a\|$.

Let $\lambda \in \Lambda_x(N, D)$; it must have the form

$$\left(l_1, \dots, l_n, N\Re\left(\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right), m_1, \dots, m_n, N\Im\left(\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right)\right) \quad \text{where} \quad l_k, m_k \in \mathbb{Z}$$

Now suppose that $\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k \neq 0$ (i.e., $(l_1 + m_1\,\omega, \dots, l_n + m_n\,\omega)$ is not a $\mathbb{Q}(\sqrt{D})$-integer relation for x). Then

$$\begin{aligned}
\|\lambda\| &= \sqrt{l_1^2 + \dots + l_n^2 + \left(N\Re\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right)^2 + m_1^2 + \dots + m_n^2 + \left(N\Im\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right)^2} \\
&> \sqrt{N^2\,\Re\left(\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right)^2 + N^2\,\Im\left(\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right)^2} \\
&= N\left|\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k\right| \geq N\left|\sum_{k=1}^n y_k\,x_k\right| > 2^{n/2}\|a\| > 2^{(n-1)/2}\|a\|
\end{aligned}$$

We know that a is a $\mathbb{Q}(\sqrt{D})$-integer relation for x, so $\sum_{k=1}^n (l_k + m_k\,\omega)\,x_k = 0$, and hence from Lemma 3.3.3 we know that $\lambda_a := (l_1, \dots, l_n, 0, m_1, \dots, m_n, 0) \in \Lambda_x(N, D)$. Furthermore observe that

$$\begin{aligned}
\|a\| &= \sqrt{|l_1 + m_1\,\omega|^2 + \dots + |l_n + m_n\,\omega|^2} \\
&= \begin{cases} \sqrt{l_1^2 + \dots + l_n^2 + |D|\left(m_1^2 + \dots + m_n^2\right)} & \text{if } D \equiv 2, 3 \pmod 4 \\ \sqrt{\left(l_1^2 + \dots + l_n^2\right) + \frac{|D|+1}{4}\left(m_1^2 + \dots + m_n^2\right) + m_1\,l_1 + \dots + l_n\,m_n} & \text{if } D \equiv 1 \pmod 4 \end{cases} \\
&\geq \sqrt{l_1^2 + \dots + l_n^2 + 0 + m_1^2 + \dots + m_n^2 + 0} \\
&= \|(l_1, \dots, l_n, 0, m_1, \dots, m_n, 0)\| = \|\lambda_a\|
\end{aligned}$$

So it must be the case that $\|\lambda\|$ is greater than that $2^{(n-1)/2}$ times the norm of a vector in $\Lambda_x(N)$ (specifically, the vector λ_a). Consequently, λ cannot be the reduced basis vector b_1 by Theorem 2.3.2 on page 12. The result follows. □

Note that, as suggested above, Proposition 2.3.8 on page 16 is a special case of the above proposition, corresponding to $D = -1$.

As was the case for the real classical case, we note that the a from the beginning of the proof is not necessarily the integer relation embedded in the reduced basis vector b_1. Additionally, there may well be multiple vectors in the reduced basis with 0 discriminants, all of which embed integer relations (or, in the case of numeric computation, all of which may be considered candidate integer relations).

3.3.3 Algebraic PSLQ

A more direct approach to computing algebraic integer relations is desirable. To this end we modify the PSLQ algorithm to compute them directly. We call this modified algorithm *Algebraic PSLQ*, or APSLQ.

We observe that the reducing matrix is the source of integers in the algorithm. The reducing matrix, in turn, relies on the nearest integer function. The theorems bounding the number of iterations needed to find an integer relation rely only on the τ, ρ, and γ parameters, the latter of which is arbitrarily chosen and the others of which are determined by the properties of the integer lattice.

In order to utilise as much of the existing theory as possible we replace the nearest integer function in the computation of the reducing matrix with a nearest algebraic integer function. Additionally, we require the specification of the intermediate quadratic extension field as input to the algorithm. The algorithm remains otherwise unmodified.

This immediately causes a problem. In the case of a real quadratic extension field (when $D > 0$) the algebraic integers are dense in $\mathbb{R}$. This leaves us without a well defined nearest integer and hence no integer lattice. We put this case aside pending further algorithmic modifications and restrict our attention to complex quadratic extension fields $D < 0$.

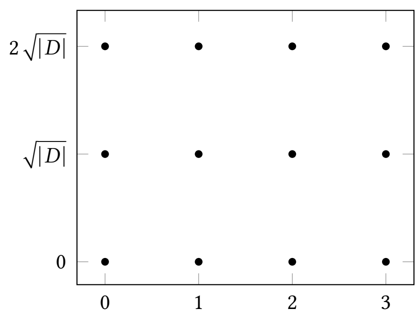
In order to calculate the nearest integer for an arbitrary $z \in \mathbb{C}$, and in in order to verify that conditions (2.1) to (2.3) on page 8 hold, we appeal to the geometric properties of the integer lattices. Figure 3.1 depicts the geometry of these lattices.

We start with the nearest integer question. For $z \in \mathbb{C}$ we write $z = \mu_1 + \mu_2\,\omega$ (noting that the coefficients μ_1, μ_2 are real numbers) and calculate the nearest integer in the $m_1 + m_2\,\omega$ form. When $D \equiv 2, 3 \pmod 4$ we have

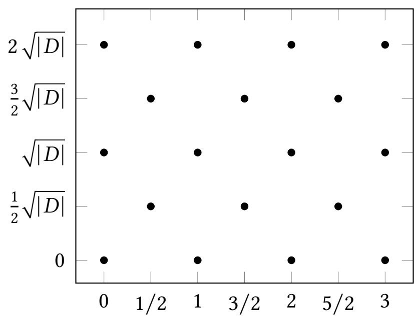
$$\lceil z \rfloor_{\mathcal{O}_{\mathbb{Q}(\sqrt{D})}} = \lceil \mu_1 \rfloor_{\mathbb{Z}} + \lceil \mu_2 \rfloor_{\mathbb{Z}}\,\omega$$

and when $D \equiv 1 \pmod 4$ we have two candidates

$$\lceil z \rfloor_{\mathcal{O}_{\mathbb{Q}(\sqrt{D})}} = \left\lceil \Re z - \frac{\lfloor \mu_2 \rfloor}{2} \right\rfloor_{\mathbb{Z}} + \lfloor \mu_2 \rfloor\,\omega \quad \text{or} \quad \lceil z \rfloor_{\mathcal{O}_{\mathbb{Q}(\sqrt{D})}} = \left\lceil \Re z - \frac{\lceil \mu_2 \rceil}{2} \right\rfloor_{\mathbb{Z}} + \lceil \mu_2 \rceil\,\omega$$



(a) $D \equiv 2, 3 \pmod 4$



(b) $D \equiv 1 \pmod 4$

Figure 3.1: Geometric Depiction of Quadratic Integer Lattices

from which we choose the one closest to z. Note the ceiling and floor functions applied to μ_2 in the different candidates. We also note that the above corrects a mistake in Skerritt and Vrbik [67] for the $D \equiv 1 \pmod 4$ case. We prove the correctness of these expression in Proposition 3.3.5, below.

Proposition 3.3.5. *Fix a complex quadratic extension field $\mathbb{Q}(\sqrt{D})$. Let $z \in \mathbb{C}$ and write $z = \mu_1 + \mu_2\,\omega$. The nearest quadratic integer to z is given by $\lceil z \rfloor_{\mathcal{O}_{\mathbb{Q}(\sqrt{D})}} = \arg\min_{\xi\in\Xi} |z - \xi|$ where*

$$\Xi = \begin{cases} \{\lceil \mu_1 \rfloor_{\mathbb{Z}} + \lceil \mu_2 \rfloor_{\mathbb{Z}}\,\omega\} & \textit{if } D \equiv 2, 3 \pmod 4 \\ \left\{ \left\lceil \Re z - \frac{f(\mu_2)}{2} \right\rfloor_{\mathbb{Z}} + f(\mu_2)\,\omega \,\middle|\, f = \lfloor \cdot \rfloor \textit{ or } f = \lceil \cdot \rceil \right\} & \textit{if } D \equiv 1 \pmod 4 \end{cases}$$

Proof. We consider each case separately.

Case 1 ($D \equiv 2, 3 \pmod 4$) Write $z = \Re z + \mathrm{i}\,\Im z = \mu_1 + \mu_2\,\omega$, and calculate that $\mu_1 = \Re z$ and $\mu_2 = \Im z/\sqrt{|D|}$. Observe that μ_1 depends only on $\Re z$ and μ_2 depends only on $\Im z$; this corresponds to the square lattice we see in Fig. 3.1. Consequently the coefficient of ω for the nearest integer must be $\lceil \mu_2 \rfloor_{\mathbb{Z}}$ and the constant term must be $\lceil \mu_1 \rfloor_{\mathbb{Z}}$.In other words $\lceil z \rfloor_{\mathcal{O}_{\mathbb{Q}(\sqrt{D})}} = \lceil \mu_1 \rfloor_{\mathbb{Z}} + \lceil \mu_2 \rfloor_{\mathbb{Z}}\,\omega$.[5]

Case 2 ($D \equiv 1 \pmod 4$) Write $z = \Re z + \mathrm{i}\,\Im z = \mu_1 + \mu_2\,\omega$, yielding the simultaneous equations $\Re z = (2\,\mu_1 + \mu_2)/2$ and $\Im z = \left(\mu_2\sqrt{|D|}\right)/2$ from which we calculate that

$$\mu_1 = \Re z - \frac{\mu_2}{2} \qquad\qquad \mu_2 = \frac{2\,\Im z}{\sqrt{|D|}}$$

Let $\alpha = \lfloor 2\,\Re z \rfloor/2$ and observe that $\alpha \le \Re z < \alpha + 1/2$ and that exactly one of α or $\alpha + 1/2$ is a rational integer. Let $\beta = \lfloor \mu_2 \rfloor/2$ and observe that $\beta\sqrt{|D|} \le \Im z < (\beta + 1/2)\sqrt{|D|}$ and that

[5]In the case that there are multiple nearest rational integers to either μ_1 or μ_2 then any of them will produce a nearest quadratic integer to z.

exactly one of β and $\beta+1/2$ is a rational integer. Note that $(\beta+1/2)\sqrt{|D|} = (\lceil \mu_2 \rceil/2)\sqrt{|D|}$ so long as $\mu_2 \notin \mathbb{Z}$.

By the geometry of the integer lattice, exactly one of $\alpha + \mathrm{i}\,\beta\sqrt{|D|}$ and $(\alpha+1/2) + \mathrm{i}\,\beta\sqrt{|D|}$ is a quadratic integer. Similarly, exactly one of and $\alpha + \mathrm{i}\,(\beta+1/2)\sqrt{|D|}$ and $(\alpha+1/2) + \mathrm{i}\,(\beta+1/2)\sqrt{|D|}$ is a quadratic integer. Call these quadratic integers ξ_1 and ξ_2 respectively. There are no other quadratic integers within these bounds.

Consider complex numbers with imaginary part $\beta\sqrt{|D|}$. All such numbers must be able to be written in the form $\mu_1' + \lfloor \mu_2 \rfloor \omega$ for some $\mu_1' \in \mathbb{R}$. In particular it must be that $\mu_1' = \Re - \lfloor \mu_2 \rfloor/2$ (where $\Re$ is used to denote the real part of the complex number in question). Similarly complex numbers of with imaginary part $(\beta+1/2)\sqrt{|D|}$ but be able to be written as $(\Re - \lceil \mu_2 \rceil/2) + \lceil \mu_2 \rceil \omega$ so long as $\mu_2 \notin \mathbb{Z}$.

Project z onto the line $t + \mathrm{i}\,\beta\sqrt{|D|}$ $(t \in \mathbb{R})$ to get

$$z_1' = \Re z + \mathrm{i}\,b\sqrt{|D|} = \left(\Re z - \tfrac{\lfloor \mu_2 \rfloor}{2}\right) + \lfloor \mu_2 \rfloor \omega$$

and note that its real part remains bounded by $\alpha \le \Re z' < \alpha + 1/2$. We know that $\Im z' = \Im \xi_1$ so $|z' - \xi_1| = |\Re z' - \Re \xi_1| < 1/2$. We also know that $\xi_1 = m_1 + \lfloor \mu_2 \rfloor \omega$ for some $m_1 \in \mathbb{Z}$. So

$$\begin{aligned} 1/2 > |z' - \xi_1| &= \left|\left(\Re z - \tfrac{\lfloor \mu_2 \rfloor}{2}\right) + \lfloor \mu_2 \rfloor \omega - (m_1 + \lfloor \mu_2 \rfloor \omega)\right| \\ &= \left|\left(\Re z - \tfrac{\lfloor \mu_2 \rfloor}{2}\right) - m_1\right| \end{aligned}$$

and because $m_1 \in \mathbb{Z}$ it must be the case that $m_1 = \lceil \Re z - \lfloor \mu_2 \rfloor/2 \rfloor_{\mathbb{Z}}$. So we have that $\xi_1 = \lceil \Re z - \lfloor \mu_2 \rfloor/2 \rfloor_{\mathbb{Z}} + \lfloor \mu_2 \rfloor \omega$.

Project z onto the line $t + \mathrm{i}\,(\beta+1/2)\sqrt{|D|}$ $(t \in \mathbb{R})$ to get.

$$z_2' = \Re z + \mathrm{i}\left(\beta + \tfrac{1}{2}\right)\sqrt{|D|} = \left(\Re z - \tfrac{\lfloor \mu_2 \rfloor + 1}{2}\right) + \lfloor \mu_2 \rfloor \omega$$

Apply the same argument applied to z_1' to see that $\xi_2 = (\Re z - (\lfloor \mu_2 \rfloor + 1)/2) + (\lfloor \mu_2 \rfloor + 1)\,\omega$.

If $\mu_2 \notin \mathbb{Z}$ then $\lfloor \mu_2 \rfloor + 1 = \lceil \mu_2 \rceil$. In this case $\xi_2 = \lceil \Re z - \lceil \mu_2 \rceil/2 \rfloor_{\mathbb{Z}} + \lceil \mu_2 \rceil \omega$.

If $\mu_2 \in \mathbb{Z}$ then $\lfloor \mu_2 \rfloor = \lceil \mu_2 \rceil = \mu_2$ and so $z_1' = z$. In this case it must be the case that ξ_1 is the nearest quadratic integer to z. An appeal to Pythagoras theorem combined with the fact that $|D| \ge 3$ shows that ξ_2 must be further away.[6]

The result follows $\square$

We now turn our attention to bounding $|z - \lceil z \rfloor|$. That is, finding ϵ such that $|z - \lceil z \rfloor| \le \epsilon$ for all $z \in \mathbb{C}$.

[6]We note that in this case, by symmetry, there is a 3rd candidate with imaginary part $(b - 1/2)\sqrt{|D|}$ that lies 1 unit directly below ξ_2 but it is equidistant with ξ_2 from z.

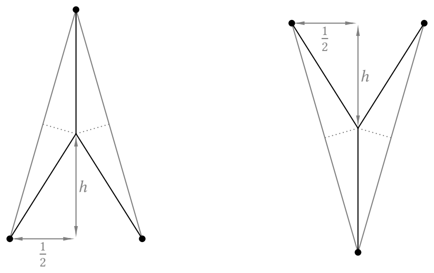


Figure 3.2: Details for calculating furthest distance from a quadratic integer when $D \equiv 1 \pmod 4$.

In the case that $D \equiv 2, 3 \pmod 4$ we observe from the square lattice in Fig. 3.1a that the furthest any complex number can be from a quadratic integer is in the centre of one of the squares. Using Pythagoras' theorem we calculate that $|z - \lceil z \rfloor| = \sqrt{1 + |D|}/2$.

In the case that $D \equiv 1 \pmod 4$ we observe from Fig. 3.1b that any complex number is bounded by triangle defined by three quadratic integers. The perpendicular bisector of the line segment joining any two of them contains the complex numbers equidistant from both. Consequently the complex number that is equidistant from all three quadratic integers lies on the intersection of the three perpendicular bisectors. The distance from this complex number to any of the three quadratic integers is the bound, ϵ that we seek.

The geometric details needed to calculate this distance are shown in more detail in Fig. 3.2. We know that the quadratic integers defining the triangle have imaginary part $(m/2)\sqrt{|D|}$ or $((m+1)/2)\sqrt{|D|}$ for some $m \in \mathbb{Z}$. So the height of the triangle is $(1/2)\sqrt{|D|}$ and it must be that $\epsilon = \sqrt{|D|}/2 - h$. We also know that $\epsilon = \sqrt{(1/4) + h^2}$. We calculate

$$\begin{aligned}
\frac{\sqrt{|D|}}{2} - h = \sqrt{(1/4) + h^2} &\implies \frac{|D|}{4} - h\sqrt{|D|} + h^2 = \frac{1}{4} + h^2 \\
&\implies \frac{|D|}{4} - \frac{1}{4} = h\sqrt{|D|} \\
&\implies h = \frac{|D| - 1}{4\sqrt{|D|}}
\end{aligned}$$

which allows us to get

$$\epsilon = \frac{\sqrt{|D|}}{2} - \frac{|D| - 1}{4\sqrt{|D|}} = \frac{|D| + 1}{4\sqrt{|D|}}$$

Combining the cases we have that

$$\epsilon = \begin{cases} \frac{1}{2}\sqrt{|D| + 1} & \text{if } D \equiv 2, 3 \pmod 4 \\ \frac{1}{4}\frac{|D|+1}{\sqrt{|D|}} & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

from which we can compute the corresponding value of ρ

$$\rho = \begin{cases} \frac{2}{\sqrt{|D|+1}} & \text{if } D \equiv 2,3 \pmod 4 \\ \frac{4\sqrt{|D|}}{|D|+1} & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

In order to calculate γ_1 we use Definition 2.2.1 on page 8 to get that

$$\frac{1}{\gamma_1^2} = 1 - \epsilon^2 = \begin{cases} 1 - \frac{1}{4}(|D|+1) & \text{if } D \equiv 2,3 \pmod 4 \\ 1 - \frac{1}{16}\frac{(|D|+1)^2}{|D|} & \text{if } D \equiv 1 \pmod 4 \end{cases}$$
$$= \begin{cases} \frac{3-|D|}{4} & \text{if } D \equiv 2,3 \pmod 4 \\ -\frac{1}{16}\frac{|D|^2-14|D|+1}{|D|} & \text{if } D \equiv 1 \pmod 4 \end{cases}$$

and so

$$\gamma_1 = \begin{cases} \frac{2}{\sqrt{3-|D|}} & \text{if } D \equiv 2,3 \pmod 4 \\ 4\sqrt{-\frac{|D|}{|D|^2-14|D|+1}} & \text{if } D \equiv 1 \pmod 4 \end{cases} \tag{3.8}$$

However, we can see that as $|D|$ increases, the value of ρ decreases, and eventually $\rho < 1$ making it impossible to satisfy condition (2.2) on page 8, and causing γ_1 to become complex. This leaves us with $D = -2$, $D = -3$, $D = -7$, and $D = -11$ as the only values of D (other than those representing the classical cases) for which the existing theory holds. Observe that these (along with the classical case which corresponds to $D = -1$) are precisely the complex quadratic norm Euclidean fields.

We note in passing that an attempt to circumvent this problem by simply scaling the integer lattice down failed, perhaps unsurprisingly. We were motivated by the fact that if a is an integer relation for x, then $\sum_{k=1}^{n} a_k x_k = 0$, but also $\lambda \sum_{k=1}^{n} a_k x_k = \sum_{k=1}^{n}(\lambda a_k)x_k = 0$ for any $\lambda \in \mathbb{R}$. So we simply shrunk our integer lattice by a factor which brought the value of ϵ below 1 in the hopes of being able to scale the resulting “integers” up by the inverse factor after completing the computation. The technique failed because the elements of the new lattice no longer formed a ring, and so the arithmetic calculations performed on the integers in the PSLQ algorithm were no longer guaranteed to stay within the shrunk lattice.

We will see that even for the cases where the existing theory does not hold (i.e., when conditions (2.1) to (2.3) do not hold) the algorithm can still be effective (see Section 3.5.3, Table 3.7).

The Algebraic PSLQ algorithm itself is simply Algorithm 2.4.2 on page 19 using the above values for ρ and γ_1, and the above nearest integer function. In order to know which quadratic integers we wish to find a relation with, we must also specify as input to the algorithm the quadratic extension field, $\mathbb{K}$, in which the integers reside. In practice for the results we report below, this was achieved by giving the algorithm the value of D (from $\mathbb{Q}(\sqrt{D})$), but any means of uniquely identifying the field in question would suffice. Algorithm A.1.1 on page 142 presented in Appendix A includes these modifications.

3.4 Experimental Methodology

We experimentally tested the efficacy of the methods described in the previous section. We note that, as stated in Section 2.5 on page 24, that the testing reported in both this and the previous chapter were performed as part of a single suite of tests. In fact, the testing reported in Sections 2.5 and 2.6 is simply a particular case of the testing reported herein. Recall, also, that the testing reported in this entire thesis extends the testing reported in Skerritt and Vrbik [67].

The code and results are available through GitHub [66].

We reiterate the main points of the testing methodology here, but omit detail and discussion found in Section 2.5. We make sure to point out, and discuss in necessary detail any changes or generalisations from those reported in the previous chapter.

We created collections of instances of quadratic integer relation problems, noting that this includes the classical cases. Each collection, referred to as a *test set*, consisted of 1000 quadratic integer relation problems each of which has a known quadratic integer relation.

We tested APSLQ, PSLQ, and LLL to see how many of the problems in each test set could be solved (i.e., the known relation recovered) by the algorithm in question. For those problems that were able to be solved we found and recorded the smallest numeric precision that was able to solve the problem, as well as the time taken to solve the problem at that minimal precision. We also measured the time taken to compute the entire test set.

In the case of LLL we additionally recorded parameters from the LLL Integer Relation procedure. We recorded the starting value of 10^k and the number of LLL computations attempted. In addition, for the problems which could be solved, we recorded the value of 10^k with which a relation was found, and the number of attempted LLL computations needed.

For PSLQ and APSLQ testing we used *Maple*'s native PSLQ implementation, as well as our own implementation of APSLQ (written in *Maple*). For the classical cases we used both implementations directly. For the non-classical quadratic cases we used our own APSLQ directly, and *Maple*'s PSLQ for the reduction technique as described in Section 3.3.1.

For LLL testing we used *Maple*'s implementation of LLL for all cases, using the appropriate $\Lambda_x(N)$ or $\Lambda_x(N, D)$ depending on the case being tested. Recall that the techniques described in Section 2.3.3 on page 14 are a special case of those described in Section 3.3.2

Recall that *Maple*'s PSLQ implementation does not allow for any choice of the parameters for that algorithm. All algorithmic parameters have been decided by the developers; one simply calls the function and passes to it the list of constants (x in Algorithm 2.2.7). As such we are unable to report on the effects of changing them.

Our implementation of APSLQ is the one presented in Appendix A (Algorithm A.1.1 on page 142). We have not implemented any of the advanced (and faster) variants. We note that although we discussed norm lower bound in some depth in Section 2.4.3 and used it in Algorithm 2.4.2 on page 19, our implementation uses neither a norm lower bound threshold for a termination condition, nor reports the norm lower bound at the end of the algorithm. This keeps it in line with *Maple*'s PSLQ implementation (although we do allow for more parameter choices then than *Maple*).

3.4.1 Test Set Generation

The test sets are classified by three parameters: a quadratic extension field ($\mathbb{K}$), a set of constants ($C \subset \mathbb{F}$), and a size ($\iota \in \mathbb{N}$) of the integers generated as part of the individual integer relation problems within the test set. We created a single test set for each valid combination of these parameters.

The extension field, $\mathbb{K} = \mathbb{Q}(\sqrt{D})$, indicates the intermediate extension field from Definition 3.2.1 on page 51 for the quadratic integer relations we wish to generate for the test set (and consequently that we will search for when testing). This also indicates that the quadratic integers will be $\mathcal{O}_{\mathbb{K}}$.

The field $\mathbb{K}$ also dictates the type of the test set. A test set is either *real* if $\mathbb{K}$ is a real quadratic extension field (i.e., $D \geq 0$) or it is *complex* if $\mathbb{K}$ is a complex quadratic extension field (i.e., $D < 0$).

Note that this is an extension of the classical testing as described in Section 2.5.1 wherein we had a field, $\mathbb{F}$ which was either $\mathbb{R}$ or $\mathbb{C}$. These classical cases are subsumed here by $\mathbb{K} = \mathbb{Q}$ and $\mathbb{K} = \mathbb{Q}(\sqrt{-1})$ for the classical real and complex cases respectively.

The set of constants, C, contains the constants we will draw from when generating quadratic integer relation problems. Exactly two sets of constants were used: one containing real constants, the other complex. The real and complex test sets, respectively, were

$$\begin{aligned} C_{\mathbb{R}} &= \left\{\pi^k : k \in \mathbb{N}, k \leq 9\right\} \cup \left\{e^k : k \in \mathbb{N}, k \leq 9\right\} \cup \left\{\gamma^k : k \in \mathbb{N}, k \leq 9\right\} \\ &\quad \cup \{\sin k : k \in \mathbb{N}, k \leq 9\} \cup \{\ln 2, \ln 3, \ln 5, \ln 7\} \\ C_{\mathbb{C}} &= \left\{5\,e^{-9\,i}, 4\,e^{-8\,i}, 9\,e^{-7\,i}, 5\,e^{-6\,i}, 2\,e^{-5\,i}, 9\,e^{-4\,i}, 8\,e^{-3\,i}, 3\,e^{-2\,i}, 2\,e^{-i}, \right. \\ &\quad \left. 4, 4\,e^{i}, 5\,e^{2\,i}, 2\,e^{3\,i}, 7\,e^{4\,i}, 6\,e^{5\,i}, 3\,e^{6\,i}, 3\,e^{7\,i}, 5\,e^{8\,i}, 5\,e^{9\,i}\right\} \end{aligned}$$

Integer size, ι is the size (in number of decimal digits) of the integers. We used two integer sizes in our testing: $\iota = 1$ and $\iota = 6$ referred to as *small* and *large* respectively. For quadratic integers the coefficients, m_1 and m_2, of $m_1 + m_2\,\omega$ are each of the indicated size ($-10^\iota < m_k < 10^\iota$).

A single test set was generated for each valid combination of the parameters used in our testing. For a fixed $\mathbb{K}, C$, and ι, the test set corresponding to these parameters was created by generating 1,000 integer relation problems with a known solution. The creation procedure is detailed in the Generate Test Instance procedure.

```
Procedure Generate Test Instance($\mathbb{K}, C, \iota$)

Input  :  $\mathbb{K}$   (a quadratic extension field $\mathbb{Q}(\sqrt{D})$)
          $C$   (a set of constants)
          $\iota$   (number of decimal digits of the integers to be generated)
Output:   $x$   (an integer relation input vector)
          $\mathfrak{a}$   (known integer relation of $x$)

1 $n \leftarrow$ random integer($2 \ldots 10$)        /* Generate a uniform random integer $2 \leq n \leq 10$ */
2 $C' \leftarrow$ random permutation($C$)        /* Randomise the order of the constants */
3 for $k$ from 1 to $n$ do        /* Generate $n$ uniform random quadratic integers each with $\iota$ digits */
4     $a_k \leftarrow$ random integer($-10^\iota + 1 \ldots 10^\iota - 1$)
5     if $\mathbb{K} \neq \mathbb{Q}$ then
6         $a_k \leftarrow a_k + \omega \cdot$ random integer($-10^\iota + 1 \ldots 10^\iota - 1$)        /* $\omega = \sqrt{D}$ or $(1+\sqrt{D})/2$ */
7 $x \leftarrow \left(\sum_{k=1}^{n}(a_k C'_k), C'_1, \ldots, C'_n\right)$        /* Construct the integer relation problem */
8 $\mathfrak{a} \leftarrow (-1, a_1, \ldots, a_n)$        /* Construct a known integer relation solution */
9 return $x, \mathfrak{a}$
```

3.4.2 Testing Procedure

For each test set, we attempted to solve the problems within it using PSLQ, APSLQ, and LLL in *Maple*. Our aim was to see if the algorithm could recover the known quadratic integer relation, $\mathfrak{a}$, from the input vector x. Any quadratic integer multiple of the known relation was considered to be an equivalent relation for this purpose.

Each test set was used twice. Once with so called *short input* wherein only the coefficients in the known relation $\mathfrak{a}$ were used, and once with so called *long input* wherein twice as many coefficients were used (or all the coefficients from the set if there were not enough). The purpose of this was to verify the robustness of the algorithm in the presence of unnecessary information, and to see the effect on the needed precision and time.

We compute at multiple precisions until we find the known integer relation, or until the precision becomes too large. If the known relation can be found, we define the most favourable computation as the one which finds the known relation using the smallest precision. All recorded measurements are either for the most favourable computation (if it exists), or for the computation performed with the largest precision otherwise.

We measure and record the precision used, and the time taken[7] for all computations. For LLL computations we also measure and record the number of LLL attempts before the relation was recovered, and the number of candidate integer relations.

The result of a computation on an individual test instance are either GOOD, UNEXPECTED, BAD, or FAIL as as outlined in Table 2.1 on page 28. We simply counted the number of occurrences of each result. No UNEXPECTED results were found during our testing.

The procedure to diagnose a result is unchanged from the Diagnose procedure on page 29. For a candidate relation a we diagnose a FAIL condition is immediately if no result is produced (usually because the maximum number of iterations was exceeded). Otherwise, we look to see if $(-a_1)\mathfrak{a} = a$, and if so we diagnose a GOOD result. If that is not the case, we then test the computed algebraic integer relation to 1000 decimal digits of precision, and if the result is within 10^{-998} of 0 we diagnose an UNEXPECTED result. If none of the above apply, then we diagnose a BAD result. In the case that there are multiple candidate relations (which currently only happens with LLL) we diagnose each individually, and take the best result using the order GOOD $>$ UNEXPECTED $>$ BAD $>$ FAIL.

The problem with the reconstructed relation for the reduction method in the complex case as described in Section 3.3.1 is not addressed at all by this diagnosis method. It is entirely possible that $(-a_1)\mathfrak{a} = a$ even if a_1 is not a valid algebraic integer for the extension field in question. We accounted this by checking to see if the entries in the recovered relation consisted only of valid algebraic integers from the appropriate field. This check was performed after the usual diagnosis, so that we could compare the updated results with the originally diagnosed result. If any entries were not appropriate algebraic integers then we applied both techniques described in Section 3.3.1. When applying the decomposition method we favour ξ_1 but were careful to check for the case that it was the zero vector. When applying the conjugate method we produced multiple transformed candidates; one for each non-zero element of the original candidate. We then processed the collection of new candidates as normal, using the Diagnose procedure. We were

[7] Measured in CPU seconds using *Maple*'s `time()` function

careful to record which of the methods corresponded to the candidate with the best result, and also to record the diagnosis of every candidate for comparative purposes.

We note that of the two methods presented the decomposition method is the simplest to apply. We further note that the conjugate requires multiplication, and so increases the size (both in terms of absolute value, and also in terms of the number of digits) of the integers in the candidate relation, risking overflow or exhaustion of numeric precision.

When testing with APSLQ on appropriate test sets (i.e., classical cases and complex quadratic cases) each such test set is separately tested using different values of γ. The values of γ used were $\gamma = \gamma_1$, $\gamma = 2.0$, and $\gamma = 3.0$. Note that although, strictly speaking, we require $\gamma > \gamma_1$ in order for conditions (2.1) to (2.3) on page 8 to be satisfied, the choice of $\gamma = \gamma_1$ seems to be common in practice, and the results do not seem to suffer.[8] All APSLQ testing used the threshold $\epsilon = 10^{-(d-1)}$ (see Algorithm 2.4.2 on page 19 on which APSLQ was modified only in the manner described in Section 3.3.3).

3.5 Experimental Results

Herein we present the results from the testing methodology described above. The results are extensive due to the large number of combinations of parameters. We present our discussion and and select graphs to illustrate the main points. The complete collection of graphs, of which there are many, can be found in Appendix B on page 145.

For each test set we present graphs for minimum required precision, and also for time taken.[9] We further sub-divide this information into graphs of the data for all three algorithms (PSLQ, LLL, and APSLQ), choosing only the $\gamma = \gamma_1$ case for APSLQ, and a separate graph showing only APSLQ for all the tested γ values. This was done to prevent the totality of the APSLQ data from dominating the graphs and obscuring the relationship between the three algorithms. The reader may use the $\gamma = \gamma_1$ case a a reference to correlate the two graphs.

When looking at the time taken, it is not meaningful to compare *Maple*'s inbuilt PSLQ and LLL times to those of our own implementation of APSLQ. This is because our implementation is written in *Maple* itself (an interpreted language sitting on top of *Maple*'s computation engine) and have not implemented any of the efficient methods described in the literate (some of which are also outlined in Appendix A on page 140). So we omit the APSLQ timing data from the graphs containing the PSLQ and LLL data. The relative performance within APSLQ of the different γ values if of interest, however, and so we retain the APSLQ-only graphs for this data.

[8]It is likely that much of the time the rounding inherent in floating point arithmetic results in a value slightly higher than the exact value of γ_1, although we have taken no pains to ensure this.
[9]Measured in CPU seconds using *Maple*'s `time()` function, recall.

3.5.1 Classical Integer Relations

We tested our implementation of APSLQ when computing classical integer relations (i.e., for the cases $\mathbb{K} = \mathbb{Q}$ and $\mathbb{K} = \mathbb{Q}(\sqrt{-1})$). We compared the results against those reported in Section 2.6. This testing acted as a “sanity check” that our implementation was correct for the classical cases.

Table 3.1: APSLQ performance on test sets for classical integer relations

Test Set Parameters			Short Input	Long Input
$\mathbb{K}$	C	ι	APSLQ (all γ)	APSLQ (all γ)
$\mathbb{Q}$	$C_\mathbb{R}$	1	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}$	$C_\mathbb{R}$	6	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-1})$	$C_\mathbb{R}$	1	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-1})$	$C_\mathbb{R}$	6	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-1})$	$C_\mathbb{C}$	1	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-1})$	$C_\mathbb{C}$	6	1000G 0B 0F	1000G 0B 0F

The results are tabulated in Table 3.1 and should be compared against Table 2.2 on page 30. Recall that it was impossible to create real test sets that use complex constants ($\mathbb{K} = \mathbb{Q}$ and $C = C_\mathbb{C}$), so we were only able to test a single field with complex constants.

Precision

The precisions required for the classical test sets are shown in Figs. 3.3 to 3.8. Observe that Figs. 3.3, 3.5 and 3.7 in particular present the same information as Fig. 2.3 on page 35 and Figs. 2.10 and 2.11 on page 42 and on page 43, but with the addition of the APSLQ precision data.

We note that the PSLQ and APSLQ, while not identical, are very strongly clustered to the point of being difficult to tell apart; we consider this a strong sign that our implementation is good. We also note that as the value of γ increases, so does the required precision of the computation. The (rare) cases where the minimum precision is below the theoretical minimum are discussed in Section 2.6.1 on page 31

Timing

The relative timing data for PSLQ and LLL has already been presented in the previous chapter (see Fig. 2.3 on page 35 and Figs. 2.10 and 2.11 on page 42 and on page 43). We do not repeat this data here.

The timing data for APSLQ is presented in Figs. 3.9 to 3.11. We note that as γ increases, that computation time tends to decrease. In the real case we see this difference can be as much as half an order of magnitude, although in the complex cases the difference is

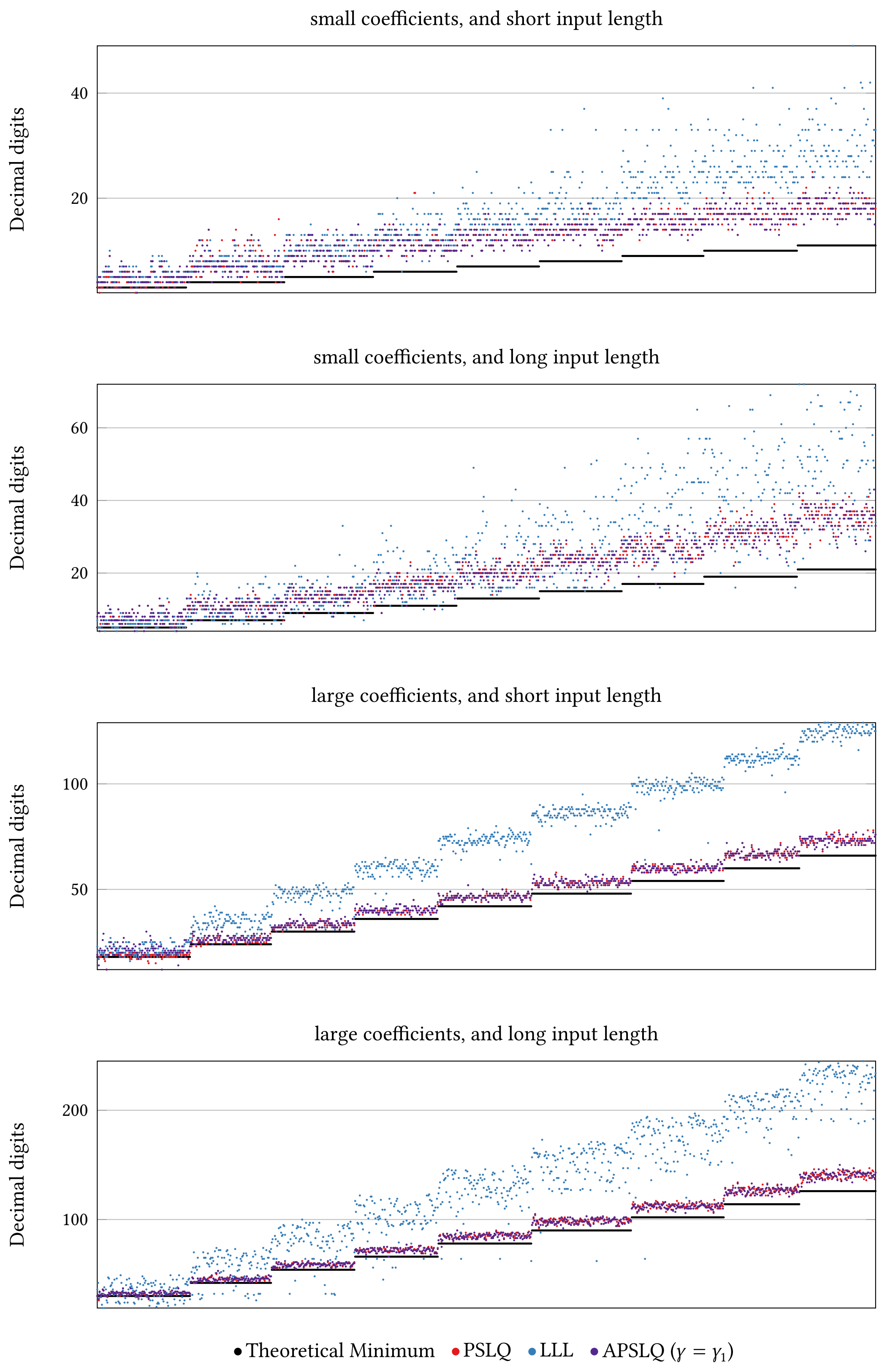


Figure 3.3: Precision required for $\mathbb{K} = \mathbb{Q}, C = C_{\mathbb{R}}$.

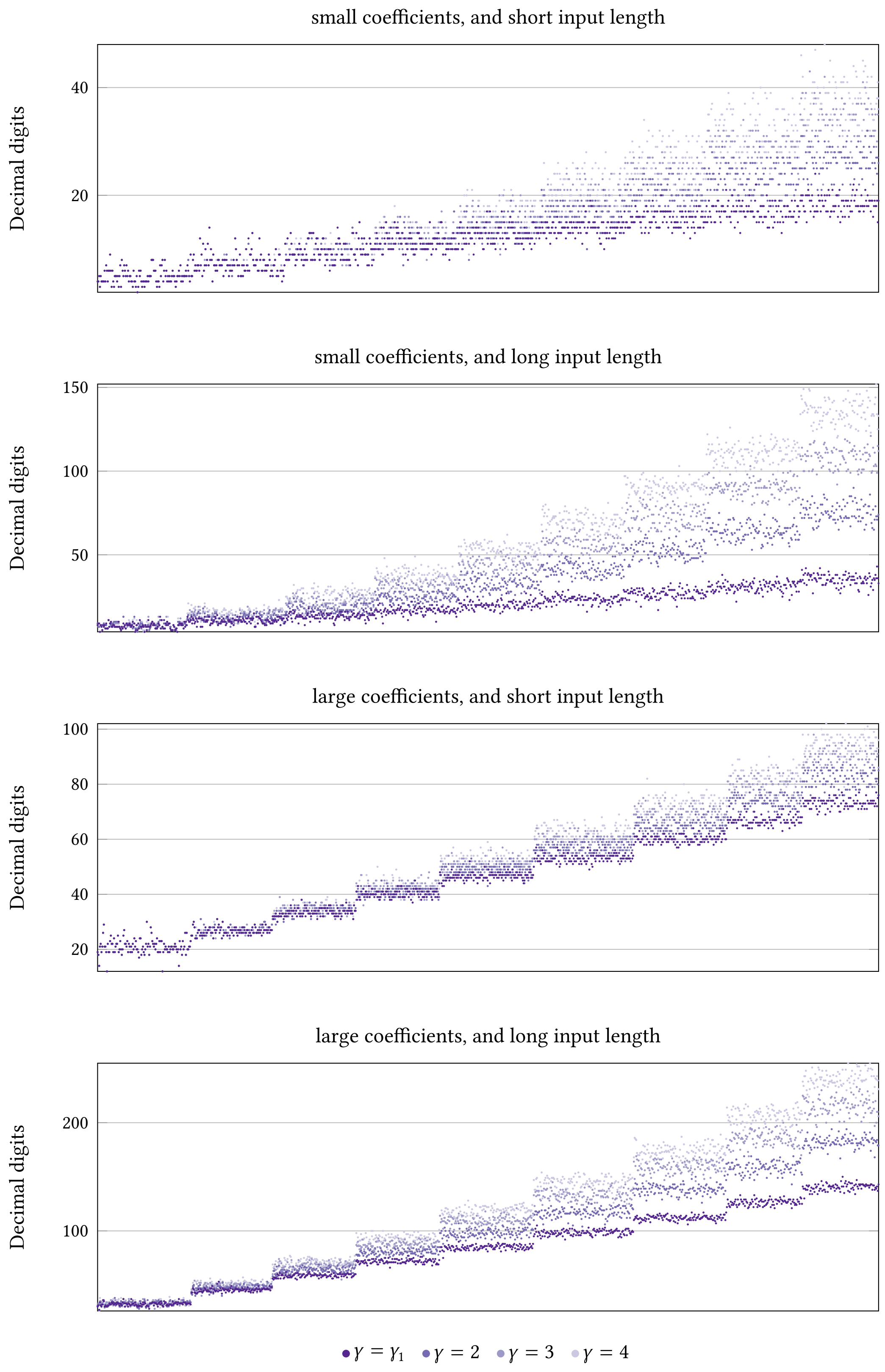


Figure 3.4: Precision required for $\mathbb{K} = \mathbb{Q}, C = C_{\mathbb{R}}$ (APSLQ only).

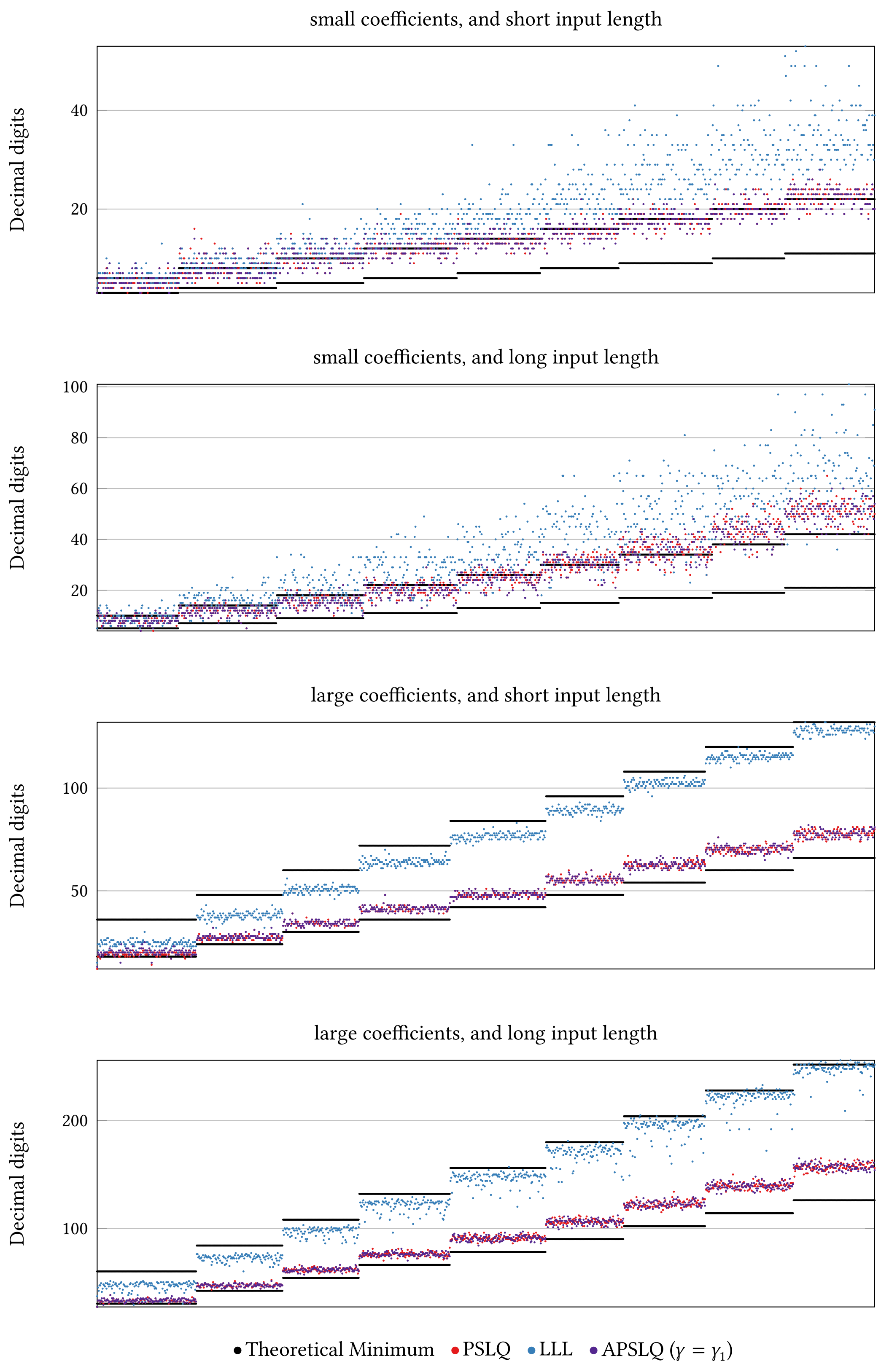


Figure 3.5: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$.

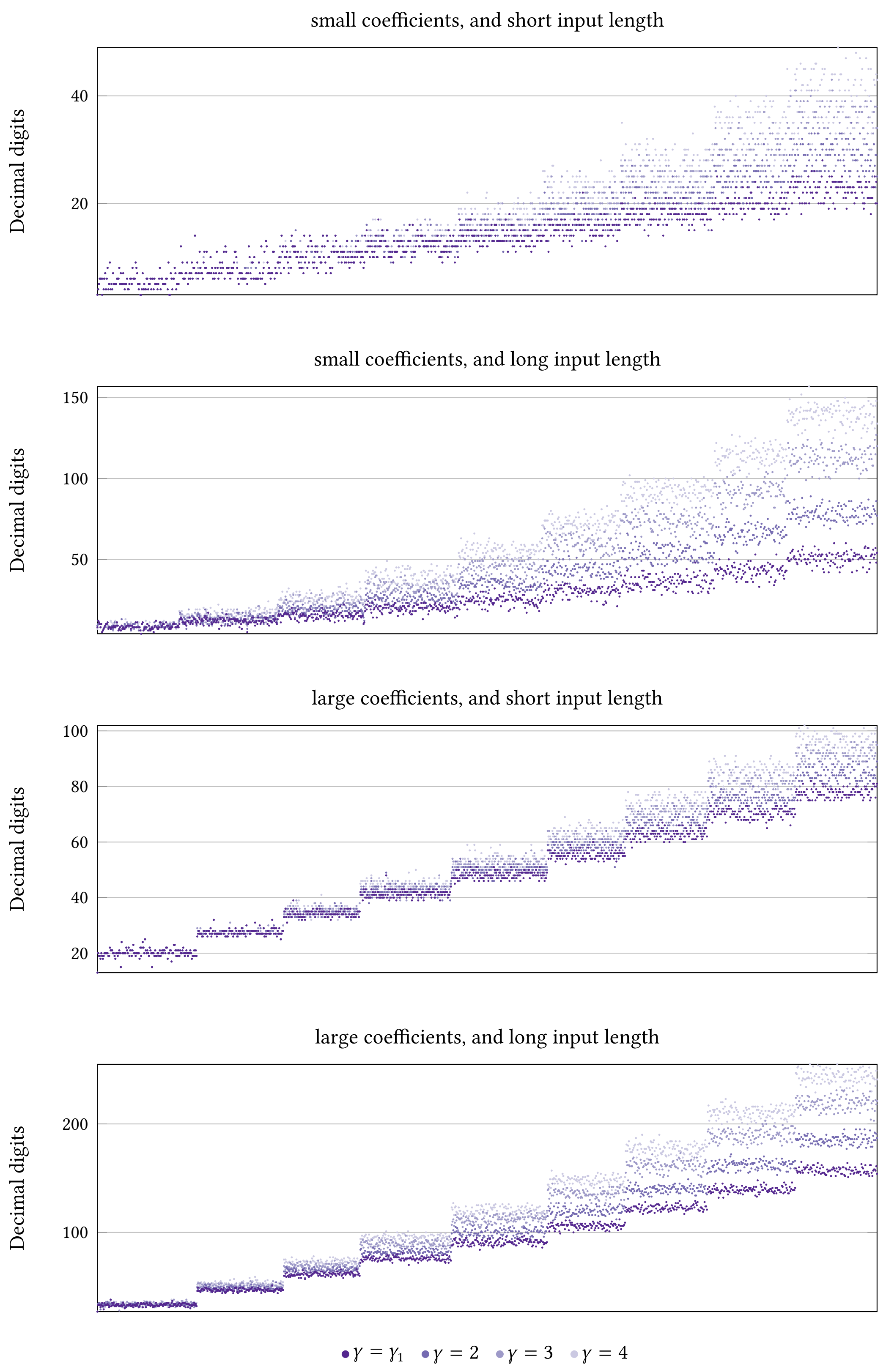


Figure 3.6: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$ (APSLQ only)).

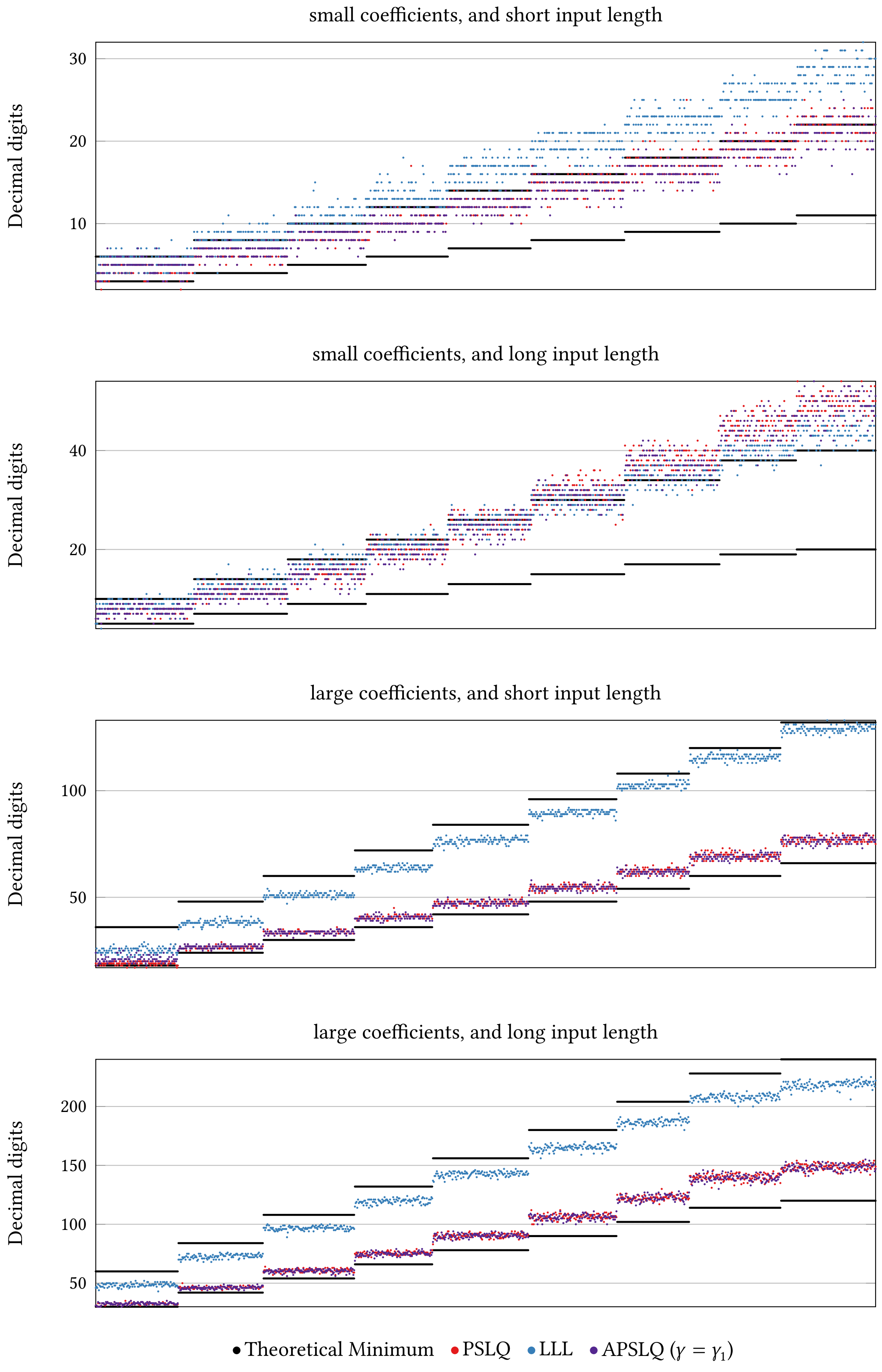


Figure 3.7: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$.

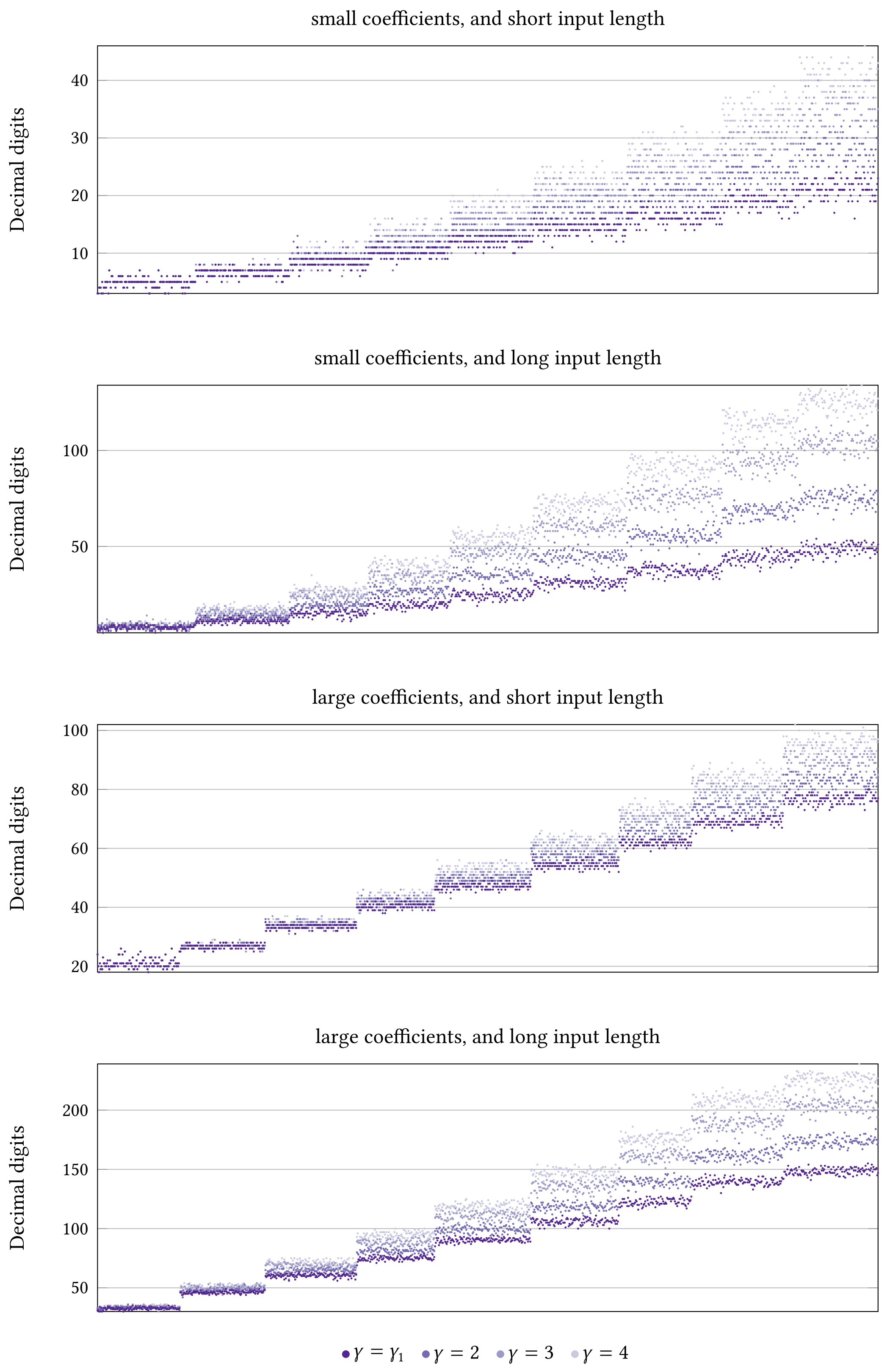


Figure 3.8: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$ (APSLQ only)).

less distinct. We note a stark exception to this tendency in the particular the case where $\mathbb{K} = \mathbb{Q}(\sqrt{-1})$ using small coefficients and long input length. In combination with the observation, above, that required precision increases with γ suggests a trade-off between computation time and required precision.

We note that some care should be taken in interpreting these graphs, especially in regard to evaluating the extent of the suggested trade-off, as the time reported by our testing is for the most favourable computation, and (as we can see in the graphs, above, for minimum precision) even for a fixed test problem, the integer relation calculations were performed at potentially different precisions for the different γ parameter values. From a practical standpoint, these graphs represent in a sense the best possible times (under the assumption that calculation at greater precision will require more time).

3.5.2 Real Quadratic Extension Fields

For the real quadratic algebraic integer relations we tested the following real quadratic extension fields:

$$\mathbb{K} = \mathbb{Q}(\sqrt{D}) \quad \text{for} \quad D \in \{2, 3, 5, 6, 7, 10, 11\}$$

Recall that APSLQ is not appropriate for these extension fields, so only the reduction method was tested. We solved the reduced classical integer relation problem with both PSLQ and LLL.

Table 3.2: Reduction method for real quadratic fields

Test Set Parameters			Short Input		Long Input	
$\mathbb{K}$	C	ι	PSLQ	LLL	PSLQ	LLL
$\mathbb{Q}(\sqrt{2})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{2})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{3})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{3})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{5})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{5})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{6})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{6})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{7})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{7})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{10})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{10})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{11})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{11})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F

The results are tabulated in Table 3.2; the reduction method has given all correct results with both PSLQ and LLL. We note that since we are testing real quadratic extension fields

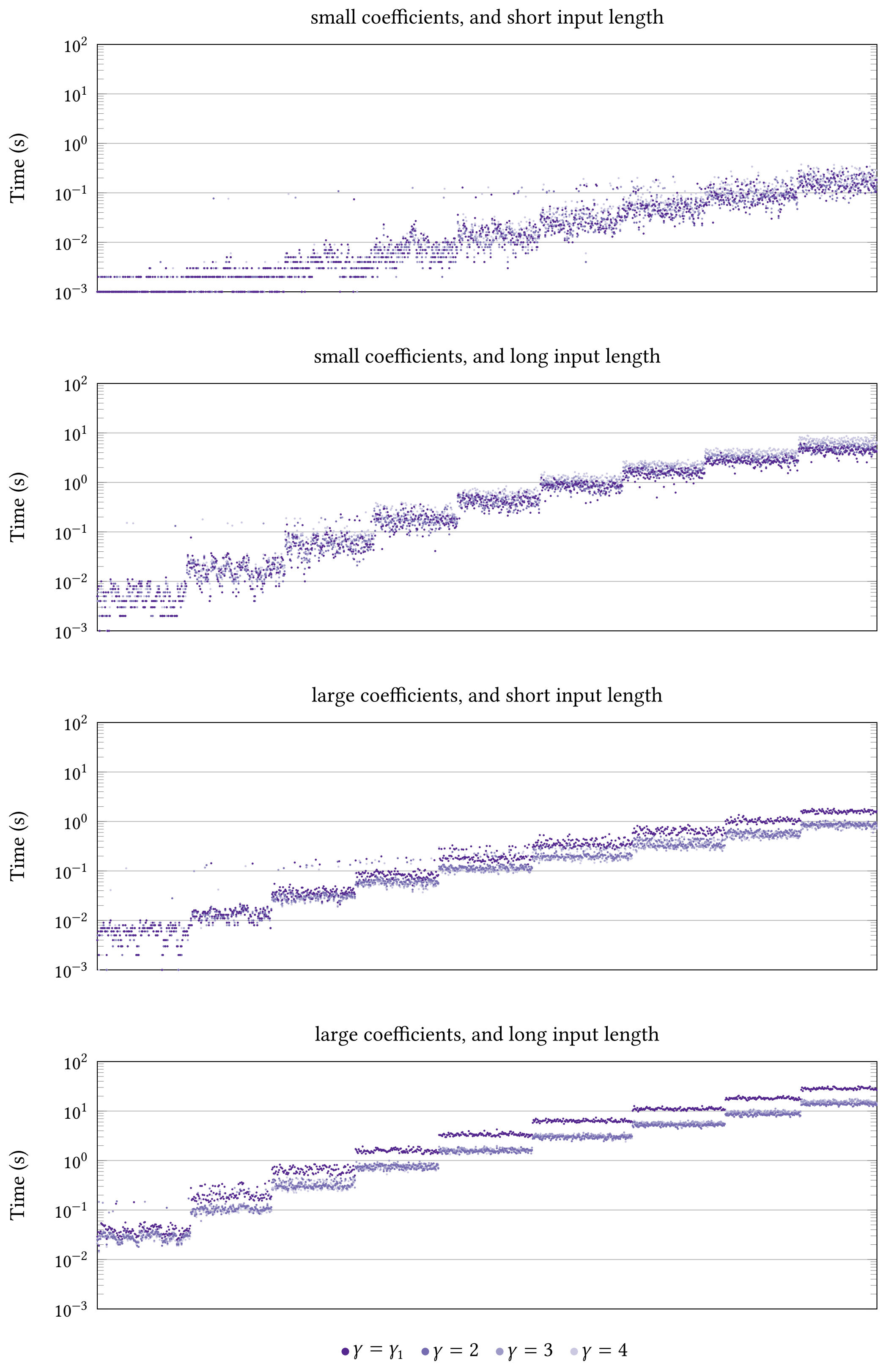


Figure 3.9: Computation time for $\mathbb{K} = \mathbb{Q}, C = C_{\mathbb{R}}$ (APSLQ only).

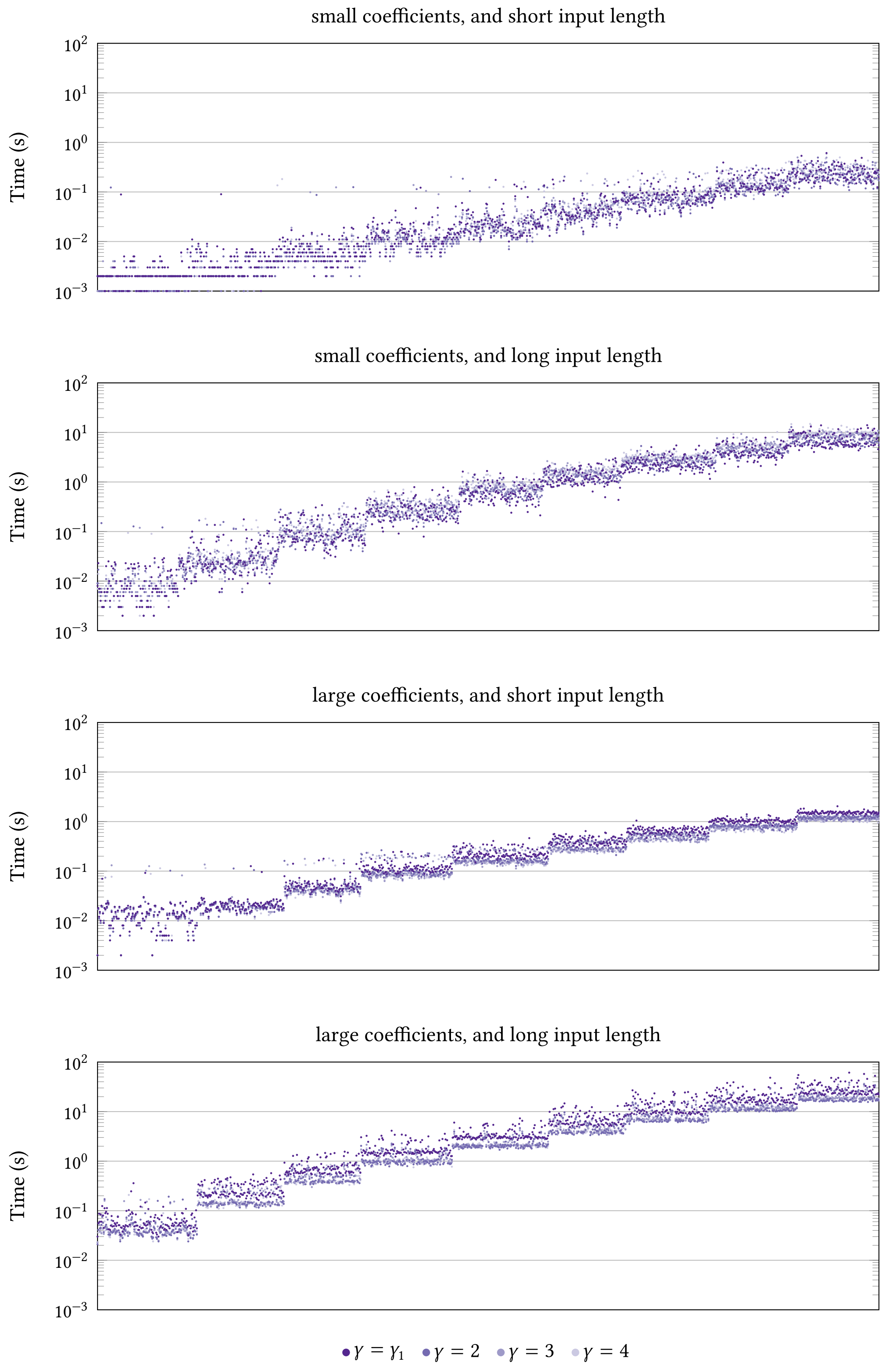


Figure 3.10: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$ (APSLQ only)).

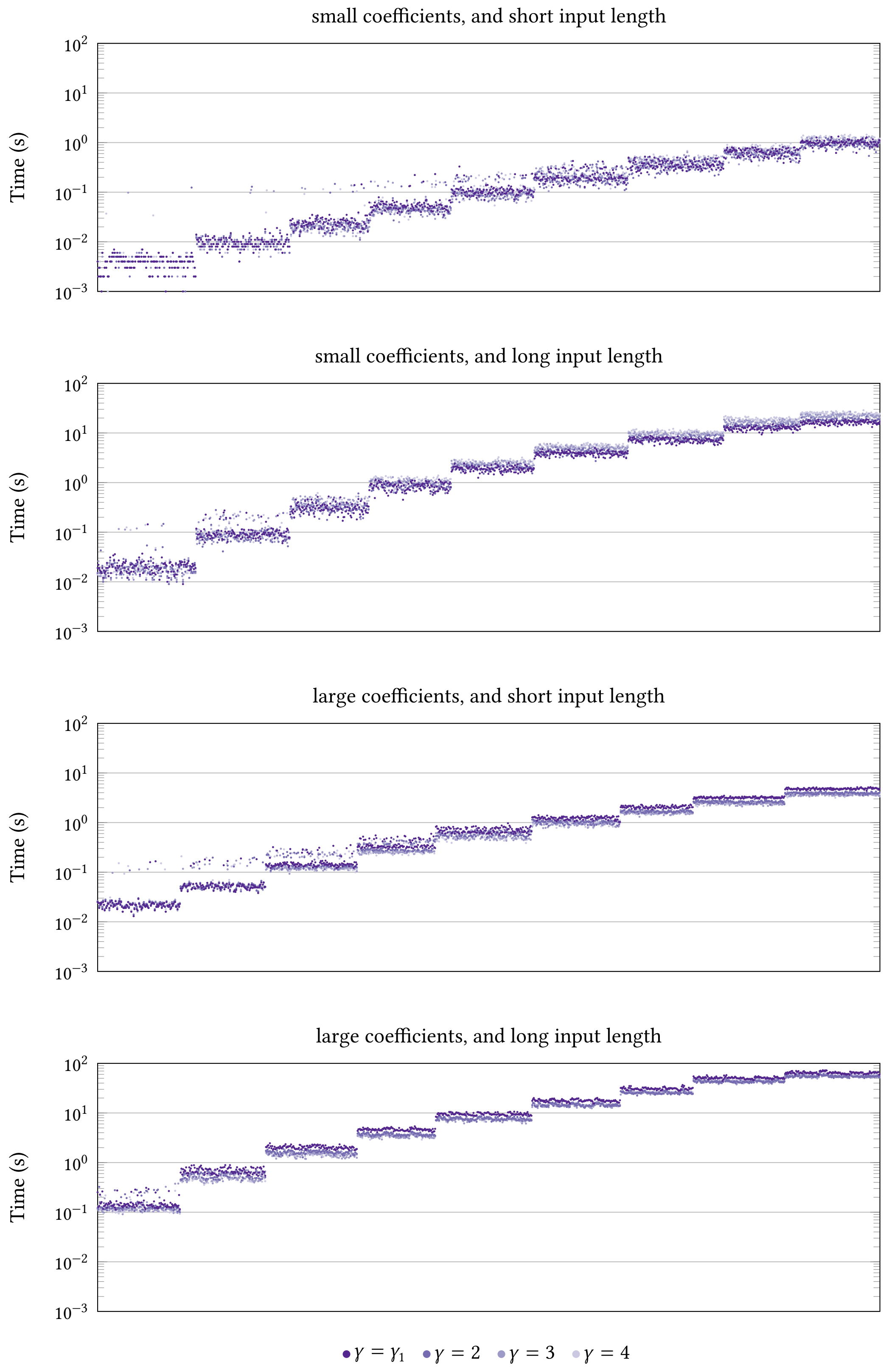


Figure 3.11: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$ (APSLQ only).

(i.e., where $\mathbb{K} \subset \mathbb{R}$) then, as discussed in Section 3.3.1, any candidate relation found must definitely have been algebraic integer relations. Contrast this to the complex quadratic extension field testing, below.

Precision

We present the graph for $\mathbb{K} = \mathbb{Q}(\sqrt{2})$ in Fig. 3.12, and note that it is illustrative of all cases, as presented in Figs. B.2 to B.8 on pages 147–153.

We see that the required precision for PSLQ typically sits just above the double theoretical minimum precision mark. This should be unsurprising given that the reduction method doubles the size of the input vector. We also see that LLL typically requires about twice as much precision as PSLQ, which is in keeping with the behaviour we saw in the classical cases. Note that we are using the reduction method with both algorithms here so they are computing integer relations for the same input vector (notwithstanding that the LLL procedure constructs an integer lattice basis from this input vector.)

Timing

We present the graph for $\mathbb{K} = \mathbb{Q}(\sqrt{2})$ in Fig. 3.13, and note that it is illustrative of all cases, as presented in Figs. B.10 to B.16 on pages 155–161.

We note simply that we see LLL takes usually two orders of magnitude more time to compute the same integer relation problem. This is in keeping with our observations from the classical cases. Furthermore we see that in extreme cases LLL takes longer than 100 seconds (corresponding to problem for which PSLQ takes close to 10 seconds).

3.5.3 Complex Quadratic Extension Fields

For complex quadratic algebraic integer relations we were able to test all of our techniques: reduction (using PSLQ), LLL using $\Lambda_x(N, D)$, and APSLQ. We tested the fields:

$$\mathbb{K} = \mathbb{Q}(\sqrt{D}) \quad \text{for} \quad D \in \{-2, -3, -5, -6, -7, -10, -11\}$$

Euclidean Fields

The cases where $D \in \{-2, -3, -7, -11\}$ correspond to the norm Euclidean fields, and for these fields conditions (2.1) to (2.3) on page 8 are satisfied. In particular, for these cases there is a real value for γ_1 which is the lower bound for the γ parameter of PSLQ. The results are summarised in Tables 3.3 to 3.5. All algorithms give flawless results.

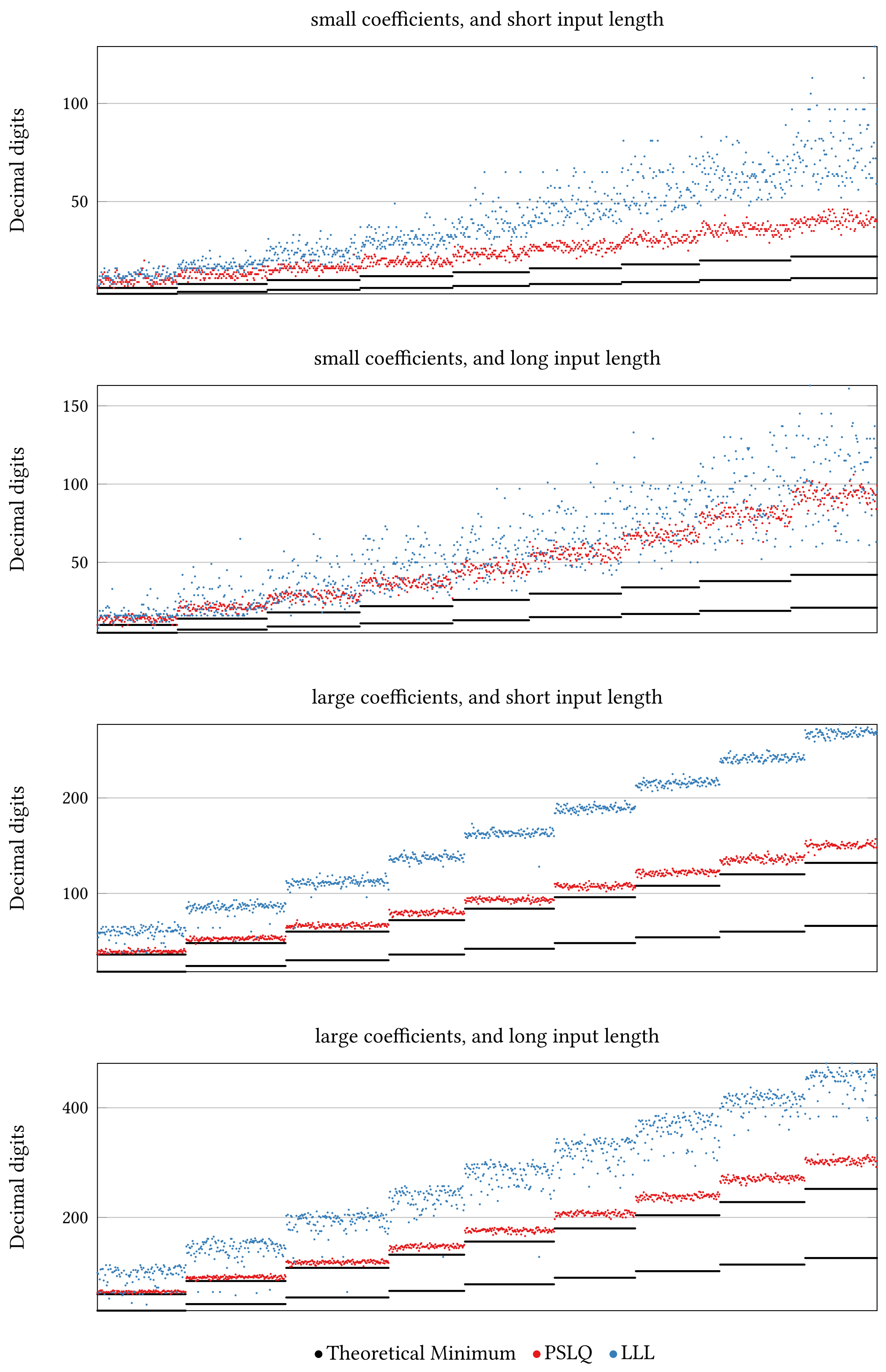


Figure 3.12: Precision required: $\mathbb{K} = \mathbb{Q}(\sqrt{2}), C = C_{\mathbb{R}}$.

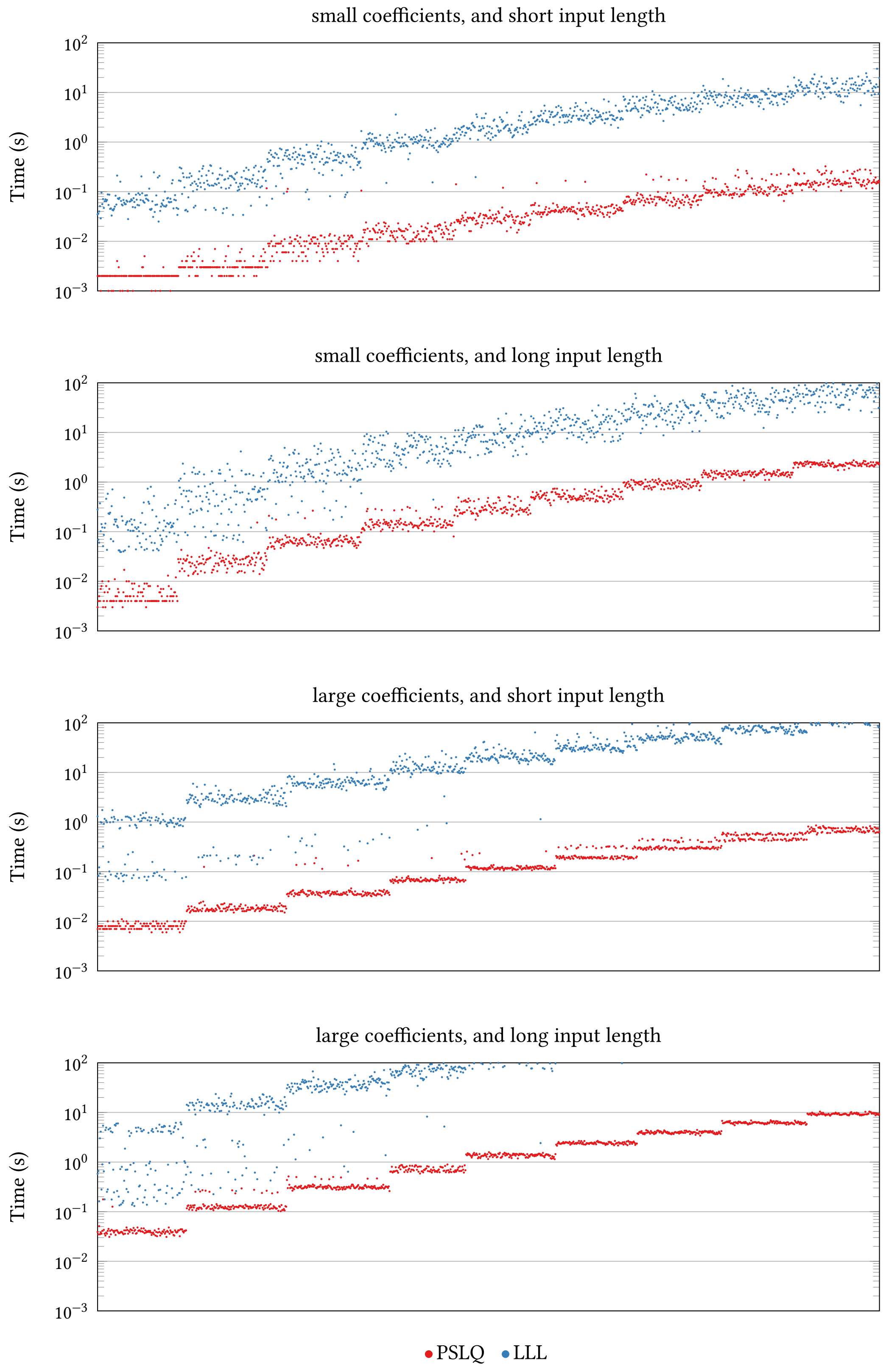


Figure 3.13: Computation time: $\mathbb{K} = \mathbb{Q}(\sqrt{2}), C = C_{\mathbb{R}}$.

Table 3.3: Complex quadratic norm Euclidean fields

Test Set Parameters			Short Input		
$\mathbb{K}$	C	ι	PSLQ	LLL	APSLQ[†]
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F

[†]All γ values except for the specific cases when $\mathbb{K} = \mathbb{Q}(\sqrt{-11})$ and $\gamma = 2$. See Table 3.5 for those cases.

Table 3.4: Complex quadratic norm Euclidean fields

Test Set Parameters			Long Input		
$\mathbb{K}$	C	ι	PSLQ	LLL	APSLQ[†]
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	1000G 0B 0F

[†]All γ values except for the specific cases when $\mathbb{K} = \mathbb{Q}(\sqrt{-11})$ and $\gamma = 2$. See Table 3.5 for those cases.

Table 3.5: APSLQ results for $\mathbb{Q}(\sqrt{-11})$ with $\gamma = 2$

Test Set Parameters			Short Input			Long Input		
$\mathbb{K}$	C	ι	APSLQ ($\gamma = 2$)			APSLQ ($\gamma = 2$)		
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	1	1000G	0B	0F	999G	0B	1F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	6	1000G	0B	0F	999G	0B	1F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	1	999G	0B	1F	990G	0B	10F
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	6	995G	0B	5F	959G	0B	41F

Observe that when testing the field $\mathbb{Q}(\sqrt{-11})$ with complex constants $C = C_{\mathbb{R}}$ and $\gamma = 2.0$ the results were slightly worse than when $\gamma = \gamma_1$. This is likely because for this field $\gamma_1 = \sqrt{22}/2 > 2$, so $\gamma = 2.0$ is too small to satisfy conditions (2.1) to (2.3) from Section 2.2. This supposition is strengthened by the observation that when $\gamma = 3.0 > \sqrt{22}/2$ the results are all GOOD again.

The PSLQ results from these tables hide some complexity. Recall from Section 3.3.1 on page 52 that in complex cases, the reduction technique might yield integers from a larger field than desired ($\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$ instead of $\mathbb{Q}(\sqrt{D})$). We presented two techniques for transforming a candidate relation of $\mathbb{Q}(\mathrm{i} + \sqrt{|D|})$-integers into a candidate relation of $\mathbb{Q}(\sqrt{D})$-integers. In our testing we have been careful to use a transformation method only when it is needed. To our surprise, it turns out that in the vast majority of cases, the reduction method was directly successful in recovering the relation (i.e., without the need for any transformation at all). Table 3.6 shows the number of test cases requiring transformation.

Table 3.6: No. of Problems Requiring Transformation after Reduction with PSLQ

Test Set Parameters		Short Input		Long Input	
$\mathbb{K}$	C	$\iota = 1$	$\iota = 6$	$\iota = 1$	$\iota = 6$
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{R}}$	85	83	142	112
$\mathbb{Q}(\sqrt{-2})$	$C_{\mathbb{C}}$	91	84	123	124
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{R}}$	83	92	124	122
$\mathbb{Q}(\sqrt{-3})$	$C_{\mathbb{C}}$	120	100	172	119
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{R}}$	57	75	95	107
$\mathbb{Q}(\sqrt{-7})$	$C_{\mathbb{C}}$	83	64	107	109
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{R}}$	32	61	51	61
$\mathbb{Q}(\sqrt{-11})$	$C_{\mathbb{C}}$	41	50	40	69

When transformation was necessary, we used both transformation methods, and recorded the results of each. When applying the conjugate method we individually applied conjugate multiplication from each non-zero element of the candidate relation, yielding many resulting candidate relations. We found, to some surprise, that for those tests problems that required a transformation method every transformed candidate yielded a GOOD result.

Non-Euclidean Fields

The cases where $D \in \{-5, -6, -10\}$ correspond to fields which are not norm Euclidean and for these fields conditions (2.1) to (2.3) on page 8 are not satisfied. In particular, for these cases there is no real value for γ_1 (it will be complex if computed using Eq. (3.8)) and so we have no viable lower bound for the γ PSLQ parameter. Moreover the existing theory does not hold in these cases. We nonetheless attempted calculations with $\gamma \in \{2, 3, 4\}$.

Table 3.7: Complex quadratic fields that are not norm Euclidean

Test Set Parameters			Short Input				
$\mathbb{K}$	C	ι	PSLQ	LLL	APSLQ ($\gamma = 2$)	APSLQ ($\gamma = 3$)	APSLQ ($\gamma = 4$)
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	997G 0B 3F	1000G 0B 0F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	983G 2B 15F	992G 2B 6F	995G 2B 3F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	158G 0B 842F	187G 0B 813F	221G 0B 779F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	164G 0B 836F	182G 0B 818F	192G 0B 808F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	997G 0B 3F	999G 0B 1F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	986G 1B 13F	996G 1B 3F	999G 1B 0F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	137G 0B 863F	144G 0B 856F	158G 0B 842F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	59G 0B 941F	60G 0B 940F	60G 0B 940F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	999G 0B 1F	999G 0B 1F	1000G 0B 0F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	993G 0B 7F	994G 0B 6F	997G 0B 3F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	40G 0B 960F	42G 0B 958F	43G 0B 957F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	0G 0B 1000F	0G 0B 1000F	0G 0B 1000F

Table 3.8: Complex quadratic fields that are not norm Euclidean

Test Set Parameters			Long Input				
$\mathbb{K}$	C	ι	PSLQ	LLL	APSLQ ($\gamma = 2$)	APSLQ ($\gamma = 3$)	APSLQ ($\gamma = 4$)
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	904G 0B 96F	994G 0B 6F	978G 0B 22F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	727G 0B 273F	793G 0B 207F	821G 0B 179F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	11G 0B 989F	23G 0B 977F	24G 0B 976F
$\mathbb{Q}(\sqrt{-5})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	0G 0B 1000F	0G 0B 1000F	0G 0B 1000F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	909G 0B 91F	991G 0B 9F	967G 0B 33F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	766G 0B 234F	824G 0B 176F	872G 0B 128F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	2G 0B 998F	10G 0B 990F	7G 0B 993F
$\mathbb{Q}(\sqrt{-6})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	0G 0B 1000F	0G 0B 1000F	0G 0B 1000F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{R}}$	1	1000G 0B 0F	1000G 0B 0F	932G 0B 68F	989G 0B 11F	975G 0B 25F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{R}}$	6	1000G 0B 0F	1000G 0B 0F	808G 0B 192F	861G 0B 139F	885G 0B 115F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{C}}$	1	1000G 0B 0F	1000G 0B 0F	1G 0B 999F	0G 0B 1000F	0G 0B 1000F
$\mathbb{Q}(\sqrt{-10})$	$C_{\mathbb{C}}$	6	1000G 0B 0F	1000G 0B 0F	0G 0B 1000F	0G 0B 1000F	0G 0B 1000F

The results are summarised in Tables 3.7 and 3.8. For real constants ($C = C_{\mathbb{R}}$) the results are mostly good, despite the algorithm conditions not being satisfied. This is similar to the results for $\mathbb{Q}(\sqrt{-11})$, highlighted above, that also failed those conditions. Contrast these to the cases where $C = C_{\mathbb{C}}$ which perform exceptionally poorly with APSLQ. This ought not be especially surprising since these fields do not satisfy the required conditions, and it is more remarkable that the results for the cases using real constants are so good.

Both the reduction method (using PSLQ), and LLL give flawless results, as we also saw for the Euclidean fields. The number of test problems requiring a transformation is shown in Table 3.9 and we see significantly fewer cases as compared to the Euclidean fields. We note that, similar to the Euclidean cases, every transformed candidate yielded a GOOD result.

Table 3.9: No. of Problems Requiring Transformation after Reduction with PSLQ

Test Set Parameters		Short Input		Long Input	
$\mathbb{K}$	C	$\iota = 1$	$\iota = 6$	$\iota = 1$	$\iota = 6$
$\mathbb{Q}(\sqrt{-5})$	$C_\mathbb{R}$	8	12	23	13
$\mathbb{Q}(\sqrt{-5})$	$C_\mathbb{C}$	6	9	26	21
$\mathbb{Q}(\sqrt{-6})$	$C_\mathbb{R}$	4	8	15	14
$\mathbb{Q}(\sqrt{-6})$	$C_\mathbb{C}$	0	8	20	18
$\mathbb{Q}(\sqrt{-10})$	$C_\mathbb{R}$	0	2	8	2
$\mathbb{Q}(\sqrt{-10})$	$C_\mathbb{C}$	0	0	5	2

Precision

We present the graphs for $\mathbb{K} = \mathbb{Q}(\sqrt{-2})$ in Figs. 3.14 to 3.17, and note that it is illustrative of all cases, as presented in Figs. B.21 to B.48 on pages 166–193.

We continue to see that the required precision for PSLQ typically sits just above the double theoretical minimum precision mark, like we was in the real quadratic cases. We also continue to see the increase in required precision as the value of γ increases for APSLQ.

The required precision for APSLQ typically sits above the theoretical minimum precision mark suggesting that, like Gaussian integers, the number of digits of a quadratic integer is the larger of the ω coefficient or the constant coefficient. The height above the minimum theoretical precision that APSLQ sits appears to grow as D grows, although $D = -2$ stands as an exception. The discrepancy in the $D = -2$ case might be that it is the only case where $D \equiv 2, 3 \pmod 4$, and that these cases have a larger lattice than the triangular one in the other case.

Of greater interest and surprise is that the LLL precision (recalling that we are using the more direct method of the $\Lambda_x(N, D)$ lattice) lies between PSLQ and APSLQ. The lattice used to compute these integer relations problems is twice as large, and the lattice vectors twice as long as would be the case for a classical integer relation (which would also be the case if we could compute LLL directly with quadratic integers and complex numbers). We do not know precisely why this is the case; we expect something about the structure of the constructed lattice allows for the reduced precision requirement.

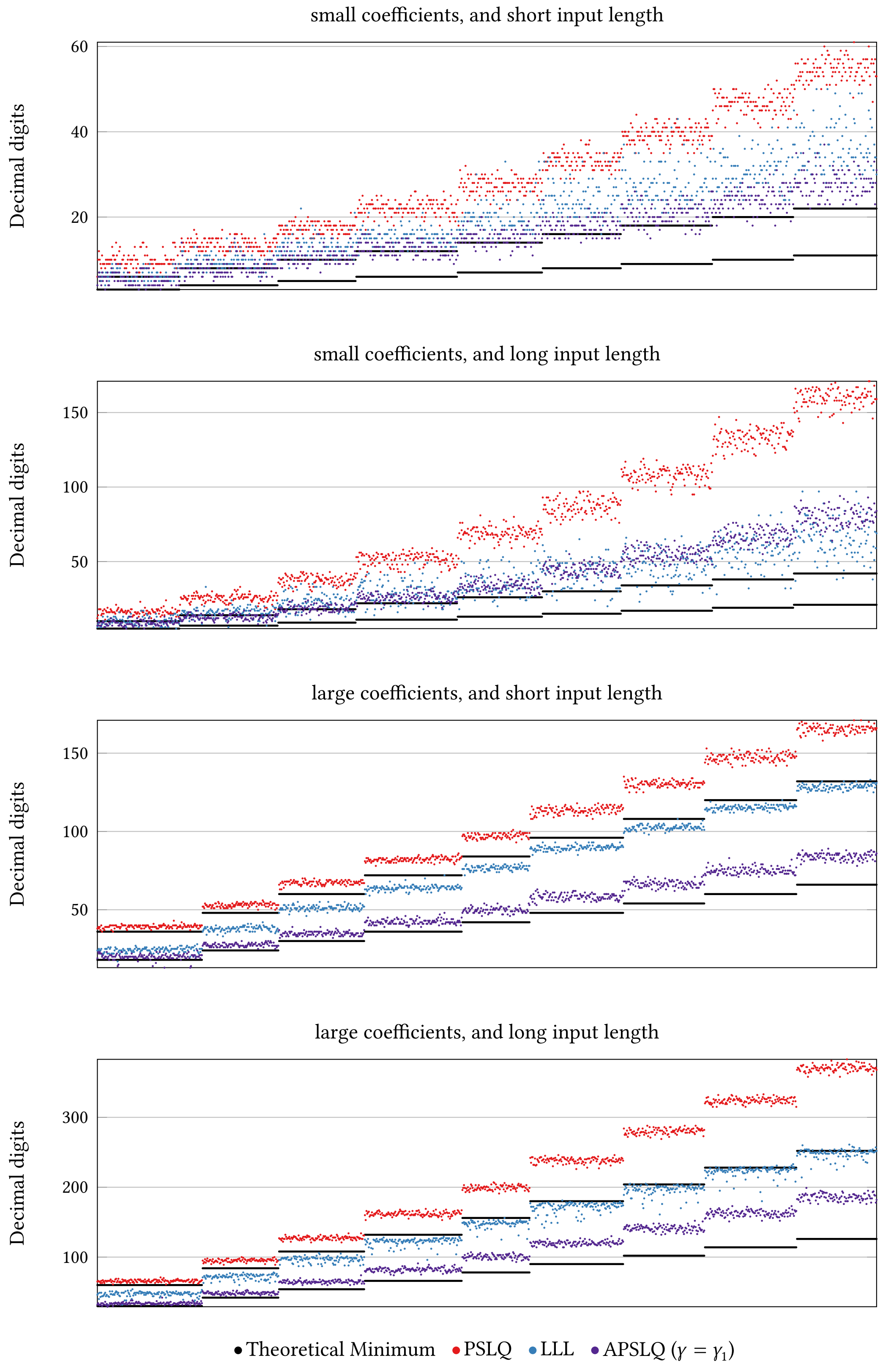


Figure 3.14: Precision required: $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$.

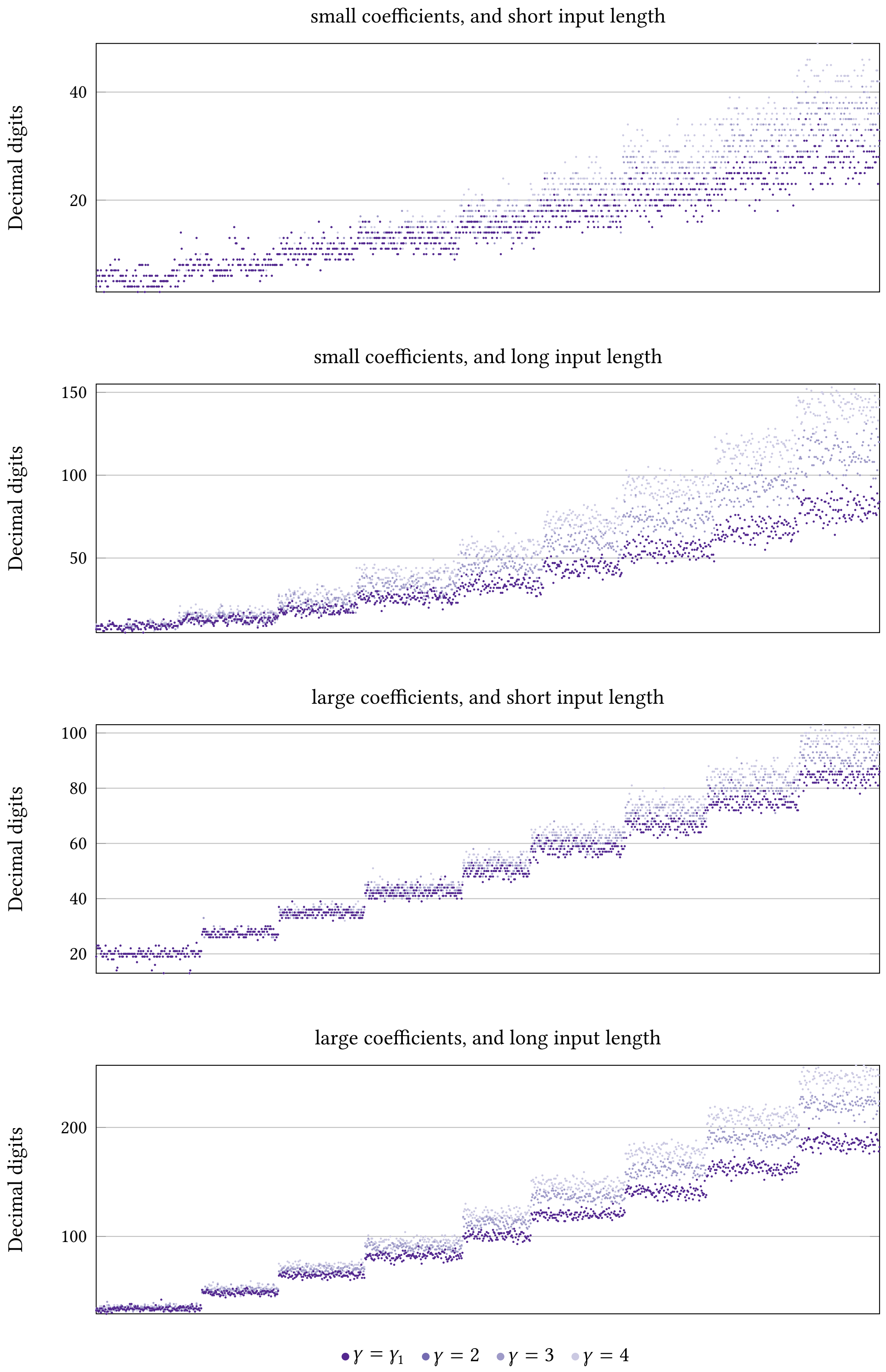


Figure 3.15: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$ (APSLQ only).

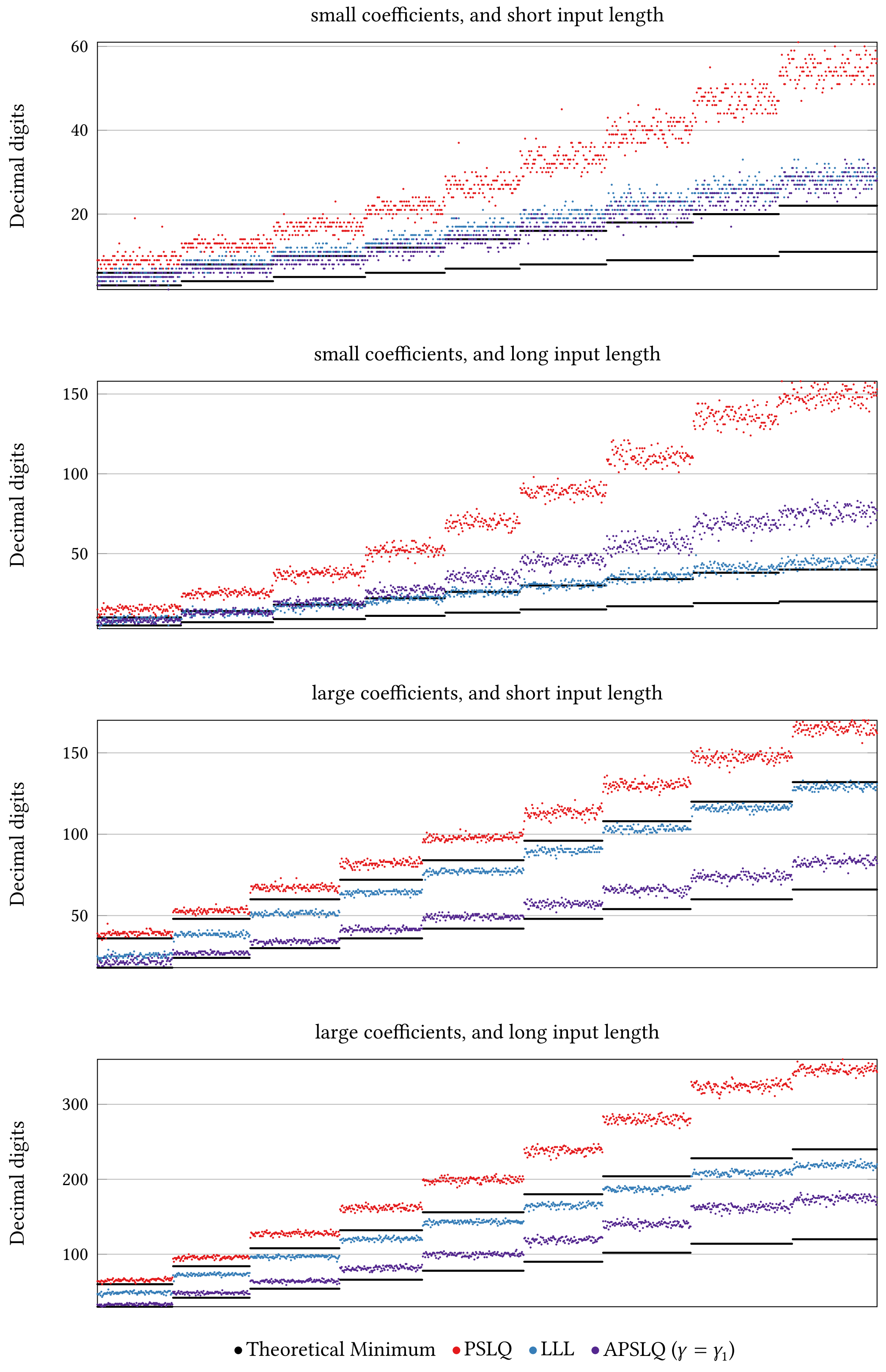


Figure 3.16: Precision required: $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$.

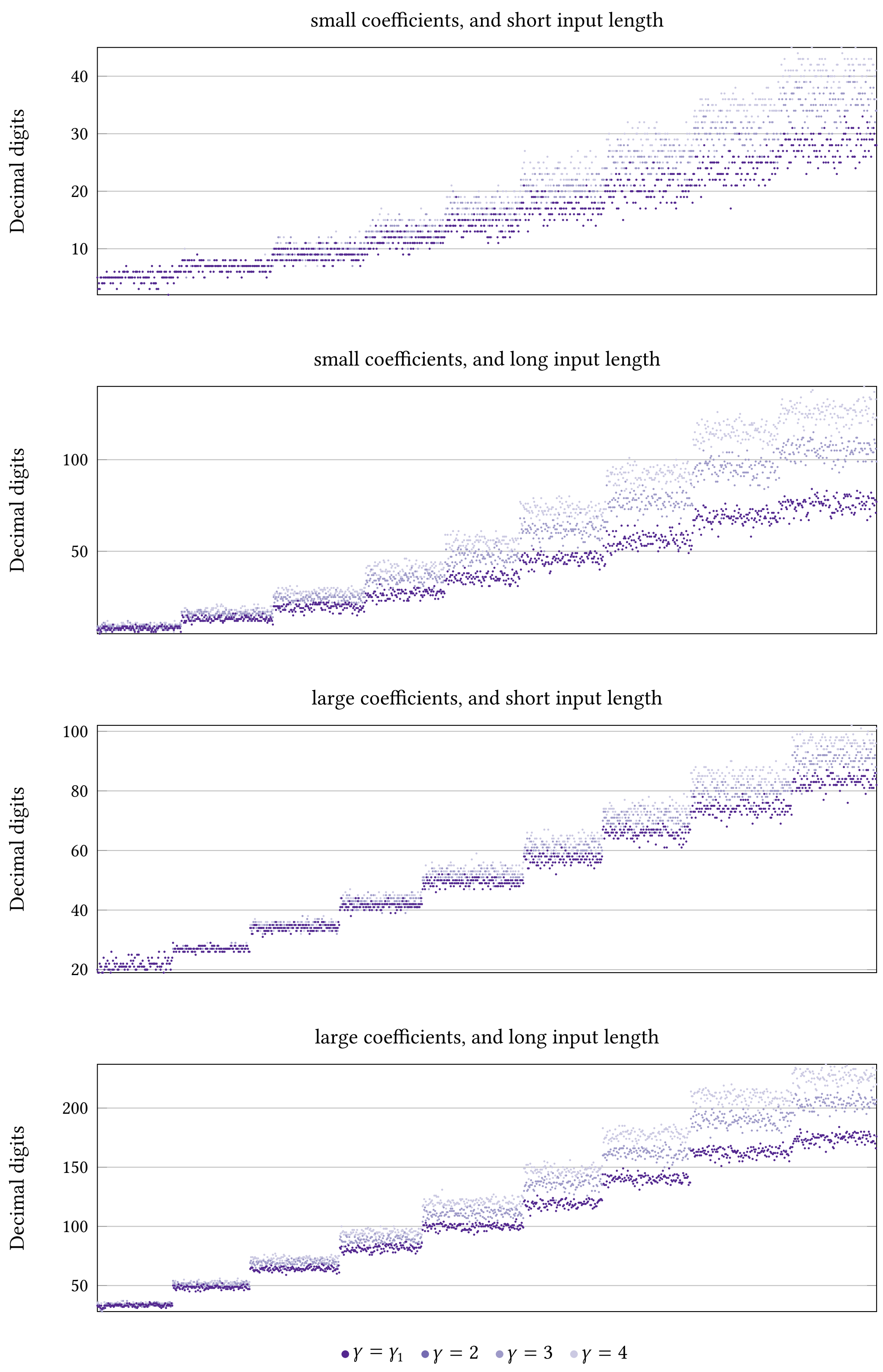


Figure 3.17: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$ (APSLQ only).

Timing

We present the graphs for $\mathbb{K} = \mathbb{Q}(\sqrt{-2})$ in Figs. 3.18 to 3.21, and note that they are illustrative of all cases, as presented in Figs. B.53 to B.80 on pages 198–225.

We see that LLL continues to be the slowest of the techniques, although the difference between it and PSLQ is less pronounced—closer to 1 order of magnitude. The smaller difference is likely related to the lower precision that the $\Lambda_x(N, D)$ lattice seems to grant.

The difference in computation times for the different γ values for APSLQ is much less clear in these cases. In no cases is there any indication of strata like we saw in the classical cases.

Interestingly we note that despite not being efficiently implemented our APSLQ implementation is consistently faster than LLL, and sometimes even faster than PSLQ. We expect that if properly implemented it should be the fastest (supported by the fact that it consistently requires significantly lower precision for the computations).

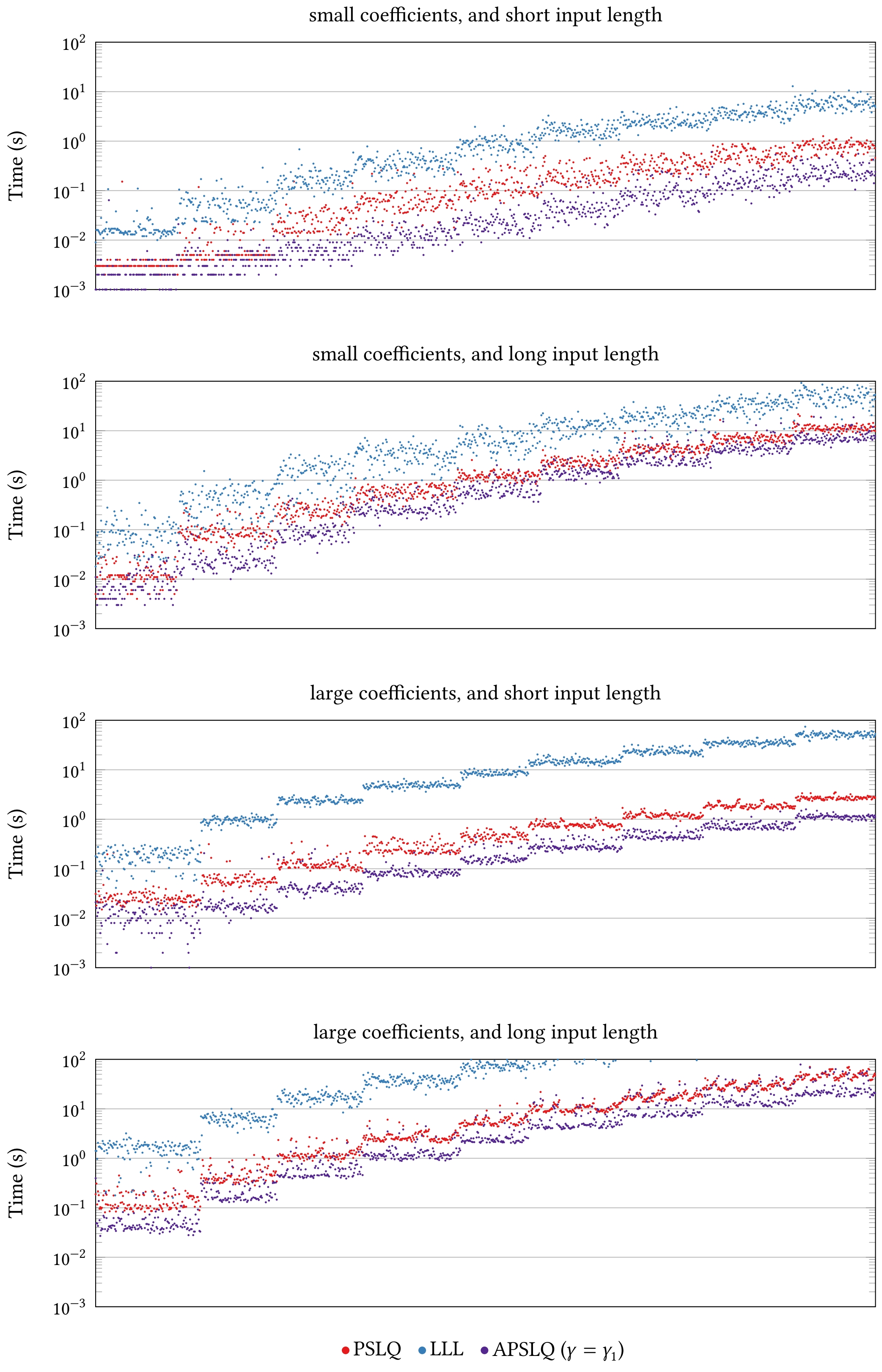


Figure 3.18: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$.

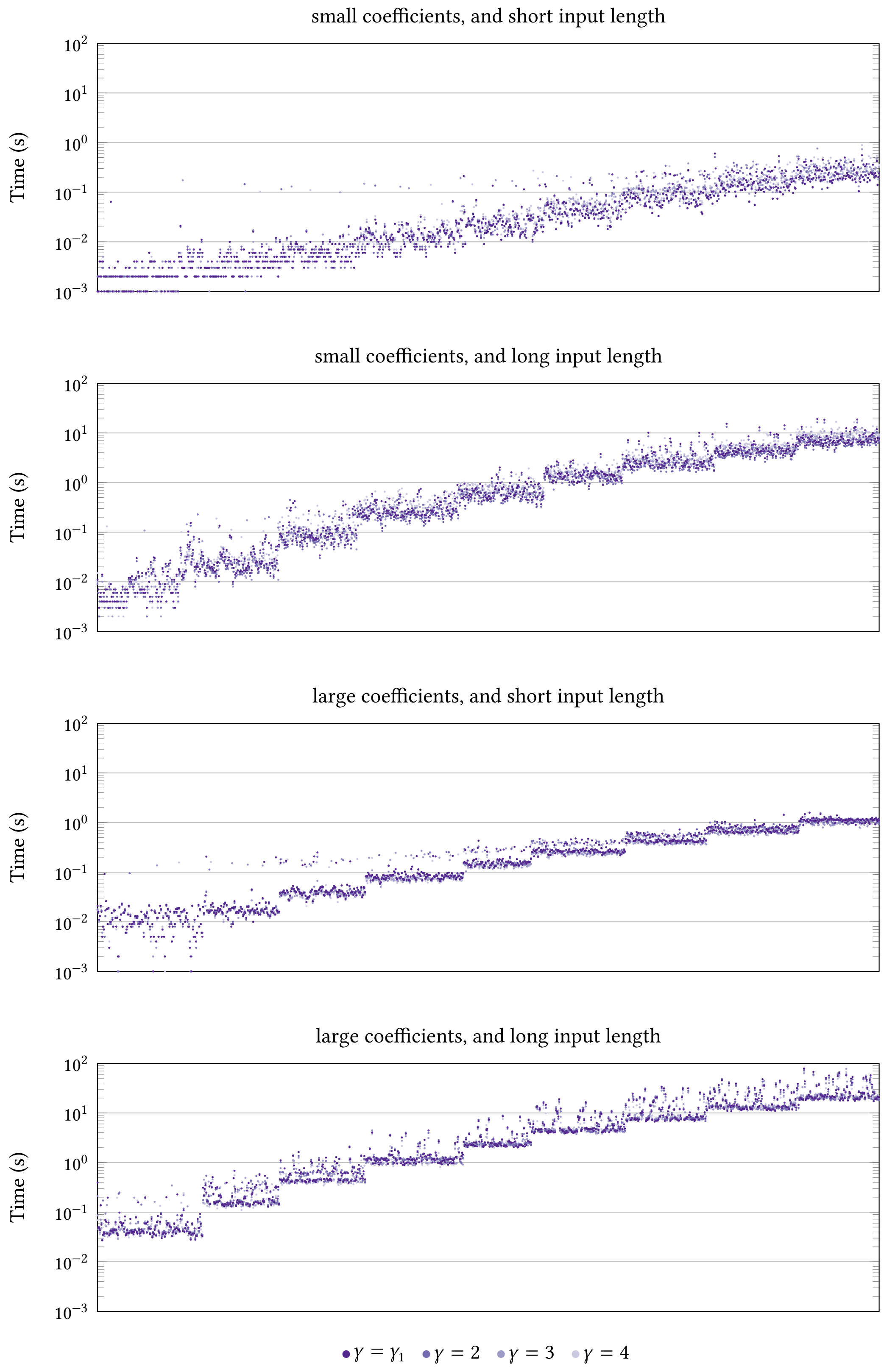


Figure 3.19: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$ (APSLQ only).

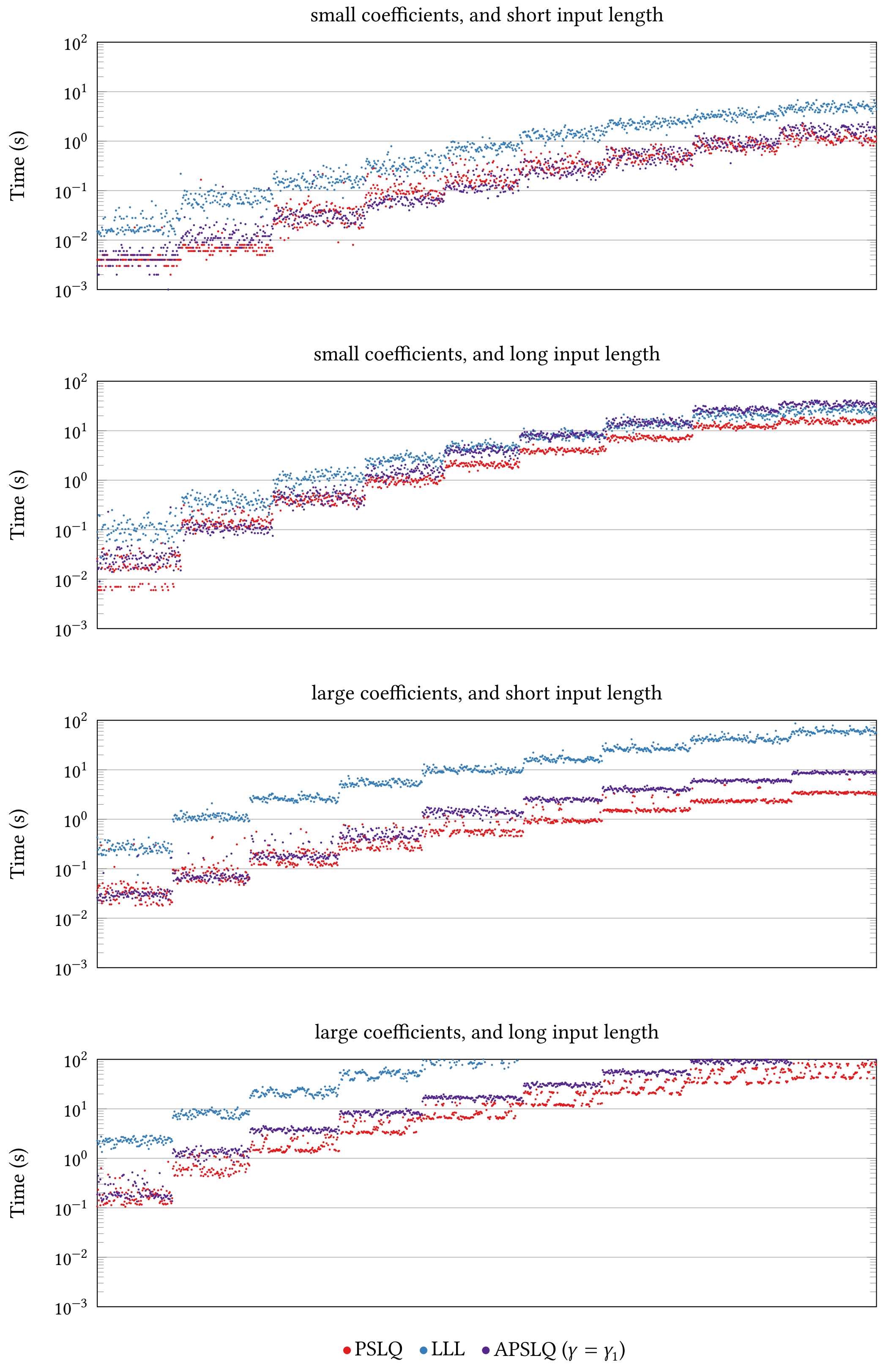


Figure 3.20: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$.

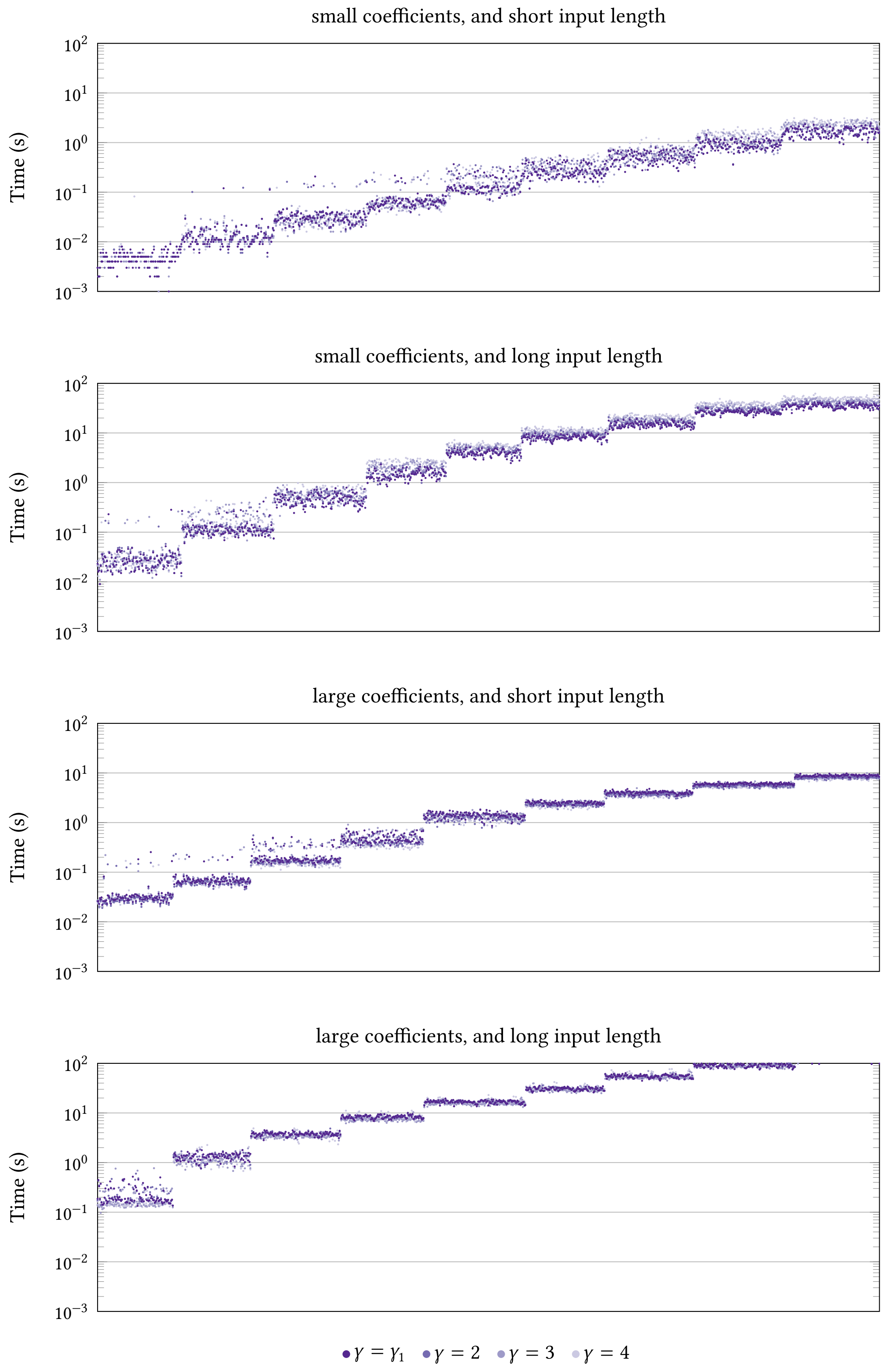


Figure 3.21: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$ (APSLQ only).

Part II

Douglas–Rachford

4 Dynamics of the Douglas-Rachford Method for Ellipses and p-Spheres

4.1 Preliminaries

Herein H is a Hilbert space. We will denote the induced norm by $\|\cdot\|$. The projection onto a proximal subset C of H is defined for all $x \in H$ by

$$P_C(x) := \left\{ z \in C \, : \, \|x - z\| = \inf_{z' \in C} \|x - z'\| \right\}$$

When C is closed and convex the projection operator P_C is single valued and firmly non-expansive. When C is a closed subspace it is also linear and self-adjoint. For additional information, see, for example, [13, Definition 3.7]. The reflection mapping through the set C is then defined by

$$R_C := 2P_C - I,$$

where I is the identity map on H.

Definition 4.1.1 (Douglas-Rachford Method). *For two closed sets A and B, and an initial point $x_0 \in H$, the Douglas-Rachford method generates a sequence $(x_n)_{n=1}^{\infty}$ as follows:*

$$x_{n+1} \in T_{A,B}(x_n) \quad \textit{where} \quad T_{A,B} := \frac{1}{2}(I + R_B R_A). \tag{4.1}$$

Figure 4.1 illustrates the construction of one iteration of the Douglas-Rachford method.

Notation 4.1.2. *Throughout, x_n, x_0 are as in Definition 4.1.1, A, B are closed. When the two sets A and B are clear from the context we will simply write T in place of $T_{A,B}$.*

Theorem 4.1.3 (Bauschke, Combettes, and Luke [14]). *Suppose $A, B \subseteq H$ are closed and convex with non-empty intersection. Given $x_0 \in H$ the sequence of iterates $T_{A,B}$ converges weakly to an $x \in \operatorname{Fix} T_{A,B}$ with $P_A(x) \in A \cap B$.*

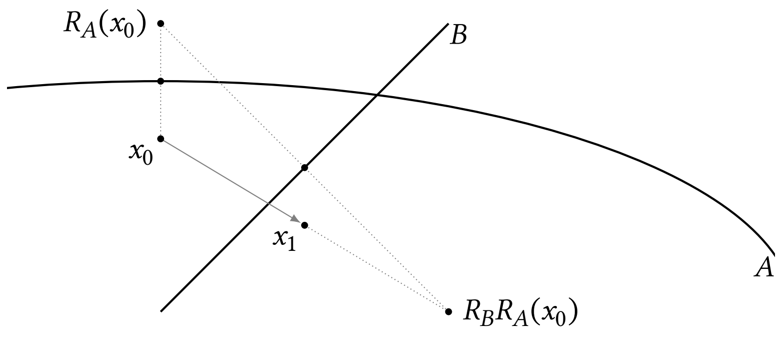


Figure 4.1: One iteration of the Douglas-Rachford method

In finite dimensions convergence in norm for convex sets is therefore assured. Notwithstanding the absence of a satisfactory theoretical justification, the Douglas-Rachford iteration scheme has been used to successfully solve a wide variety of practical problems in which one or both of the constraints are non-convex. Phase retrieval problems are one important instance, and the case of a line L and circle C in 2-dimensional Euclidean space—prototypical of such problems—was investigated by Borwein and Sims [25] as a specific case of the higher dimensional problem of a line and a sphere in Hilbert space.

Despite the seeming simplicity of the situation, the Douglas-Rachford method applied to L and C proved surprisingly difficult to analyse. Among the partial results obtained in the feasible case was local convergence to each of the two feasible points. Based on this and extensive computer experimentation, Borwein and Sims were led to ask whether this could be extended to convergence to one or other of the two intersection points for all starting points except those lying on a "singular set" S_0; the line of symmetry perpendicular to L and passing through the centre of C. Borwein and Aragón Artacho [2] established sizeable domains of attraction for each of the feasible points, and the global question was answered in the affirmative by Benoist [19] who obtained the result by constructing a suitable Lyapunov function, see figure 4.2.

The singular set S_0 is invariant under the Douglas-Rachford operator $T_{C,L}$. It contains period 2 points if and only if L passes through the centre of C in which case all the points of L inside C are period 2 points. When L is tangential to C all points on S_0 are fixed by $T_{C,L}$. For other positions of L the iterates exhibits periodic behaviours when rational commensurability is present, while in the absence of such commensurability the behaviours may be quite chaotic. See [25] for more details.

In order to gain further insights into the behaviour of the Douglas-Rachford algorithm in the case of non-convex constraint sets we consider two generalisations of a line and sphere (circle) in 2 dimensional Euclidean space, namely: that of a line together with an ellipse and that of a line together with a p-sphere.

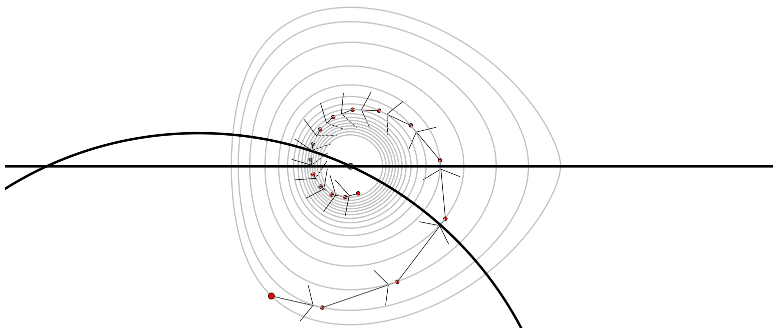


Figure 4.2: Douglas-Rachford on the 2-sphere and line showing the level sets of the Lyapunov function [19]

These seemingly innocuous generalisations, while open to exploration and local analysis about the feasible points, may be impossible to analyse in full. The singular set is no longer a simple curve but rather exhibits a complex (and fascinating) geometry involving a rich array of periodic points and associated domains of attraction. In the case of an ellipse and line we observe the appearance of higher order periodic points as the ellipticity is increased.

Definition 4.1.4 (Periodic points and domains). *The following terms, already used informally, help inform our discussion.*

1. *A point x is a* periodic point of period m *(or a* period m point*) if $T^m_{A,B}x = x$ (A period* 1 *point is simply a fixed point of $T_{A,B}$).*
2. *The* domain of attraction *(or* attractive domain*) for a period m point x is the set of all x_0 satisfying*
$$\lim_{k\to\infty} T^{km}_{A,B}(x_0) = x. \tag{4.2}$$
3. *A point x is* attractive *if its domain of attraction contains a neighbourhood of x.*
4. *The* singular set *consists of all points not belonging to a domain of attraction for any feasible point.*
5. *A period m point is said to be* repelling *if there exists a neighbourhood N_x of x such that for every $x_0 \in N_x\backslash\{x\}$ the sequence $(T^{km}_{A,B}(x_0))^\infty_{k=1}$ eventually lies outside of N_x.*

Of course if S is a domain of attraction for a period m point x then for $k = 1, 2, \cdots m - 1$ it follows that $T^k(S)$ is a domain of attraction for $T^k(x)$. This is a notable feature in many of our graphics, see for instance Figure 4.3.

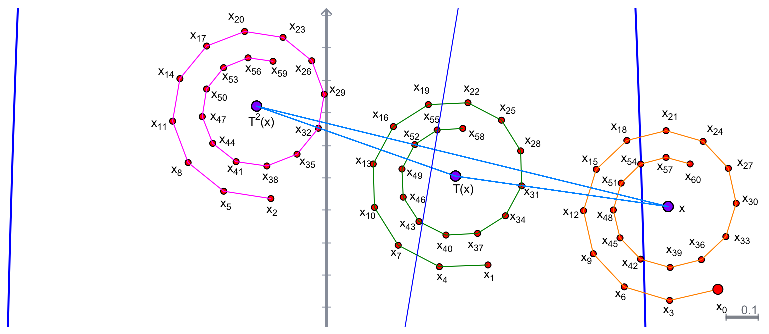


Figure 4.3: Domain of attraction for period 3 points in the case of E_6, L_8

4.1.1 Notation

By a suitable rotation and scaling of axes we may without loss of generality take our ellipse and p-sphere respectively to be

$$\begin{aligned} E_b &:= \left\{(x,y) \in \mathbb{R}^2 | \ \varphi_b(x,y) := x^2 + \left(\frac{y}{b}\right)^2 = 1\right\} \quad \text{and} \\ S_p &:= \left\{(x,y) \in \mathbb{R}^2 | \ \theta_p(x,y) := (x)^p + (y)^p = 1\right\}, \end{aligned} \tag{4.3}$$

and will write:

$$L_{m,\beta} := \{(x,y) \in \mathbb{R}^2 | y = mx + \beta\} \quad \text{and} \quad L_m := L_{m,0}.$$

When it is clear from the context what the parameters are we will simply write E, S or L respectively. Similarly, when the context makes it clear we will write T in place of $T_{E,L}$ or $T_{S,L}$.

4.1.2 Computation of Projections

For the case of the 2-sphere, the closest point projection has a simple closed form. For $x \neq 0$, $P_S(x) = x/\|x\|$. Such a simple closed form is immediately lost for any ellipse with $b \neq 1$ or any p-Sphere with $p \notin \{1,2\}$ because—where φ_b, θ_p are as in (4.3)—the induced Lagrangian problems

$$\begin{aligned} P_{E_b}(x) &= \left\{x' \ \mid \ \lambda \nabla d(x,\cdot)^2(x') = \nabla \varphi_b(x'), \quad \varphi_b(x') = 1\right\} \\ P_{S_p}(x) &= \left\{x' \ \mid \ \lambda \nabla d(x,\cdot)^2(x') = \nabla \theta_p(x'), \quad \theta_p(x') = 1\right\} \end{aligned}$$

yield implicit relations that no longer admit explicit solutions. Computation of the required projections necessitates the use of numerical methods.

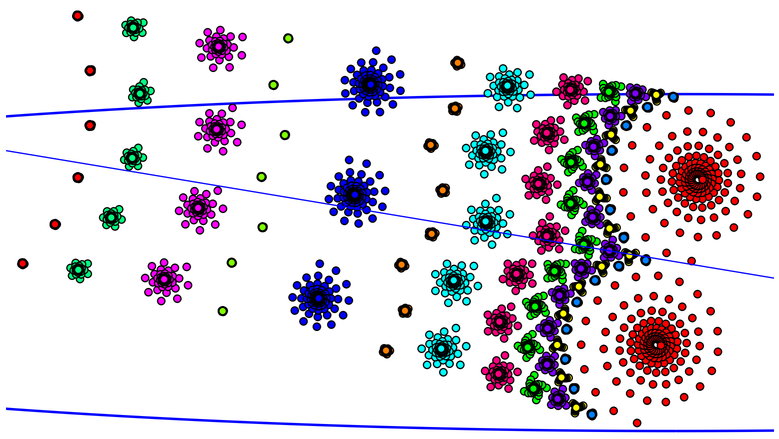


Figure 4.4: Partial domains of attraction of T_{E_8,L_6} for points of different periodicities

A description of the optimised function solvers used is available in the supplementary material [27]. Many of our implementations of these function solvers—for example, that used to generate Figure 4.3—employ the interactive geometry software *Cinderella*, available at `https://cinderella.de/`.

4.2 The Case of an Ellipse and a Line

In the case of an ellipse and a line, the singular set—in contrast to the case of a circle and a line—is no longer a simple curve, and appears to contain periodic points in many cases. For example, Figure 4.4 shows periodic points for T_{E_8,L_6} with attendant subsets of their attractive domains. The singular set is larger than suggested here (see Figure 4.10 for a more complete depiction).

For simplicity, we set the line intercept at 0 for our pictures and tables. It should be noted that, in contra distinction to the case of a circle and line, similar behaviour can be observed with nonzero intercepts (although symmetries are lost).

The number and periodicity of the points appear to be related to both the eccentricity of the ellipse, and the angle of the line. As the eccentricity is increased, we observe growth in both the number of periodic points and the maximum periodicity. Table 4.1, obtained experimentally using *Cinderella*, summarises our findings. Note that the method used required interactively moving a point in the geometry package, and visually observing the attractive domains. As such it is regrettably possible that some periodic points were missed, either because their domains of attraction were too small or they were not attractive points.

Table 4.1: Periods observed for attractive domains for various ellipse and line configurations

	E_2	E_3	E_4	E_5	E_6
L_1	2	2	2	2	2
L_2		2,3	2,3	2,3	2,3
L_3		2,3	2,3,5	2,3,4,5,7	2,3,4,5,7
L_4			2,3,5,7	2,3,4,5,7,9	2,3,4,5(×2),7,9
L_5			2,3	2,3,4,5,7,9,11,13	2,3,4,5,7,9,11,13
L_6				2,3,5,7	2,4,5(×2),7,9,11,13,15
L_7				3	2,3,4,5,7,9,11,13
L_8					2,3,5

Note: some periodicity's were observed in more than one domain.

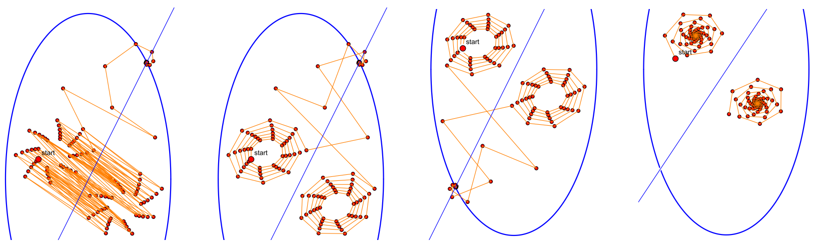


Figure 4.5: Sensitivity of behaviours to small changes in line slope

We can describe the period 2 points of T_{E_b,L_m} with a closed form that, while complicated to state, is quick to evaluate [27]. Determining period 2 points algebraically is useful for corroborating some of the behaviours we observe in *Cinderella*. However, the degree of complication associated with the analysis of even this simplest case of a non-fixed periodic point suggests that fully describing all behaviour globally with explicit forms would be an impractical undertaking. This, in part, led us to pursue the computer assisted evidence-gathering approach we describe in subsections 4.2.1.

The nature of the periodic points is also sensitive to small perturbations of the line. We can see above that for lines of small slope there are only a few attractive periodic points, and as the slope increases additional points with higher periodicity emerge. As the slope becomes large, some of the attractive domains appear to shrink in size until eventually the associated periodic point ceases to be attractive. This appears to be the eventual fate of all periodic points.

This sensitivity to perturbations can be seen in Figures 4.5 and 4.6. In the former we have connected every second iterate, and see how a small change in slope can affect which feasible point is converged to, as well as the appearance/disappearance of attractive domains. In the latter we plot every second iterate for T_{E_2,L_m} with 300 iterates, starting at $m = 2$ (top left) we slowly rotate the line until we have $m = 3/2$ (bottom right). Part of the line is visible in the bottom right corner of each frame. In the initial configuration, the subsequence of iterates started near to the periodic point are repelled from it. As we rotate

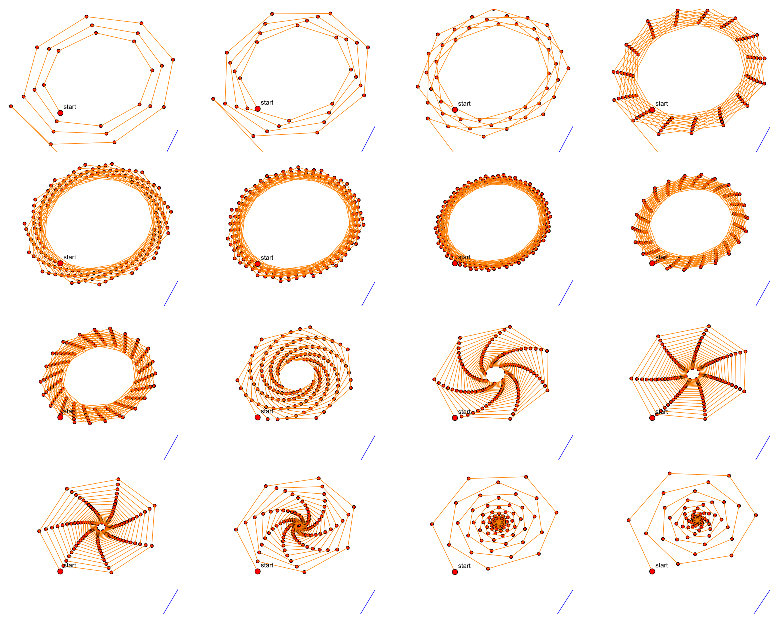


Figure 4.6: Evolution in behaviours near two period 2 points as the line slope is changed

the line, we see that the "speed" at which they are repelled decreases until eventually the periodic point becomes an attractive point instead of a repelling point.

4.2.1 Studying Convergence: Numerical Motivations

The complicated nature of the singular set precludes any possibility of constructing a Lyapunov function in any sizeable region about the feasible point. Indeed, attempts to even numerically construct the level curves such a function might have near a feasible point proved unstable. Instead we refine our numerical-graphical method of discovery. The method we used for Figure 4.4, though useful for discovery, is not, in itself, sufficient for fully understanding the behaviours, even for one specific ellipse and line. There are several reasons for this.

1. There may be other periodic points we cannot see because they are repelling or their attractive domains are too small.
2. The potential for numerical error is accentuated by the fact that the projection onto the ellipse is specified as the root function—induced by the Lagrangian system—whose calculation is, in some configurations, complicated by the presence of nearby incorrect roots.
3. This method of visualisation may be deceptive, as it precludes us from seeing accurately the extent and shape of attractive domains.

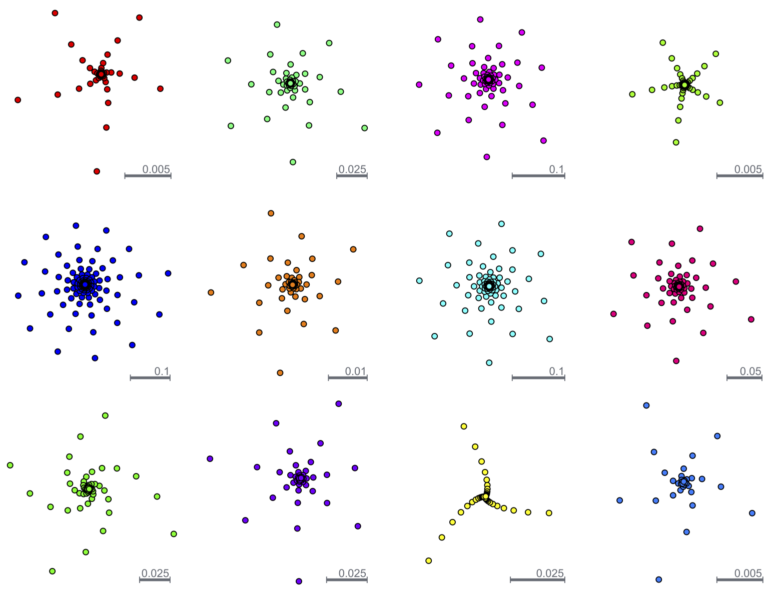


Figure 4.7: Close up view of the spirals in seen in Figure 4.4

As an example of the latter, notice how the patterns of the iterates in Figure 4.4 form orderly spirals. This lovely pattern seems to hold for all the cases we have looked at. If we zoom in on the spirals we see what look like twisting galaxies (see Figure 4.7). Intuition would suggest to us—incorrectly—that this is perhaps indicative of smooth boundaries for the domains.

4.2.2 Visualisation Through Parallelisation

Seeking clearer pictures with finer resolution, we implemented a new version of our code. In this new version we specify a resolution and for each pixel we compute the midpoint, calling it x_0. Once computed, the location of $x_{1,000}$ is checked against a list containing the feasible points and approximate periodic points, and the pixel is coloured according to which list member $x_{1,000}$ is nearest to.

The efficacy of this technique is demonstrated in Figures 4.8 and 4.9. The former clearly depicts the complex structure of the domains of attraction for the two feasible points in the case of T_{L_2,E_2} where two period 2 repelling points are present. Note the interweaving of the attractive domains near the repelling points. The latter shows the domains of attraction for T_{E_8,L_6} in the same region as shown in Figure 4.4.

This new implementation was written in such a way as to leverage the highly parallel nature of GPU devices, although it may also be run on regular CPUs. This allowed us to compute the colourings for many pixels simultaneously, reducing the time needed to produce the images and simultaneously affording us the ability to produce images with a

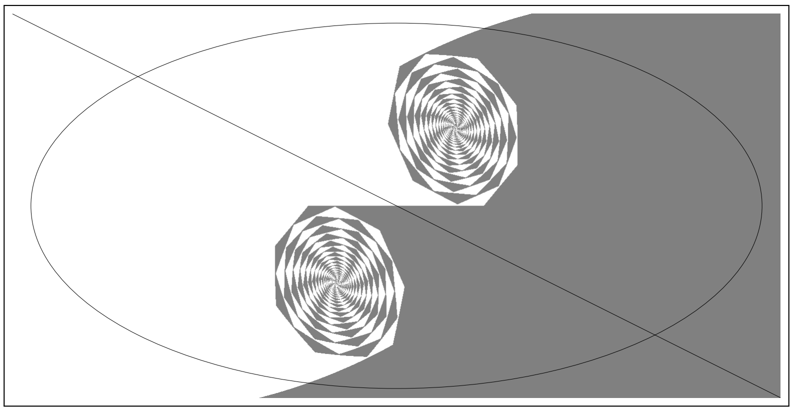


Figure 4.8: Domains of attraction for the two feasible points of T_{E_2,L_2}

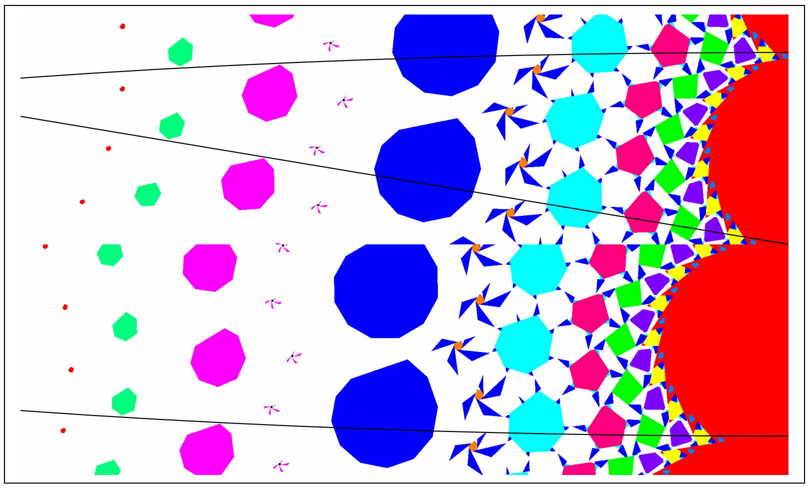


Figure 4.9: Domains of attraction for T_{E_8,L_6}. Compare with Figure 4.4

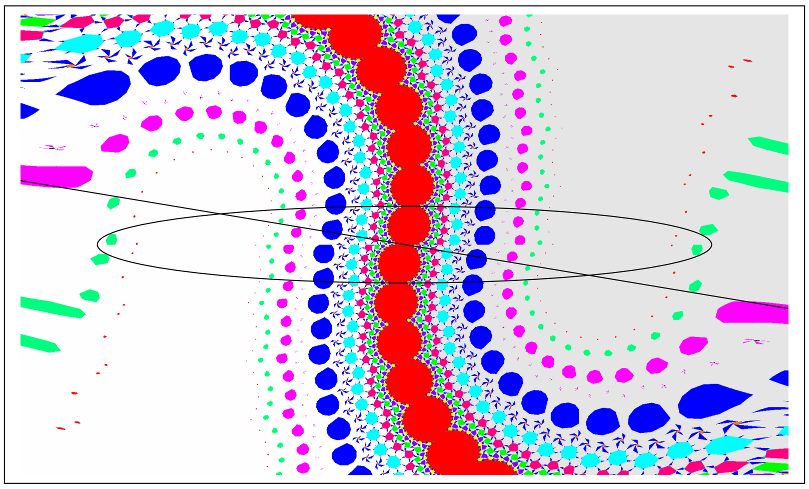


Figure 4.10: Domains of attraction for T_{E_8,L_6}

much finer resolution. With this method we were able to see the behaviour of the system over a larger area, as shown in Figure 4.10. Note that these images benefit greatly from colour coding of the domains.

4.2.3 Correctness and Reproducibility

We pause to discuss the reliability of the images produced. We do not know a priori where the periodic points are, and so we are at the mercy of the correctness of our numerical methods.

As an initial step we note that early pictures were produced by applying a continuous colour map to the region in question (of $\mathbb{R}^2$). After performing the Douglas–Rachford iterations each pixel was coloured based on it's final position, using this colour map. We saw the same shapes we see in the pictures presented herein, although not with the distinct, contrasting, and uniform colours. We note that these early pictures resulted, in the case of T_{E_8,L_6}, in the discovery of a set of attractive points which had hitherto been missed.

We have not yet detailed the means with which we compute the projection onto an ellipse. The supplementary material [27] (to the original paper [28] from which the results in this chapter were published) gives some details, but obscures the details of the actual ellipse projection stating only "numerically solve the Lagrangian system [...]". We present this information here to further our discussion.

To project onto the ellipse, we exploited symmetry in the ellipse, and considered only projection from a point in the positive quadrant. Given a point $(x, y) \in \mathbb{R}^2$ with $x \geq 0$ and $y \geq 0$ we compute the projection, P, by solving $f(z) = 0$ where

$$f(z) = \begin{cases} (b^4 - 2b^2 + 1)\,z^4 + (2b^2x - 2x)\,z^3 \\ \quad + (-b^4 + b^2y^2 + 2b^2 + x^2 - 1)\,z^2 + (-2b^2x + 2x)\,z - x^2 & \text{if } b > 1 \\ (b^4 - 2b^2 + 1)\,z^4 + (-2b^3y + 2by)\,z^3 \\ \quad + (-b^4 + b^2y^2 + 2b^2 + x^2 - 1)\,z^2 + (2b^3y - 2by)\,z - b^2y^2 & \text{if } b < 1 \end{cases}$$

and compute

$$P = \begin{cases} \left(z, b\sqrt{1 - x^2}\right) & \text{if } b > 1 \\ \left(\frac{1}{b}\sqrt{b^2 - y^2}, z\right) & \text{if } b < 1 \end{cases}$$

Observe that the function depends on the value of b (from the ellipse equation $x^2 + (y/b)^2 = 1$). In particular when $b > 1$ we solve for x and when $b < 1$ we solve for y.[1] The reason for this is that (as noted in the supplementary material [27]) the induced Lagrangian function for x has multiple zeros in the desired range when $b < 1$, but only a single zero when $b > 1$. Similarly the induced Lagrangian function for y has multiple zeros in the desired range when $b > 1$, but only a single zero when $b < 1$.

The function $f(z)$ is quite poorly behaved, and so to find the correct zero we perform five iterations of the bisection method before utilising Newton's method—starting at the lower bisection bound—to find the zero. We note that, as a simple means of checking, some pictures were produced using only the bisection method (iterating until the difference between the bounds was less than machine epsilon), and the images produced agreed with those produced using the hybrid method.

Motivated by concern regarding the complexity of both the computation method, and the code to implement it, a simpler projection calculation was devised. This new method exploited the parametric equation of the ellipse $(u, v) = (\cos(\theta), b\sin(\theta))$, and the fact that the tangent line to a point on the ellipse is parallel to the vector $(-\sin(\theta), b\cos(\theta))$.

For a given point $(x, y) \in \mathbb{R}^2$ with $x \geq 0$ and $y \geq 0$ we compute the projection, P, by solving $f(\theta) = 0$ where

$$\begin{aligned} f(\theta) &= \langle\, (x, y) - (\cos(\theta), b\sin(\theta)), (-\sin(\theta), b\cos(\theta))\,\rangle \\ &= by\cos(\theta) - x\sin(\theta) + (1 - b^2)\sin(\theta)\cos(\theta) \end{aligned}$$

[1]In the case that $b = 1$ we simply have a circle, and the projection is given by $(x, y)/\|(x, y)\|$.

and $\langle \cdot, \cdot \rangle$ denotes the usual dot product on $\mathbb{R}^2$. In other words, we search for the value of θ for which the line segment from $(u, v) = (\cos(\theta), b\sin(\theta))$ to (x, y) is perpendicular to the tangent line at (u, v).

The function is periodic with a period of 2π. The range of θ we are interested in is $0 \le \theta \le \pi/2$ and in this range there is exactly one zero. At $\theta = 0$ we have $f(0) = by$ and $f'(0) = 1 - x - b^2$ which is always negative. At $\theta = \pi/2$ we have $f(\pi/2) = -x$ and $f'(\pi/2) = b^2 - by - 1$ which may be positive or negative, depending on the value of y.

In addition to being much simpler, we found experimentally that this approach is significantly better behaved with regard to finding zeros. It needs no initial conditioning using the bisection method; Newton's method starting at $\theta = 0$ performed superbly. Observing the shape of the function in Cinderella we see that the zeros, of the function outside the range we are interested in never appear to be close to the zero we wish to find. We also note, in passing, that the function may have 2 or 4 zeros inside the periodic cycle depending on the value of y.

We did have one small issue, however. When producing images of the basins of T_{E_8,L_6} for over regions of the plane larger than any shown in this document, we observed strange discrepancies between the original function and the new function. After some exploration, we found that Newton's method was converging on a zero outside the desired range for points with sufficiently large y-value. Much trial and error eventually yielded the simple solution wherein we start our Newton iterations at either $\theta = 0$ or $\theta = \pi/2$ depending on which has the steeper negative tangent line to the function. Moreover, it is easy to show that the steeper negative gradient will correspond to $\theta = 0$ precisely when $2 - x \le 2b^2 - yb$.

With the above modified method, the images agree with those produced by the original. This greatly increases confidence in the correctness of the images, and of the numerical stability of the more complicated function $f(z)$. When considered with the original colour maps, and the discussion in Section 5.4.2 on page 128 we are extremely confident the images accurately portray the situation. Additionally, we note that the code to compute using this modified method is both smaller, and simpler to understand.

4.3 Line and p-sphere

Projections onto the 1-sphere can be determined explicitly, so exact analysis is possible. Consequently, much of the behaviour is readily determined in particular periodic points with periods higher than 2 are observed. When $p = 2$, we recover the circle for which the convergence properties are known, (see Section 4.1). Our observations from the case $p > 2$ suggest a conjecture: that when the line is not parallel to either of the axes there is at most one pair of periodic points and they are repelling.

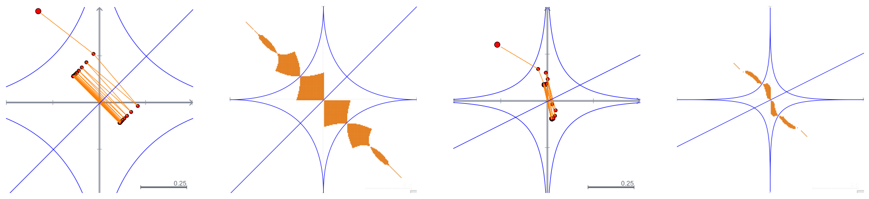


Figure 4.11: Subsequential convergence to—and attractive domains for—period 2 points. Left: $T_{S_{1/2},L_1}$, right: $T_{S_{1/3},L_{1/2}}$

For $1/n$-spheres where $n \geq 2$ is a natural number, we see the appearance of period 2 points with attendant local domains of attraction, examples are shown in Figure 4.11. It can be seen—proven easily—that, for the sphere $S_{1/n}$ with line L_1, any point $(-t, t)$ or $(t, -t)$ for $t \in (0, \frac{1}{2^n}]$ is a period 2 point. This continuum of period 2 points is analogous to what is observed in the case of a 2-sphere. More interesting is the apparent emergence of attractive domains which have nonzero measure.

These observations already hint at the larger measure and greater complexity of the singular manifold in the case of a line and p-sphere, when $p \neq 2$, compared to that for a line and 2-sphere. When we rotate the line to, say, $L_{1/2}$ there appear to be only finitely many period 2 points, but they are no longer constrained to lie in an affine submanifold.

4.4 A Theoretical Interlude: Local Convergence to a Feasible Point

Borwein and Sims [25] used the Perron theorem on the stability of almost linear difference equations [51, Corollary 4.7.2] to establish local convergence of the Douglas-Rachford algorithm, $x_{n+1} = T_{K,L}(x_n)$, to an isolated point $f \in L \cap K$ when L is a line and K is the (non convex) unit sphere in n-dimensional Euclidean space. We outline a strategy for extending this to the case when L is still a line, but K is a smooth hypersurface ($(n-1)$-manifold). We consider its application when L and K lie in $\mathbb{R}^2$; L is the line $\alpha x + \beta y = \gamma$, K is the ellipse E_b as in (4.3).

The strategy is to show that, in a neighbourhood of the feasible point f, the reflection in the supporting hyper-plane H_f to K at f—as in Figure 4.12—provides an o-order approximation to the reflection in K so that the Perron theorem can be applied to the system of difference equations corresponding to the Douglas-Rachford algorithm. Succinctly, we want

$$R_K(p) = R_{H_f}(p) + \Delta, \quad \text{where } \|\Delta\| = o(\|p - f\|) \text{ for } p \text{ sufficiently near } f.$$

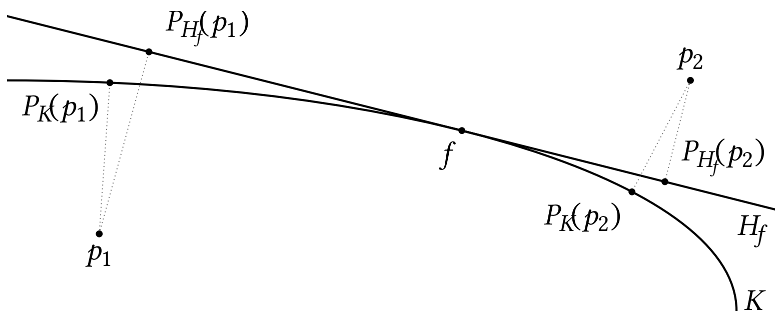


Figure 4.12: Approximation of P_K by P_{H_f} near f

For the Euclidean reflection this follows if $\|P_K(p) - P_{H_f}(p)\| = o(\|p - f\|)$. When this happens we have, for p in a neighbourhood of f,

$$\begin{aligned}
T_{K,L}(p) &= \frac{1}{2}[p + R_L(R_K(p))] \\
&= \frac{1}{2}\Big[p + R_L\Big(R_{H_f}(p) + \Delta\Big)\Big] \\
&= \frac{1}{2}\Big[p + R_{L-f}\Big(R_{H_f}(p) + \Delta - f\Big) + f\Big] \\
&= \frac{1}{2}\Big[p + R_{L-f}\Big(\Big(R_{H_f-f}(p-f) + f\Big) + \Delta - f\Big) + f\Big] \\
&= \frac{1}{2}\Big[p + R_{L-f}\Big(R_{H_f-f}(p-f)\Big) + R_{L-f}(\Delta) + f\Big], \quad \text{since } R_{L-f} \text{ is linear} \\
&= \frac{1}{2}\Big[(p-f) + R_{L-f}\Big(R_{H_f-f}(p-f)\Big)\Big] + \frac{1}{2}R_{L-f}(\Delta) + f
\end{aligned}$$

Thus we have that

$$T_{K,L}(p) = f + T_{(H_f-f),(L-f)}(p-f) + \Delta'$$

where $\Delta' = \frac{1}{2}R_{L-f}(\Delta)$ has $\|\Delta'\| = o\|p - f\|$ since R_{L-f} is a bounded linear operator.

Thus, by the theorem of Perron (see, [25] theorem 6.1 or [51] Corollary 4.7.2), the system of difference equations corresponding to the Douglas-Rachford algorithm for K and L,

$$x_{n+1} = T_{KL}(x_n)$$

is exponentially asymptotically stable at f (in particular $\|x_n - f\| \to 0$ for x_0 sufficiently near f) provided all the eigenvalues of the linear operator $T_{(H_f-f),(L-f)}$ have moduli less than one.

When M is a subspace, the projection P_M is linear (as is the case when $M = L - f$) and the Douglas-Rachford operator for N and M becomes

$$\begin{aligned} T_{N,M} &= \frac{1}{2}[I + (2P_M - I)(2P_N - I)] && (4.4)\\ &= 2P_M P_N - P_M - P_N + I && (4.5)\\ &= P_M P_N + (I - P_M)(I - P_N). && (4.6) \end{aligned}$$

When N is also a subspace (for instance when $N = H_f - f$) this may be written as

$$T_{NM} = P_M P_N + P_{M^\perp} P_{N^\perp}$$

where $^\perp$ denotes the orthogonal complement.

As a curiosity, we observe that if in the case of two subspaces we define a *twisted Douglas-Rachford operator* by $V_{NM} := P_M P_N + P_{N^\perp} P_{M^\perp}$, then, since $P_M P_{M^\perp} = P_N P_{N^\perp} = 0$, the iterates are $x_n = V_{NM}^n(x_0) = u_n + v_n$, where $u_{n+1} = P_M P_N(u_n)$ and $v_{n+1} = P_{N^\perp} P_{M^\perp}(v_n)$. The sequence of twisted Douglas-Rachford approximants is thus the sum of two sequences (u_n) and (v_n) resulting from the application of von Neumann's alternating projection algorithm to the pairs of subspaces M and N, and $N^\perp$ and $M^\perp$ respectively. Since the Friedrichs angle θ between M and N is the same as the angle between $M^\perp$ and $N^\perp$, the twisted Douglas-Rachford algorithm converges with the same rate as the von Neumann algorithm; namely at a linear rate proportional to $\cos^2\theta$ [34], the same as the rate exhibited by the standard Douglas-Rachford algorithm [18].

Now we consider the special case of an ellipse and a line. Without loss of generality we consider the ellipse E_b as in (4.3), and the line $L : \alpha x + \beta y = \gamma$, where $b \geq 1$, $\beta \geq 0$ and, to ensure the existence of $f = (x_0, y_0) \in L \cap E \cap |\mathbb{R}|^2$, either $\alpha \leq \gamma \leq \beta b$ or $\beta b < \gamma$ and $\alpha \geq \sqrt{\gamma^2 - \beta^2 b^2}$.

Following the strategy outlined above leads us to consider the eigenvalues of T for two lines through the origin

$$L_1 : \alpha x + \beta y = 0 \quad \text{and} \quad L_2 : Ax + By = 0$$

where, in our context, the latter line, being parallel to the tangent to E at f, has $A = x_0$ and $B = y_0/b^2$. It is readily verified that the orthogonal projection onto L_1 has the matrix

$$[P_{L_1}] = \frac{1}{\alpha^2 + \beta^2}\begin{pmatrix} \beta^2 & -\alpha\beta \\ -\alpha\beta & \alpha^2 \end{pmatrix} \tag{4.7}$$

with a matching expression for $[P_{L_2}]$. Substituting these expressions into $T_{L_2,L_1} = 2P_{L_1}P_{L_2} - P_{L_1} - P_{L_2} + I$ yields

$$[T_{L_1,L_2}] = \frac{\psi}{\Delta}\begin{pmatrix} \psi & \omega \\ -\omega & \psi \end{pmatrix}$$

where $\psi = \alpha A + \beta B$, $\omega = \alpha B - \beta A$ and $\Delta = (\alpha^2 + \beta^2)(A^2 + B^2)$, which has eigenvalues $\frac{\psi}{\Delta}(\psi \pm i\omega)$ with modulus squared equal to

$$\begin{aligned}\frac{\psi^2}{\Delta^2}\left(\psi^2 + \omega^2\right) &= \frac{(\alpha A + \beta B)^2\left((\alpha A + \beta B)^2 + (\alpha B - \beta A)^2\right)}{\left(\alpha^2 + \beta^2\right)^2\left(A^2 + B^2\right)^2} \\ &= \frac{(\alpha A + \beta B)^2}{\left(\alpha^2 + \beta^2\right)\left(A^2 + B^2\right)} < 1\end{aligned}$$

Thus, as expected, for any two lines intersecting in a single point the Douglas-Rachford algorithm with any starting point spirals exponentially to their common point.

Therefore the Douglas-Rachford algorithm for a line and an ellipse E is locally convergent at each of the feasible points f provided

$$\|P_E(p) - P_{H_f}(p)\| = o(\|p - f\|),$$

for all p in some neighbourhood of f.

To see this we follow an argument suggested by Asen Dontchev [35]. While we present the argument in the particular case of $E_b = \{x|\varphi_b(x) - 1 = 0\}$, the astute reader will observe that it applies to any smooth hypersurface $K := \{g(x) = 0\}$ at any point $f = (x_0, y_0) \in K$ at which the gradient ∇g is non-singular (true for the ellipse as $\nabla g(x) = (2x, 2y/b^2)$) and so applies to p-spheres except near the extreme points of the sphere when $0 < p \leq 1$.

We begin by noting that for the supporting hyperplane (tangent) to E at f

$$P_{H_f}(p) = f + P_{H_f - f}(p - f)$$

where

$$\left[P_{H_f - f}\right] = \frac{b^4}{b^4 x_0^2 + y_0^2}\begin{pmatrix} y_0^2/b^4 & -x_0y_0/b^2 \\ -x_0y_0/b^2 & x_0^2 \end{pmatrix}.$$

Next we observe that the nearest point projection $(u(p), v(p)) = P_E(p)$ at $p = (\zeta, \eta)$ is the solution of

$$\begin{aligned}&\text{minimise: } \frac{1}{2}\|P_E(p) - p\|^2 = \frac{1}{2}\left((u - \zeta)^2 + (v - \eta)^2\right) \\ &\text{subject to: } g(P_E(p)) = u^2 + \left(\frac{v}{b}\right)^2 - 1 = 0,\end{aligned}$$

which, since $\nabla g(P_E(p))$ is non-singular, is characterised via the method of Lagrange multipliers by $\nabla\frac{1}{2}\left((u-\zeta)^2+(v-\eta)^2\right)+\lambda\nabla g(P_E(p)) = 0$ together with $g(P_E(p)) = 0$, that is,

$$\begin{aligned} f_1 &: u-\zeta+2\lambda u = 0 \\ f_2 &: v-\eta+2\lambda\frac{v}{b^2} = 0 \\ g &: u^2+\left(\frac{v}{b}\right)^2-1 = 0. \end{aligned}$$

As an aside, this yields the implicit specification

$$P_E(\zeta,\eta) = \left(\frac{\zeta}{1+2\lambda}, \frac{b^2\eta}{b^2+2\lambda}\right), \quad \text{where } \frac{\zeta^2}{(1+2\lambda)^2}+\frac{b^2\eta^2}{\left(b^2+2\lambda\right)^2} = 1.$$

In order to apply the implicit function theorem to ensure that u, v and λ are differentiable functions of ζ and η in a neighborhood of f we require the Jacobian of the above system of equations with respect to the dependent variables, u, v and λ at f, $J(f)$, to be non-singular. Since $P_E(f) = f$ we see from the first (and second) equation that for $p = f$ the corresponding Lagrange multiplier is necessarily 0. Thus,

$$J(f) = \left.\frac{\partial(f_1,f_2,g)}{\partial(u,v,\lambda)}\right|_{(x_0,y_0,0)} = \begin{pmatrix} 1 & 0 & 2x_0 \\ 0 & 1 & 2y_0/b^2 \\ 2x_0 & 2y_0/b^2 & 0 \end{pmatrix},$$

more generally $J(f) = \begin{pmatrix} I & \nabla g(f)^T \\ \nabla g(f) & 0 \end{pmatrix}$, which is indeed non-singular, and in our case

$$J(f)^{-1} = \frac{b^4}{b^4x_0^2+y_0^2}\begin{pmatrix} y_0^2/b^4 & -x_0y_0/b^2 & x_0/2 \\ -x_0y_0/b^2 & x_0^2 & y_0/2b^2 \\ x_0/2 & y_0/2b^2 & -1/4 \end{pmatrix}.$$

Thus the implicit function theorem applies, yielding

$$\begin{pmatrix} [P_E'(f)] \\ [\lambda'(f)] \end{pmatrix} = \left.\frac{\partial(u,v,\lambda)}{\partial(\zeta,\eta)}\right|_{(x_0,y_0))} = J(f)^{-1}\frac{\partial(f_1,f_2,g)}{\partial(\zeta,\eta)} = J(f)^{-1}\begin{pmatrix} I \\ 0 \end{pmatrix},$$

whence $\left[P_E'(f)\right] = \frac{b^4}{b^4x_0^2+y_0^2}\begin{pmatrix} y_0^2/b^4 & -x_0y_0/b^2 \\ -x_0y_0/b^2 & x_0^2 \end{pmatrix}$ which we recognise as $\left[P_{H_f}'(f)\right]$ and we are able to conclude that near f

$$\|P_E(p)-P_{H_f}(p)\| = \|f+P_E'(f)(p-f)+\Delta - \left(f + P_{H_f-f}(p-f)\right)\| = \|\Delta\|$$

where $\|\Delta\| = o(\|p-f\|)$ as required.

Summarizing the above, we have the following local convergence result.

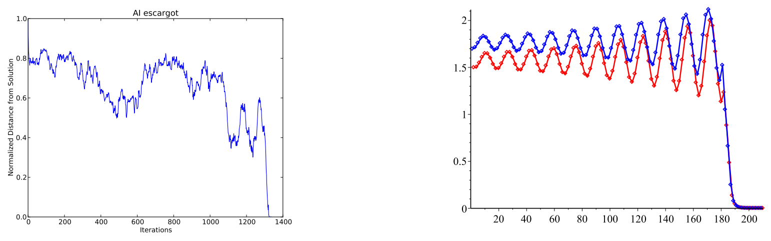


Figure 4.13: Distance from iterates to solution. Left: for a sudoku puzzle [4]. Right: for T_{E_2,L_2}

Theorem 4.4.1. *Let K be a hypersurface in $\mathbb{R}^n$ and $f \in K$ such that K is smooth in a neighbourhood of f and let H_f be the unique supporting hyperplane to K at f. If L is a line such that $L \cap H_f = \{f\}$, then the Douglas-Rachford algorithm $T_{K,L}$ is locally convergent to f.*

4.5 Important Lessons About Global Behaviour

What we have observed in our computer-assisted study of these two simple cases of a line together with an ellipse or a p-sphere is remarkably informative: it suggests likely explanations for the behaviour of the algorithm both for feasible and infeasible cases. We consider feasible cases first.

4.5.1 The Feasible Case

Aragón Artacho, Borwein, and Tam experimented with using the Douglas-Rachford method to solve Sudoku puzzles [4]. The left hand images of Figures 4.13 and 4.14, illustrate the distance to the solution by iterations of Douglas-Rachford for two different sudoku puzzles. First consider Figure 4.13. On the left, we see the algorithm struggle for a long period of time before finally converging. Compare this to the image on the right: for T_{E_2,L_2} with 210 iterates, distance of each iterate—to the particular feasible point the sequence converges to—is plotted. The subsequences x_{2k} and x_{2k-1} are coloured light and dark grey respectively. They correspond respectively to iterates landing in the domain of attraction for the left and right feasible points; see Figure 4.8. Without the geometric intuition gleaned from Figure 4.8, we would not know to colour these two subsequences distinctly, and the error plot would not reflect the behaviour as clearly. Once the iterates have finally climbed free of the influence of the repelling period 2 points, we see a sudden rapid convergence to the relevant feasible point.

Now consider Figure 4.14. At left, we see distance from the solution of the iterates for a different sudoku puzzle. This time the error stabilises after a time without any indication of impending convergence. At right, we see the iterates for T_{E_{14},L_9}. The iterates

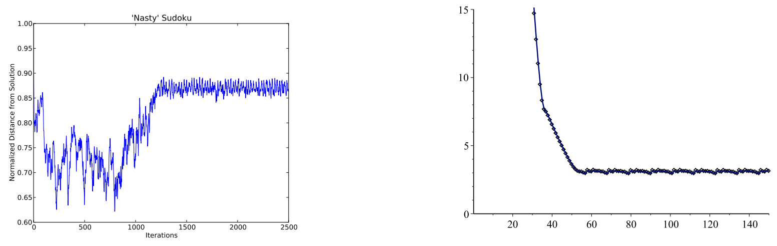


Figure 4.14: Distance from iterates to a solution. Left: a sudoku puzzle [26]. Right: T_{E_{14},L_9}

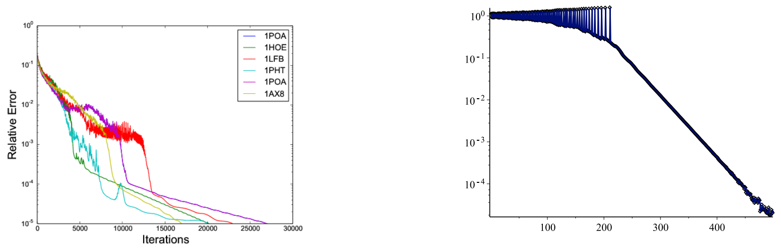


Figure 4.15: Distance of iterates from solution (scale logarithmic). Left: for five proteins [26], Right: for the iterates of T_{E_8,L_6} pictured in Figure 4.16

approach the ellipse before being pulled into the attractive domain for some period 11 points, preventing convergence.

Experiments have also been conducted using the Douglas-Rachford method to solve matrix completion problems associated with incomplete Euclidean distance matrices for protein mapping [3, 6, 26]. Consider Figure 4.15. The left image shows the relative error of iterates when solving the Euclidean distance matrices for various proteins. The right image shows the relative error for the iterates of T_{E_8,L_6} when the sequence of iterates is started near to domains of attraction.

The exact iterates used to generate this data are shown in Figure 4.16; they appear to trace out the shapes of the attractive domains for periodic points, narrowly avoiding

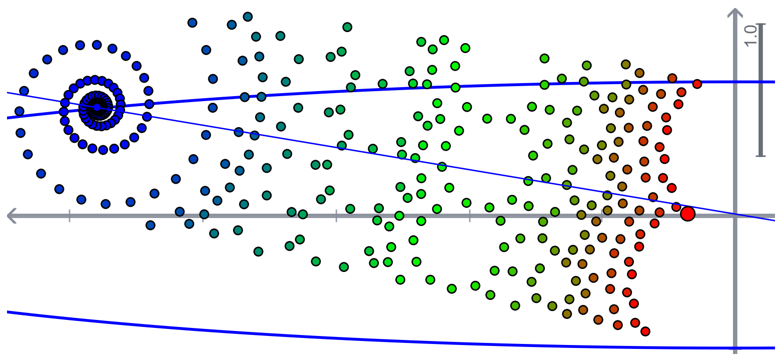


Figure 4.16: A convergent sequence of iterates of T_{E_8,L_6} traces the outline of the domains

them on their way to eventual convergence. Points started relatively close to domains of attraction for periodic points (as in the right hand side of Figure 4.13) appear to take longer to converge than those started elsewhere.

While we cannot say with any real certainty that the behaviour when solving these Euclidean distance matrices is analogous to iterates climbing away from repelling points or dodging and weaving between a nest of attractive domains, it is surprising that a system as simple as that of a line and ellipse can create behaviour so similar to that observed in far more complicated scenarios.

4.5.2 Infeasible Cases

For the infeasible cases of line and the p-sphere or ellipse, we observed that the iterates of the Douglas-Rachford algorithm appear to walk to infinity with a roughly linear step size. In both infeasible cases, it is possible to strictly separate the two sets in question. This led to the following theorem.

Theorem 4.5.1. *Let x_0 be a point in, and A and B be closed subsets of a Hilbert space H and let $x_n := T^n_{A,B}(x_0)$. Suppose one of the following hold:*

1. *A is compact and $\mathrm{co}(A)$ and $\mathrm{cl}(\mathrm{co}(B))$ are disjoint.*
2. *B is compact and $\mathrm{cl}(\mathrm{co}(A))$ and $\mathrm{co}(B)$ are disjoint.*

Then $\|x_n\|$ tends linearly to ∞ with a step size of at least $d(A, B)$.

Proof. Suppose that condition 1. applies. We can strictly separate $\mathrm{co}(A)$ and $\mathrm{cl}(\mathrm{co}(B))$ with a hyperplane $\mathbb{H} = f^{-1}(\alpha)$ for some linear functional f. See [30, Theorem 1.7] for details. By translation invariance, let $\alpha = 0$. Then $\mathbb{H}$ is a subspace, so we can uniquely describe any $x \in X$ as $x = h_x + y_x$ where $h_x \in \mathbb{H}$ and $y_x \in \mathbb{H}^\perp$. If, instead, condition 2. applies then we determine an f to separate $\mathrm{co}(B)$ and $\mathrm{cl}(\mathrm{co}(A))$ in the same manner.

In either case we are free to impose the following additional properties on f:

$$|f(x)| = |f(h_x + y_x)| = \|y_x\|_X \quad \text{for all } x \in X \tag{4.8}$$

$$f(x) < 0 \quad \text{for all } x \in A \tag{4.9}$$

$$f(x) > 0 \quad \text{for all } x \in B. \tag{4.10}$$

Equations (4.8), (4.9), and (4.10) imply that $f(x) \geq d(A,B)$ for all $x \in B - A$. Now

$$\begin{aligned} x_{n+1} - x_n = \frac{R_B(R_A(x_n)) + x_n}{2} - x_n &= \frac{R_B(R_A(x_n)) - x_n}{2} \\ &= \frac{2P_B(R_A(x_n)) - R_A(x_n) - x_n}{2} \\ &= \frac{2P_B(R_A(x_n)) - (2P_A(x_n) - x_n) - x_n}{2} \\ &= P_B(R_A(x_n)) - P_A(x_n) \in B - A. \end{aligned}$$

Now $x_{n+1} - x_n \in B - A$ implies that $f(x_{n+1} - x_n) \geq d(A,B)$. Thus we have that, for all n, $f(x_{n+1}) \geq d(A,B) + f(x_n)$. This shows that $\|x_n\|_n \to \infty$ with a linear step size of at least $d(A,B)$. ☐

From this result we obtain the following corollary, the computer-assisted discovery of which motivated the pursuit of the more general Theorem.

Corollary 4.5.2. *In the infeasible case of a line L with an ellipse E or a p-sphere S, we have that $\|x_n\| \to \infty$ with a linear step size greater than or equal to $d(E,L)$ or $d(S,L)$ respectively.*

Using these results and the following remark, we can naturally extend some of the convex theory to the non-convex case.

Remark 4.5.3. As a consequence of [17, Theorem 4.5] we have, for convex subsets U, V of a Hilbert space H, with $U \cap (v + V) \neq \emptyset$, where $v = P_{\mathrm{cl}(\mathrm{ran}(\mathrm{Id} - T_{U,V}))}(0)$ is the *minimal displacement vector*, and for $x \in X$, that $(P_U T_{U,V}^n x)_n$ converges weakly to a point in $U \cap (v + V)$.

We extend this result in our context using Theorem 4.5.1.

Theorem 4.5.4. *Let A, B be the respective boundaries of two disjoint, compact, convex sets U, V in a Hilbert space H (in $\mathbb{R}^n$ it suffices to have only one compact set but the other must be closed) so that A, B satisfy the requirements of Theorem 4.5.1. Let $x_{n+1} = T_{A,B}(x_n)$ and $v := P_{\mathrm{cl}(\mathrm{ran}(\mathrm{Id} - T_{U,V}))}(0)$, the uniquely defined element in $\mathrm{cl}(\mathrm{ran}(\mathrm{Id} - T_{U,V}))$ such that $\|v\| = \inf_{x \in X} \|x - T_{U,V}x\|$, and let $v' = P_{\mathrm{cl}(\mathrm{ran}(\mathrm{Id} - T_{A,B}))}(0)$, the uniquely defined element in $\mathrm{cl}(\mathrm{ran}(\mathrm{Id} - T_{A,B}))$ such that $\|v'\| = \inf_{x \in X} \|x - T_{A,B}x\|$. Then, for $x \in X$, we have that $(P_A T_{A,B}^n x)_n$ converges weakly to a point in $A \cap (v' + B)$.*

Proof. By the closure and compactness we have attainment of elements which minimise the distance. Thus $A \cap (v' + B) = U \cap (v + V) \neq \emptyset$ where v is defined as in Remark 4.5.3.

Let f be defined as in Theorem 4.5.1 so that $f(u) < 0$ for all $u \in U$. Then the sequence $f(x_n)$ is monotone increasing and so there exists some $n' \in \mathbb{N}$ such that $f(x_n) \geq 0$ for all $n \geq n'$. Suppose $n \geq n'$. Then we have that $x_n \notin U$. Thus $P_A(x_n) = P_U(x_n)$, and so $R_A(x_n) = R_U(x_n)$. We also have that $R_A(x_n) \notin V$. Thus $P_B(R_A(x_n)) = P_V(R_A(x_n))$, and so $R_B(R_A(x_n)) = R_V(R_A(x_n))$. Thus we have that $T_{A,B}(x_n) = T_{U,V}(x_n)$, and so

$$(P_A T_{A,B}^{n'+n} x)_n = (P_A T_{A,B}^{n} x_{n'})_n = (P_A T_{U,V}^{n} x_{n'})_n \tag{4.11}$$

for all $n \in \mathbb{N}$. We have from Remark 4.5.3 that the sequence on the right converges to a point y in $U \cap (v + V)$. □

5 Computing Intersections of Implicitly Specified Plane Curves

5.1 Preliminaries

We investigate the problem of computing a point in the intersection of two analytic plane curves specified implicitly by $f((x,y)) = 0$ and $g((x,y)) = 0$ for $(x,y) \in \mathbb{R}^2$ which we often identify with the complex plane $\mathbb{C}$. We will also identify the curves themselves with their respective graphs $\mathcal{G}_f := \{(x,y) \in \mathbb{R}^2 : f((x,y)) = 0\}$ and $\mathcal{G}_g := \{(x,y) \in \mathbb{R}^2 : g((x,y)) = 0\}$. Regarding the graphs as constraint sets our problem becomes the non convex feasibility problem of finding a point (x,y) in $\mathcal{G}_f \cap \mathcal{G}_g$. To avoid degeneracies we assume that (x,y) is an isolated point of $\mathcal{G}_f \cap \mathcal{G}_g$ at which the curves have contact of order zero (that is: $\nabla f((x,y))$ and $\nabla g((x,y))$ are both non-zero and not parallel).

In order to better exploit the analytic nature of the curves, we propose to solve the feasibility problem using an adaptation of the Douglas-Rachford algorithm (also known as *reflect-reflect-average*) [36] in which Euclidean reflections are replaced by Schwarzian reflections.

The Schwarzian reflection of a point $z \in \mathbb{C}$ in an analytic curve K [[61] pages 254 to 257] is $\mathcal{R}_K = \overline{S_K(z)}$, where S_K is the *Schwarz function* for K; an analytic function such that $S_K(z) = \bar{z}$ for all $z \in K$.

The Schwarz function for $K : k(x,y) = 0$ can often by found by substituting $\frac{z+\bar{z}}{2}$ for x and $\frac{z-\bar{z}}{2i}$ for y in the specification of K and solving for $\bar{z}$.

Example 5.1.1. *The Schwarz function for the ellipse $E : x^2 + \left(\frac{y}{b}\right)^2 = 1$ can be found by solving*

$$\left(\frac{z+\bar{z}}{2}\right)^2 + \left(\frac{z-\bar{z}}{2ib}\right)^2 = 1$$

for $\bar{z}$, yielding

$$S_E(z) = \frac{1}{1-b^2}\left[\left(1+b^2\right)z - 2b\sqrt{z^2-1+b^2}\right].$$

So, the Schwarzian reflection in E is given by

$$\mathscr{R}_E(z) = \overline{S_E(z)} \; = \frac{1}{1-b^2}\left[(1+b^2)\bar{z} \; - \; 2b\sqrt{\bar{z}^2 - 1 + b^2}\right].$$

Similarly, it is readily verified that the Schwarzian reflection in the unit circle $|z| = 1$ is the same as the inversion $z \to z/|z|^2$. A comparison of Euclidean and Schwarzian reflection in the case of the circle can be seen in figure 5.1. For the line Ł : $\alpha x + \beta y = 0$ Schwarzian reflection coincides with Euclidean reflection in Ł; that is the linear transformation with matrix

$$[\mathscr{R}_Ł] = \frac{1}{\alpha^2 + \beta^2}\begin{pmatrix} \beta^2 - \alpha^2 & -2\alpha\beta \\ -2\alpha\beta & \alpha^2 - \beta^2 \end{pmatrix}. \tag{5.1}$$

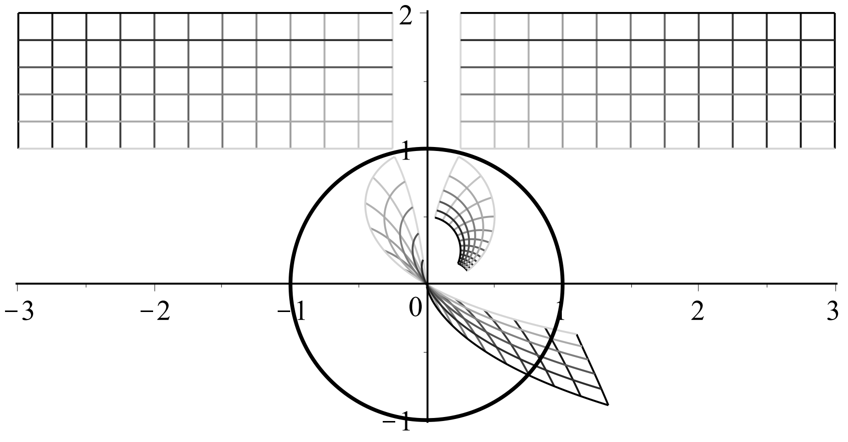


Figure 5.1: Euclidean (left) and Schwarzian (right) reflection.

For more details on Schwarz functions the interested reader is referred to Davis [33]

The Douglas-Rachford algorithm using Schwarzian reflections for finding a point in $\mathscr{G}_f \cap \mathscr{G}_g$ from a given initial point x_0 is the iterative scheme

$$x_{n+1} \;=\; T(x_n), \tag{5.2}$$

where $T = T_{\mathscr{G}_f\mathscr{G}_g}$ is the Douglas-Rachford operator

$$T_{\mathscr{G}_f\mathscr{G}_g} \;=\; \frac{1}{2}\left(I \;+\; \mathscr{R}_{\mathscr{G}_g}\mathscr{R}_{\mathscr{G}_f}\right). \tag{5.3}$$

5.2 Local Convergence of the Modified Douglas-Rachford Algorithm

For starting points x_0 sufficiently near to $p \in \mathscr{G}_f \cap \mathscr{G}_g$ we apply the theorem of Perron [see, [25] theorem 6.1 or [51] Corollary 4.7.2] to the system of difference equations in (5.2) to show the sequence of Douglas-Rachford iterates converges at a linear rate to p (that is, the iteration scheme (5.2) is exponentially asymptotically stable at p).

To ensure the conditions of Perron's theorem are satisfied we need to show:

(i) the Douglas-Rachford operator (5.3) is almost linear about p. That is, for x near p

$$T_{\mathscr{G}_f\mathscr{G}_g}(x) \;=\; p \;+\; L_{\mathscr{G}_f\mathscr{G}_g}(x-p) \;+\; \Delta, \tag{5.4}$$

where $L_{\mathscr{G}_f\mathscr{G}_g} : \mathbb{R}^2 \to \mathbb{R}^2$ is a linear operator and $\|\Delta\| = o(\|x-p\|)$.

and

(ii) both eigenvalues of $L_{\mathscr{G}_f\mathscr{G}_g}$ have modulus less than 1.

Proof of (i)
To prove (i) it suffices to show for an analytic curve K that near a point p in K the Schwarz function

$$S_K(x) \;=\; S_{H_K(p)}(x) \;+\; \Delta, \tag{5.5}$$

where $H_K(p)$ is the tangent (supporting hyperplane) to K at p and $\Delta \;=\; \Delta(K,p,x)$ has $\|\Delta\| = o(\|x-p\|)$. As then, for x in a neighbourhood of p we have

$$\begin{aligned}
T_{\mathscr{G}_f\mathscr{G}_g}(x) &= \frac{1}{2}\Big[x + R_{\mathscr{G}_g}\Big(R_{\mathscr{G}_f}(x)\Big)\Big] \\
&= \frac{1}{2}\Big[x + R_{\mathscr{G}_g}\Big(R_{H_{\mathscr{G}_f}(p)}(x) + \overline{\Delta'}\Big)\Big] \\
&= \frac{1}{2}\Big[x + R_{H_{\mathscr{G}_g}(p)}\Big(R_{H_{\mathscr{G}_f}(p)}(x) + \overline{\Delta'}\Big) + \overline{\Delta''}\Big] \\
&= \frac{1}{2}\Big[x + p + R_{H_{\mathscr{G}_g}(p)-p}\Big(\Big(p + R_{H_{\mathscr{G}_f}(p)-p}(x-p) + \overline{\Delta'}\Big) - p\Big) + \overline{\Delta''}\Big] \\
&= \frac{1}{2}\Big[x + p + R_{H_{G_g}(p)-p}\Big(R_{H_{G_f}(p)-p}(x-p)\Big) + R_{H_{\mathscr{G}_g}(p)-p}(\Delta') + \Delta''\Big], \\
&\qquad \text{since } R_{H_{\mathscr{G}_g}(p)-p} \text{ is linear} \\
&= \frac{1}{2}\Big[x + p + R_{H_{\mathscr{G}_g}(p)-p}\Big(R_{H_{\mathscr{G}_f}(p)-p}(x-p)\Big)\Big] + \frac{1}{2}\Big(R_{H_{\mathscr{G}_g}(p)-p}(\Delta') + \Delta''\Big)
\end{aligned}$$

So, as required

$$T_{\mathscr{G}_f\mathscr{G}_g}(p) \;=\; p \;+\; L_{\mathscr{G}_f\mathscr{G}_g}(x-p) + \Delta^*$$

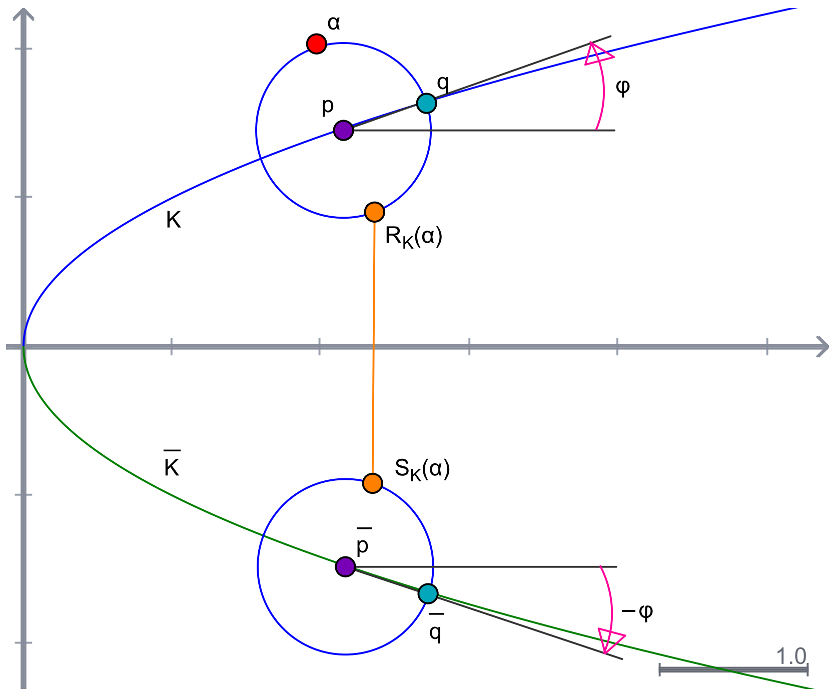


Figure 5.2: Construction of a Schwarz function and Schwarzian reflection

where $L_{\mathscr{G}_f\mathscr{G}_g} := T_{\left(H_{\mathscr{G}_g}(p)-p\right)\left(H_{\mathscr{G}_f}(p)-p\right)}$ and, since $R_{H_{\mathscr{G}_g}(p)-p}$ is a bounded linear operator, $\Delta^* = \frac{1}{2}\left(R_{H_{\mathscr{G}_g}(p)-p}(\Delta') + \Delta''\right)$ has $\|\Delta^*\| = o(\|x-p\|)$.

To establish (5.5) we argue as follows.

Since $S_K(z)$ is analytic it may be expressed as a Taylor series about $p \in K$, so

$$S_K(z) \;=\; S_K(p) \;+\; S_K'(p)(z-p) \;+\; \Delta = \overline{p} \;+\; e^{-2\phi i}(z-p) \;+\; \Delta,$$

where $|\Delta| = O(|z-p|^2)$ and ϕ is the angle between $H_K(p)$, the tangent to K at p, and the real axis, refer to figure 5.2, from which we infer $|S_K'(p)| = 1$ and $\arg\big(S_K'(p)\big) = -2\varphi$, also see [61], page 255.

Thus

$$\mathscr{R}_K(z) \;=\; p \;+\; e^{2\phi i}\overline{(z-p)} \;+\; \overline{\Delta} \;=\; \overline{(z-p)e^{-\phi i}}e^{\phi i} \;+\; p \;+\; \overline{\Delta}$$

which we recognise as

$$\mathscr{R}_K(z) \;=\; R_{H_K(p)}(z) \;+\; \overline{\Delta} \quad \text{(see [61], exercise 30(i), page 265).}$$

We now turn to the *Proof of (ii)*
$L_{\mathscr{G}_f\mathscr{G}_g}$ is the Douglas-Rachford operator for two non parallel lines; the translated tangents

$H_{\mathscr{G}_f}(p)-p$ and $H_{\mathscr{G}_g}(p)-p$, meeting at the origin. We will take these to be $Ł_1 : \alpha x+\beta y = 0$ and $Ł_2 : Ax + By = 0$ so by (5.3)

$$T_{Ł_1Ł_2} = \frac{1}{2}\left(I + \mathscr{R}_{Ł_2}\mathscr{R}_{Ł_1}\right)$$

which, via (5.1), has matrix

$$[T_{Ł_1Ł_2}] = \frac{\psi}{\Delta}\begin{pmatrix} \psi & \omega \\ -\omega & \psi \end{pmatrix},$$

where $\psi = \alpha A+\beta B$, $\omega = \alpha B-\beta A$ and $\Delta = (\alpha^2+\beta^2)(A^2+B^2)$. The eigenvalues of $[T_{Ł_1Ł_2}]$ are $\frac{\psi}{\Delta}(\psi \pm i\omega)$ both of which have modulus squared equal to

$$\begin{aligned}\frac{\psi^2}{\Delta^2}\left(\psi^2+\omega^2\right) &= \frac{(\alpha A+\beta B)^2\left((\alpha A+\beta B)^2+(\alpha B-\beta A)^2\right)}{\left(\alpha^2+\beta^2\right)^2\left(A^2+B^2\right)^2} \\ &= \frac{(\alpha A+\beta B)^2}{\left(\alpha^2+\beta^2\right)\left(A^2+B^2\right)} \\ &< 1\end{aligned}$$

as required.

Thus, the Douglas-Rachford algorithm using Schwarzian reflections applied near a simple intersection of two plane analytic curves yields a sequence of iterates that exponentially spirals to the intersection point.

The astute reader will see that if we employ Euclidean reflections in the Douglas-Rachford algorithm our techniques and results still apply and, provided we move to a multi-set version of the Douglas-Rachford algorithm (see for example [23]), readily extend to a family of curves in higher dimensional space, where each curve is specified as an intersection of hyper-surfaces. In this case local convergence is assured by arguments similar to those found in [28]. The reversion to Euclidean reflections is necessary as Schwarzian reflection for curves in a space of dimension greater than two is no longer defined.

5.3 Alternative Approaches to the Problem

One "classical" approach to finding an intersection of two implicitly specified plane analytic curves; $f(x,y) = 0$ and $g(x,y) = 0$ would be to compute a zero of

$$F : \mathbb{R}^2 \to \mathbb{R}^2 : (x,y) \mapsto (f(x,y), g(x,y)), \tag{5.6}$$

using, for instance, Newton's method.

Method Name	Description
DR Euclidean	Douglas-Rachford with Euclidean: $T_{\mathscr{G}_f\mathscr{G}_g} = \frac{1}{2}\left(I + R_{G_g}R_{\mathscr{G}_f}\right)$ where $R_{\mathscr{G}_f} = 2P_{\mathscr{G}_f} - I$ with $P_{\mathscr{G}_f}$ the Euclidean projection onto $\mathscr{G}_f$ and similarly for $\mathscr{G}_g$
DR Schwarzian	Douglas-Rachford using Schwarzian reflection: $T_{\mathscr{G}_f\mathscr{G}_g} = \frac{1}{2}\left(I + R_{G_g}R_{\mathscr{G}_f}\right)$ where $R_{\mathscr{G}_f}$ is the Schwarzian reflection in $\mathscr{G}_f$ and similarly for $\mathscr{G}_g$
Newton on F	Newton's Method applied to find a zero of $F := (x, y) \to (f(x, y), g(x, y))$
Newton on ∇G	Newton's Method applied to find a zero of ∇G, $G(x, y) = f(x, y)^2 + g(x, y)^2$.
Gradient Descent	Gradient Descent with step size determined by a line search, applied to minimize $G(x, y) = f(x, y)^2+g(x, y)^2$

Figure 5.3: Iterative Methods used on test scenarios.

An alternative approach is to seek solutions of

$$G(f(x, y), g(x, y)) = 0,$$

where $G : \mathbb{R}^2 \to \mathbb{R}$ is any function vanishing only at $(0, 0)$. We will use $G(u, v) := u^2+v^2$. Then the problem becomes to locate the global minimum of $G(x, y) = f(x, y)^2 + g(x, y)^2$ (where the function is 0 in the feasible case) For this formulation the method of gradient descent is available, with a line search being implemented at each iteration. Alternatively one could use Newton's method to find where $\nabla G(x, y) = 0$.

5.4 Experimental Results

In this section we compare computational results for three test scenarios: two intersecting circles, an ellipse and line, and finding a zero of a function $y = \phi(x)$ which we think of as finding an intersection of the curves $f(x, y) := y - \phi(x) = 0$ and $g(x, y) := y = 0$. In each case we seek to solve the problem using the methods listed in figure 5.3 for a selection of initial points $P = (p, q)$ chosen to best illustrate different behaviours.

Colour versions of the diagrams appearing below plus additional details concerning the calculations may be accessed at: https://carma.newcastle.edu.au/DRmethods/Schwarzian/.

5.4.1 Two Circles

We consider the circles specified by $f(x, y) := x^2 + y^2 - 1 = 0$ and $g(x, y) := (x - 2)^2 + y^2 - 9/4 = 0$. The intersection points are $(11/16, \pm 3\sqrt{15}/16)$.

In order to see the relative performance of each Douglas-Rachford scheme (Schwarzian versus Euclidean reflections) we plot a fine grid of starting points, each coloured according to the ratio of the number of iterates required by the two schemes to fall within a specified threshold distance from a feasible point. The result is seen in figure 5.4 where a threshold distance of $1/400$ (half a pixel width) was used and computations were performed to double precision (64 bits, or approximately 15 decimal digits).

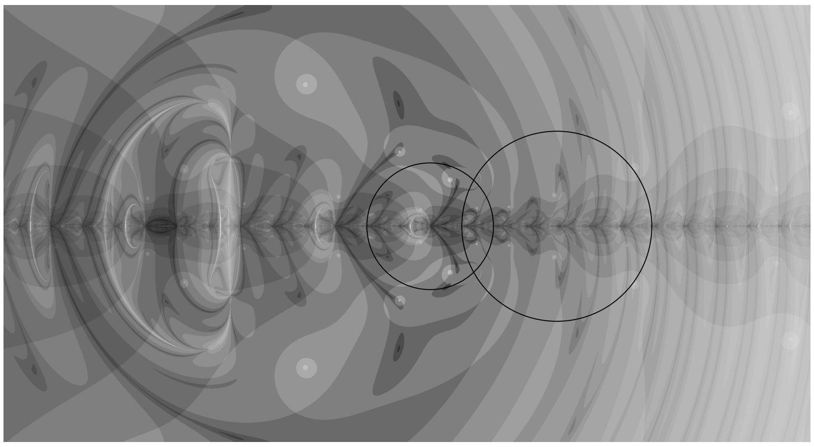


Figure 5.4: Relative performance of the Euclidean and Schwarzian Douglas-Rachford for two circles.

Relative performance of the two schemes are represented on a grey scale with white indicating points where the ratio of "Schwarzian" iterations to "Euclidean" iterations needed is close to 0 and black where the reciprocal ratio is close to 0. We see that for many points (mid-grey) there is little difference between the performance of the two schemes, nevertheless a rich and interesting pattern is revealed.

Motivated by Figure 5.4 we chose as starting points: $P_1 = (0.31, 0.725)$, $P_2 = (0.41565, 0.62135)$, $P_3 = (-1.81, 0.066)$ and $P_4 = (-4.0556, 0.4471)$. Each pair (P_1, P_2), (P_3, P_4) includes a point for which the Euclidean scheme performs better, and a point for which the Schwarzian scheme performs better.

Note that the first iteration of Newtons method applied to $F : (x, y) \mapsto (f(x, y), g(x, y))$ moves any initial point P onto the line $x = 11/16$, and each subsequent iteration remains on the line.

The results for the starting points P_1 and P_2 are shown in figure 5.5 and figure 5.6 respectively. These points are both close to one of the feasible points, and also close to each other.

Starting from P_1 we see that for thresholds up to 10^{-6} the Schwarzian Douglas-Rachford scheme is roughly twice as fast as the Euclidean Douglas Rachford scheme, although for smaller thresholds the distinction is less stark. In contrast, starting from P_2 the Euclidean Douglas Rachford is at least twice as fast as the Schwarzian Doughlas-Rachford scheme

	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
DR Euclidean	3	4	6	8	9	11	13	14
DR Schwarzian	1	1	3	5	6	8	10	11
Newton on F	2	3	3	4	4	4	4	4
Newton on ∇G	2	3	3	4	4	4	4	5
Gradient Descent	2	3	4	5	6	6	7	8

Figure 5.5: Performance starting from P_1.

	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
DR Euclidean	1	1	1	1	3	5	6	8
DR Schwarzian	2	4	6	7	9	11	12	14
Newton on F	2	3	3	3	4	4	4	4
Newton on ∇G	2	3	3	4	4	4	4	4
Gradient Descent	3	4	6	8	9	10	12	14

Figure 5.6: Performance starting from P_2.

for thresholds up to 10^{-8}, and often faster. We also note that in both these cases Newton's method is superior to both Douglas-Rachford schemes, while gradient descent is comparable.

With P_3 as starting point Schwarzian Douglas Rachford drastically outperforms the Euclidean scheme, see figure 5.7.

	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
DR Euclidean	Fails to converge in 10000 iterations							
DR Schwarzian	9	10	12	14	15	17	19	20
Newton on F	9	10	10	10	10	11	11	11
Newton on ∇G	Fails to converge in 10000 iterations							
Gradient Descent	4	6	8	10	12	14	16	17

Figure 5.7: Performance for starting point P_3.

The failure of the Euclidean Douglas Rachford scheme can be seen in figure 5.8 where the iterates appear to converge to a period 4 point. Observe that a simple calculation for the Euclidean Douglas Rachford operator $T = T_{\mathcal{G}_f \mathcal{G}_g}$ and $P = (p, 0)$, where $0 < p < 3/2$, shows that $T^4(P) = P$.

Figure 5.9 shows the results starting from P_4. Here Euclidean Douglas-Rachford significantly outperforms the Schwarzian version at all thresholds. We also note that the two schemes converge to different feasible points, see figure 5.10.

5.4.2 Ellipse and Line

We consider the ellipse $f(x, y) := x^2 + (y/8)^2 - 1 = 0$ and line $g(x, y) := y - 6x = 0$. Starting from anywhere near the ellipse the behaviour of the Schwarzian formulation

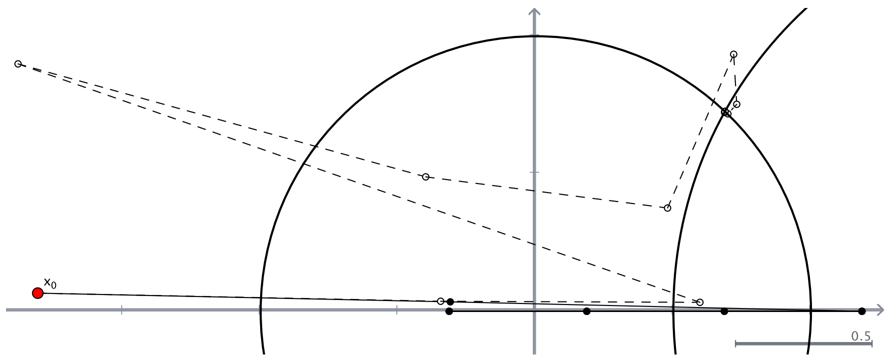


Figure 5.8: Starting point P_3. Solid black line shows Euclidean Douglas Rachford iterates, Dashed black line shows Schwarzian Douglas Rachford iterates.

	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}
DR Euclidean	3	3	3	4	6	7	9	11
DR Schwarzian	8	9	11	13	14	16	18	19
Newton on F	8	9	9	9	9	10	10	10
Newton on ∇G	8	8	9	9	9	10	10	10
Gradient Descent	4	5	7	8	10	12	13	15

Figure 5.9: Performance for starting point P_4.

of Douglas-Rachford appears quite similar to that of the Euclidean formulation. As discussed in [28], for Douglas-Rachford using Euclidean reflection, periodic points appear surrounded by basins of attraction or repulsion. This prevents convergence to the feasible points for many starting points and slows the convergence for many others. We observe the same phenomenon when Schwarzian reflection is employed, where, at least for this particular ellipse and line, the exact same periodic points are observed. This is shown in figure 5.11. Note that the subsequences have been started from different points in order to illustrate the different sizes of the local basins.

Behaviour of the different methods is solution is tabulated in figure 5.12.

Locally the behaviour of the two formulations of Douglas-Rachford are nearly identical, however, for more remote starting points, while continuing to converge, the two Douglas-Rachford schemes behave differently and like the two circle case may converge to different feasible points.

To better assess the effect of compounding numerical error on Douglas-Rachford using Euclidean reflection, where each iteration involves the numerical solution of a Lagrange multiplier problem to find the Euclidean (nearest point) projection P_{G_f}, we experimented with a further variant. For $z \in \mathbb{C}$ we computed $Q_{G_f}(z)$, the nearest intersection of G_f with the line segment from z to its Schwarzian reflection $R_{G_f}(z)$ and used this as a substitute for $P_{G_f}(z)$. Thereby avoiding the need for a computationally expensive numerical minimisation. Douglas-Rachford was then implemented using $2Q_{G_f} - I$ as a replacement for reflection in G_f. Started sufficiently near a feasible or periodic point this yielded nearly

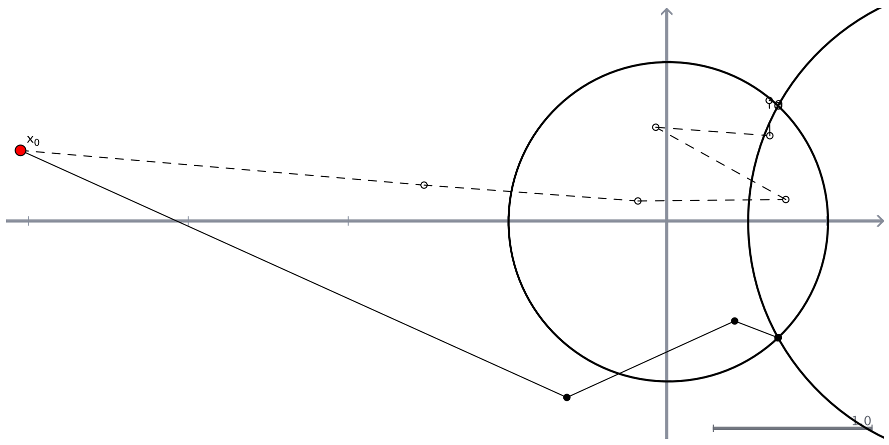


Figure 5.10: Starting point P_4. Solid black line shows Euclidean Douglas Rachford iterates, Dashed black line shows Schwarzian Douglas Rachford iterates.

identical results to the Euclidean version even after hundreds of iterations. This lack of deviation speaks to the low numerical sensitivity of the Douglas-Rachford method.

5.4.3 Finding a Zero of a Function

Thinking of this as finding an intersection of the curve $f(x,y) := y - \phi(x) = 0$ with $g(x,y) := y = 0$ leads, via (5.6), to finding a zero of

$$F : \mathbb{R}^2 \to \mathbb{R}^2 : (x,y) \mapsto (f(x,y), g(x,y)) := (y - \phi(x), y).$$

The Jacobian is

$$J(F) = \frac{\partial(f,g)}{\partial(x,y)} = \begin{pmatrix} -\phi'(x) & 1 \\ 0 & 1 \end{pmatrix}.$$

So, applying Newton's method with any initial point (x_0, y_0) leads to the iterative scheme,

$$x_{n+1} = x_n - \phi(x_n)/\phi'(x_n). \quad y_{n+1} = 0,$$

which, not surprisingly, we recognise as Newton-Raphson applied directly to $y = \phi(x)$.

Thus, for $\phi(x) = x/\sqrt{|x|}$ with any starting point other than $(0,0)$ this approach leads to cyclic iterates of period 2, and so fails to converge to the zero. By contrast, Douglas-Rachford applied to the two curve reformulation (see discussion below) is seen to rapidly spiral to $(0,0)$ and to drastically outperform the method of gradient descent applied to the function $G(x,y) = \left(y - x/\sqrt{|x|}\right)^2 + y^2$. The results with initial point (1,0) are tabulated in figure5.13.

Starting from $(1,0)$ gradient descent does not converge to within 10^{-3} after 20,000 iterations. This may be understood from the shape of the surface $z = G(x,y)$, illustrated in figure 5.14. Convergence down the sides of the trough is quite rapid, but, once near

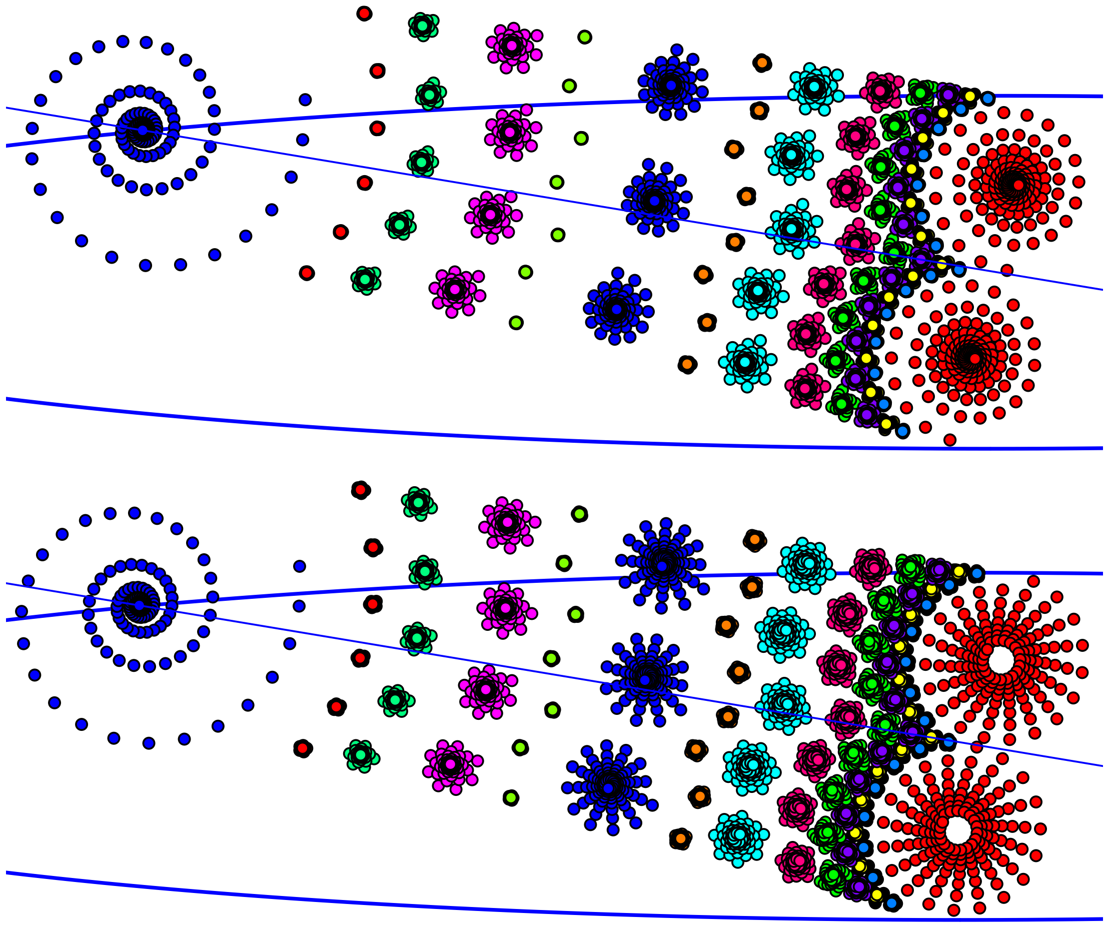


Figure 5.11: Douglas-Rachford iterates for ellipse - line, showing periodic points and their attractive basins. Top: with Euclidean Reflection. Bottom: with Schwarzian Reflection. (Images have been rotated.)

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-8}
DR Euclidean	61	129	197	265	334	402	538
DR Schwarzian	61	129	197	265	334	401	537
Newton on F	2	3	3	4	4	4	4
Newton on ∇G	3	4	4	5	5	5	5
Gradient Descent	2	4	4	6	8	10	12

Figure 5.12: Performance with $f(x,y) = x^2 + (y/8)^2 - 1$, $g(x,y) = y - 6x$, and starting point $(1.5, 4.5)$.

	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-8}
DR Euclidean	4	4	5	5	6	6	6
DR Schwarzian	4	5	5	6	6	7	7
Newton on F	cycles						
Newton on ∇G	fails						
Gradient Descent	4	415					

Figure 5.13: Locating the zero of $y = x/\sqrt{|x|}$ by the methods discussed starting from $(1,0)$.

the sharply creased bottom iterates bounce from side to side making only slow progress toward the minimum.

A more large scale view, starting from the point $(250, 500)$, is provided in figure 5.15. Here the Schwarzian formulation of Douglas-Rachford requires 293 iterations to come within 10^{-1} of the feasible point while the Euclidean formulation needs 715 iterations.

Note: To compute Schwarzian reflection in the curve $\mathscr{C} := \{(x,y) : y = x/\sqrt{|x|}\}$ we write $\mathscr{C} = \mathscr{C}_+ \cup \mathscr{C}_-$, where

$$\mathscr{C}_+ = \{(x,y) : x \geq 0, y = \sqrt{x}\} \quad \text{and } \mathscr{C}_- = \{(x,y) : x \leq 0, y = -\sqrt{-x}\}..$$

Then, if $d(z,\mathscr{C}_+) \geq d(z,\mathscr{C}_-)$ we take $R_{\mathscr{C}}(z) = R_{\mathscr{C}_+}(z) = 1 - z + \sqrt{1-4z}$, otherwise we take $R_{\mathscr{C}}(z) = R_{\mathscr{C}_-}(z) = 1 + z - \sqrt{1+4z}$.

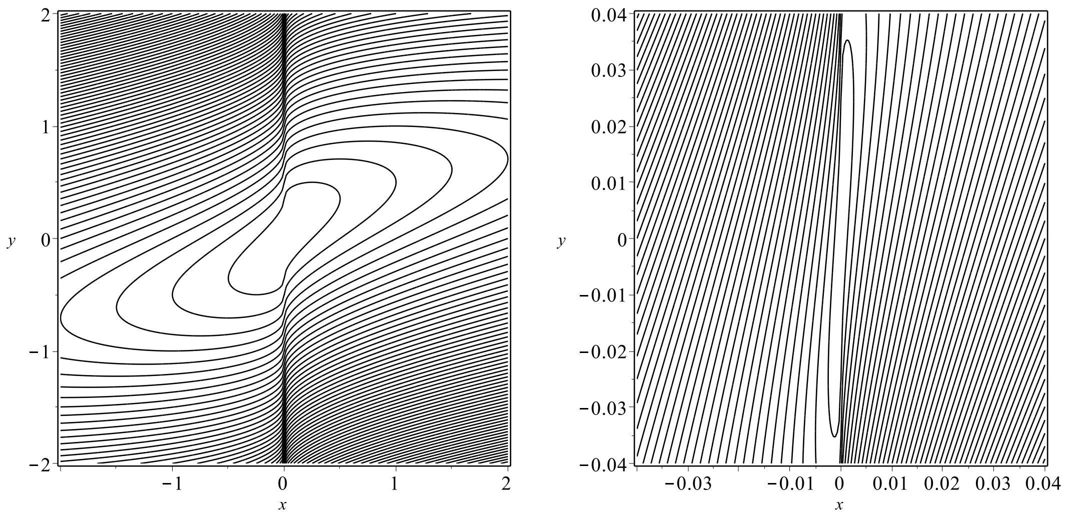


Figure 5.14: The surface $z = G(x, y) = \left(y - x/\sqrt{|x|}\right)^2 + y^2$.

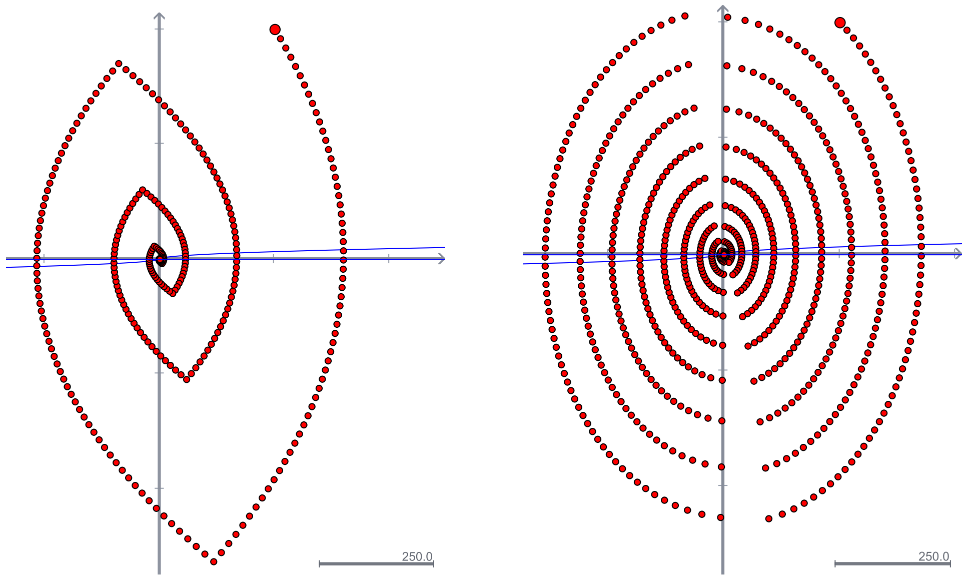


Figure 5.15: Douglas-Rachford applied to the curve $y - x/\sqrt{|x|} = 0$ and line $y = 0$. Left: using Schwarzian reflection. Right: using Euclidean reflections.

Part III

Summary

6 Final Remarks and Further Work

6.1 Integer Relations

We have looked at classical integer relations, and have generalised the notion of an integer relation to algebraic integer relations. As an initial step in understanding algebraic integer relations we looked specifically at quadratic integer relations. We have explored the computation of classical integer relations with both PSLQ and LLL, and quadratic integer relations with PSLQ through reduction (to the classical case), LLL, and APSLQ (our direct modification of the PSLQ algorithm capable of handling the complex quadratic case).

We have seen that, for the most part, PSLQ out-performs LLL for integer relation computations in terms of time taken, as well as needing lower precision for computations. This seems unsurprising when we consider that PSLQ is an algorithm dedicated to computing integer relations whereas LLL is an algorithm for lattice basis reduction—a different problem to which we happen to be able to reduce integer computations. The low performance of LLL is amplified by the need to find a parameter essentially by trial and error thus requiring, in practice, multiple executions of LLL for a single integer relation problem.

Bailey et al. [7] comment in their conclusion that "[...] research is needed in how to efficiently perform PSLQ-type integer relation computations on a highly parallel platform." It was this comment that originally motivated the author to begin to understand the PSLQ algorithm, although we note that we see no way to improve upon the methods of Bailey and Broadhurst [9]. There is, however, a parallel implementation of LLL given by Luo and Qiao [56]. We have not yet looked at this except to note its existence, and it would need to be significantly faster in order to overcome the limitations of LLL for integer relation computations, yet nonetheless it would be interesting to see the difference parallelisation has on performance.

Despite the limitations of LLL, the nature of the reduction from integer relation problem to lattice basis reduction allowed us to more reliably compute some quadratic algebraic integer relation problems that APSLQ did not handle well. Specifically, these cases are for complex $\mathbb{Q}(\sqrt{D})$-integers in cases where $\mathbb{Q}(\sqrt{D})$ is not a Euclidean field and so the existing theory behind PSLQ (and also APSLQ) does not hold. It is remarkable that even in these cases APSLQ yields surprisingly many correct results, but LLL performed flawlessly.

We present, in Table 6.1, our summary of the best choice for integer relation finding algorithm at the time of writing.

Table 6.1: Best choice of integer relation finding algorithm

Case	Algorithm
Classical	PSLQ
Real Quadratic	PSLQ using reduction
Complex Euclidean Quadratic	APSLQ
Complex Non-Euclidean Quadratic	LLL[1]

The particular case of complex non-Euclidean quadratic fields requires some more comment. In our testing we found that—in addition to LLL—PSLQ using reduction gave flawlessly correct results although needing higher precision, but taking (often significantly) less computation time. Recall from Section 3.3.1 on page 52 that the result of the reduction method in complex cases might not be from the correct integer ring, and that the techniques we provide for detecting and correcting this are not proven to work in all cases. Consequently the use of PSLQ through reduction, while quicker and requiring lower precision than LLL, is potentially less reliable, the positive results reported herein notwithstanding, so our recommendation is to use LLL for these cases. A sensible strategy might be to first try PSLQ with reduction, and to fall back on LLL if the reduction method does not yield a suitable candidate relation. A provable method to extract the correct integers from the reduction method output, or a proof of correctness of the techniques presented herein (even if in some restricted cases) would be welcome.

It seems clear that a dedicated algorithm is preferable when it can be achieved, and to that end further work should look at extending the theory to allow for the cases where no lattice exists (including the entirety of the non-trivial real quadratic fields) and the non-Euclidean complex cases. For the former case the Minkowski embedding (see Baake and Grimm [5]) has been suggested as an avenue of exploration. For the latter case we currently have no clear path forward.

Of a tangential nature to the exploration of integer relations in this thesis, the author has begun to explore the possibility of using the PSLQ algorithm to find algebraic approximations of transcendental numbers (i.e., real or complex numbers which are not algebraic) within desired bounds, or to establish that no such approximations exist. The aim is to achieve this in a similar manner to that used to determine if a number α is algebraic by attempting to find an integer relation for $(\alpha^0, \alpha^1, \dots, \alpha^n)$ for some $n \in \mathbb{N}$. Interval arithmetic is employed and the increasing lower bound of the norm of any integer relation that PSLQ computes each iteration is exploited. Preliminary experimentation in *Maple* has shown promise, although the initial technique used the interval arithmetic in a naïve

[1]Or PSLQ using reduction for increased speed at the risk of failing to find a relation if the correction transformation fails.

manner which turned out to not be very effective at finding the desired bounds. Several refinements have been devised and are marked for exploration.

Finally, on a more practical note, the author would like to properly implement APSLQ in a higher-performing language (probably C++ or Rust) and utilising the optimisations of Bailey and Broadhurst. Additionally, termination conditions based on the norm lower bound could be incorporated (whether exclusively, or as a user selectable choice) in agreement with the discussion about the threshold in Section 2.4.3. Such an implementation, if created as a library, could be used with *Maple* and other such systems.

6.2 Douglas-Rachford

We have looked at the Douglas–Rachford algorithm in two contexts. The first was a direct constraint satisfaction problem for geometric sets, generalising the prototypical case explored by Borwein and Sims. The second was as a means to find intersections of plane curves, and to explore the use of a different reflection-like operation in the Douglas–Rachford operator. We have seen in both cases that the algorithm performs remarkably well in the non-convex setting, and yields rich and unexpected complexity.

In the first case, given that we were investigating the Douglas–Rachford method applied to some of the simplest possible instances of a non-convex set, the emergence of such complexity as we saw is extraordinary. More interesting from a technical standpoint is the similarity with which the behaviour in such simple situations appears to resemble some of what is observed for much larger and more complicated ones.

We chose p-spheres and ellipses as simple generalisations of a circle as studied by Borwein and Sims [25]. An interesting alternate generalisation to look at in future would be curves of constant width such as those described by Resnikoff [64], for example.

We note that many of the points of attraction and repulsion were found through painstaking trial and error, or by the use of colour maps. A method of automatically finding the (probable) points of attraction would be preferable, and allow for exploration of more cases. The author has some thoughts regarding the use of cluster analysis which may yield such a method.

Additionally, we suggest that the parallel techniques employed could be used to try to overcome the seeming unpredictability of which points converge to a feasible point and which do not. In low dimensional spaces a simple “blanketing” of a region of space, as we used herein, ought to be satisfactory (provided we test each resulting point). In higher dimensional spaces such a technique is less feasible due to the “curse of dimensionality”, and we suggest instead that a Monte Carlo approach of starting at many randomly generated points.

In the second case, while the local similarity of Douglas–Rachford using a Schwarzian reflection and a Euclidean reflection near a feasible point is to be expected, the extent of the similarity is fascinating to observe. Equally fascinating are the large scale differences. Clearly the intersection of complex analysis with iterative methods is a fruitful one. Douglas–Rachford based on Schwarzian reflections often equals or outperforms the same method using Euclidean reflections for the class of problems considered here, and both methods sometimes outperform gradient descent algorithms. In well conditioned problems the quadratic rate of convergence exhibited by Newton's method easily outstrips the linear rate anticipated for projection methods [34]. However, in situations where Newton cycles or diverges Douglas–Rachford often continues to work well; this demonstrates the robustness of the algorithms, and Douglas–Rachford using Schwarzian reflection is seemingly the more robust.

Future work could look at generalising these ideas to higher dimensions. Shapiro [65] would provide a good starting point.

We hope that we have succeeded in making a case for computer-assisted discovery, visualisation, and verification. We close with the following quote from Littlewood's Miscellany (p. 35 in the original 1953 edition)

> A heavy warning used to be given [by lecturers] that pictures are not rigorous; this has never had its bluff called and has permanently frightened its victims into playing for safety. Some pictures, of course, are not rigorous, but I should say most are (and I use them whenever possible myself).
>
> — Littlewood [55, p. 54]

said long before the current powerful array of graphic, visualisation and geometric tools were available.

Part IV

Appendices

A Numeric Integer Relation Algorithm Implementations

In the main body of this thesis we have deliberately avoided presenting the numeric implementation details of the integer relation finding algorithms presented. This was done in order to better explain the mathematics behind the algorithms and techniques. The author has found that too many implementation details obscure the mathematical understanding. Moreover, in the case of LLL, the details of the algorithm were deemed beyond the scope of the discussion at hand.

In this appendix we present the algorithms in greater detail, more suitable for potential implementation by the reader. We note that this is far from a complete treatment of the topic, and the interested reader should still consult the literature. Nonetheless, the material presented herein should serve as a useful supplementary reference to the more mathematical treatment of the algorithm presented earlier in the thesis.

A.1 (A)PSLQ

The algorithm implemented for the experimental explorations of PSLQ is presented in Algorithm A.1.1. It is the algorithm that was used when we were not using *Maple*'s inbuilt implementation, primarily for algebraic PSLQ. It was based on, and serves as a good reference for, classical PSLQ, and demonstrates a typical implementation.

In particular, the matrix multiplications of the algorithms as presented in Algorithms 2.2.7 and 2.4.2 on page 10 and on page 19 are very inefficient if directly translated into code. The updates to H', B, and y are more efficiently implemented as shown in the hermite reduce procedure. Therein we use embedded loops so that all the updates are handled together (instead of with three separate matrix multiplications), and moreover only the rows and columns that need to be updated are actually updated. Additionally, we omit the use and reporting of the norm lower bound.

A similar optimisation can be achieved for the step where the matrix $Q_{[H',r]}$ is used to ensure that H' is once again lower trapezoidal. Recalling that the matrix $Q_{[H',r]}$ only modifies at most two columns of the matrix H', we realise that we need only update

```
Procedure hermite reduce($H'$, $B$, $y$, $r$)
Input  : $H'$, $B$, $y$, $r$ from Algorithm A.1.1
Output: $H'$, $B$, $y$, Hermite reduced
  /* Computes the Hermite reduction from Algorithm 2.4.2 ($H' \leftarrow D_{H'} H'$) and the associated updates
     of $B$ and $y$ ($B \leftarrow B D_{H'}^{-1}$ and $y \leftarrow y D_{H'}^{-1}$ respectively)              */
1 for $i$ from $r+1$ to $n$ do
2     for $j$ from $i-1$ to $1$ by $-1$ do
3         $t \leftarrow \lceil H'_{i,j}/H'_{j,j} \rfloor$
4         $y_j \leftarrow y_j + t\, y_i$
5         for $k$ from $1$ to $j$ do $H'_{i,k} \leftarrow H'_{i,k} - t\, H'_{j,k}$ end   /* $\mathrm{row}_i(H') \leftarrow \mathrm{row}_i(H') - t\,\mathrm{row}_j(H')$ */
6         for $k$ from $1$ to $n$ do $B_{k,j} \leftarrow B_{k,j} - t\, B_{k,i}$ end        /* $\mathrm{col}_j(B) \leftarrow \mathrm{col}_j(B) - t\,\mathrm{col}_i(B)$ */
```

those columns (again, instead of with a whole matrix multiplication). Furthermore, in the case the $Q_{[H',r]}$ is the identity matrix, no update needs to be performed at all.

```
Procedure remove corner($H'$,$r$)
Input  : $H'$, $r$ from Algorithm A.1.1
Output: $H'$ modified to be lower trapezoidal
  /* Computes the result of $H' Q_{[H',r]}$ from Algorithm 2.4.2, ensuring that $H'$ is lower trapezoidal
     after the row swap step                                                                          */
1 if $r < n-1$ then
2     $\beta \leftarrow H'_{r,r}$
3     $\lambda \leftarrow H'_{r,r+1}$
4     $\delta \leftarrow \sqrt{\beta\bar{\beta} + \lambda\bar{\lambda}}$
5     for $i$ from $r$ to $n$ do                                   /* Update columns $r$ and $r+1$ of $H'$ */
6         $t_r \leftarrow H'_{i,r}$
7         $t_{r+1} \leftarrow H'_{i,r+1}$
8         $H'_{i,r} \leftarrow (t_r\,\bar{\beta} + t_{r+1}\,\bar{\lambda})/\delta$
9         $H'_{i,r+1} \leftarrow (-t_r\,\lambda + t_{r+1}\,\beta)/\delta$
```

With these optimisations we implement APSLQ (and, by extension, PSLQ) in Algorithm A.1.1 on the next page. We have endeavoured to annotate the code so as to explain what the various code blocks are doing, and to provide reference back to the earlier, more high level, presentations (Algorithms 2.2.7 and 2.4.2 on pages 10 and 19 respectively).

There are further optimisations which can be performed, but which we have not employed. These optimisations are due to Bailey and Broadhurst [9] and were subsequently reported in Borwein [29]. We discuss them in very broad terms.

The first such optimisation, called *multipair*, allows for a small amount of parallelisation in the algorithm. Instead of choosing a single value of r and swapping the rows r and $r+1$ of H' (and the analogous columns of B and the elements of y), multiple value are

Algorithm A.1.1: APSLQ

Input : $x \in \mathbb{F}^n, D \in \mathbb{Z}, \gamma \geq 0, \epsilon > 0, max_i > 0$

Output: A vector in $\mathcal{O}_{\mathbb{K}}^n$ (where $\mathbb{K} = \mathbb{Q}(\sqrt{D})$) or FAIL

```
   /* ——————————— Initialisation ——————————— */
1  $y \leftarrow x/\|x\|$                                            /* Normalise input vector */
2  $s_n \leftarrow y_n \overline{y_n}$                     /* Initialise $s_k = \sqrt{\sum_{j=k}^n y_j \overline{y_j}}$ for $1 \leq k \leq n$ */
3  for $i$ from $n-1$ to 1 by $-1$ do $s_i \leftarrow s_{i+1} + y_i \overline{y_i}$ end
4  for $i$ from 1 to $n$ do $s_i \leftarrow \sqrt{s_i}$ end
5  for $j$ from 1 to $n-1$ do                                    /* Initialise $H_y$ and $B$ */
6      for $i$ from 1 to $j-1$ do $H'_{i,j} \leftarrow 0, B_{i,j} \leftarrow 0$ end
7      $H'_{j,j} \leftarrow s_{j+1}/s_j$, $B_{j,j} \leftarrow 1$
8      for $i$ from $j+1$ to $n$ do
9          $H'_{i,j} \leftarrow -(\overline{y_i}\, y_j)/(s_j\, s_{j+1})$
10         $B_{i,j} \leftarrow 0$
11 hermite reduce($H', B, r, 1$)                              /* Initial Hermite reduction */
12 $i \leftarrow 0$                                             /* Initialise Loop counter */
   /* ——————————— Main Calculation ——————————— */
13 repeat
14     $i \leftarrow (i+1)$                                     /* Increment loop counter */
15     $r \leftarrow 1$                               /* Find $r$ such that $\gamma^r|H'_{r,r}|$ is maximal */
16     $val \leftarrow \gamma H'_{1,1}$
17     for $i$ from 2 to $n-1$ do
18         if $\gamma^i H_{i,i} > val$ then
19             $r \leftarrow i$
20             $val \leftarrow \gamma^i H_{i,i}$
21     $\text{row}_r(H') \leftrightarrow \text{row}_{r+1}(H')$           /* Swap rows $r$ and $r+1$ in $H'$ */
22     $\text{col}_r(B) \leftrightarrow \text{col}_{r+1}(B)$            /* Swap columns $r$ and $r+1$ in $B$ */
23     $y_r \leftrightarrow y_{r+1}$                          /* Swap elements $r$ and $r+1$ in $y$ */
24     remove corner($H', r$)                          /* Make sure $H'$ is lower trapezoidal */
25     hermite reduce($H', B, y, r$)              /* Hermite reduce $H'$ and update $B$ and $y$ */
26     $k \leftarrow 1$                                 /* Find $k$ such that $|y_k|$ is minimal */
27     $val \leftarrow |y_1|$
28     for $i$ from 2 to $n$ do
29         if $|y_i| < val$ then
30             $k \leftarrow i$
31             $val \leftarrow |y_i|$
32 until $|y_k|/\|\text{col}_k(B)\| < \epsilon$ or $i > max_i$
33 if $|y_k|/\|\text{col}_k(B)\| < \epsilon$ then return $\text{col}_k(B)$ else return $FAIL$
```

chosen. For each such value, a row (and column and element) swap is performed as it is in the algorithm presented here. Careful criteria are given to make sure that the swaps will not interfere with each other, and so the swaps may be performed in parallel. We do not discuss these criteria here (the interested reader should consult the literature), but we will note that Borwein gives an improved criteria which avoids a cycling problem reported (and worked around) by Bailey and Broadhurst

Another optimisation, referred to as *multi level*, is to perform the computation at different precisions to balance the computational speed and required precision. The algorithm is run in two, or sometimes three "levels". One level is full precision, another is machine precision (or otherwise some lower precision than the full precision), and if a third is used it is at an intermediate precision. The algorithm is performed at the lowest precision until it is detected that precision is exhausted, at which point the algorithm state is used to update next higher precision level, and computations are performed at that precision until precision can safely be resumed at lower precision, or until precision is exhausted at that precision level. The mechanisms for detecting exhausted precision and for updating the higher precision levels are not discussed here, but are given in the literature.

A.2 LLL

The LLL algorithm is described in Algorithm A.2.1. It is presented for reference, and in the interests of completeness.

We have based the algorithm as presented on the version as presented in Borwein [29]. We note, however, that Borwein keeps the values of b_i^* updated throughout the algorithm, but that such updating is entirely unnecessary. By inspection we can see that the values of $\|b_i^*\|$ (which we have denoted as η_i^* instead) are all updated without reference to the underlying b_i^*. Indeed, the algorithm as given originally by Lenstra, Lenstra Jr, and Lovász [52, fig. 1] does not keep the b_i^* values updated and also updates the norms (which they denote as B_i) directly. Consequently we have removed the b_i^* updating from Borwein [29]'s algorithm. We have also re-ordered the operations slightly and renamed some variables to make the algorithm a little easier to follow. Care has been taken to ensure that the underlying algorithm is unchanged, and agrees with the original by Lenstra, Lenstra Jr, and Lovász.

Algorithm A.2.1: LLL

```
Input  : $b_1, \dots, b_n$ (a lattice basis to be reduced)
Output: $b_1, \dots, b_n$ (a LLL reduced basis)
  /* ——————————— Initialisation ——————————— */
for $i$ from 1 to $n$ do
    $b_i^* \leftarrow b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$
    $\eta_i^* = \|b_i^*\|^2$
    for $j$ from $i+1$ to $n$ do
        $\mu_{i,j} \leftarrow b_j \cdot b_i^* / \eta_i^*$
$k \leftarrow 2$
  /* ——————————— Main Calculation ——————————— */
repeat
    for $j$ from $k-1$ to 1 by $-1$ do
        $q \leftarrow \lceil \mu_{k,j} \rfloor$
        $b_k \leftarrow b_k - q\, b_j$
        for $i$ from 1 to $j$ do
            $\mu_{k,1} \leftarrow \mu_{k,i} - q\, b_j$
    if $\eta_k^* \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \eta_{k-1}^*$ then
        $k \leftarrow k+1$
    else
        $b_k \leftrightarrow b_{k-1}$                     /* Swap basis vectors $b_k$ and $b_{k-1}$ */
        $t \leftarrow \eta_k^* + \mu_{k,k-1}^2\, \eta_{k-1}^*$          /* Set temporary values to be re-used below */
        $m \leftarrow \mu_{k,k-1}\, \eta_{k-1}^* / t$
        $\eta_k^* \leftarrow \eta_k^*\, \eta_{k-1}^* / t$       /* Update $\eta_k^*$ and $\eta_{k-1}^*$. The order here is important */
        $\eta_{k-1}^* \leftarrow t$
        for $i$ from 1 to $k-2$ do                  /* Update the relevant $\mu_{i,j}$ values */
            $\mu_{k,i} \leftrightarrow \mu_{k-1,i}$
        for $i$ from $k+1$ to $n$ do
            $t \leftarrow \mu_{i,k}$
            $\mu_{i,k} \leftarrow \mu_{i,k-1} - \mu_{k,k-1}\, \mu_{i,k}$
            $\mu_{i,k-1} \leftarrow t + m\, \mu_{i,k}$
        $\mu_{k,k-1} \leftarrow m$
        $k \leftarrow \max(2, k-1)$
until $k = n+1$
return $b_1, \dots, b_n$
```

B Complete Collection of Numeric Result Graphs

Contained in this appendix is the complete collection, of which there are many, of graphs of precision and time taken for all testing performed. Each graph take a page, and consequently the collection begins on the next page.

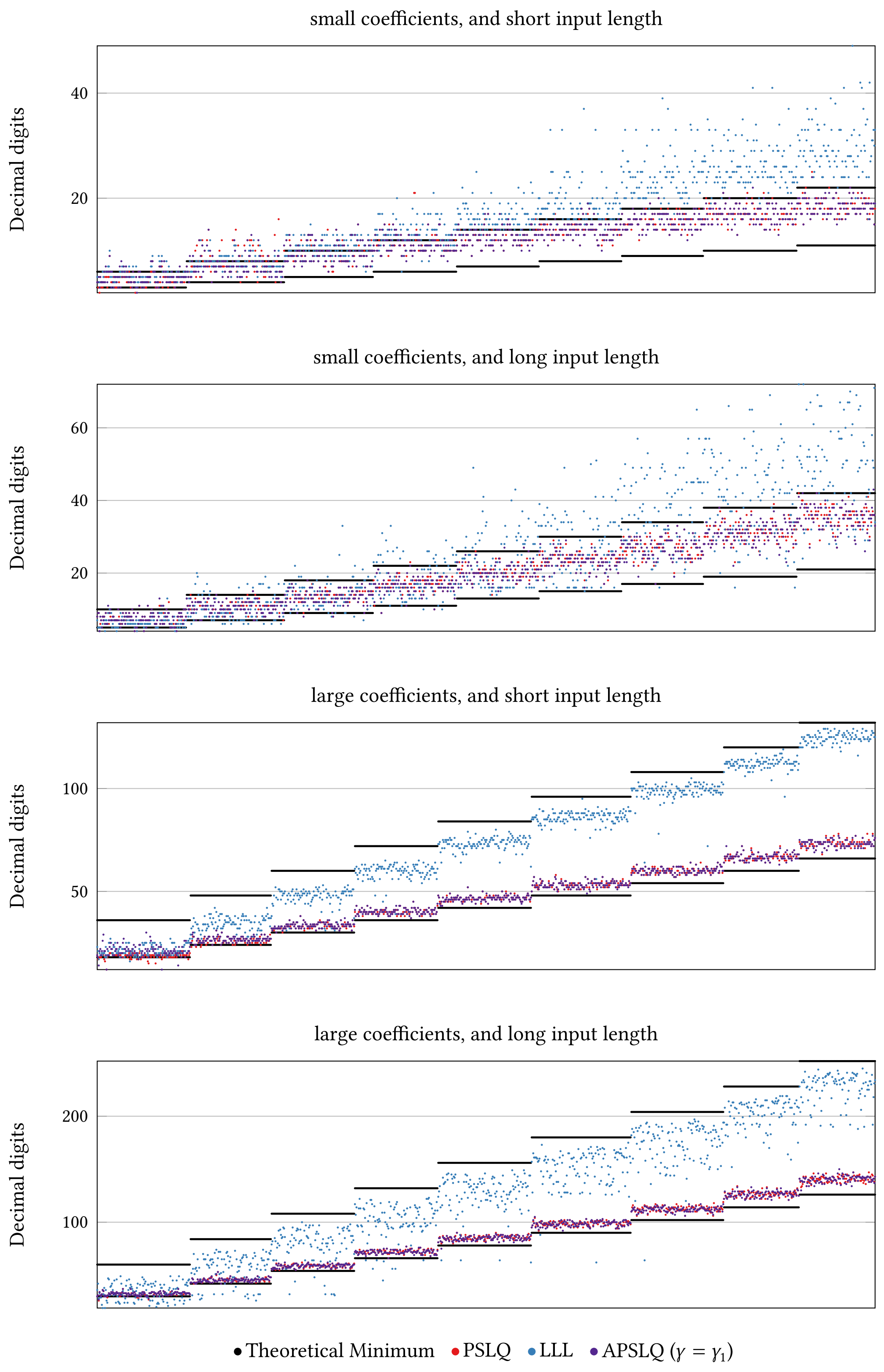


Figure B.1: Precision required for $\mathbb{K} = \mathbb{Q}, C = C_{\mathbb{R}}$.

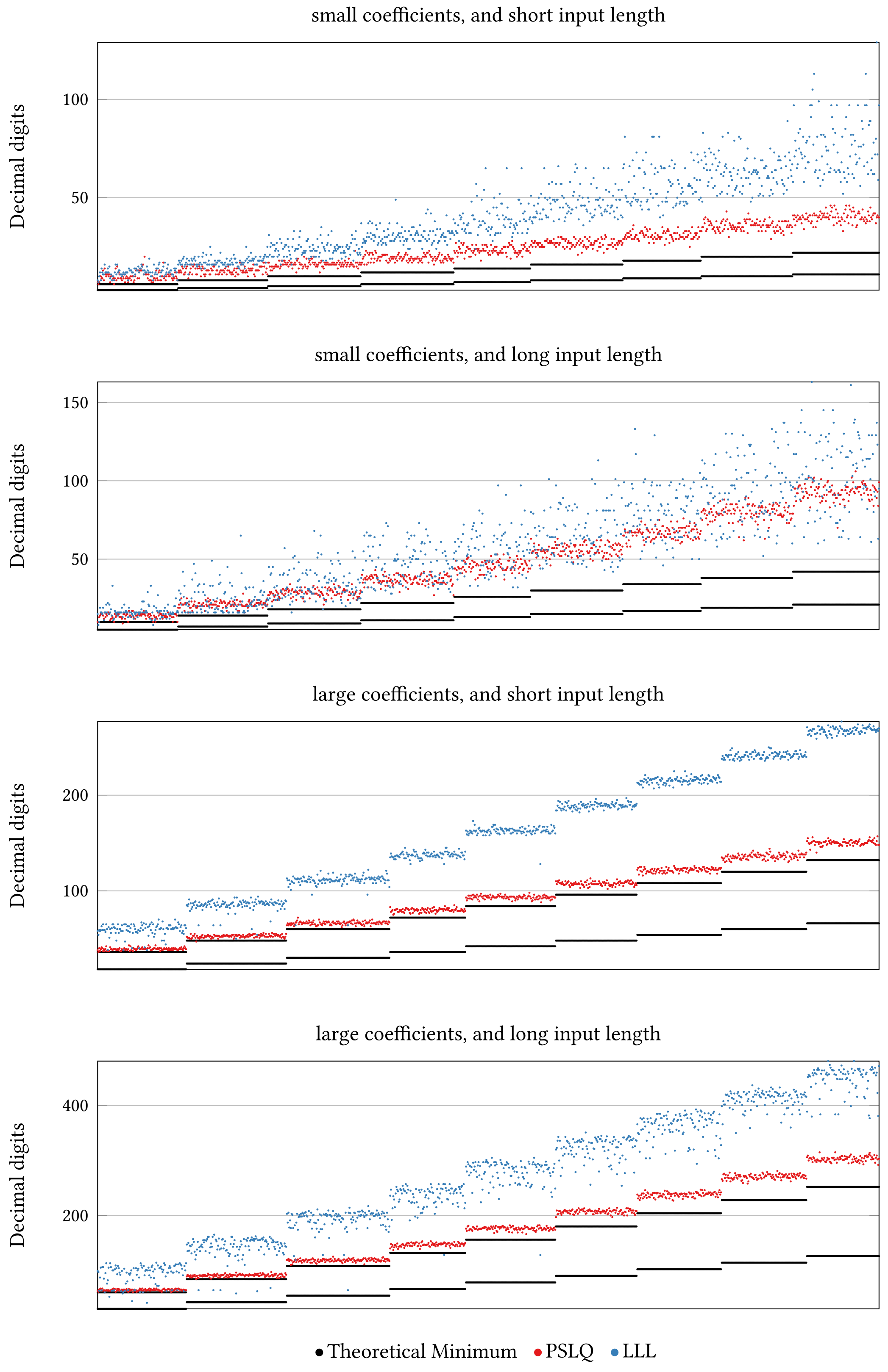


Figure B.2: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{2}), C = C_{\mathbb{R}}$.

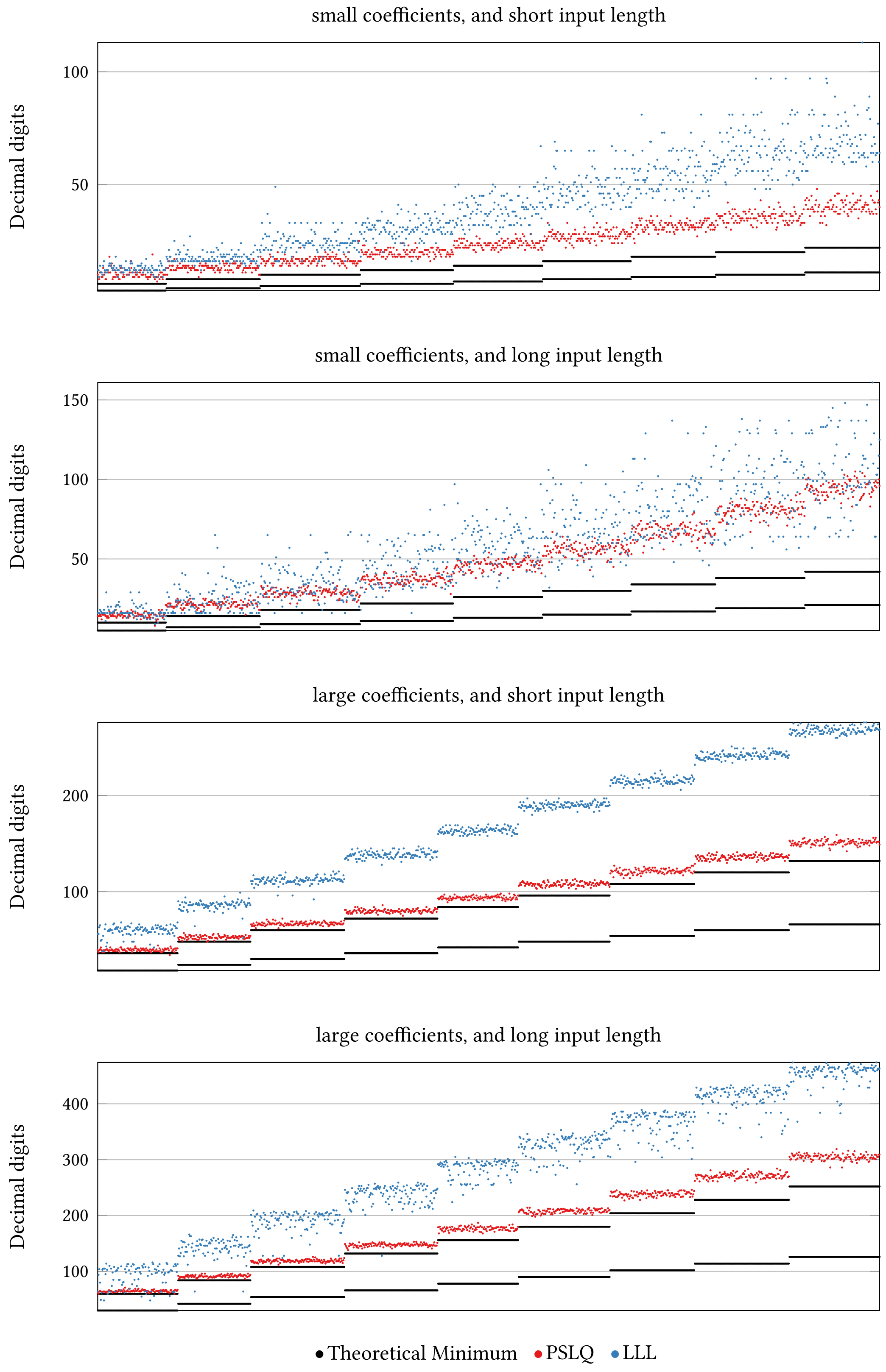


Figure B.3: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{3}), C = C_{\mathbb{R}}$.

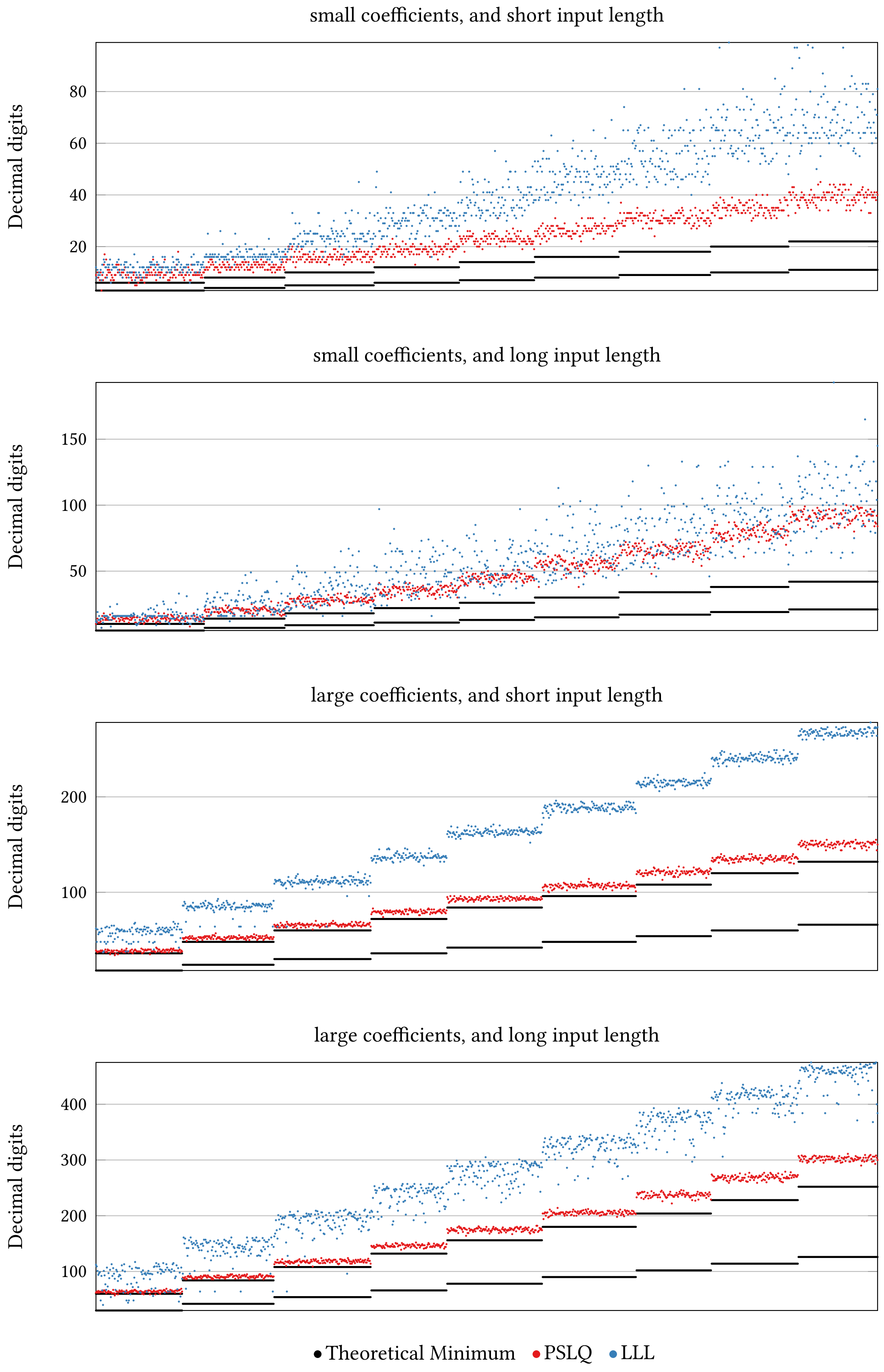


Figure B.4: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{5}), C = C_{\mathbb{R}}$.

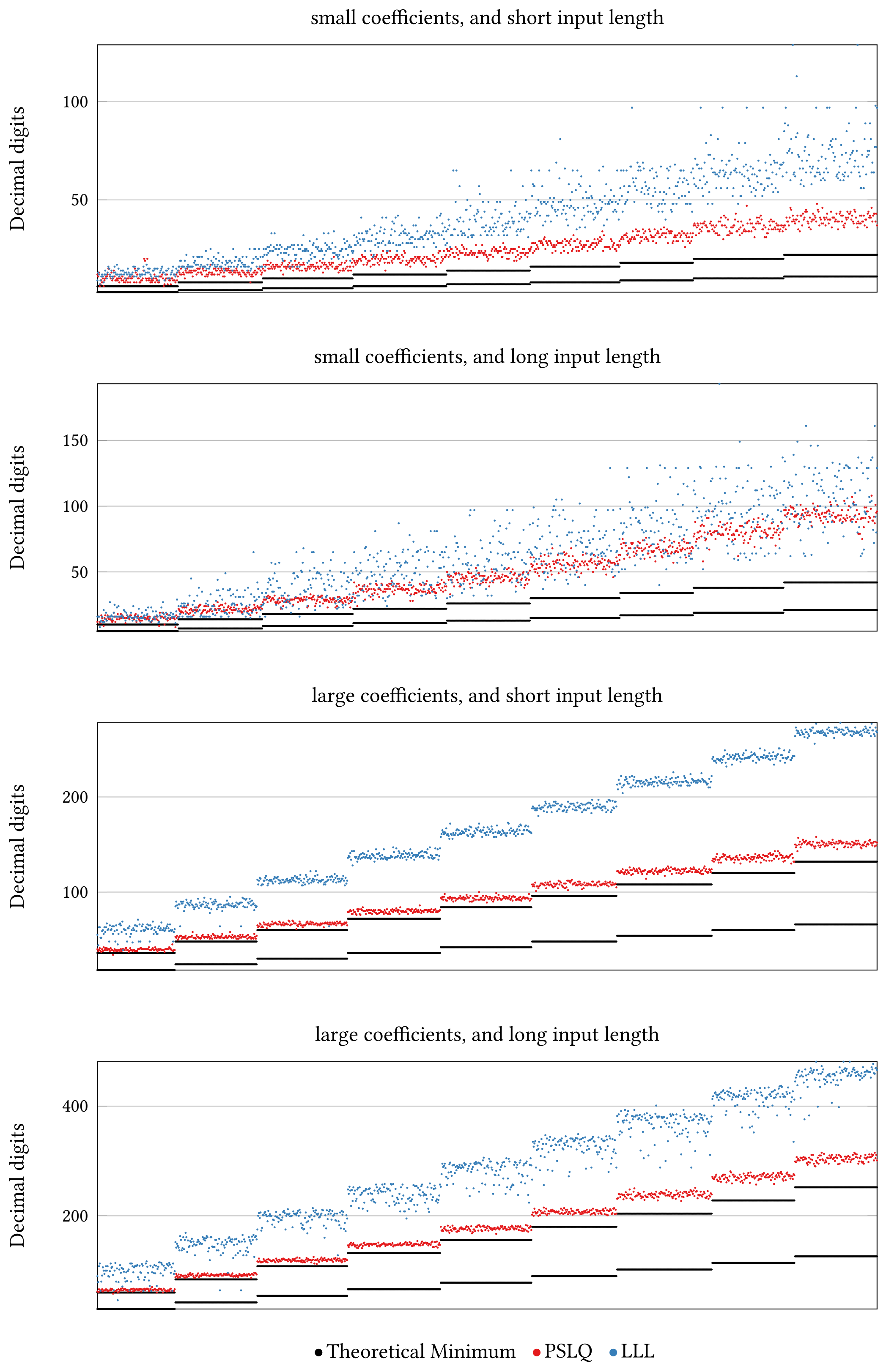


Figure B.5: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{6}), C = C_{\mathbb{R}}$.

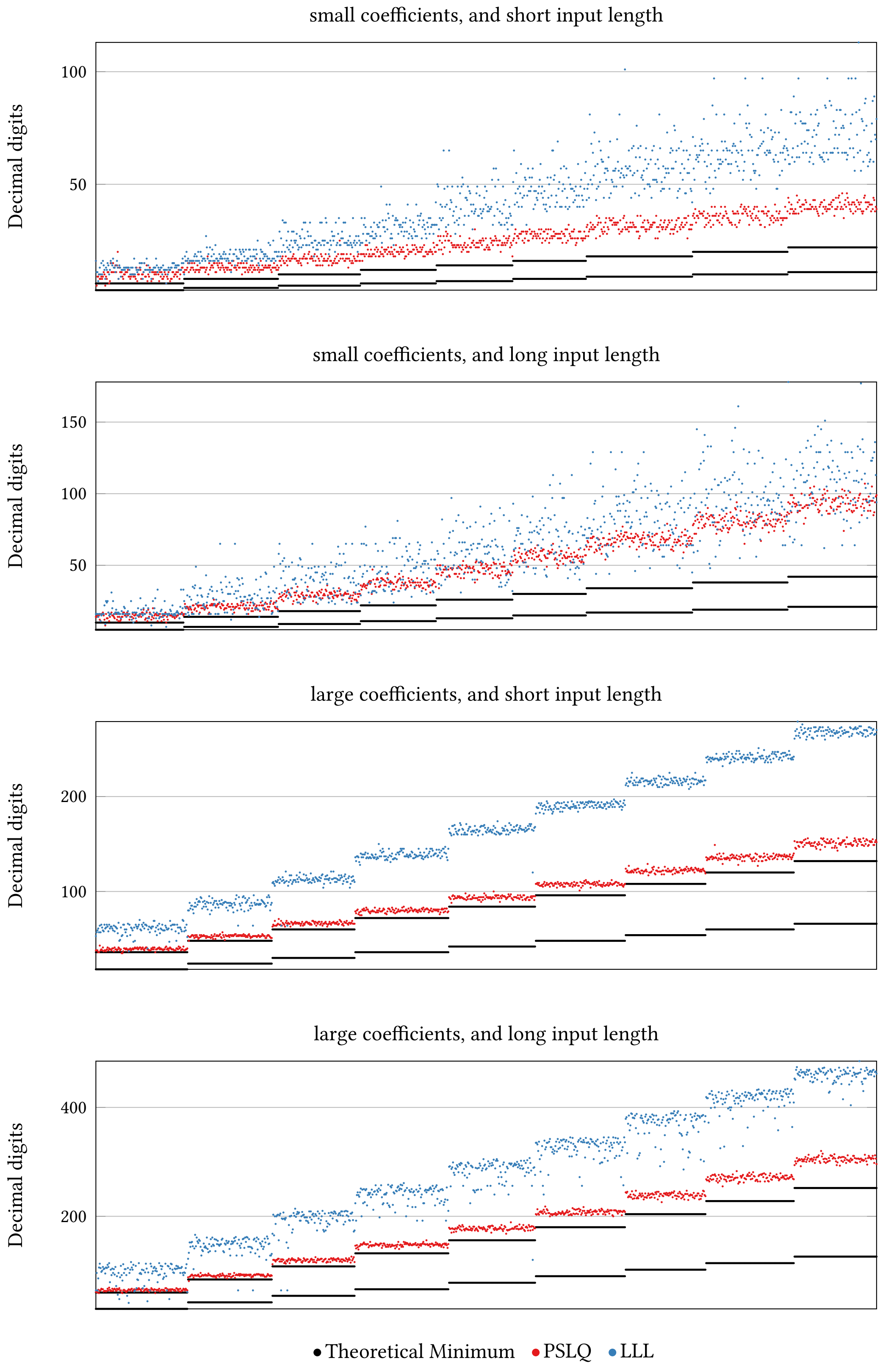


Figure B.6: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{7}), C = C_{\mathbb{R}}$.

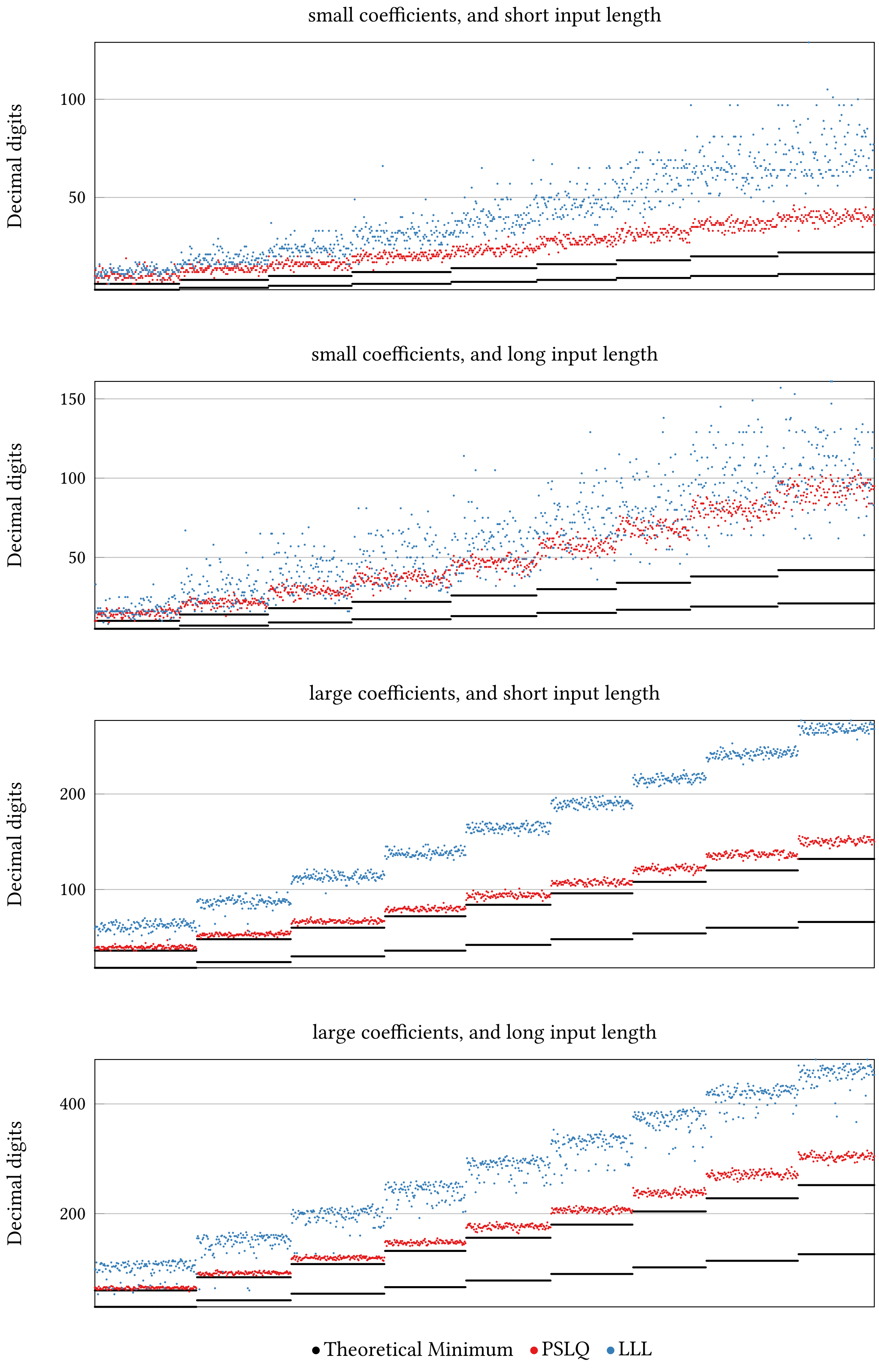


Figure B.7: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{10}), C = C_{\mathbb{R}}$.

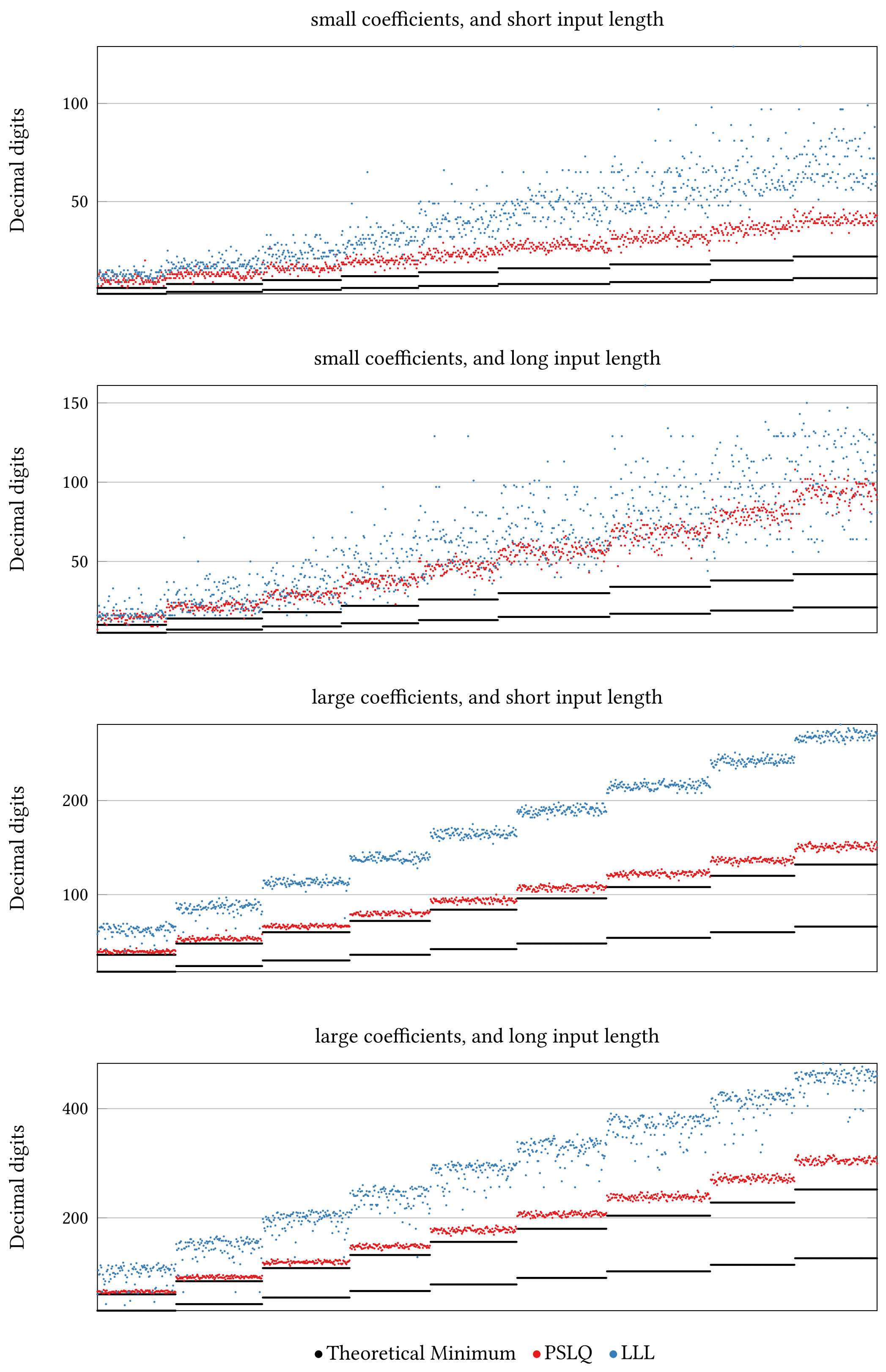


Figure B.8: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{11}), C = C_{\mathbb{R}}$.

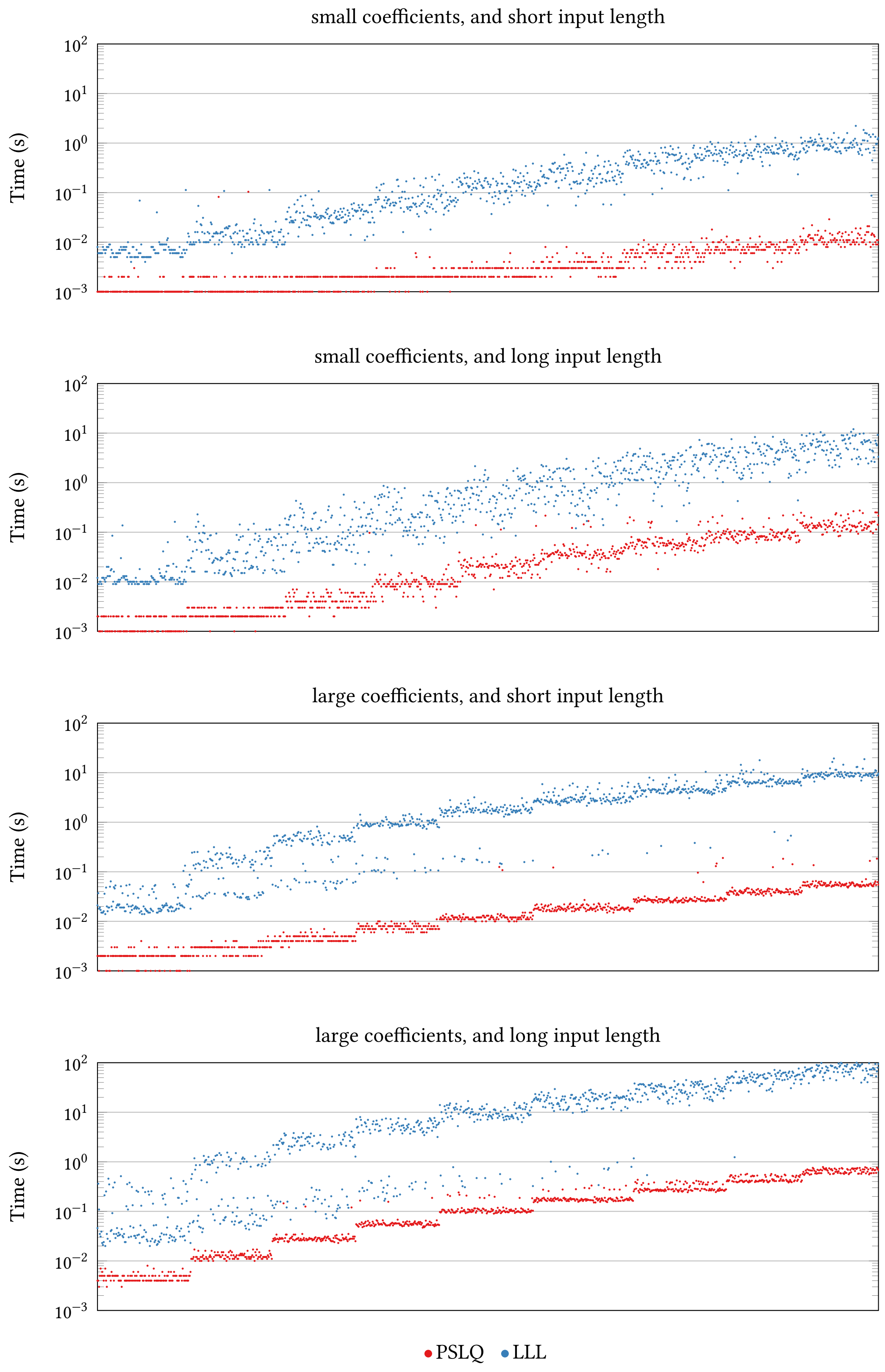


Figure B.9: Computation time for $\mathbb{K} = \mathbb{Q}, C = C_{\mathbb{R}}$.

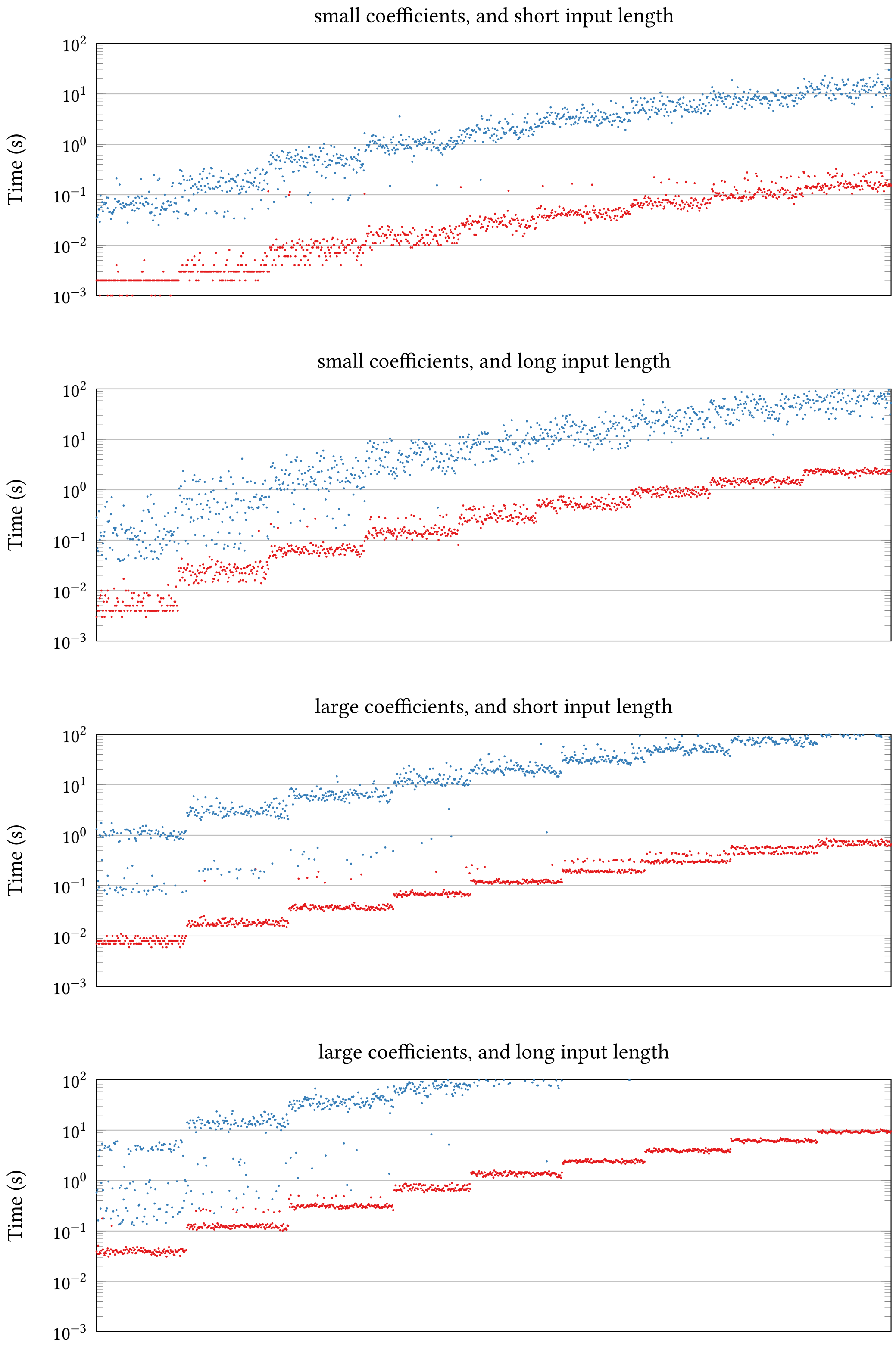


Figure B.10: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{2}), C = C_{\mathbb{R}}$.

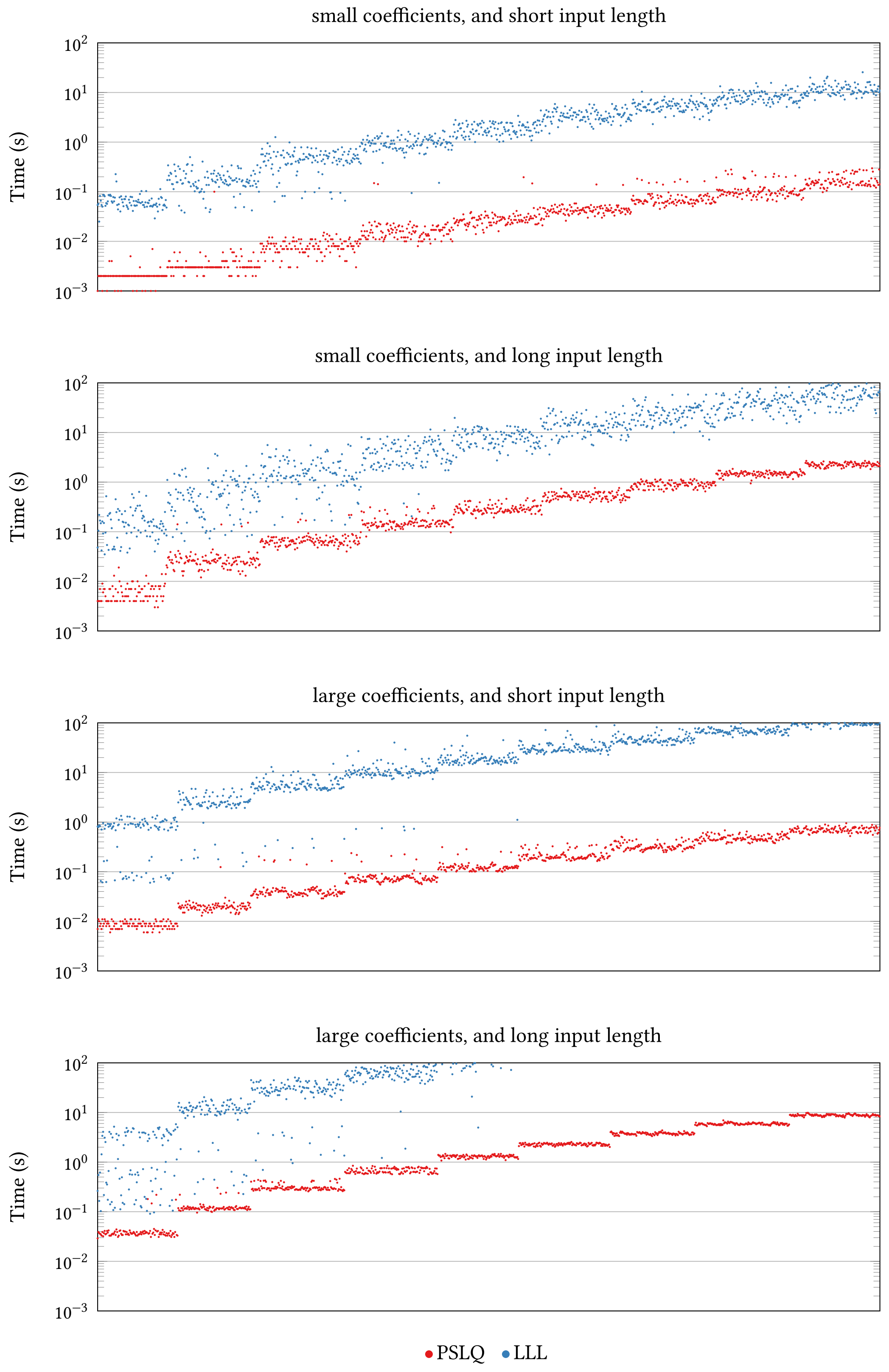


Figure B.11: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{3}), C = C_{\mathbb{R}}$.

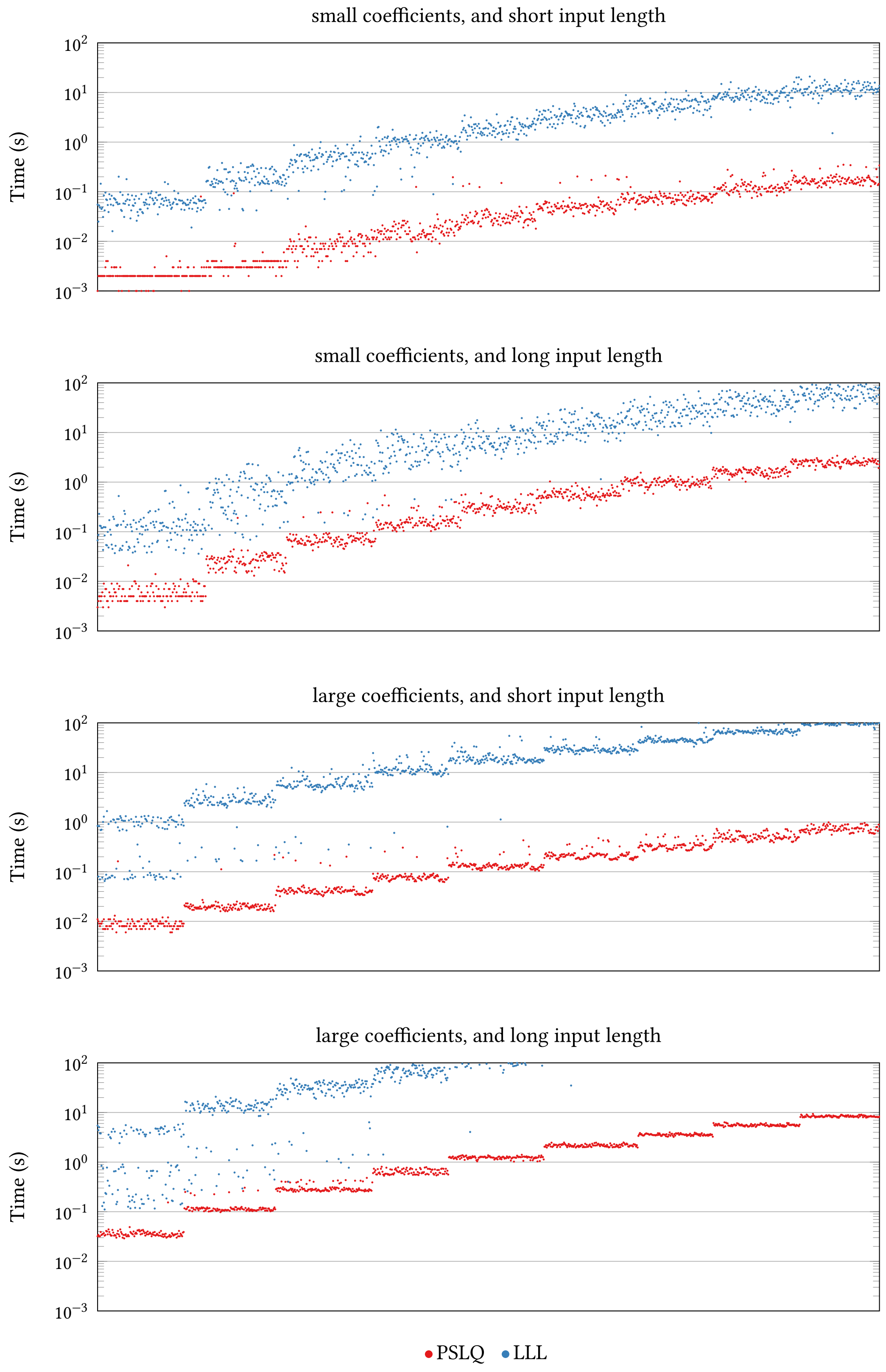


Figure B.12: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{5}), C = C_{\mathbb{R}}$.

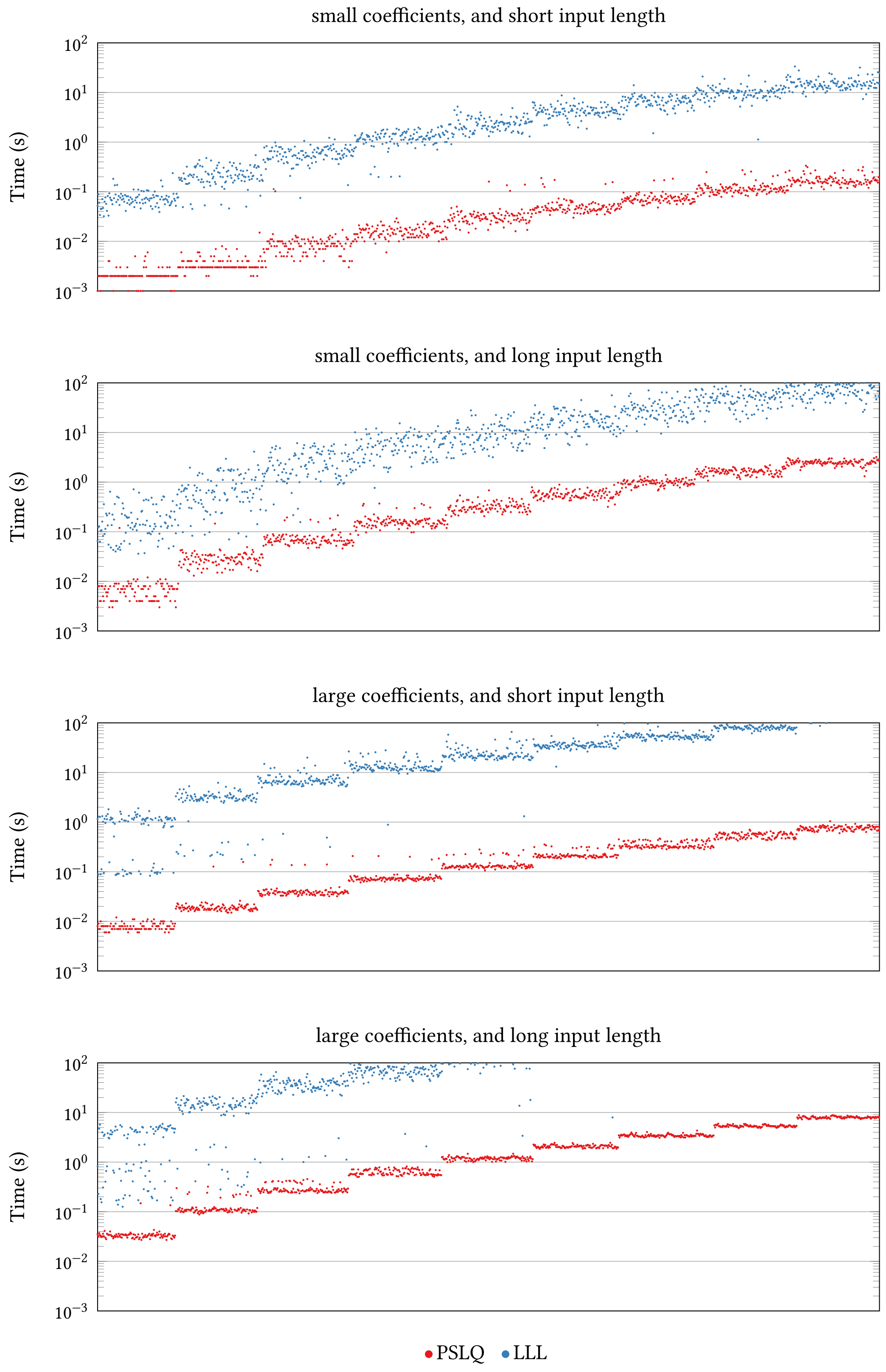


Figure B.13: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{6}), C = C_{\mathbb{R}}$.

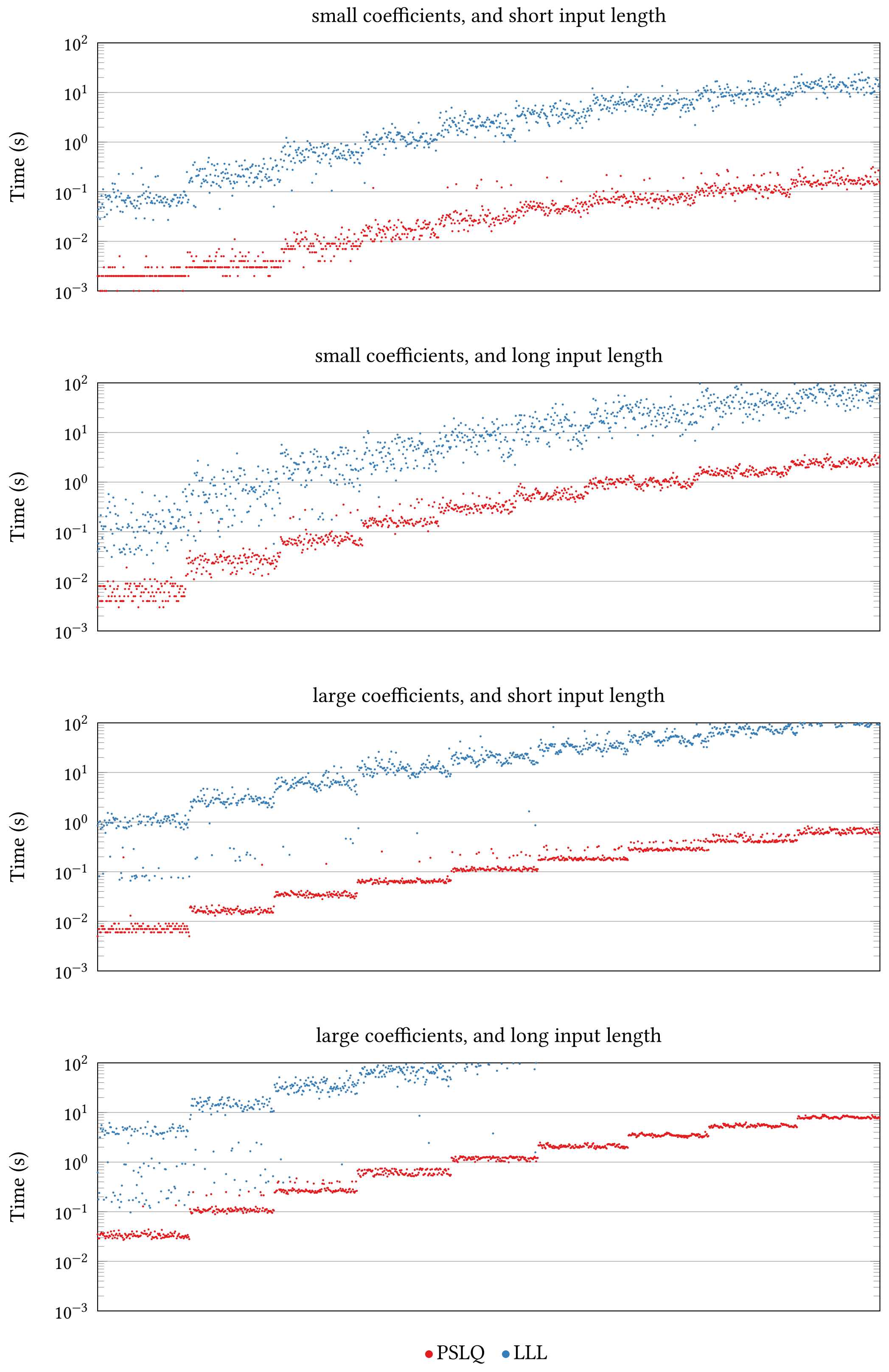


Figure B.14: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{7}), C = C_{\mathbb{R}}$.

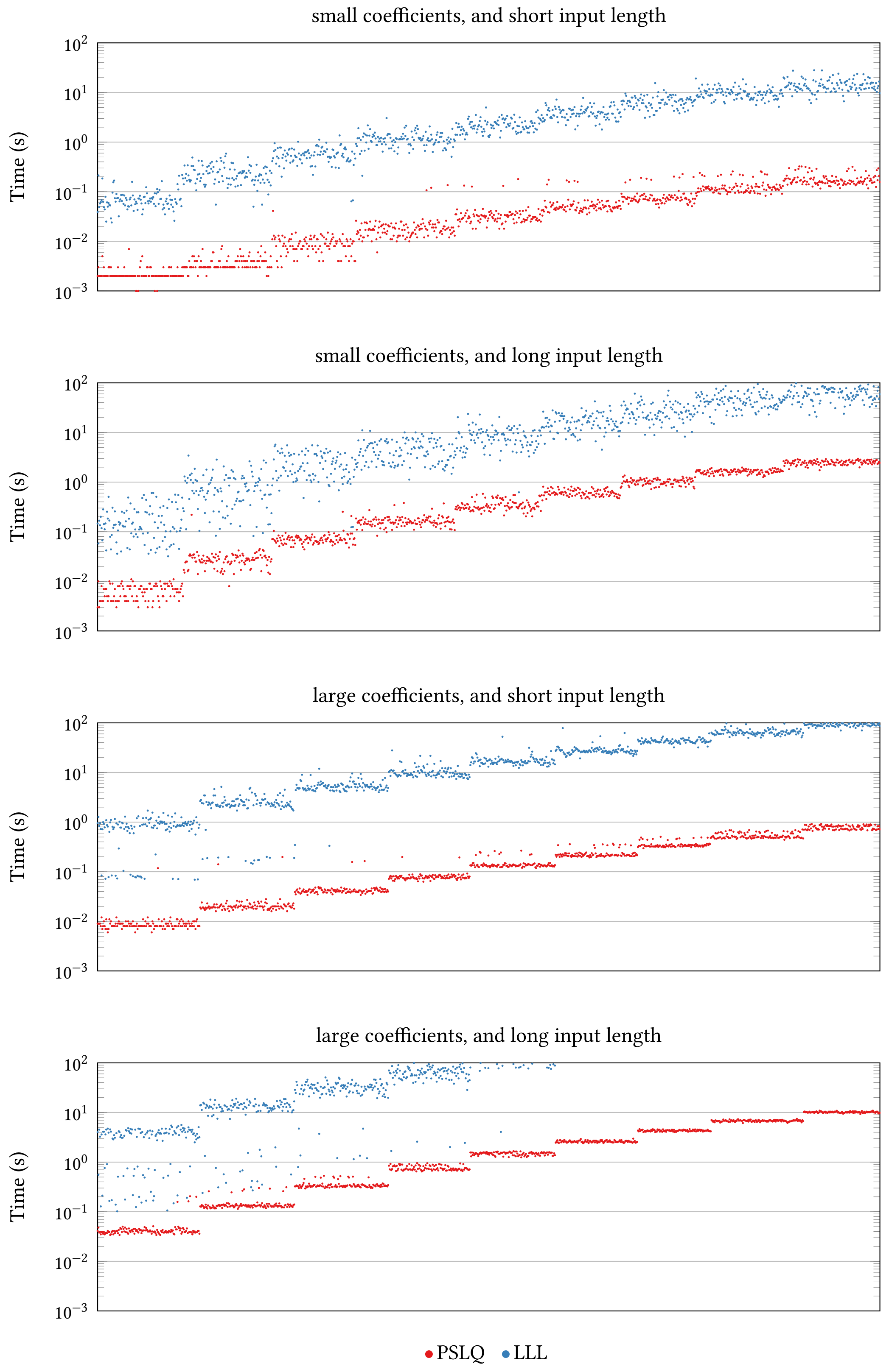


Figure B.15: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{10}), C = C_{\mathbb{R}}$.

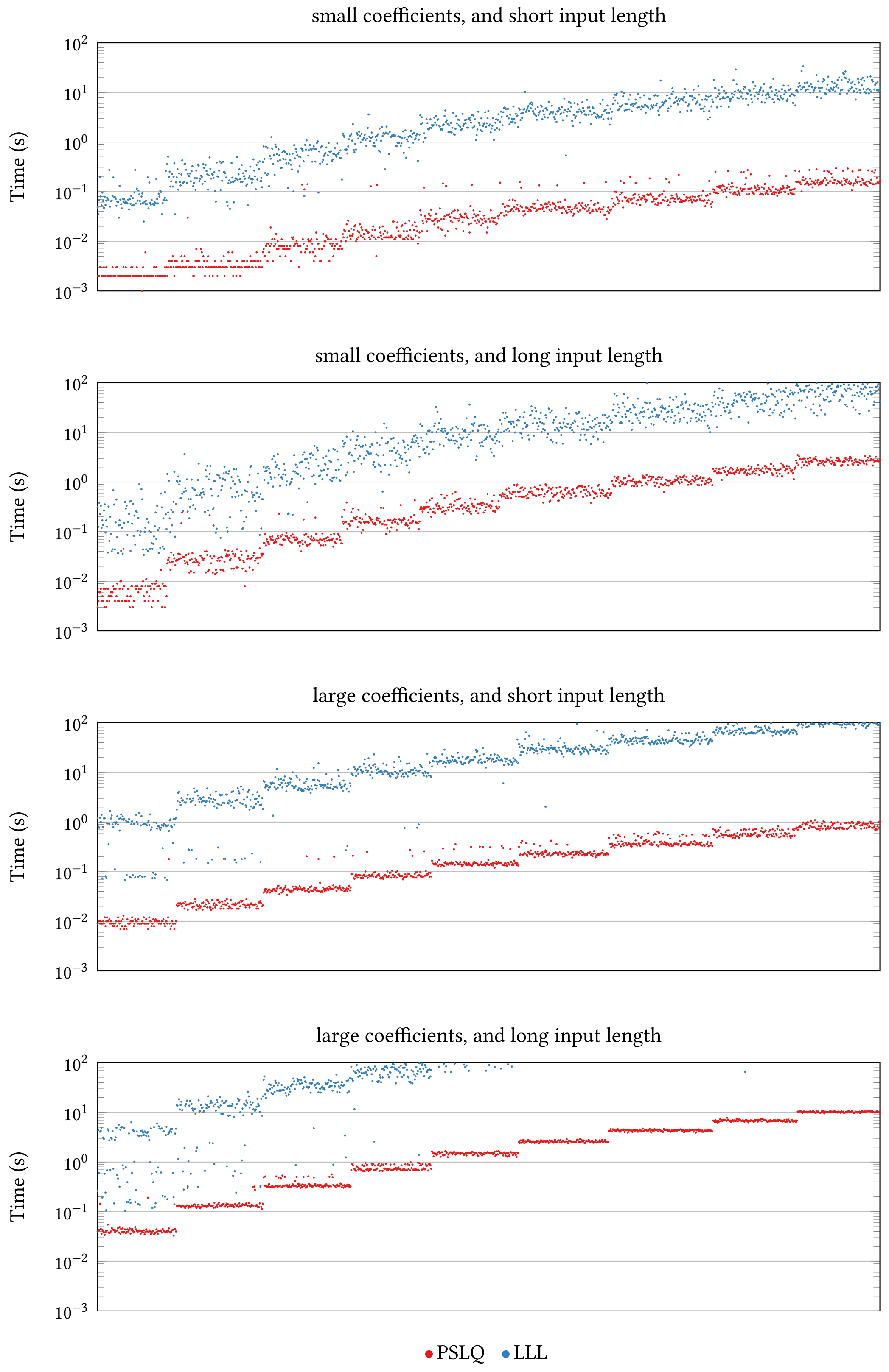


Figure B.16: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{11}), C = C_{\mathbb{R}}$.

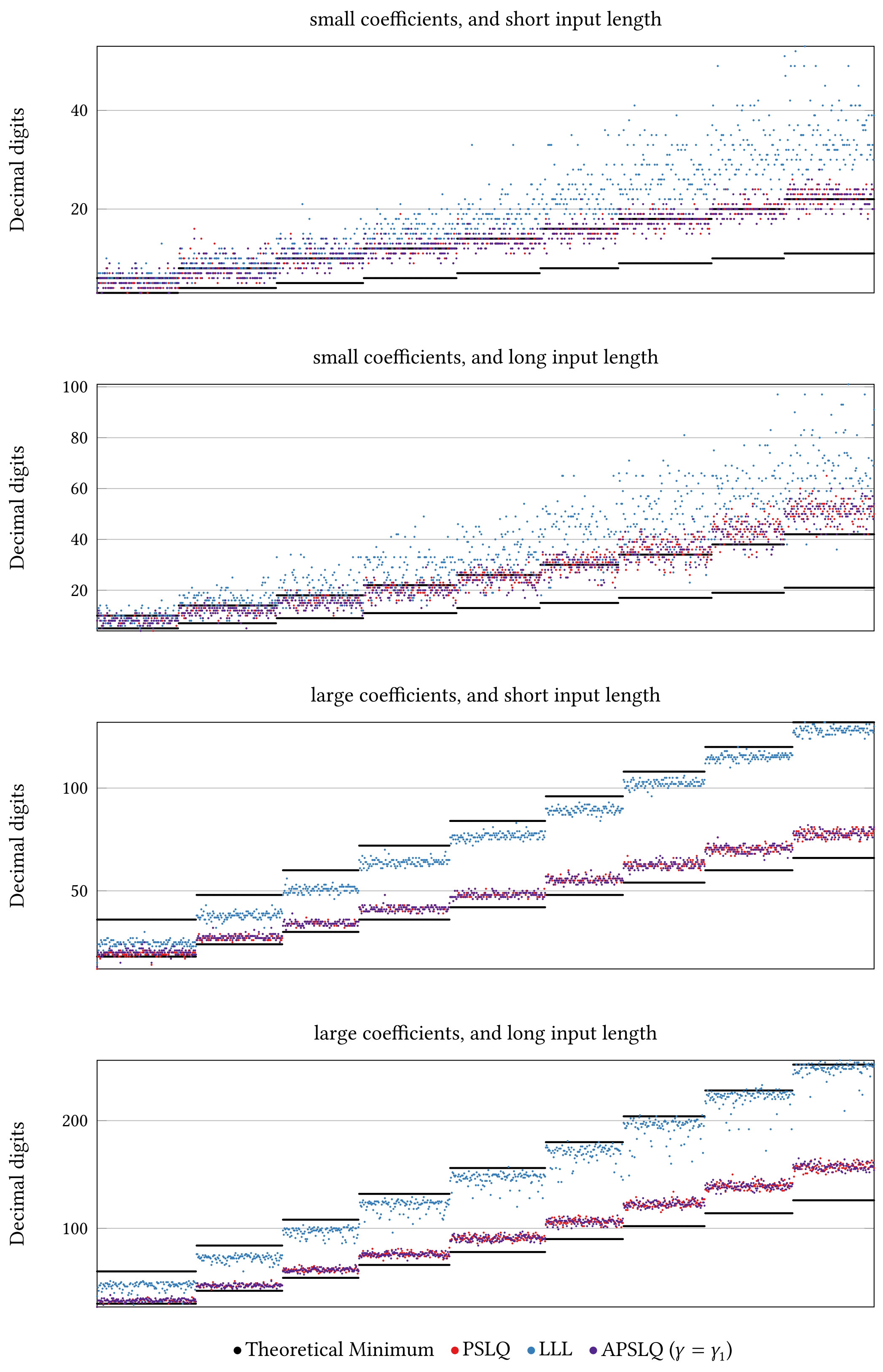


Figure B.17: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$).

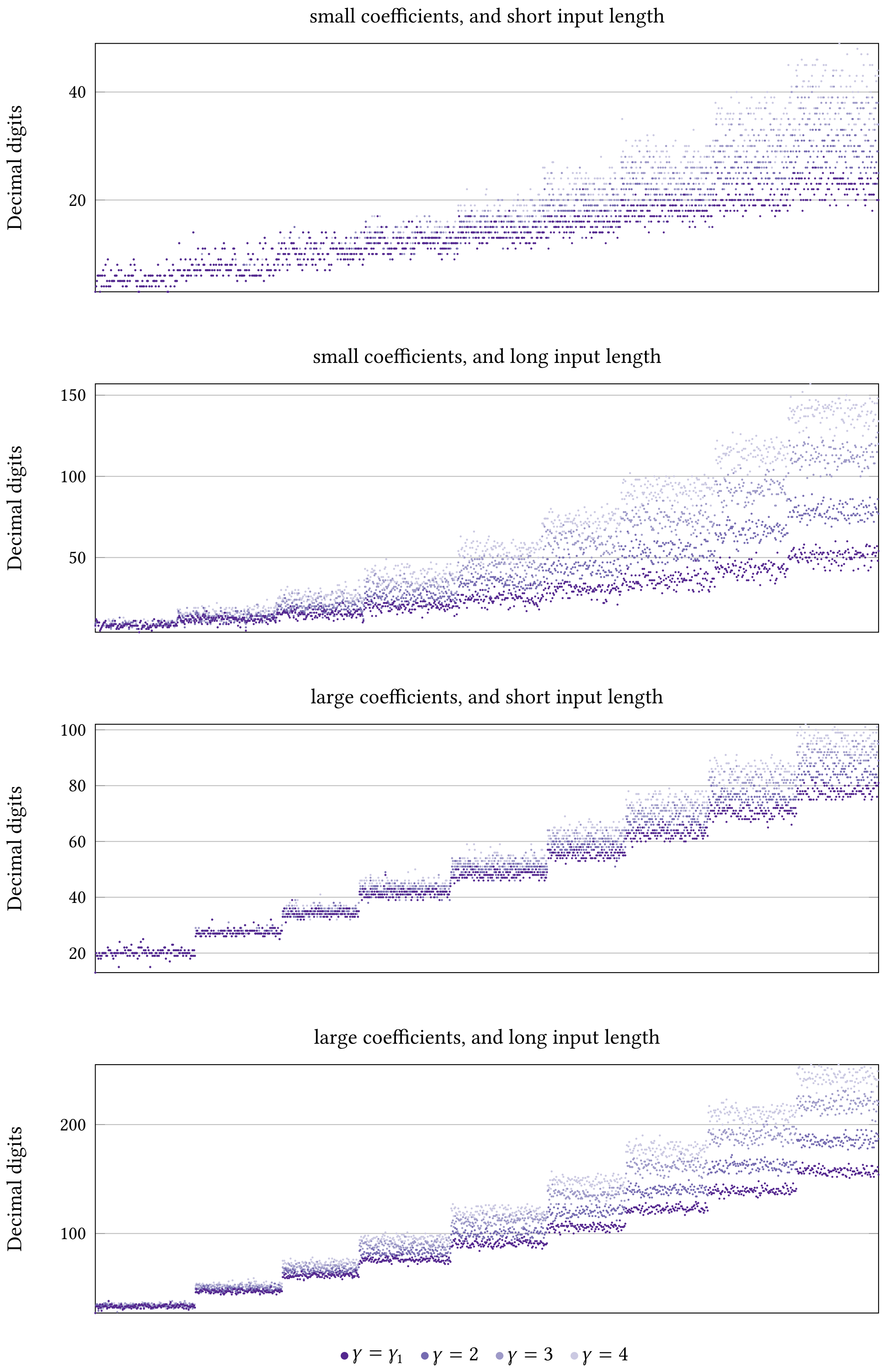


Figure B.18: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$ (APSLQ only).

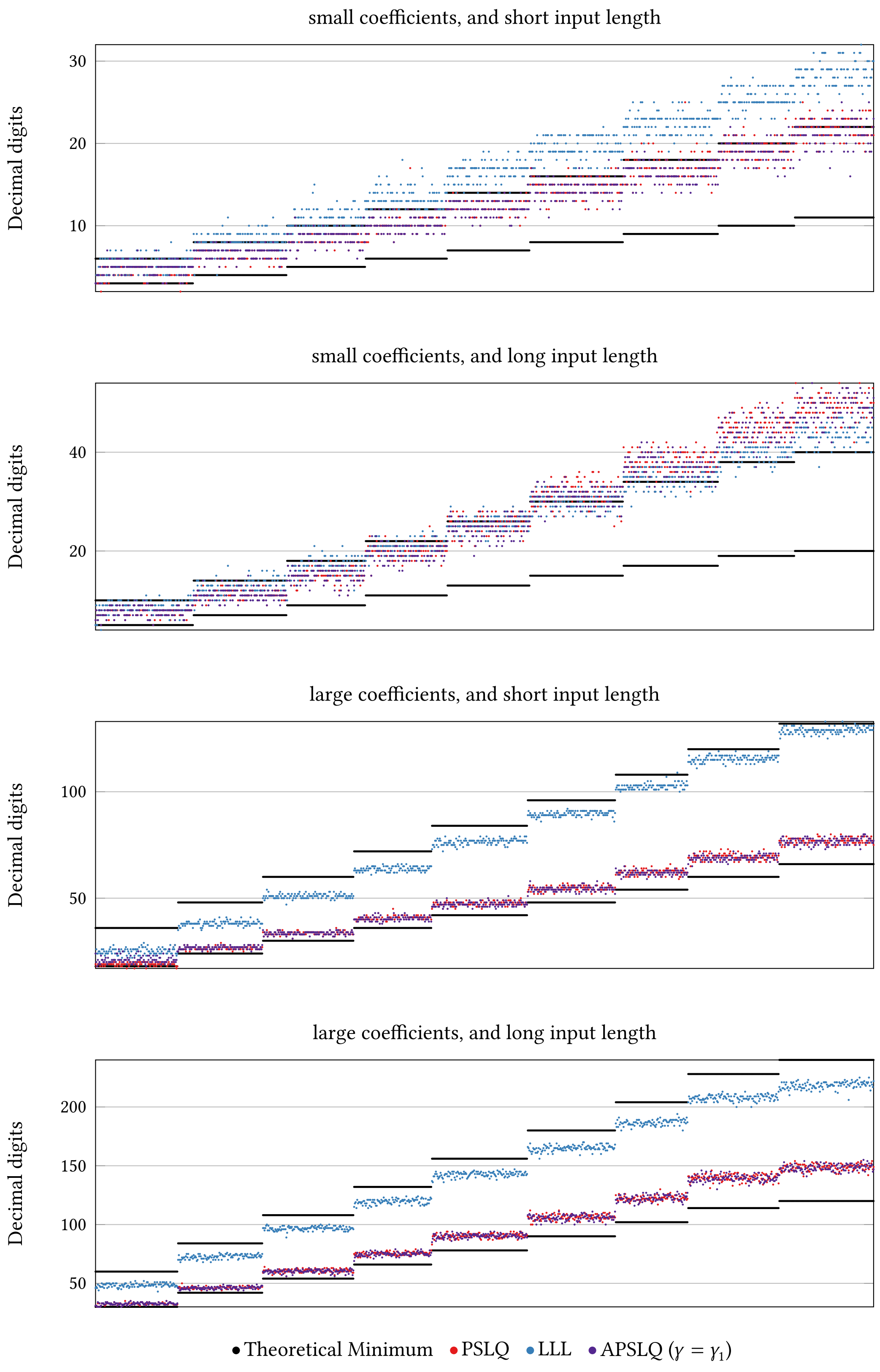


Figure B.19: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$).

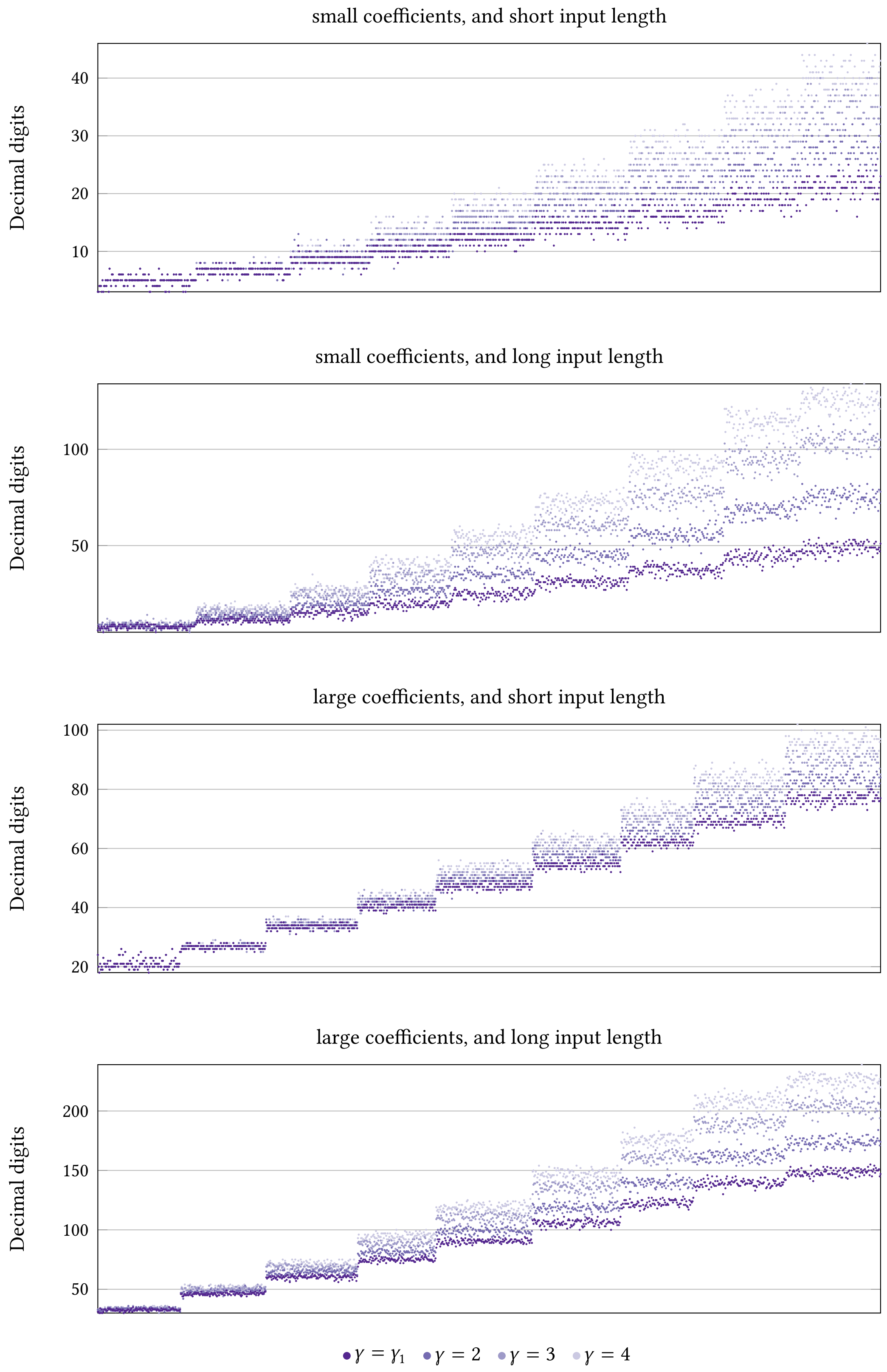


Figure B.20: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$ (APSLQ only).

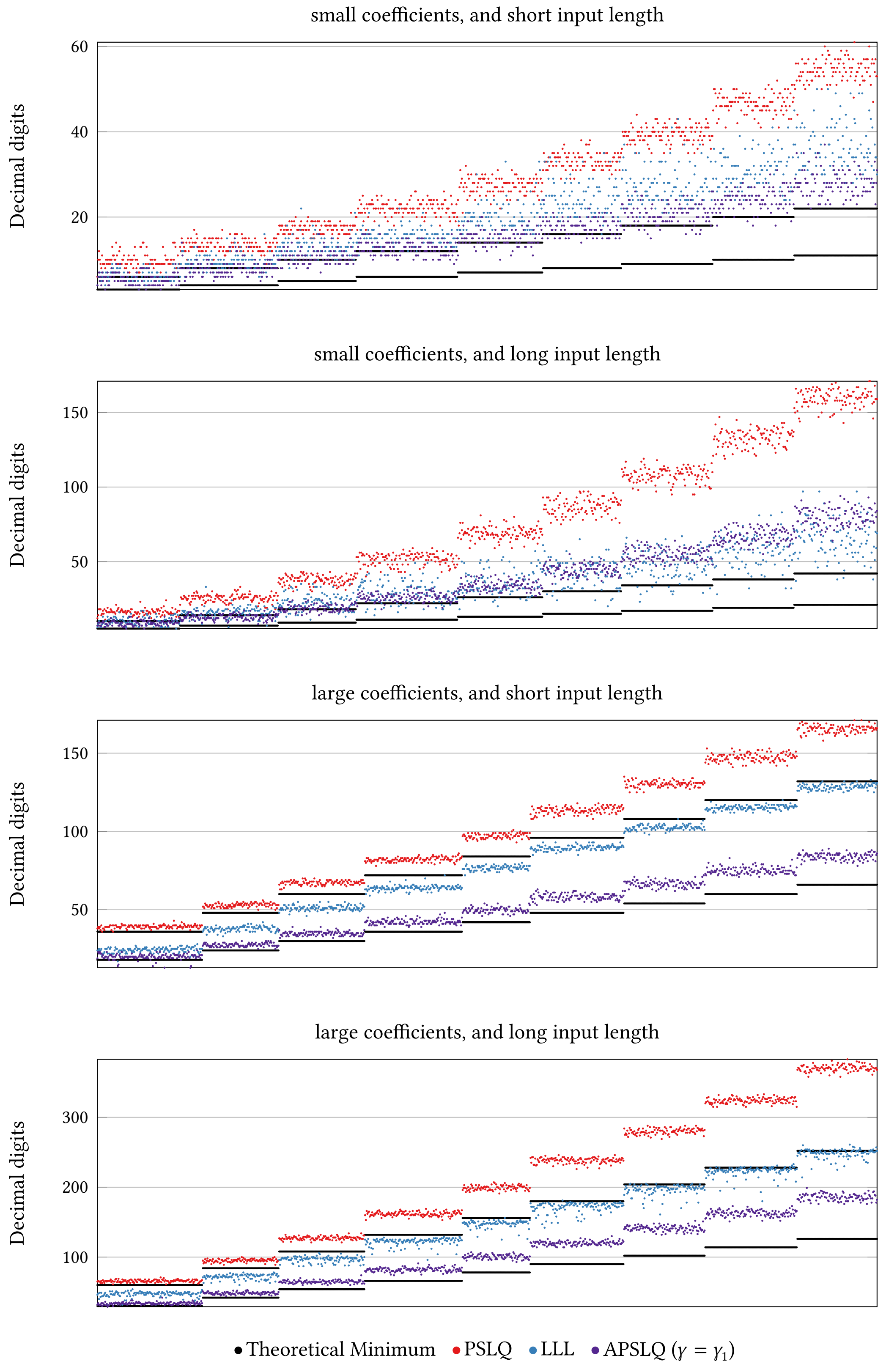


Figure B.21: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$).

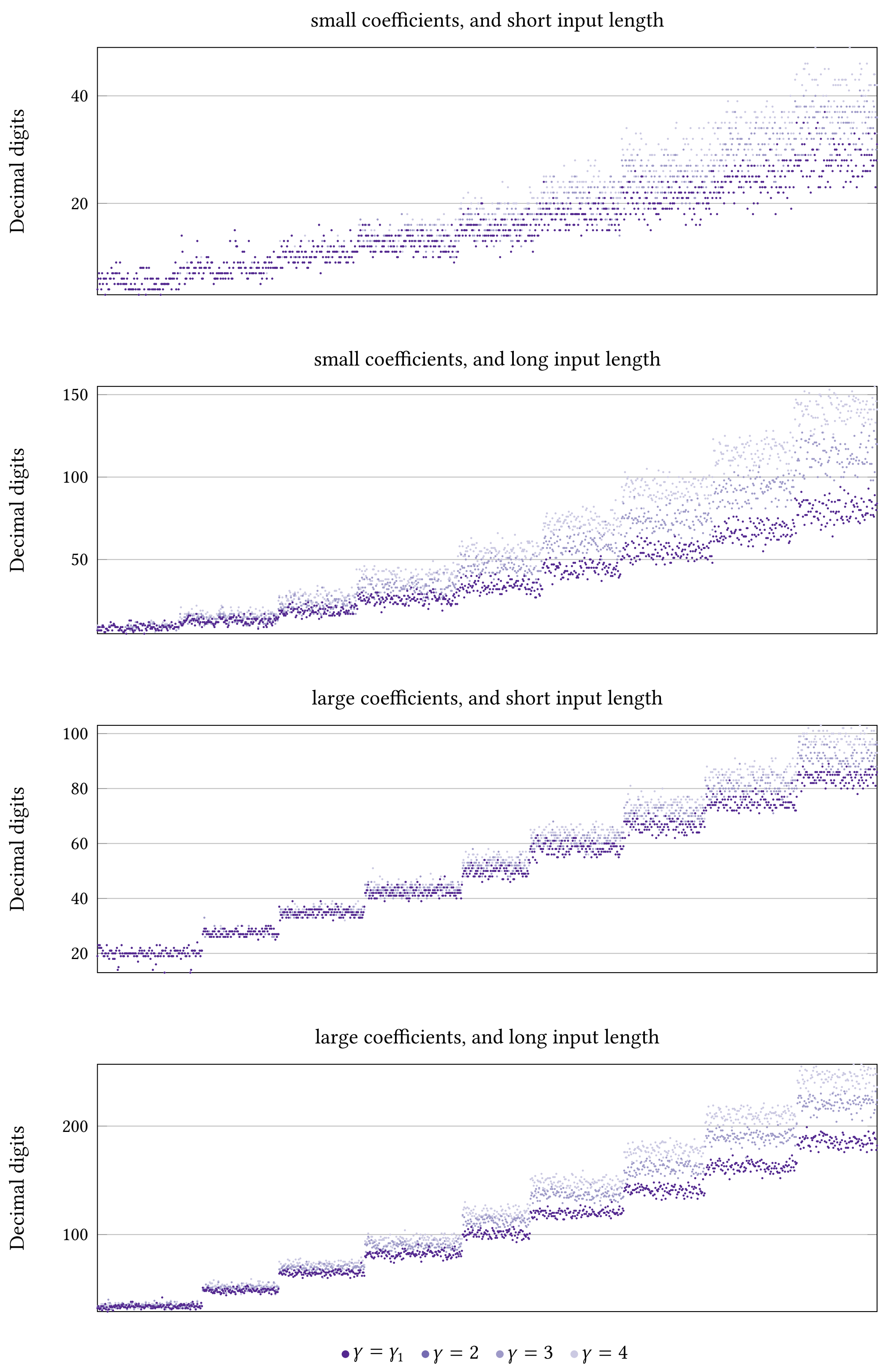


Figure B.22: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$ (APSLQ only).

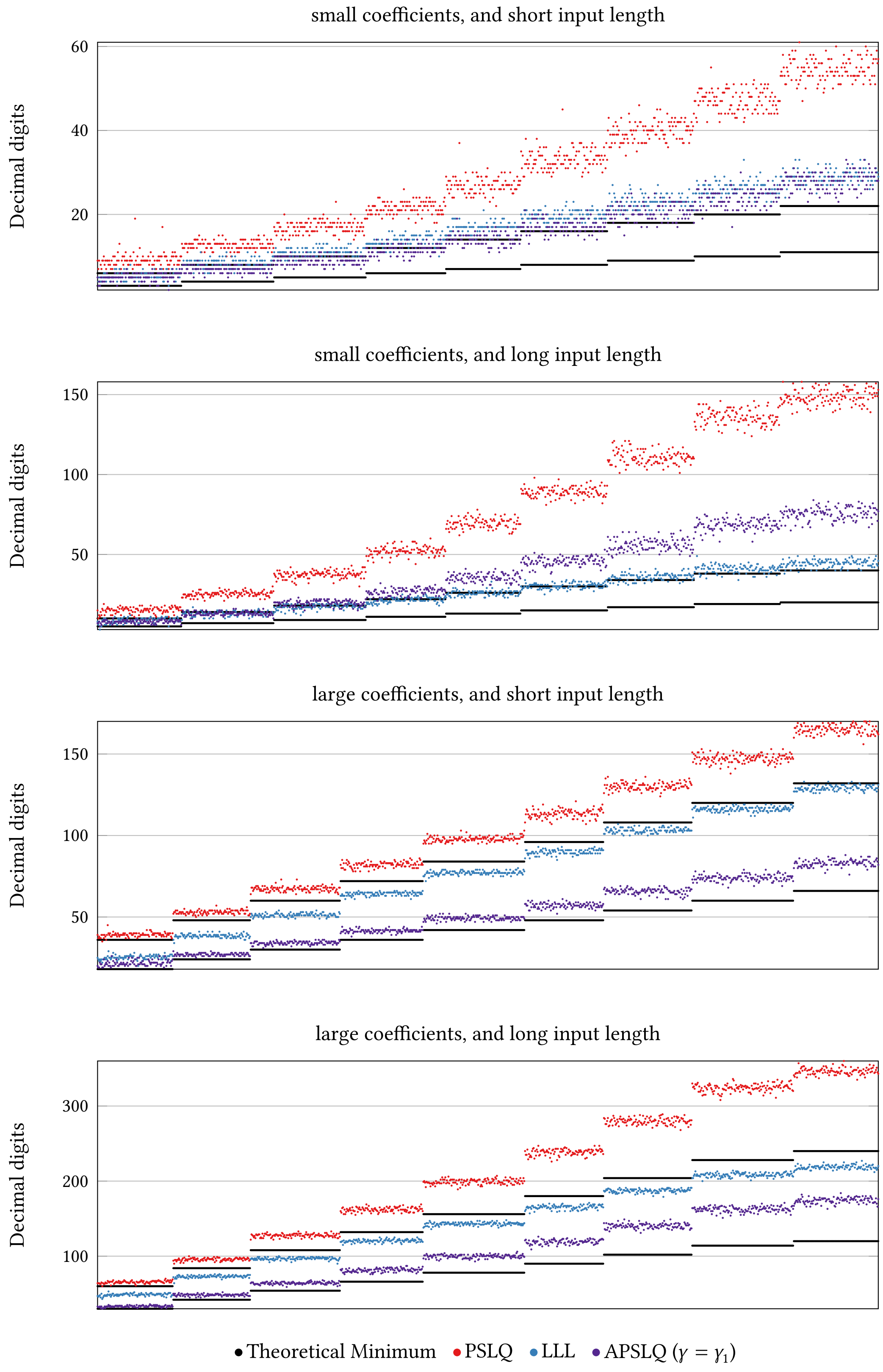


Figure B.23: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$).

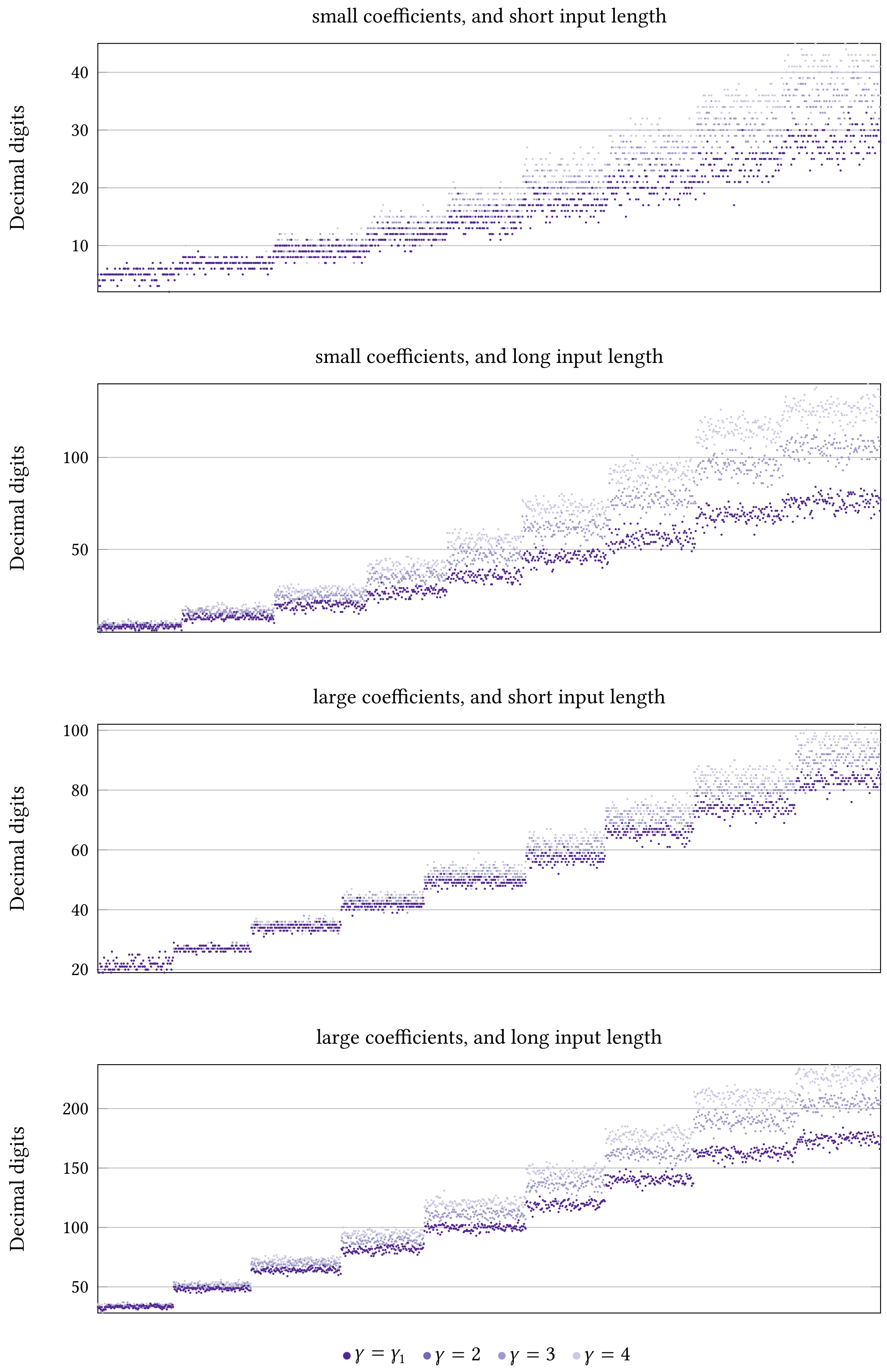


Figure B.24: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$ (APSLQ only).

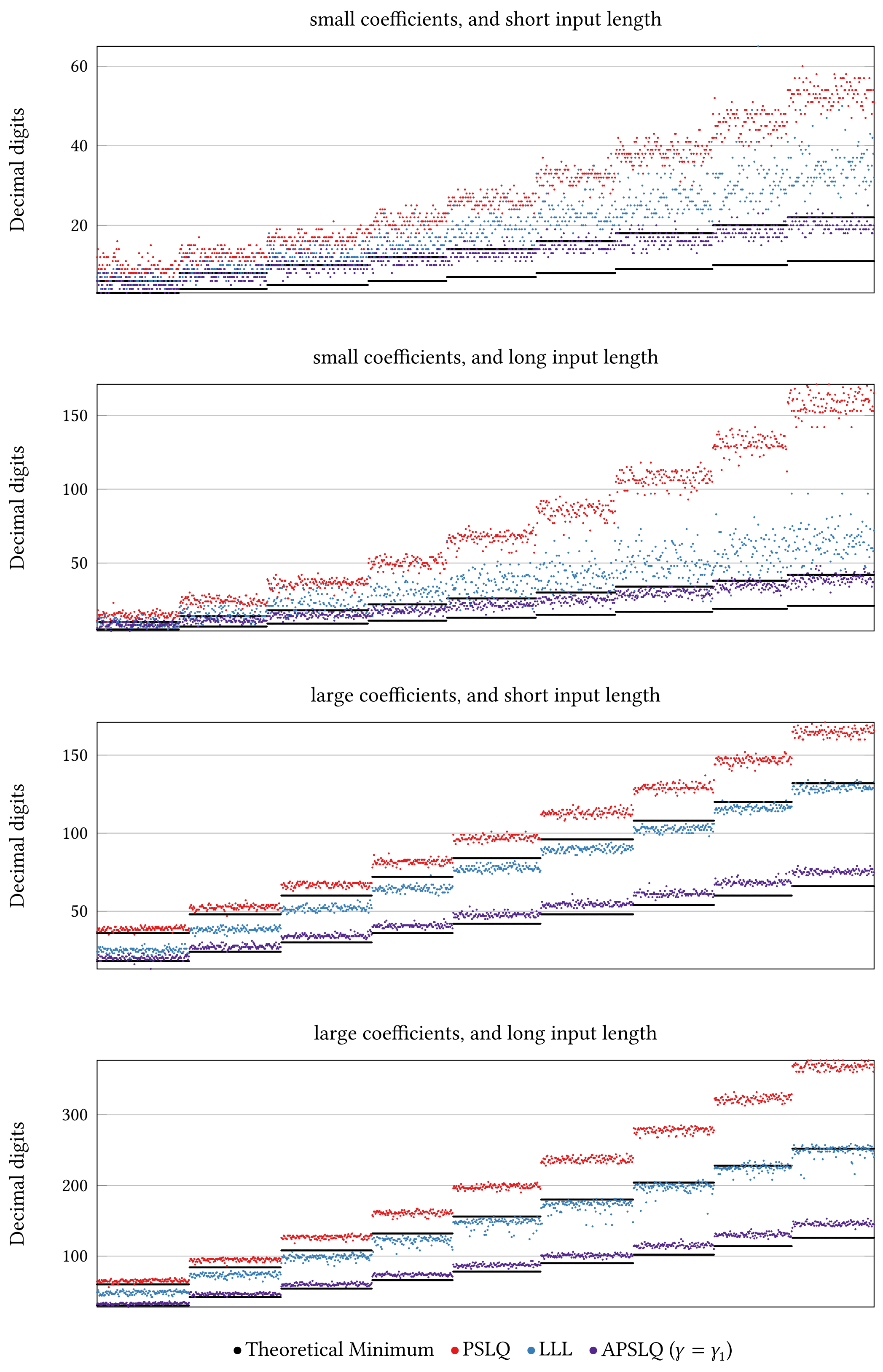


Figure B.25: Precision required for $(\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{R}})$.

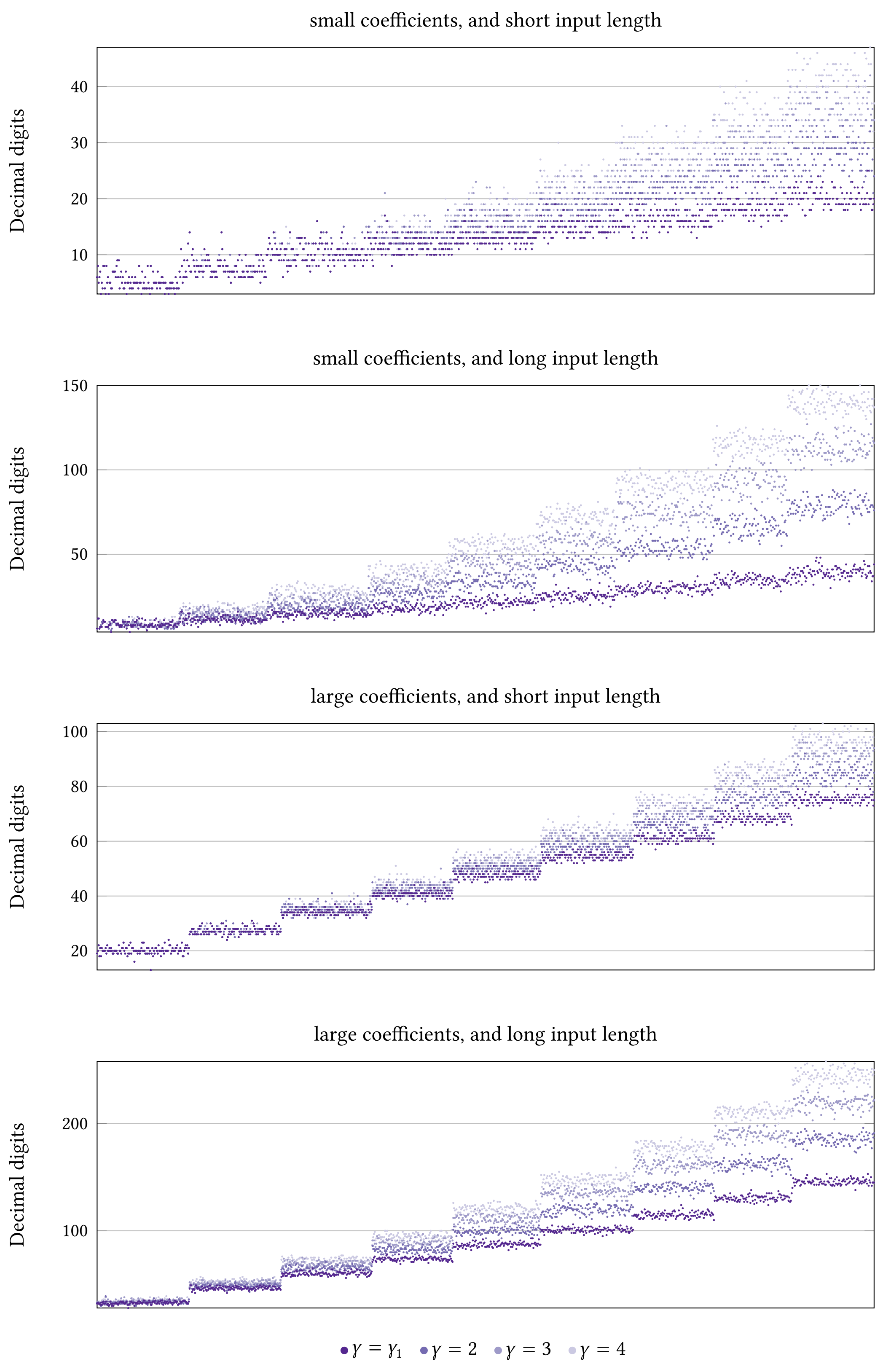


Figure B.26: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{R}}$ (APSLQ only).

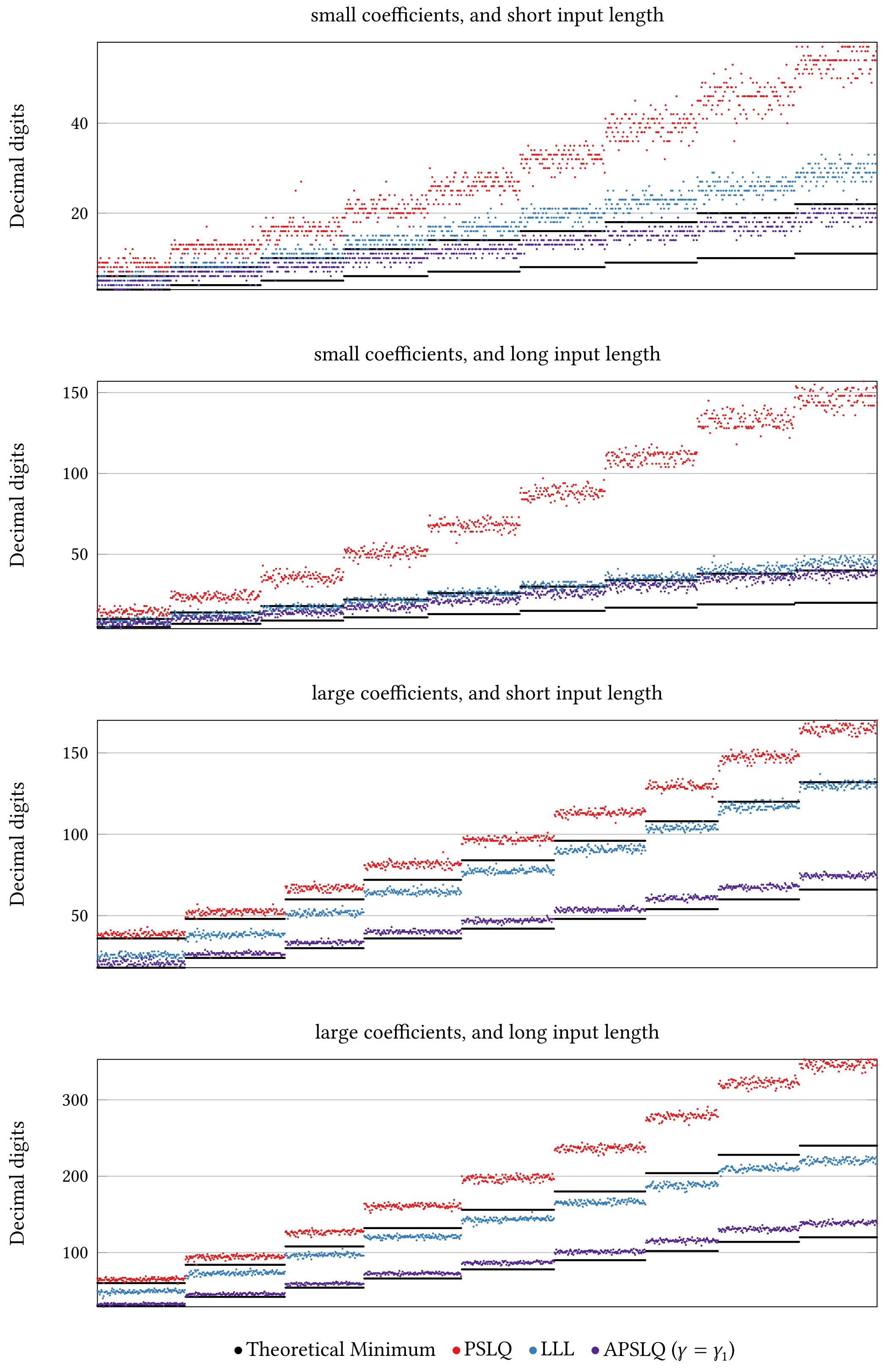


Figure B.27: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{C}}$).

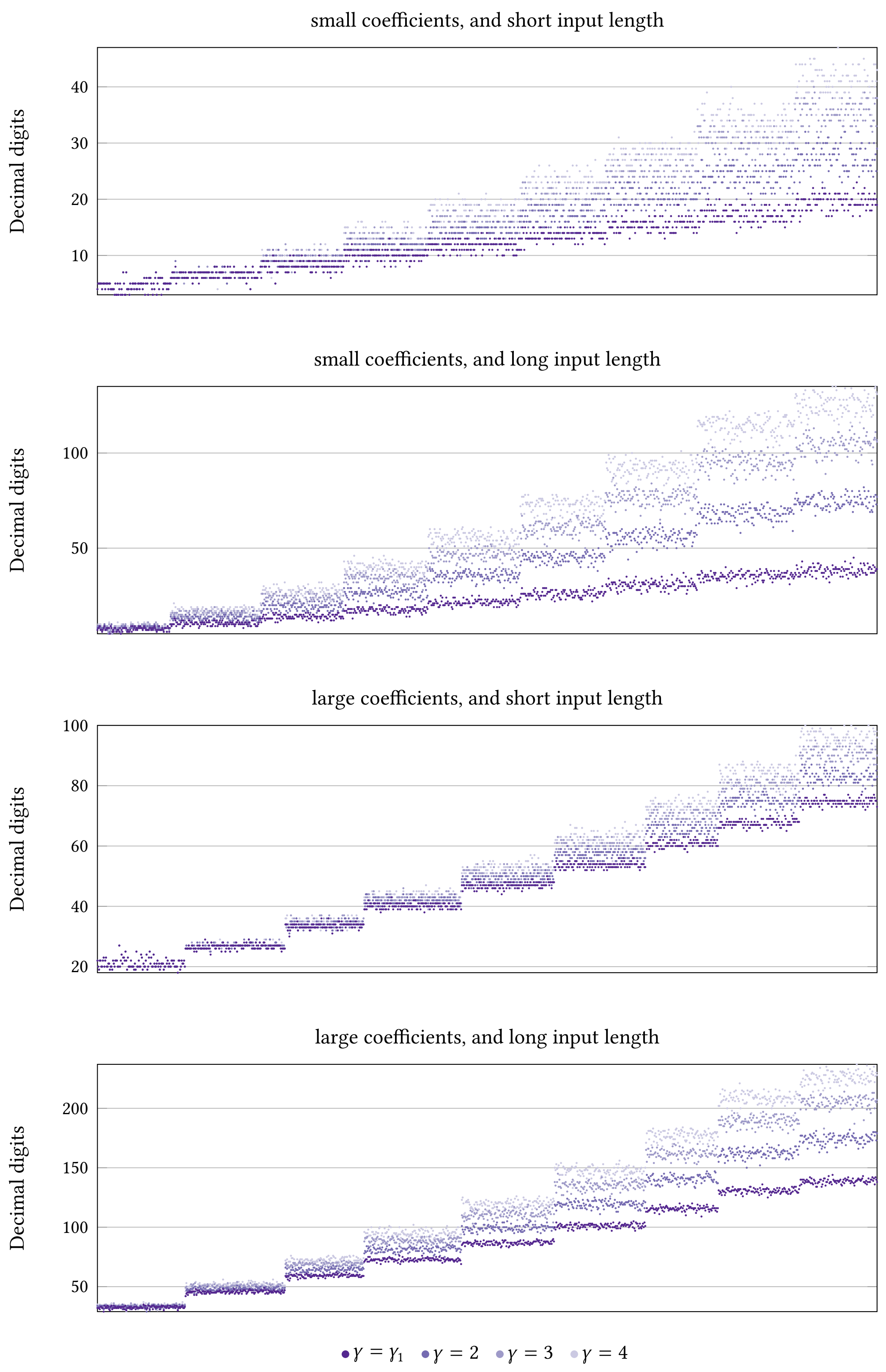


Figure B.28: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{C}}$ (APSLQ only).

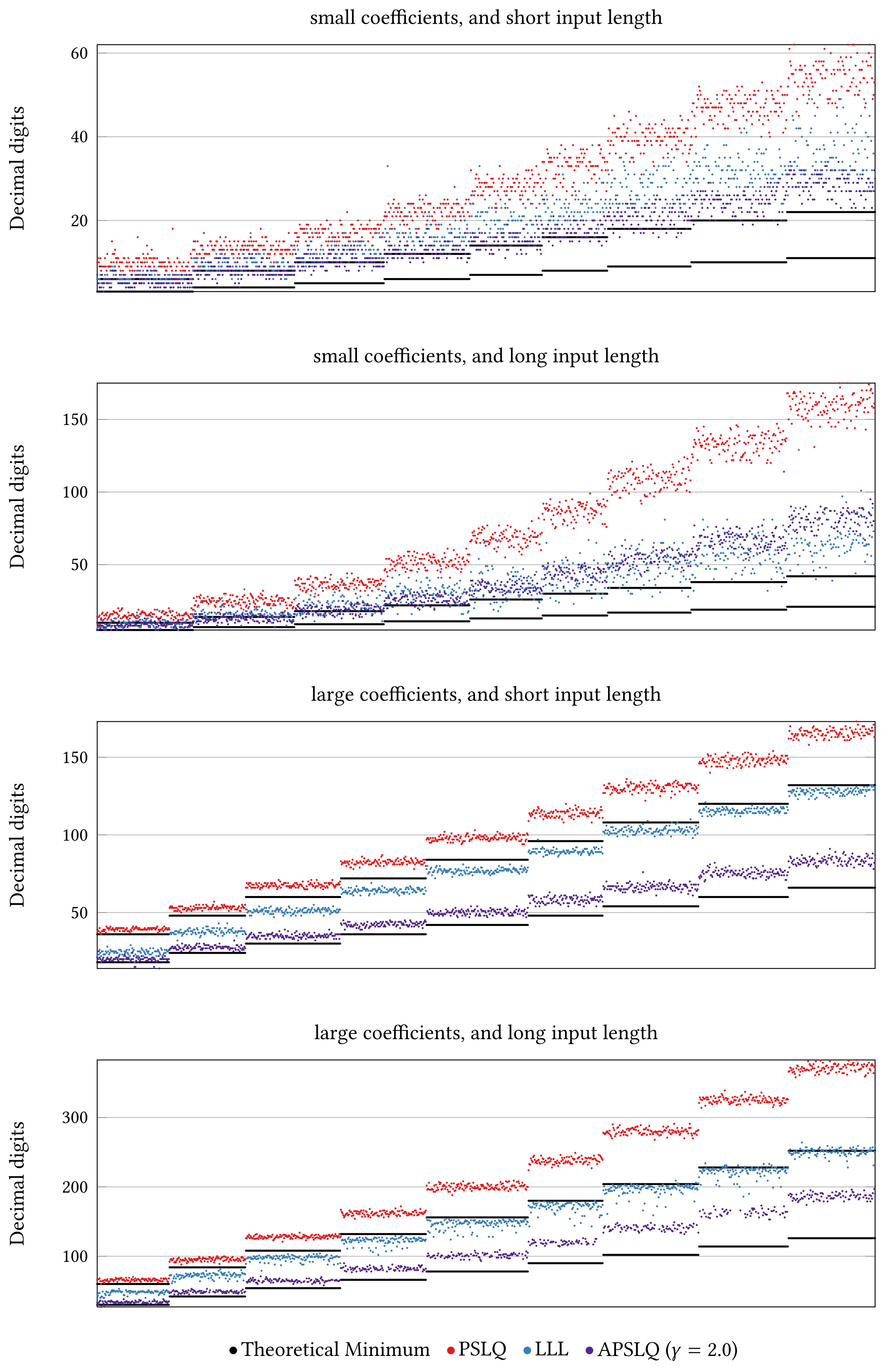


Figure B.29: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{R}}$).

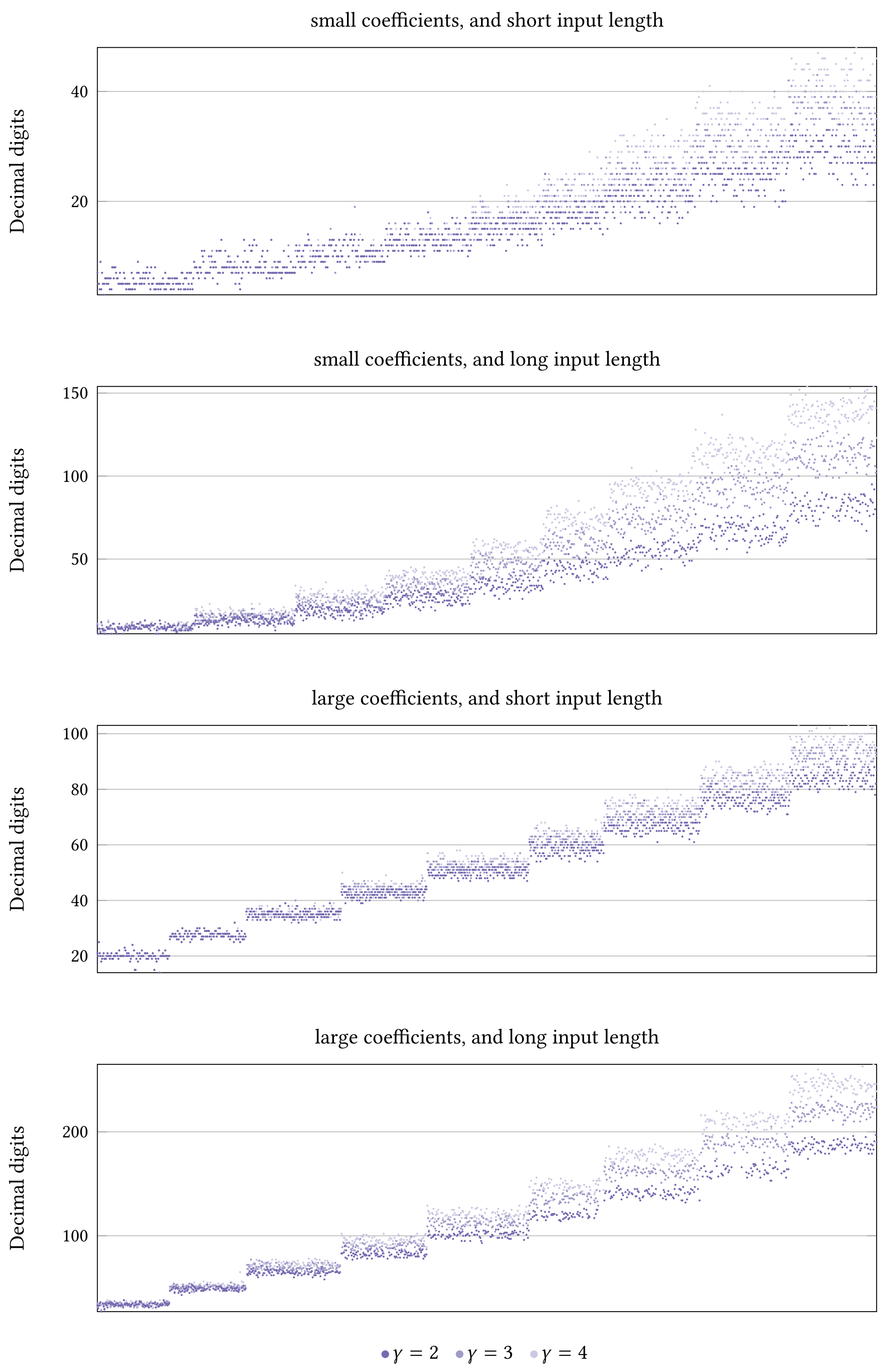


Figure B.30: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{R}}$ (APSLQ only).

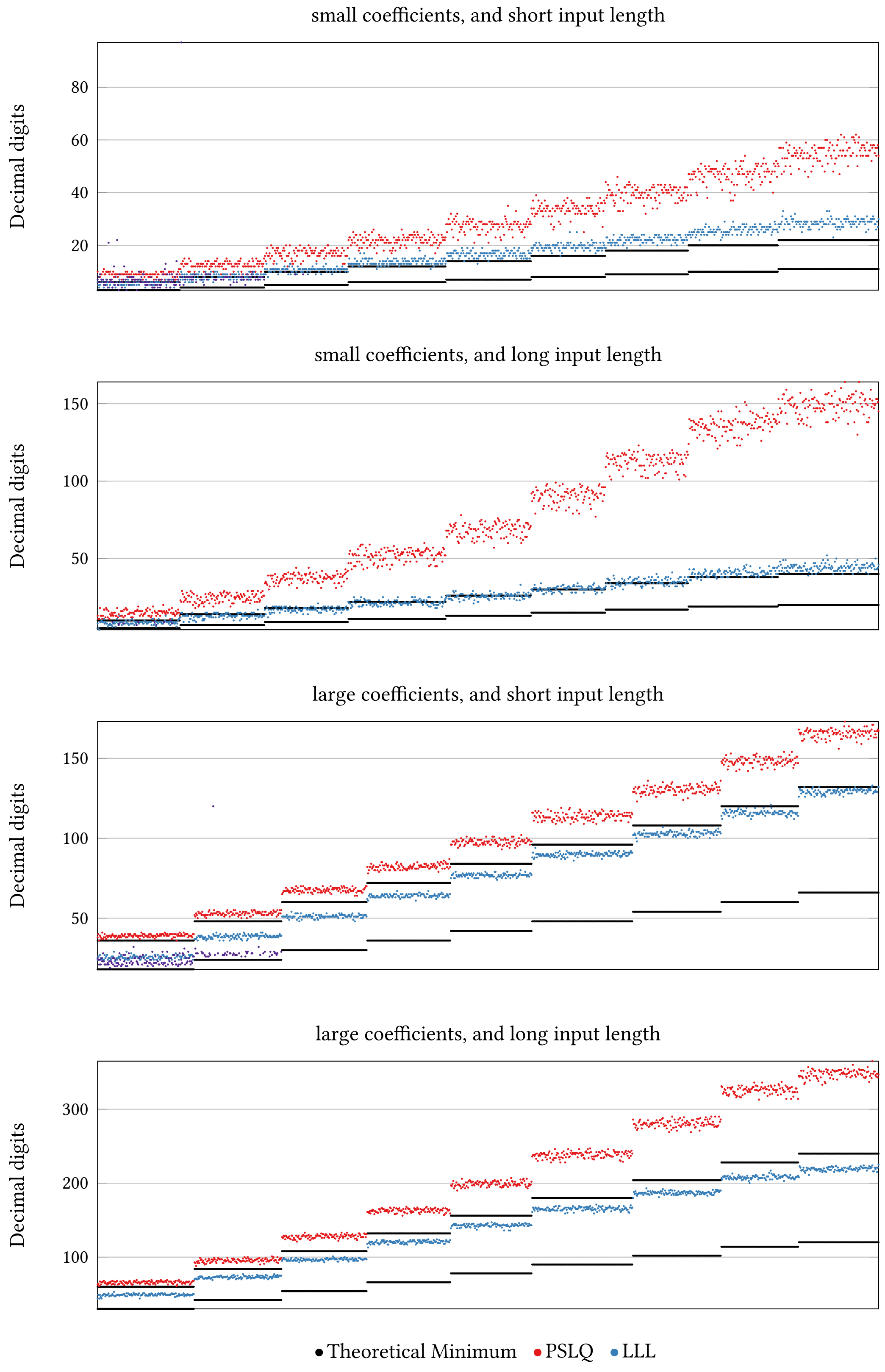


Figure B.31: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{C}}$).

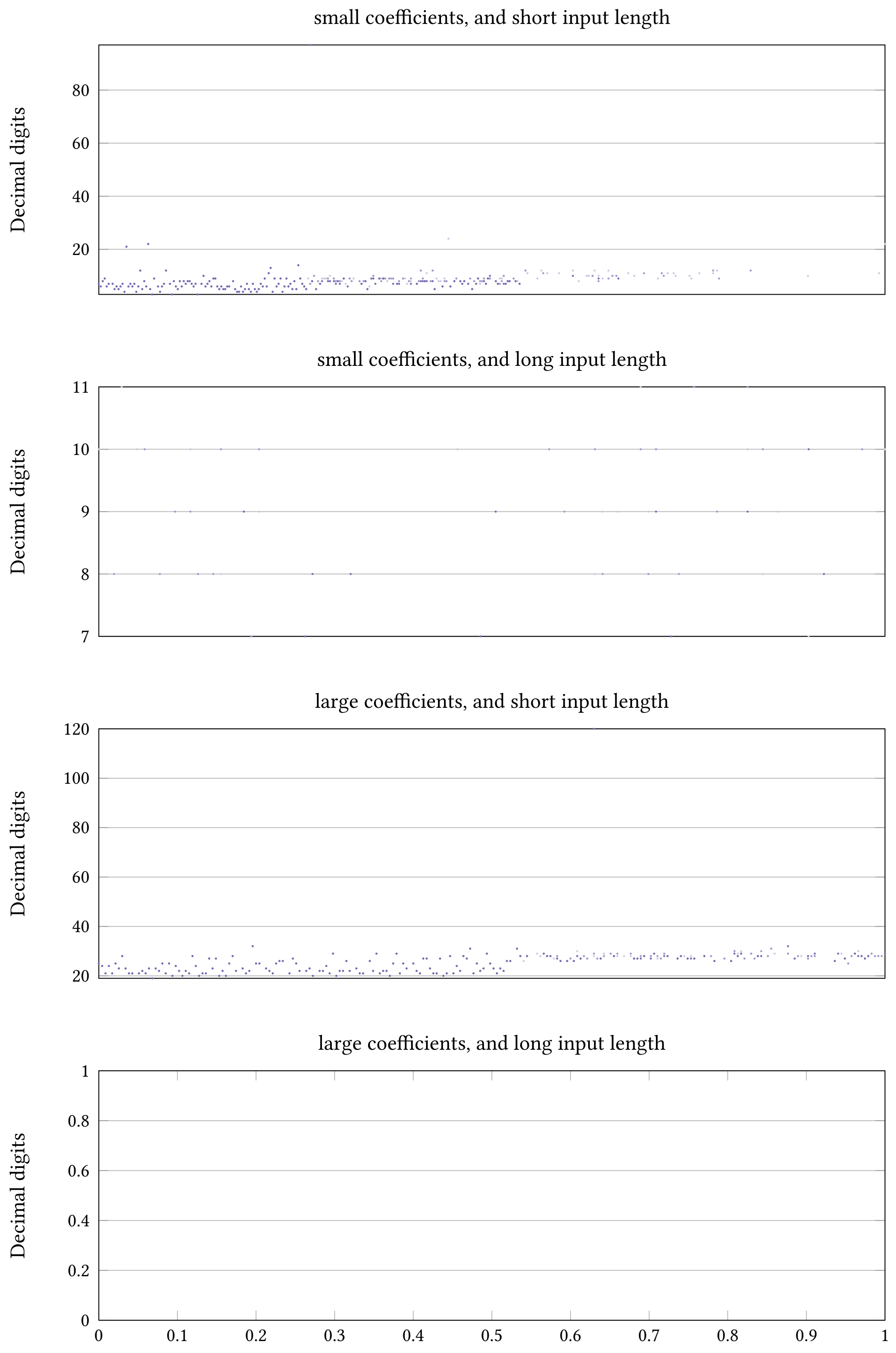


Figure B.32: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{C}}$ (APSLQ only).

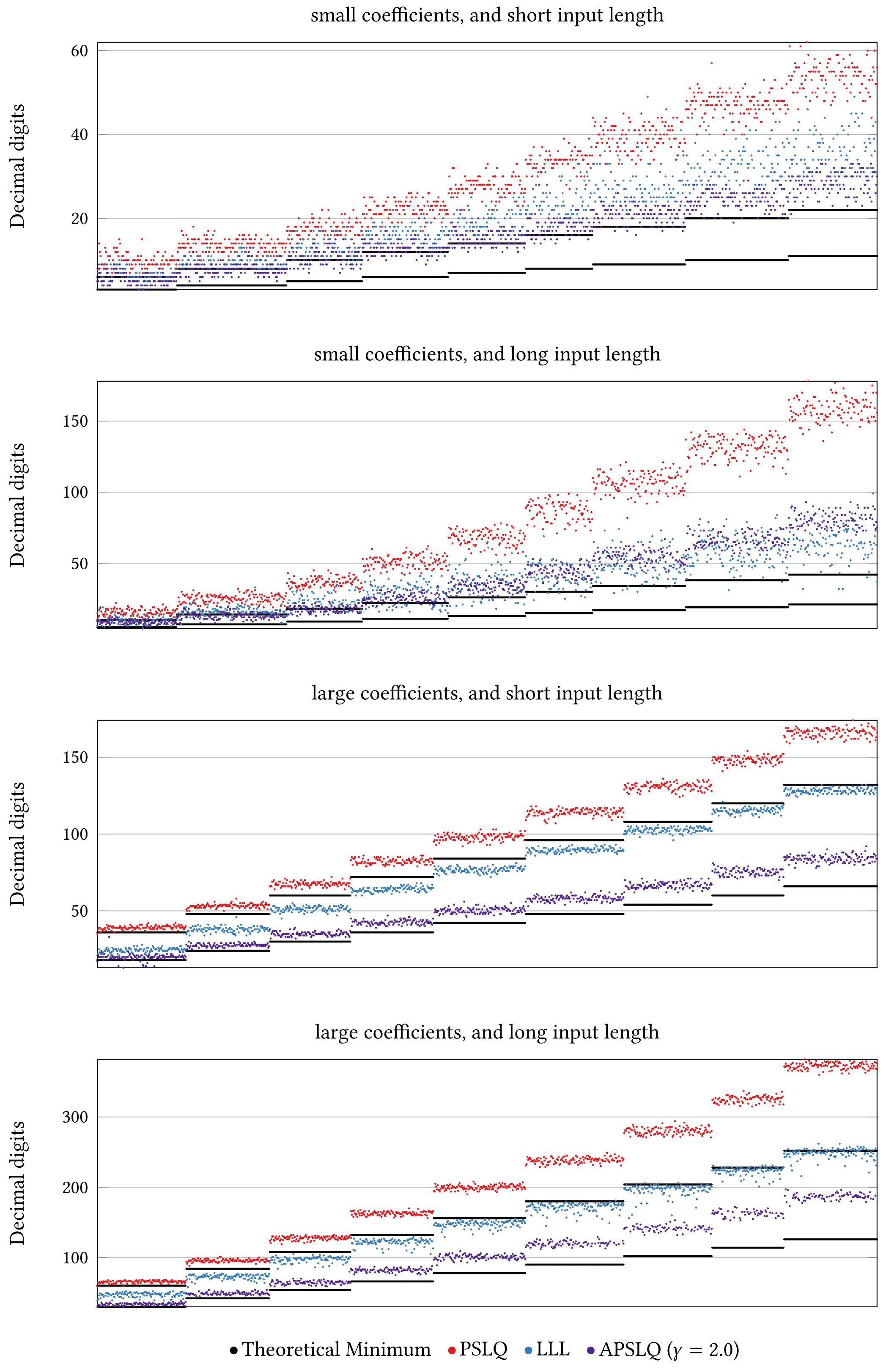


Figure B.33: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{R}}$).

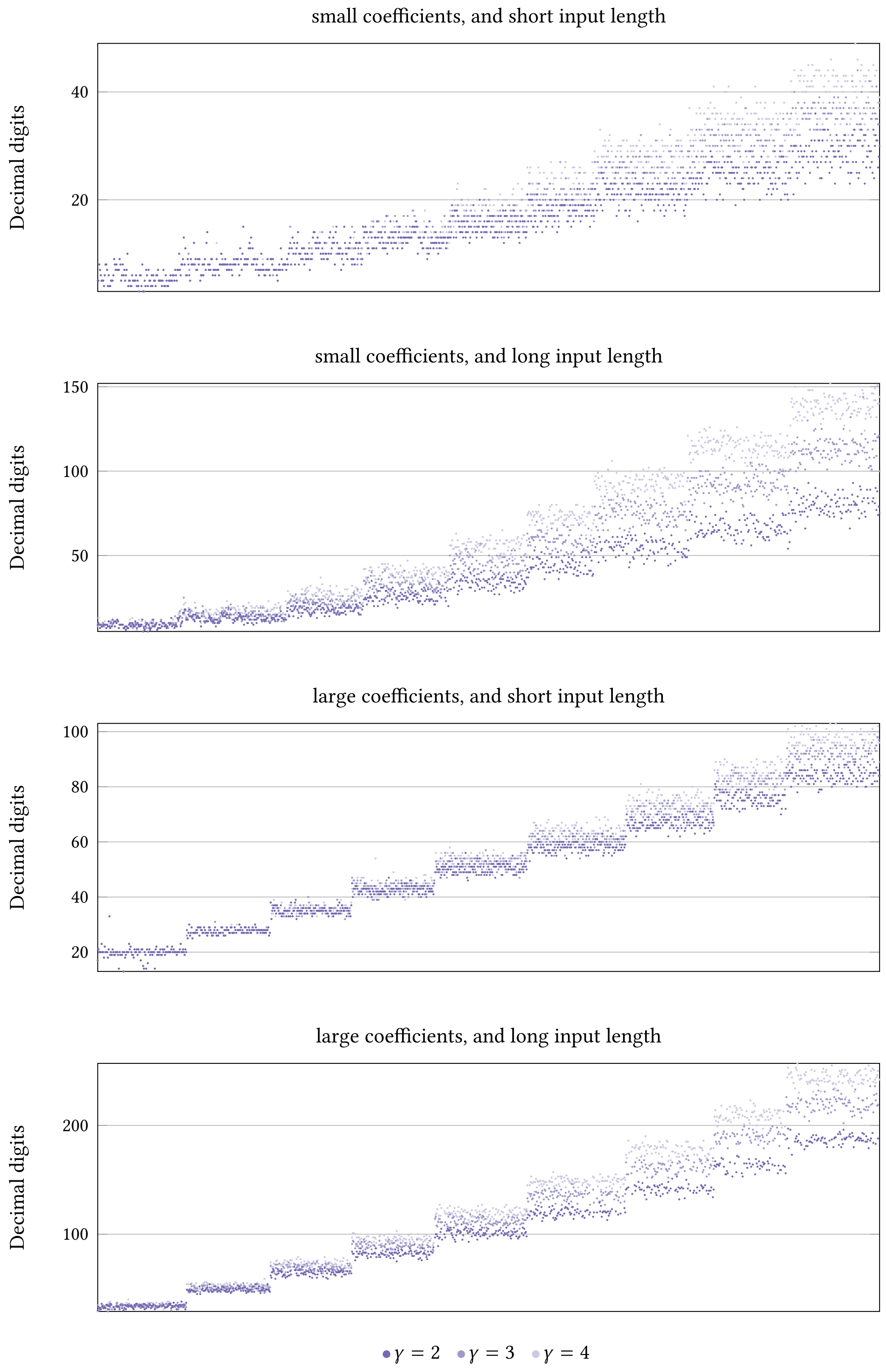


Figure B.34: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{R}}$ (APSLQ only).

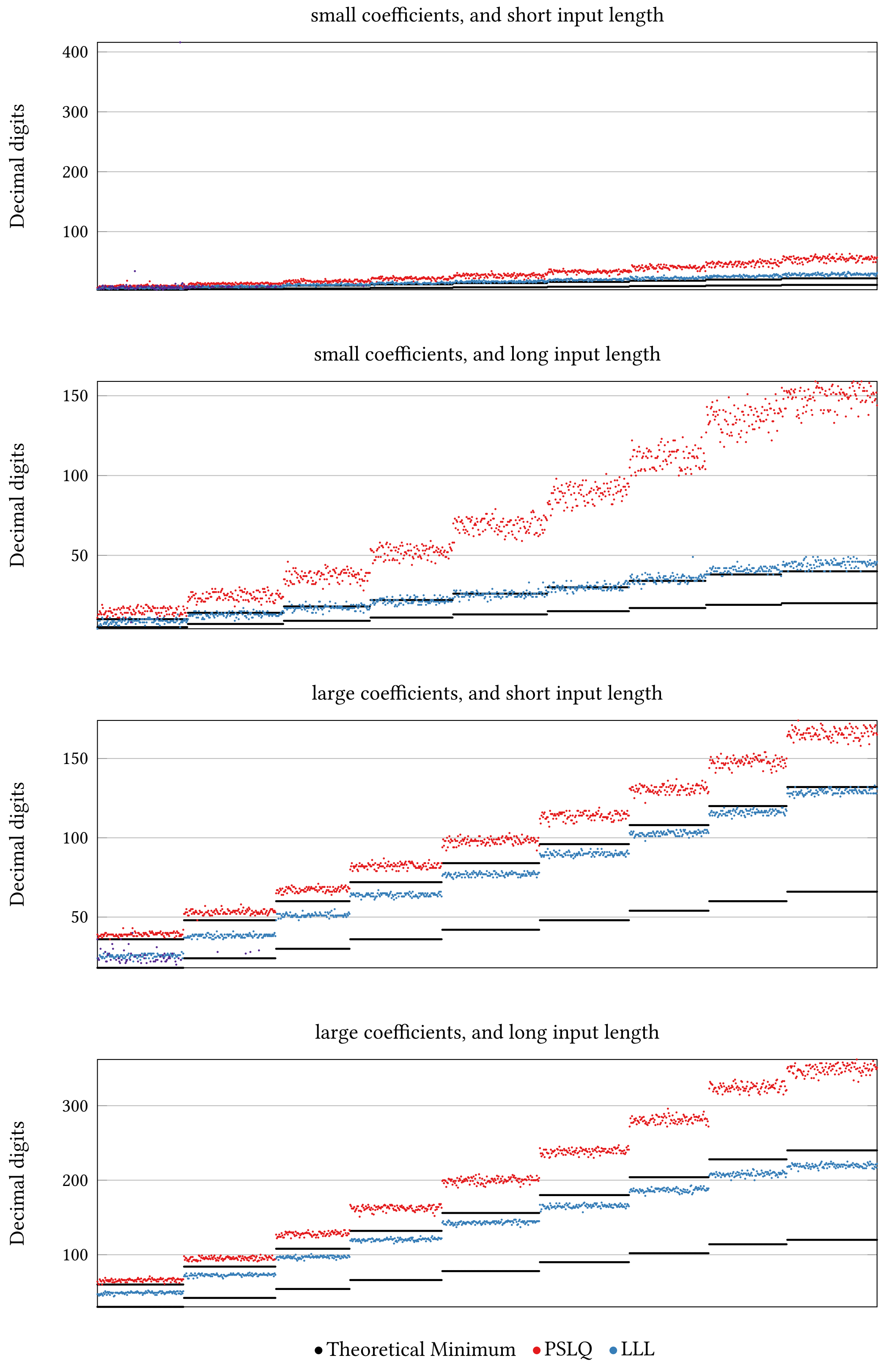


Figure B.35: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{C}}$).

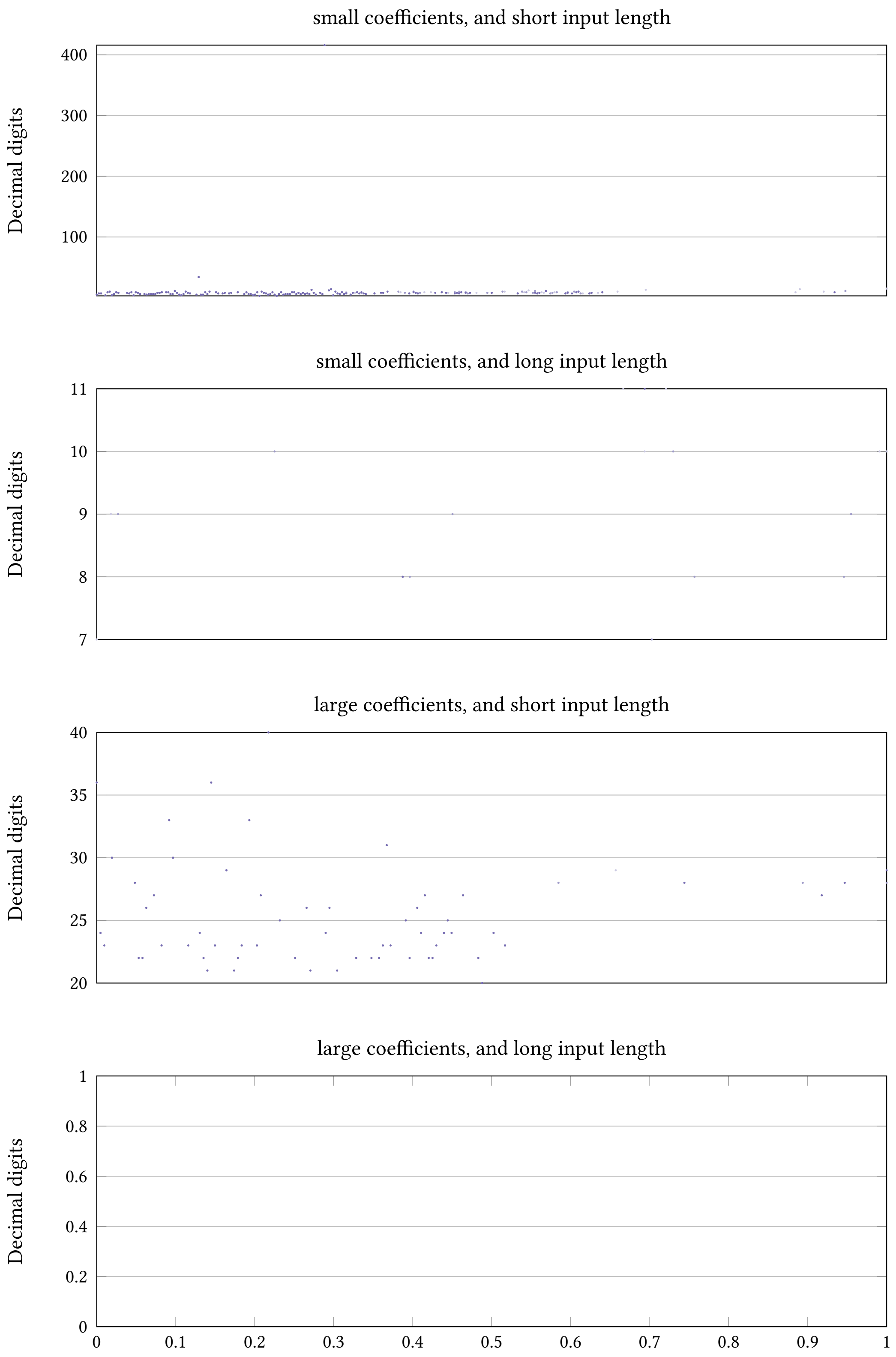


Figure B.36: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{C}}$ (APSLQ only).

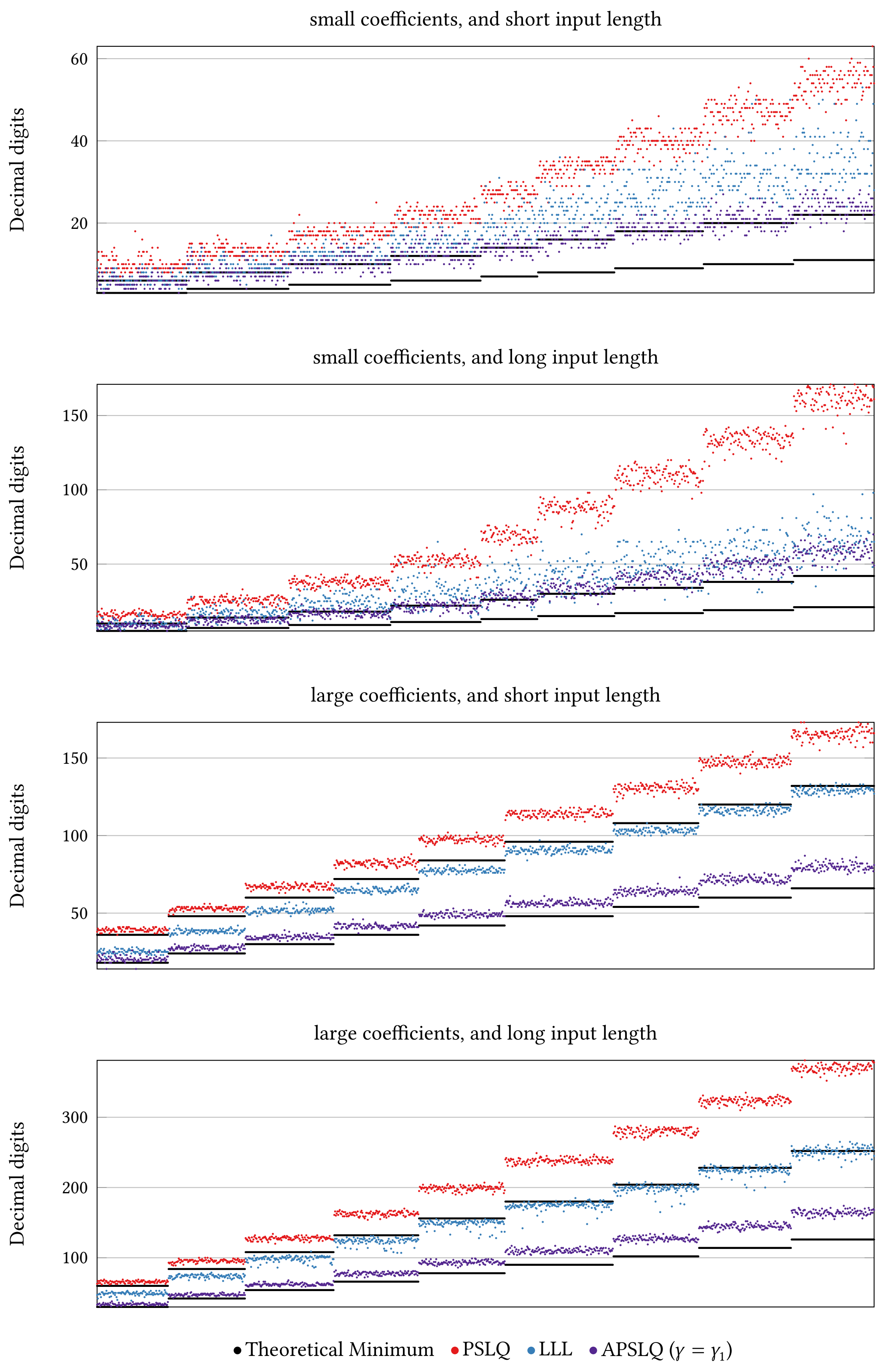


Figure B.37: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{R}}$).

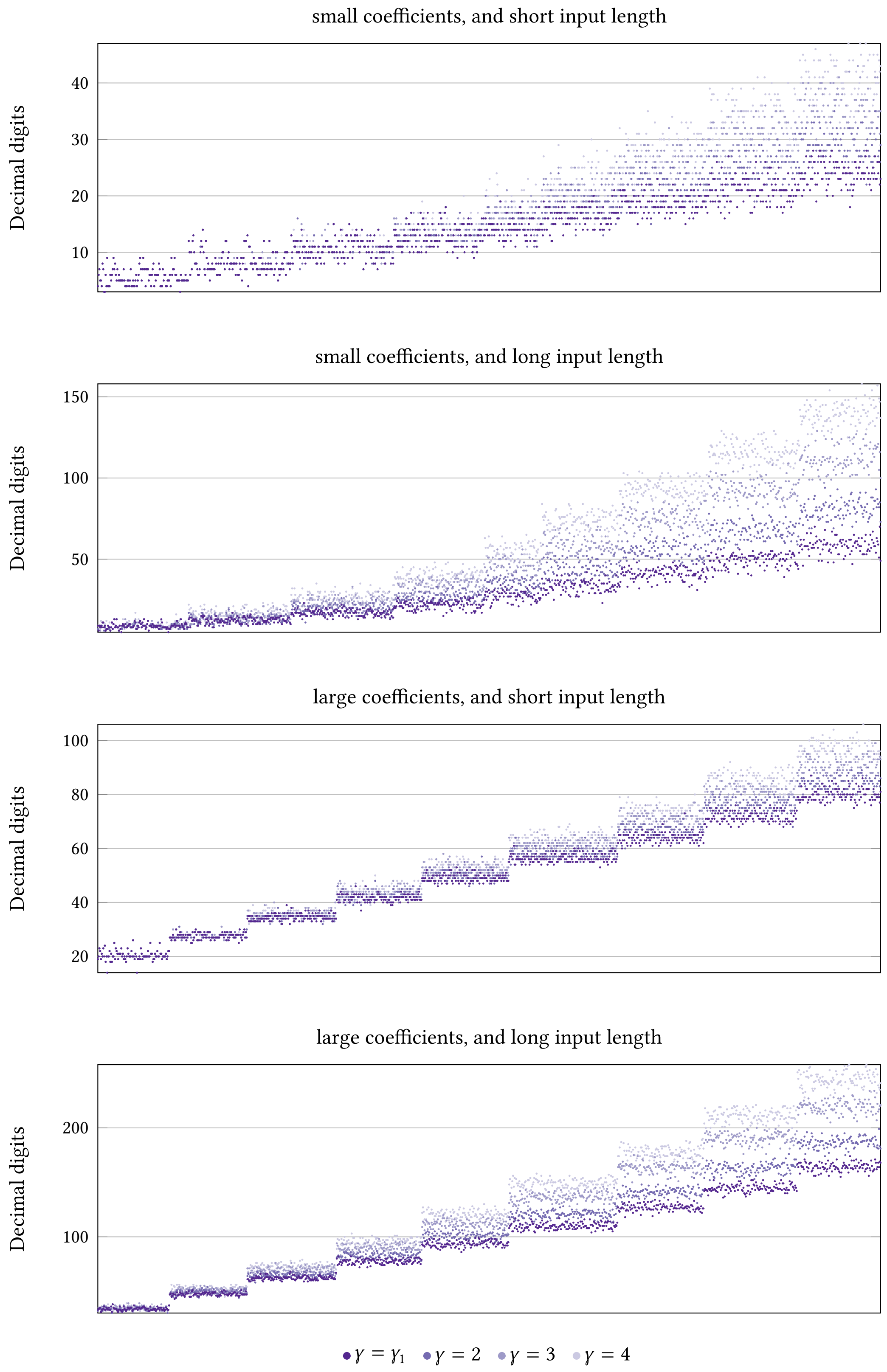


Figure B.38: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{R}}$ (APSLQ only).

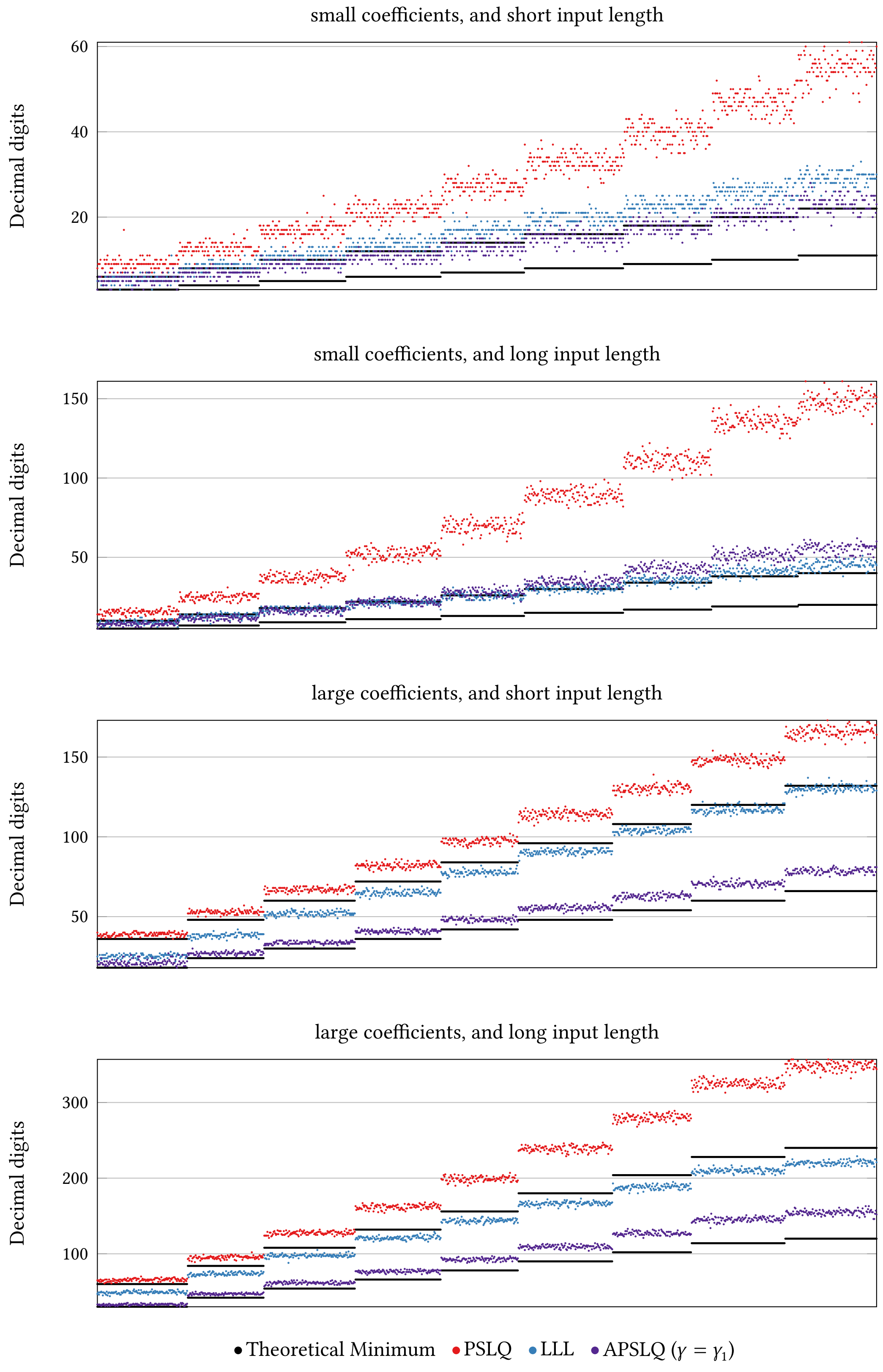


Figure B.39: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{C}}$).

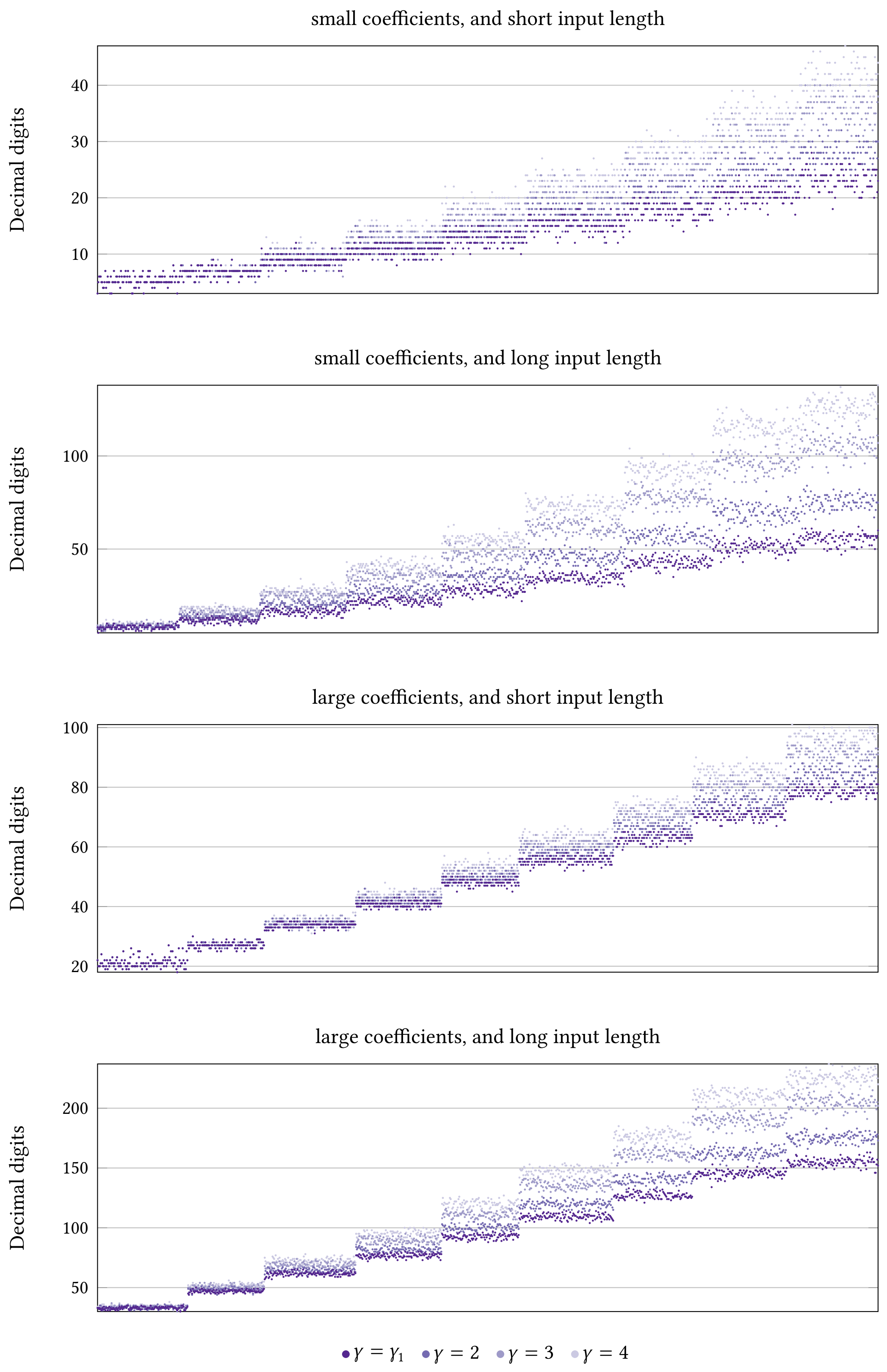


Figure B.40: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{C}}$ (APSLQ only).

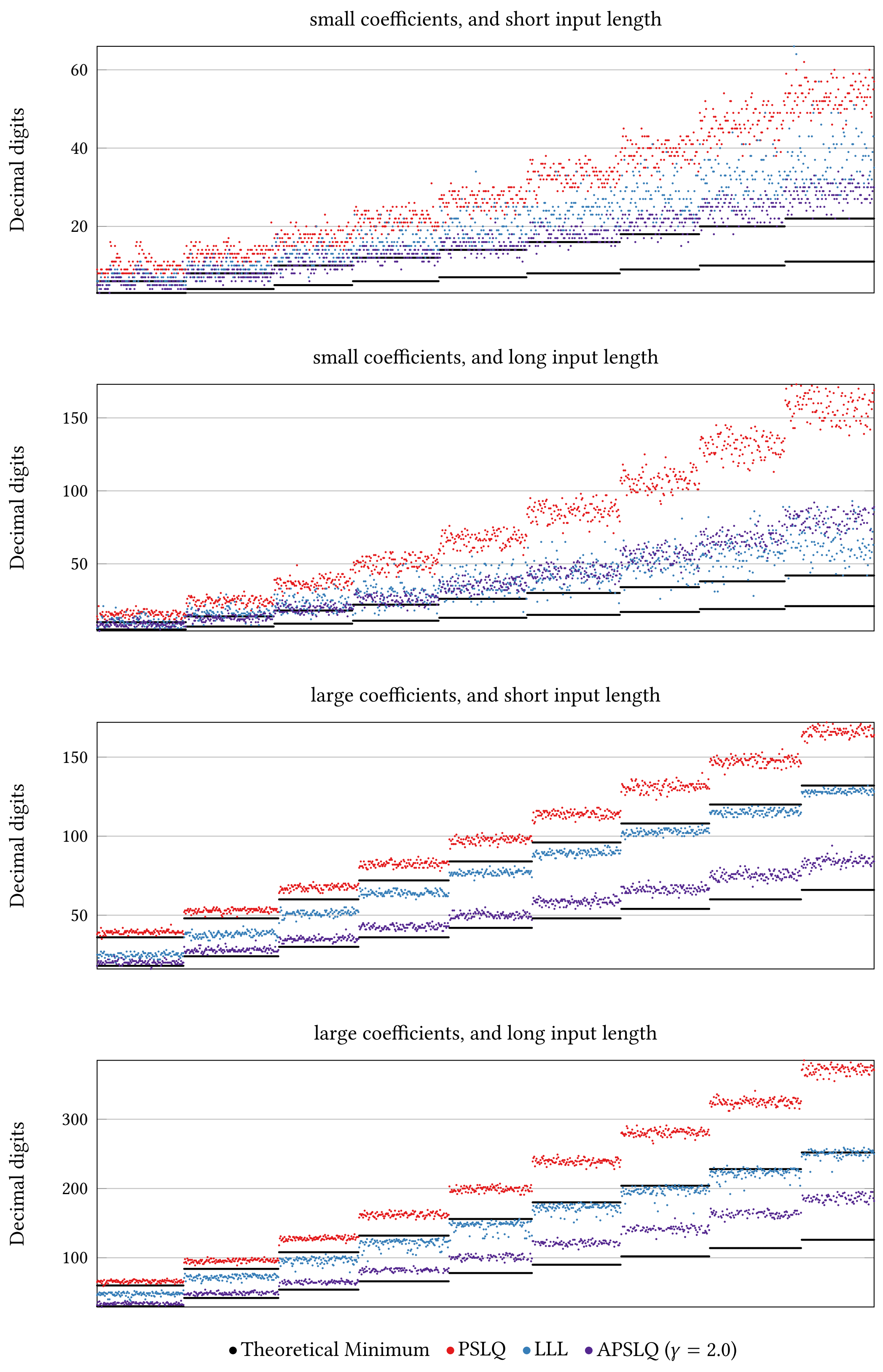


Figure B.41: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{R}}$).

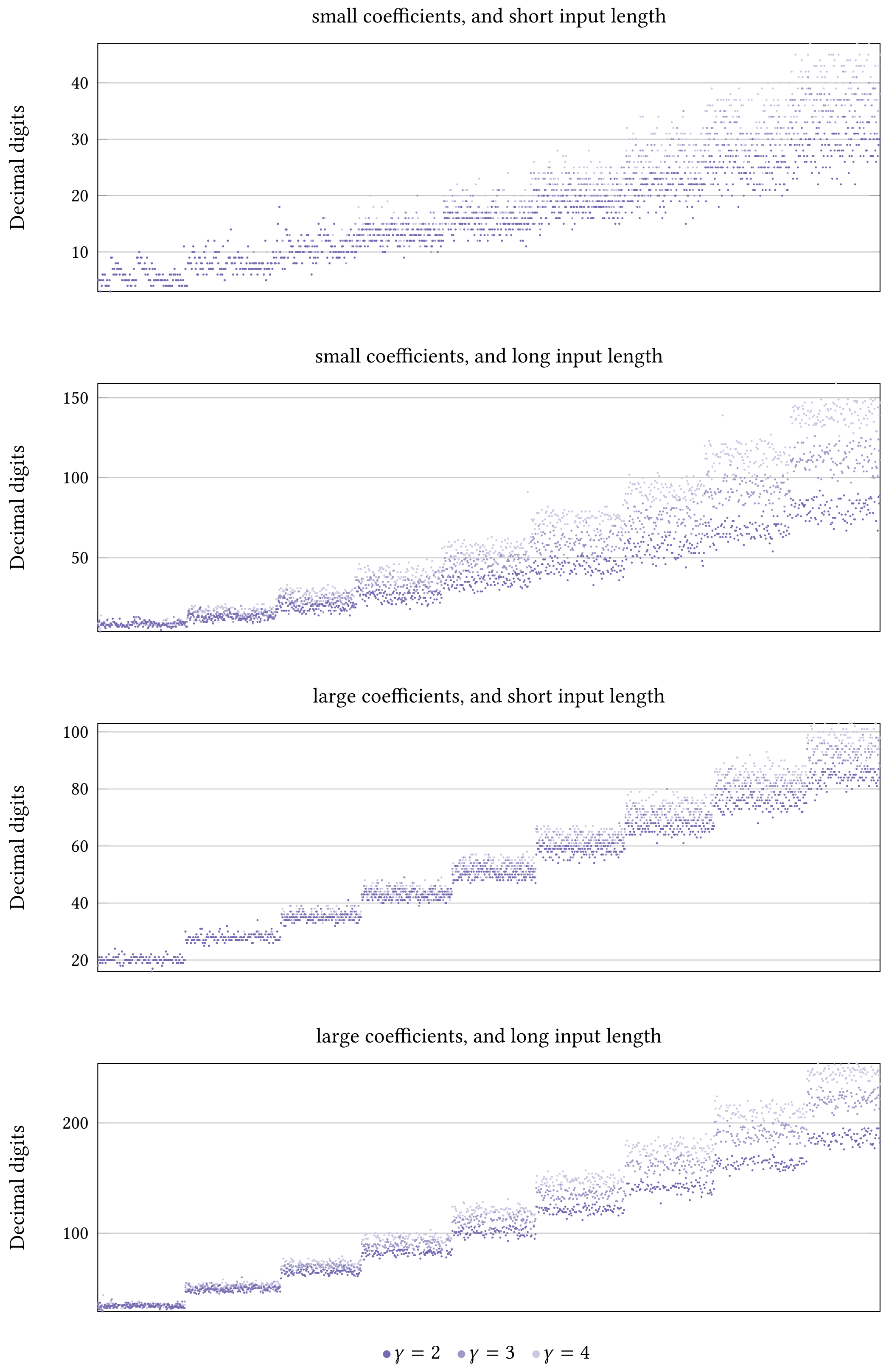


Figure B.42: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{R}}$ (APSLQ only).

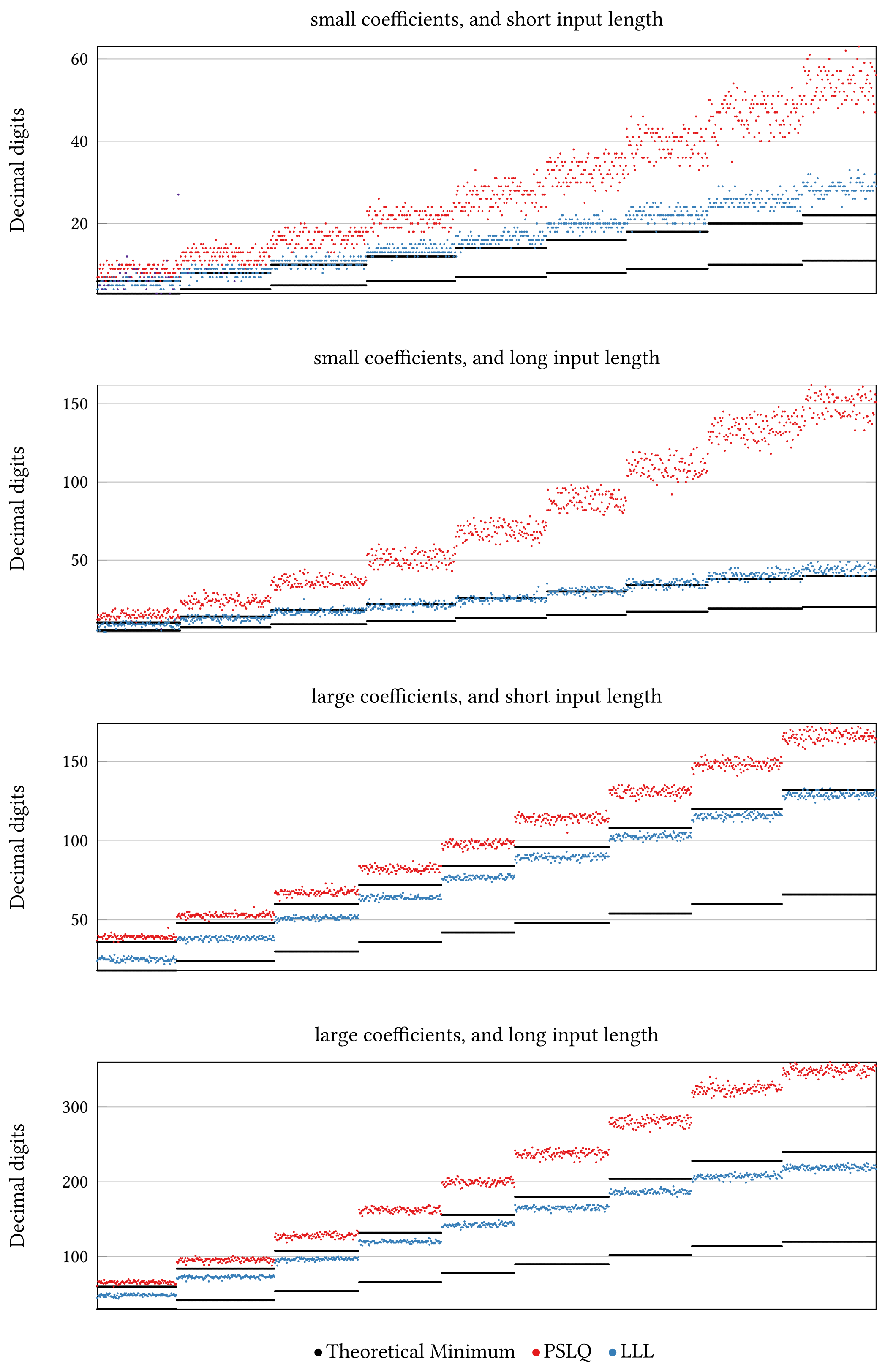


Figure B.43: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{C}}$).

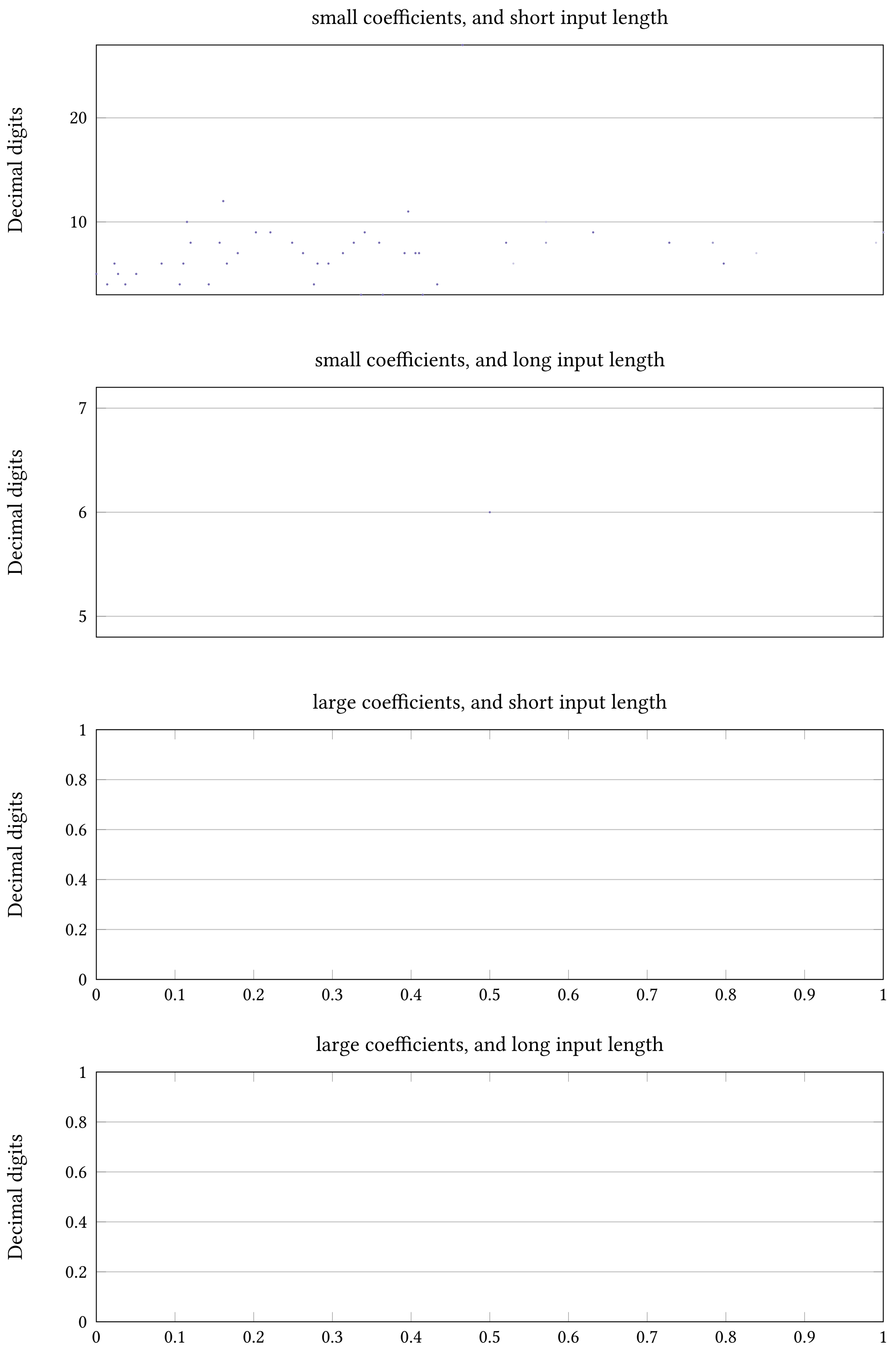


Figure B.44: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{C}}$ (APSLQ only).

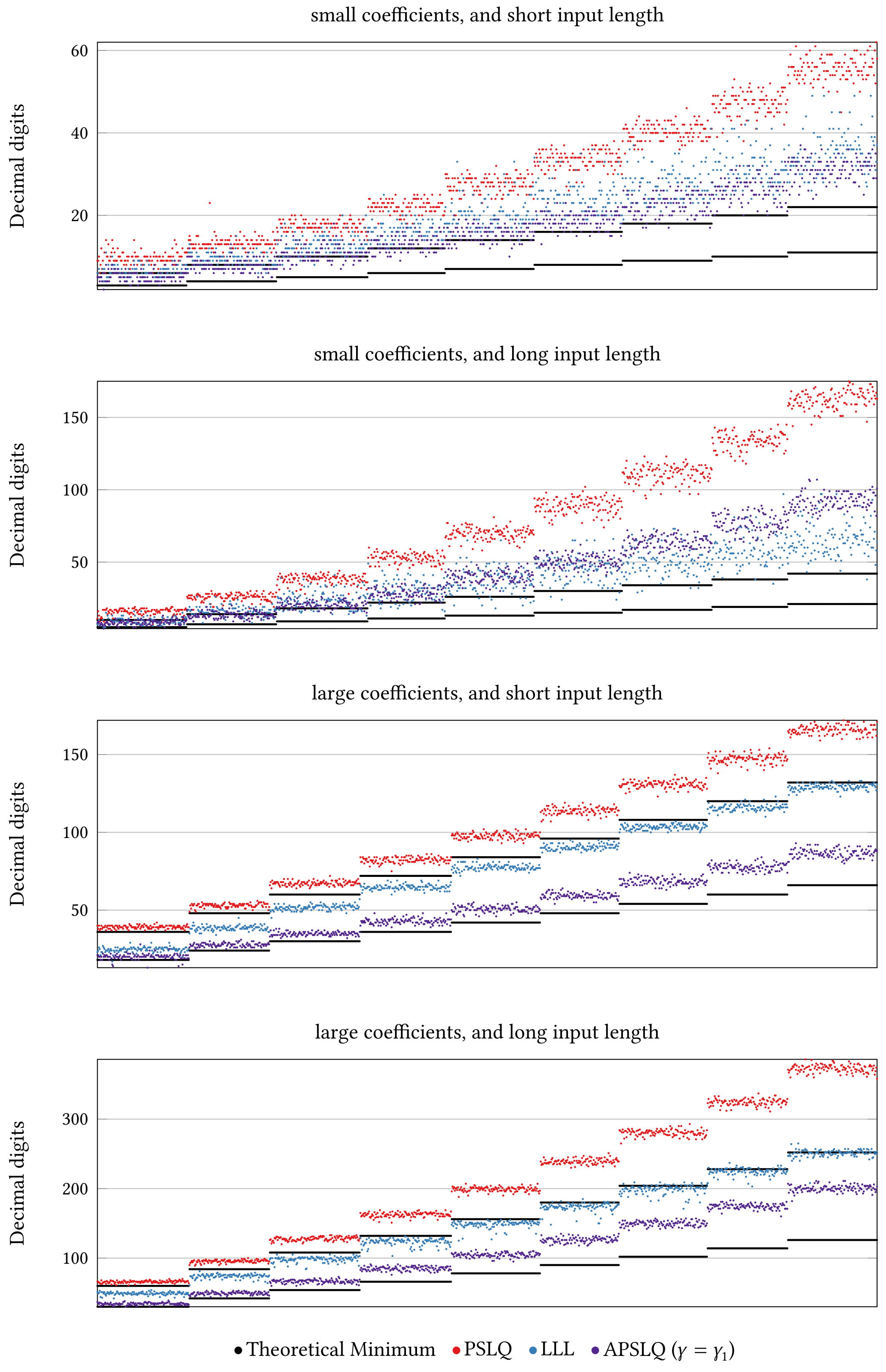


Figure B.45: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{R}}$).

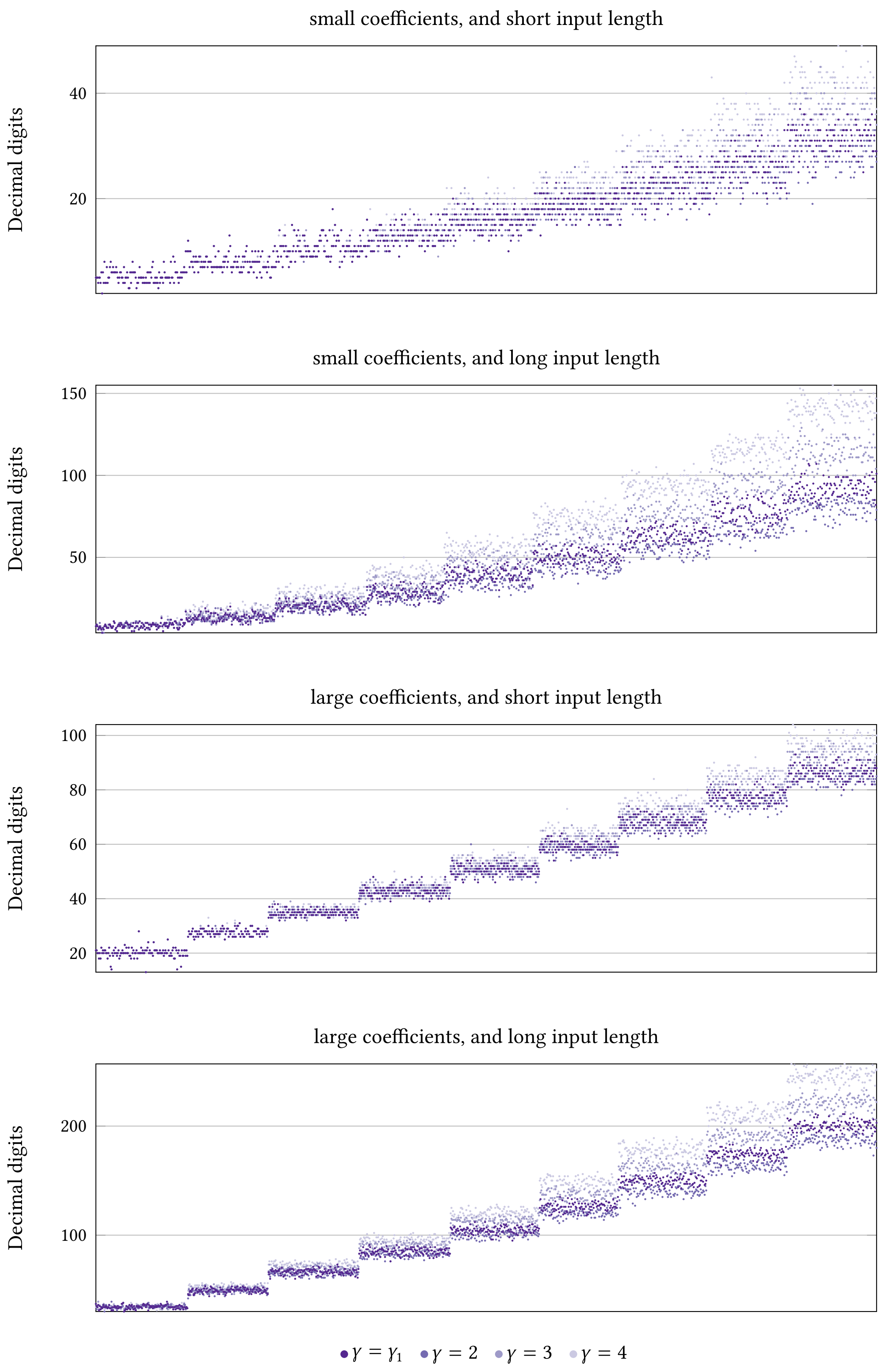


Figure B.46: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{R}}$ (APSLQ only).

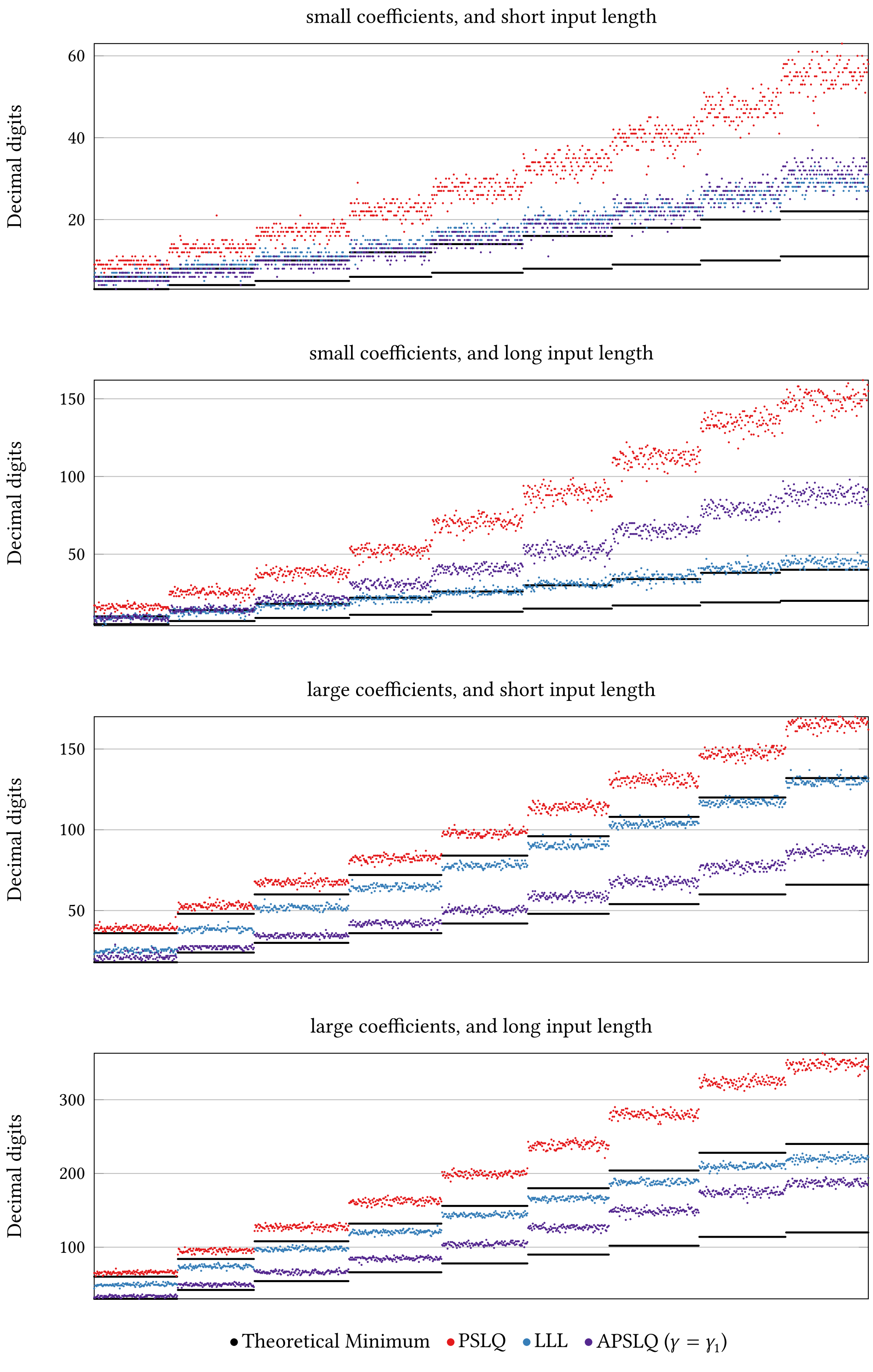


Figure B.47: Precision required for ($\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{C}}$).

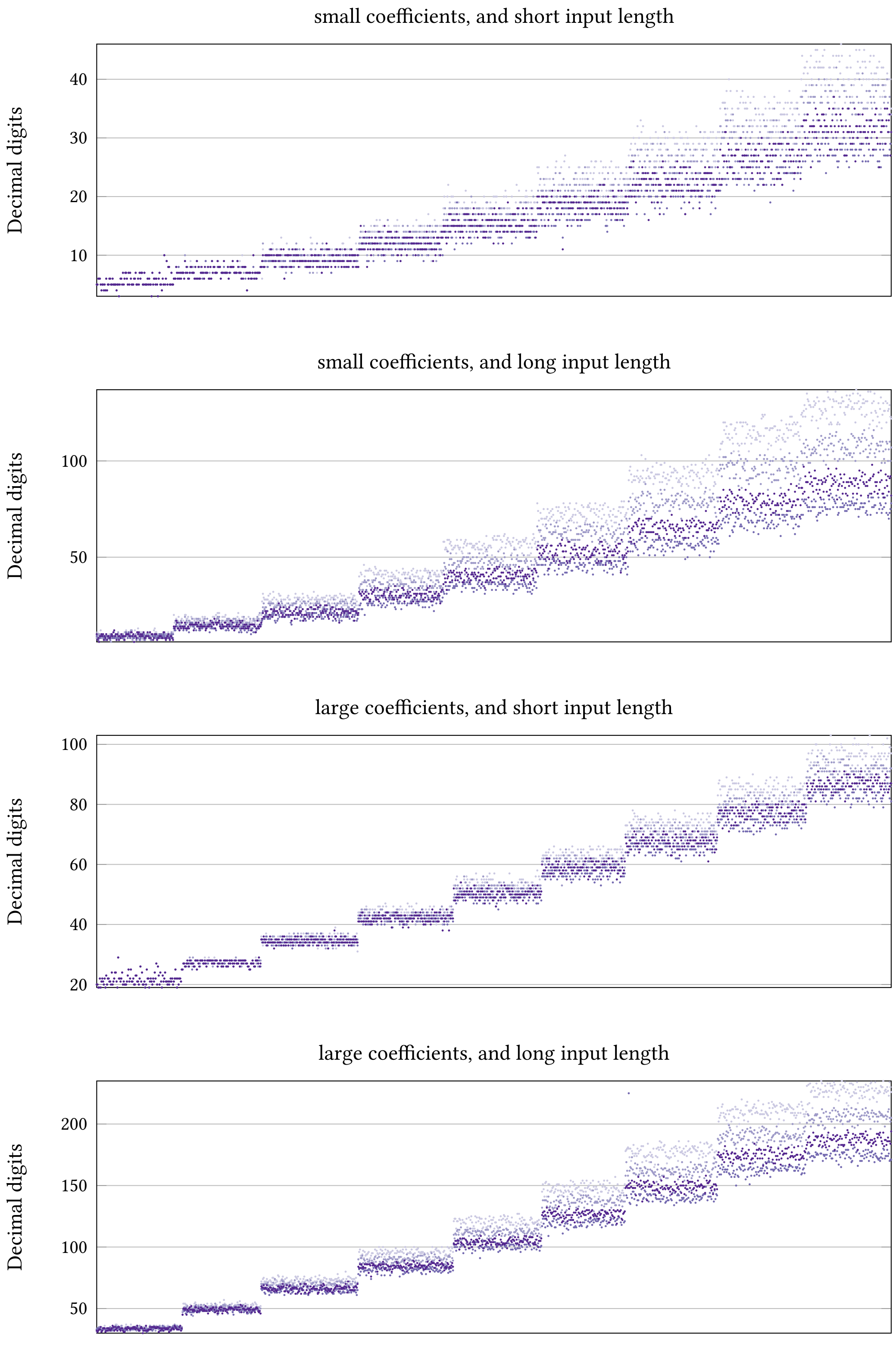


Figure B.48: Precision required for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{C}}$ (APSLQ only).

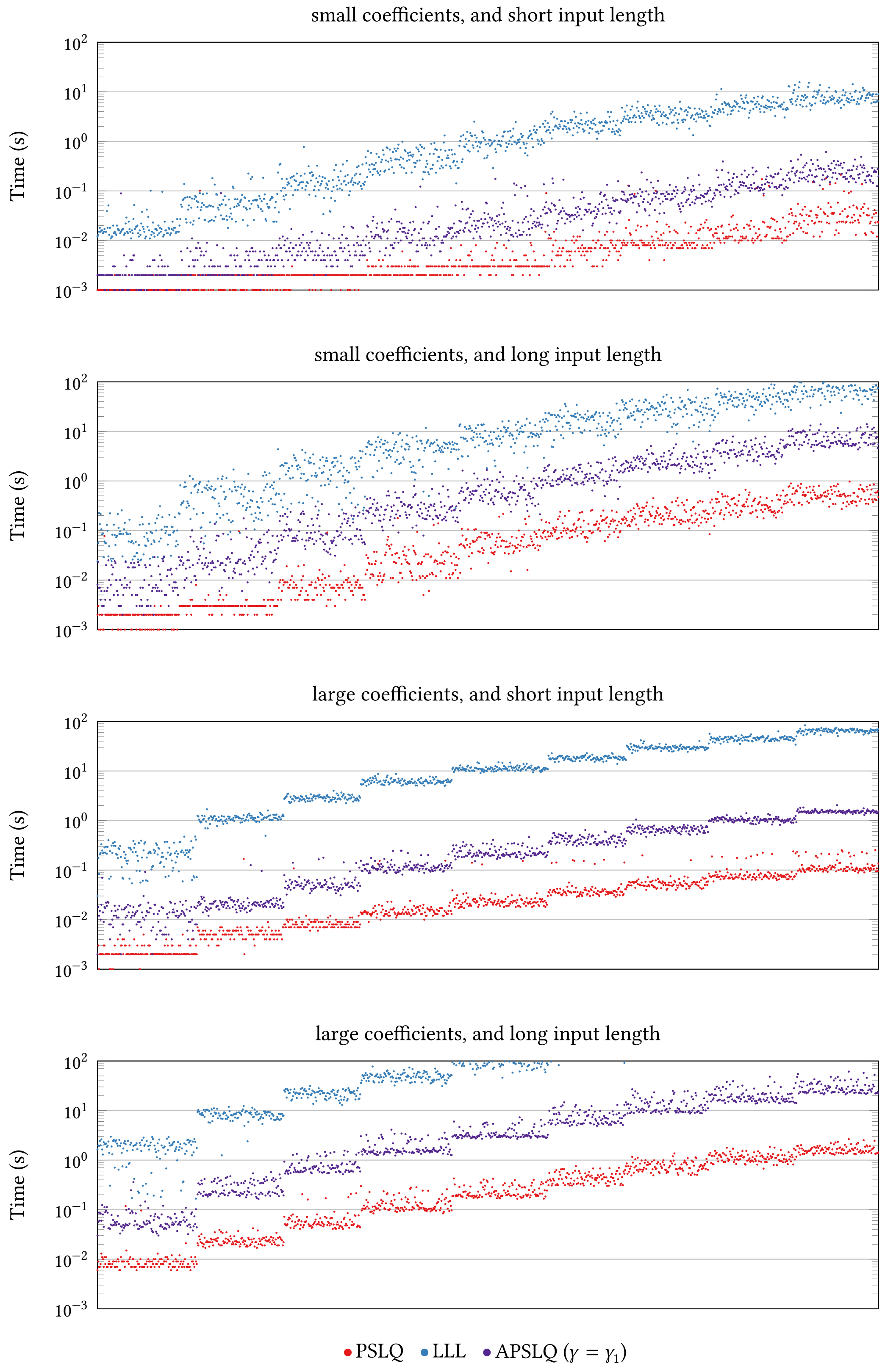


Figure B.49: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$.

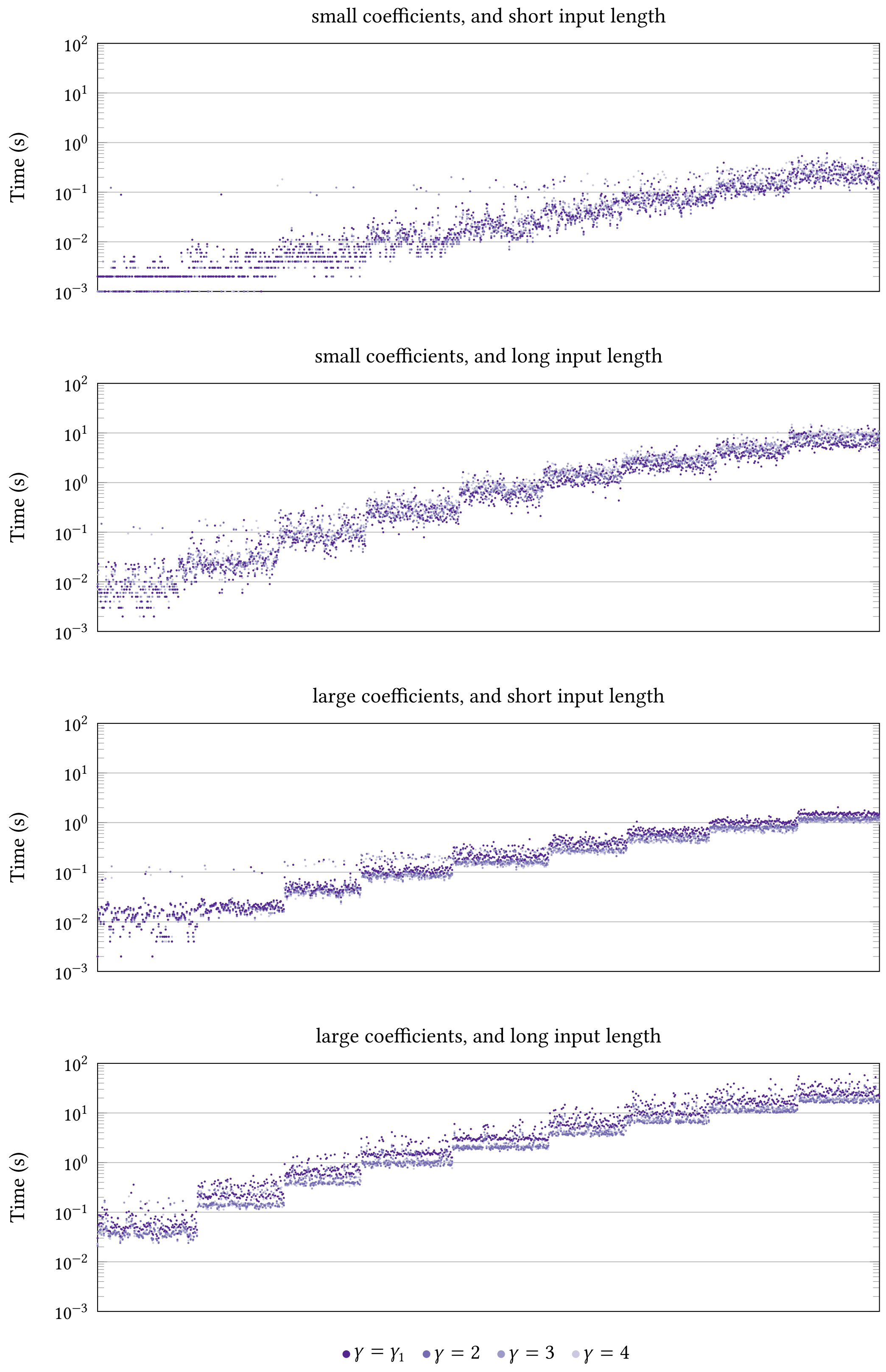


Figure B.50: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{R}}$ (APSLQ only).

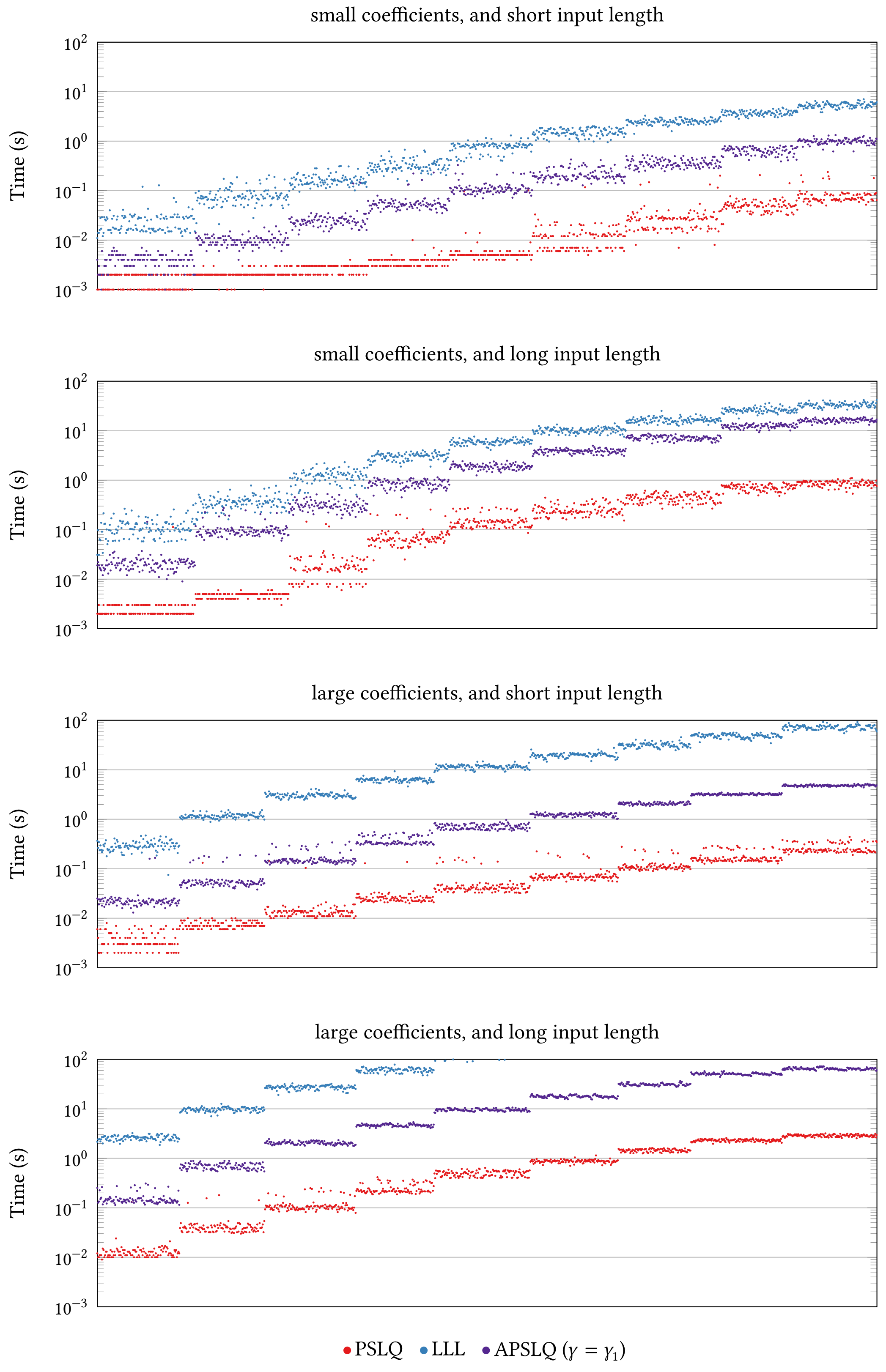


Figure B.51: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$.

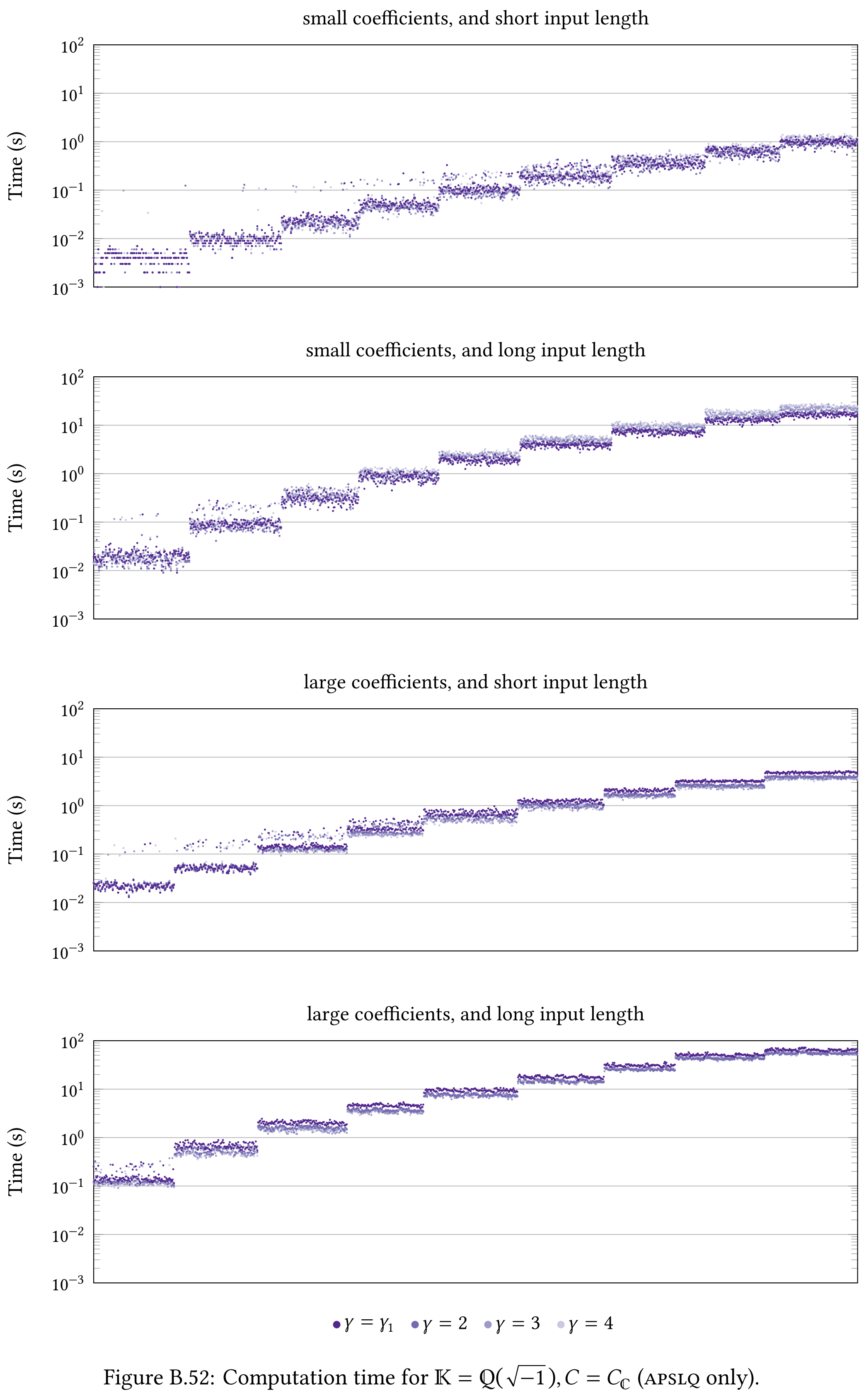


Figure B.52: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-1}), C = C_{\mathbb{C}}$ (APSLQ only).

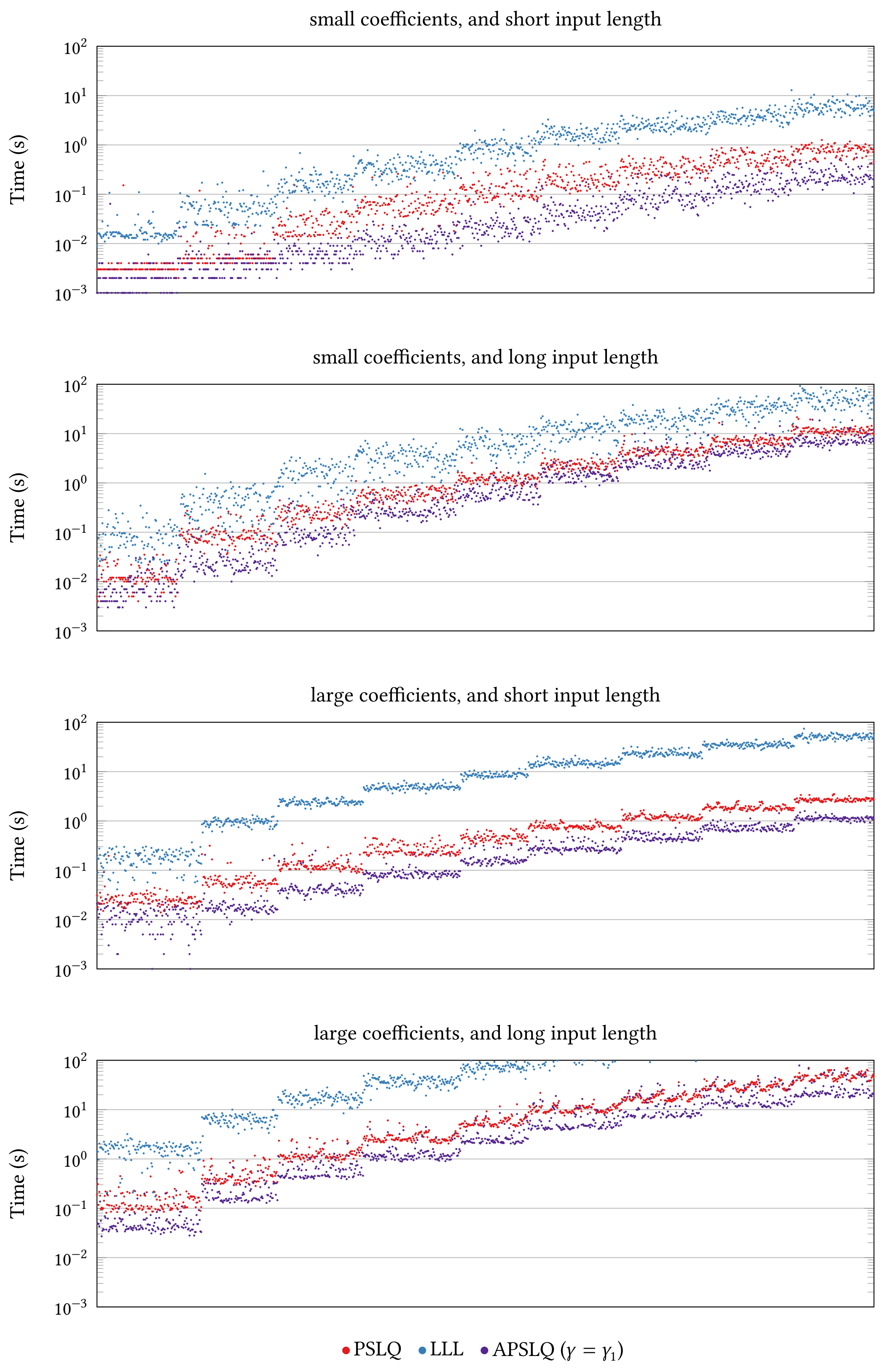


Figure B.53: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$.

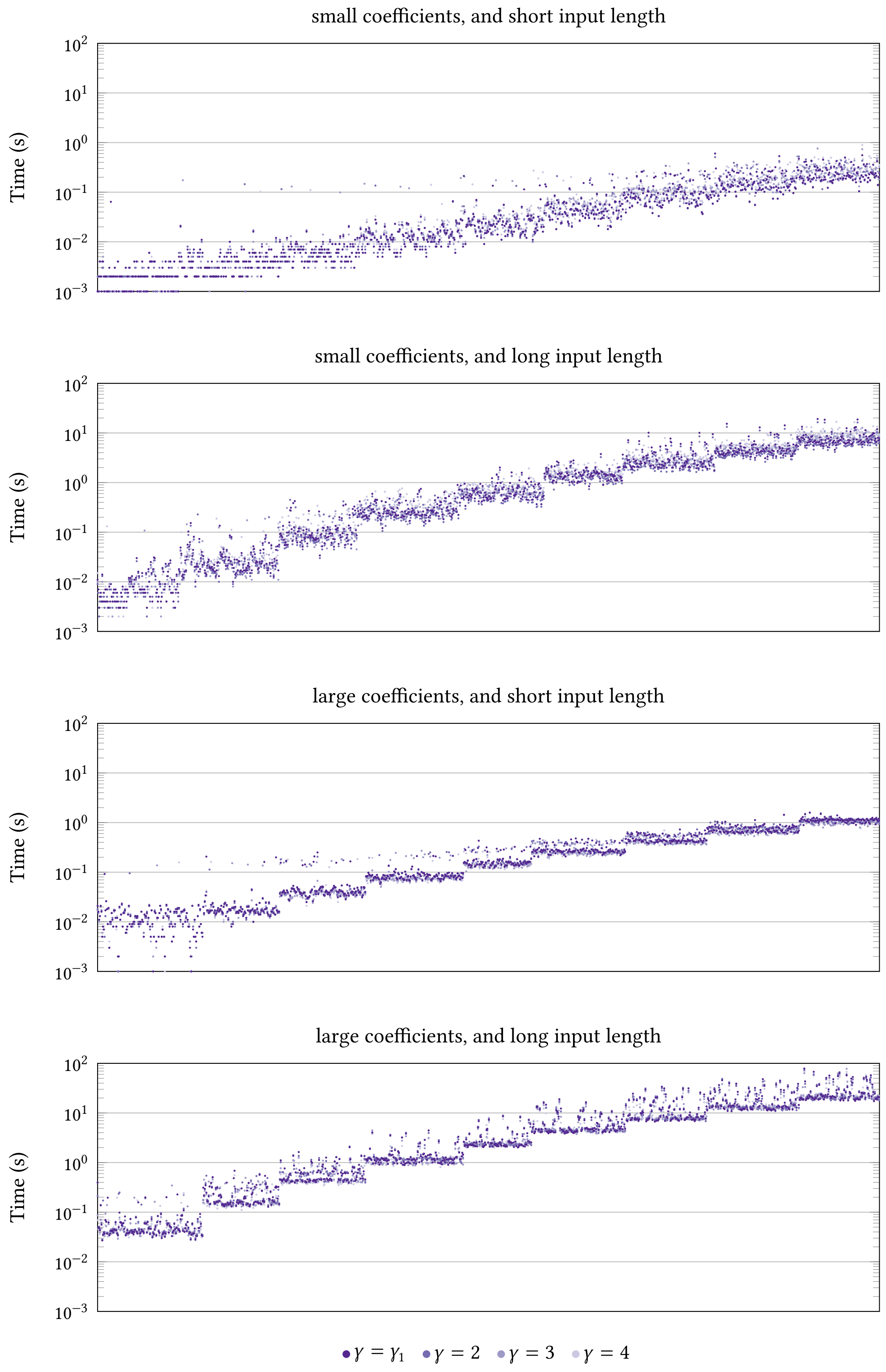


Figure B.54: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{R}}$ (APSLQ only).

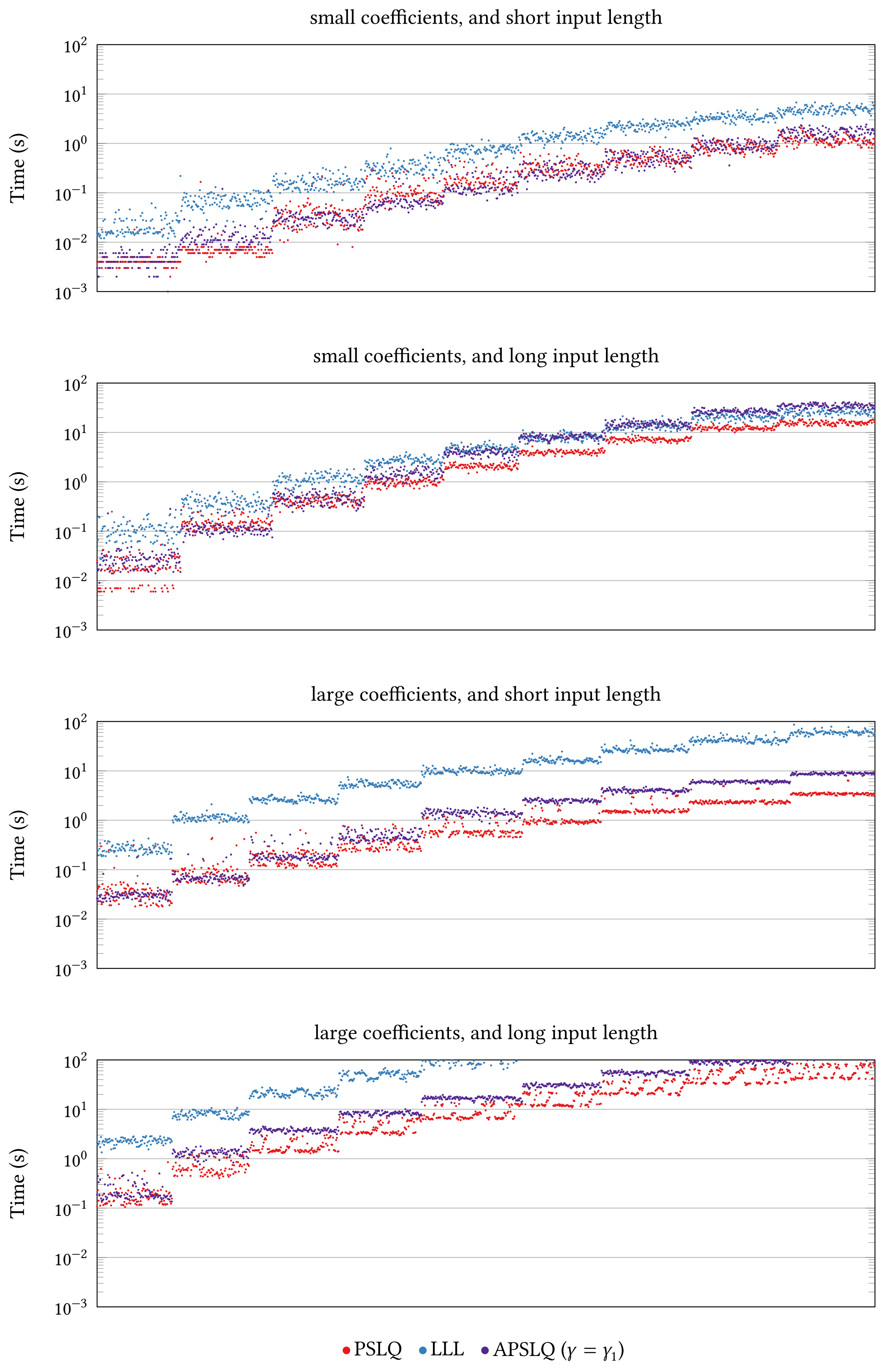


Figure B.55: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$.

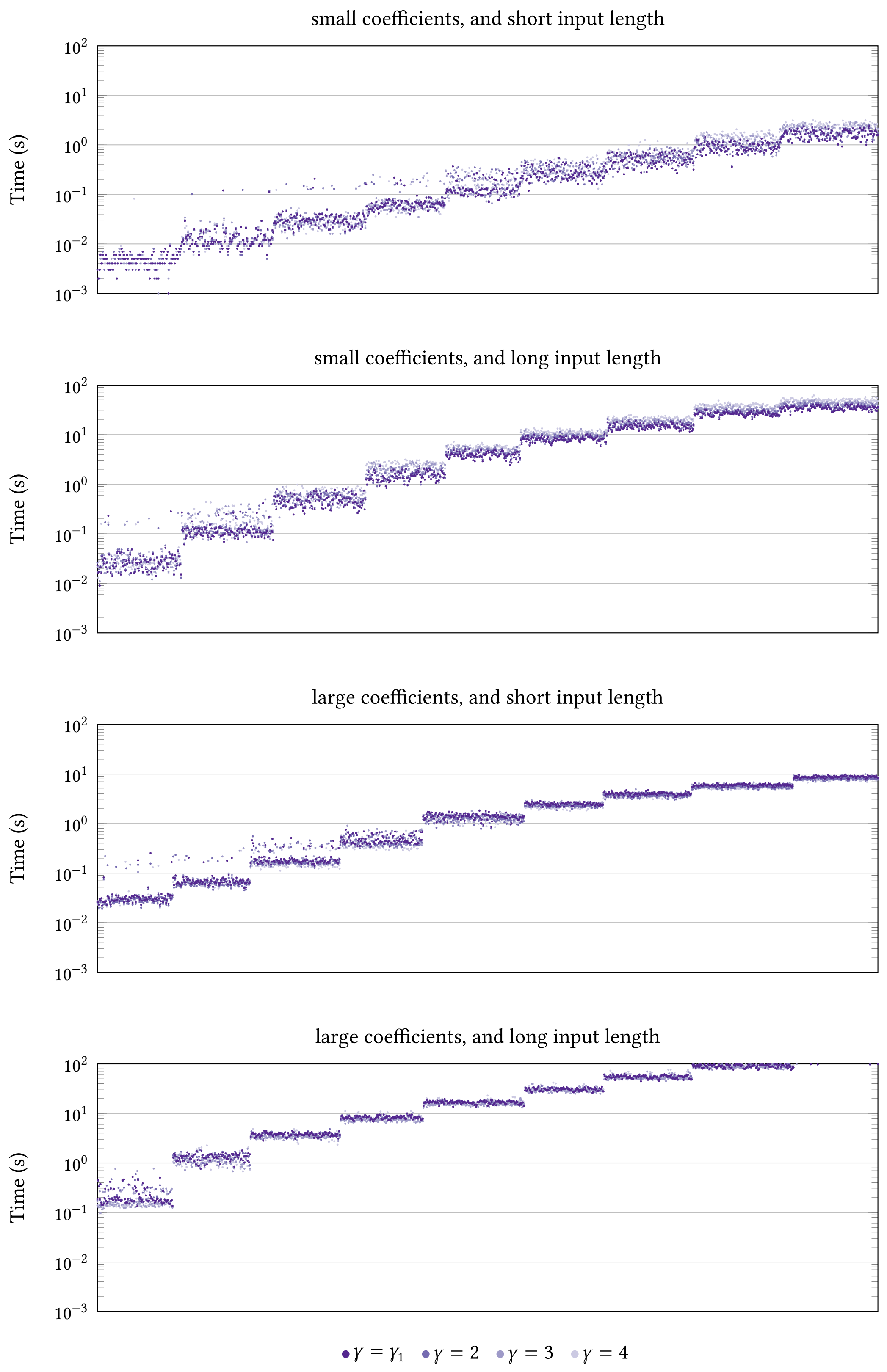


Figure B.56: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-2}), C = C_{\mathbb{C}}$ (APSLQ only).

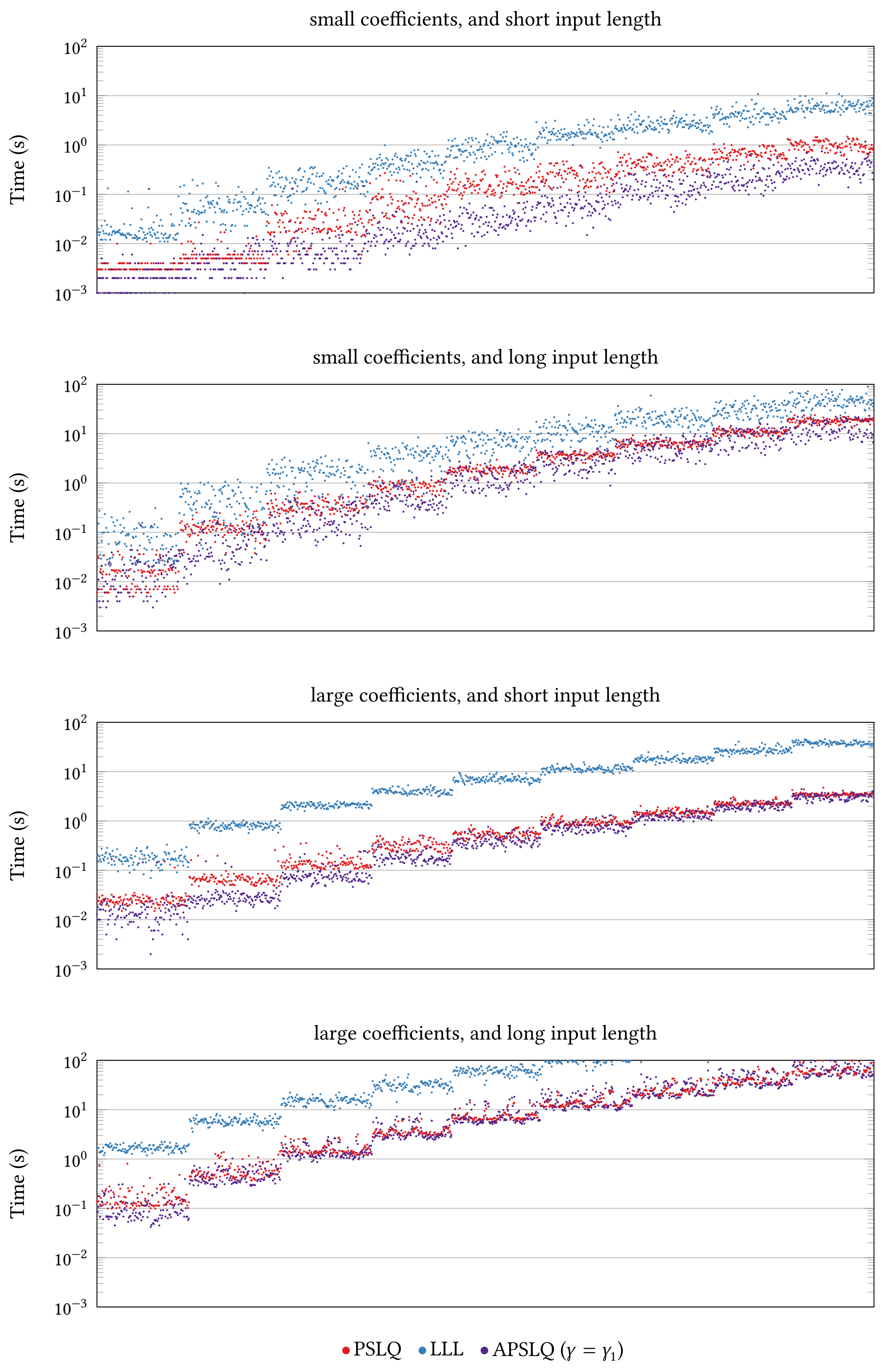


Figure B.57: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{R}}$.

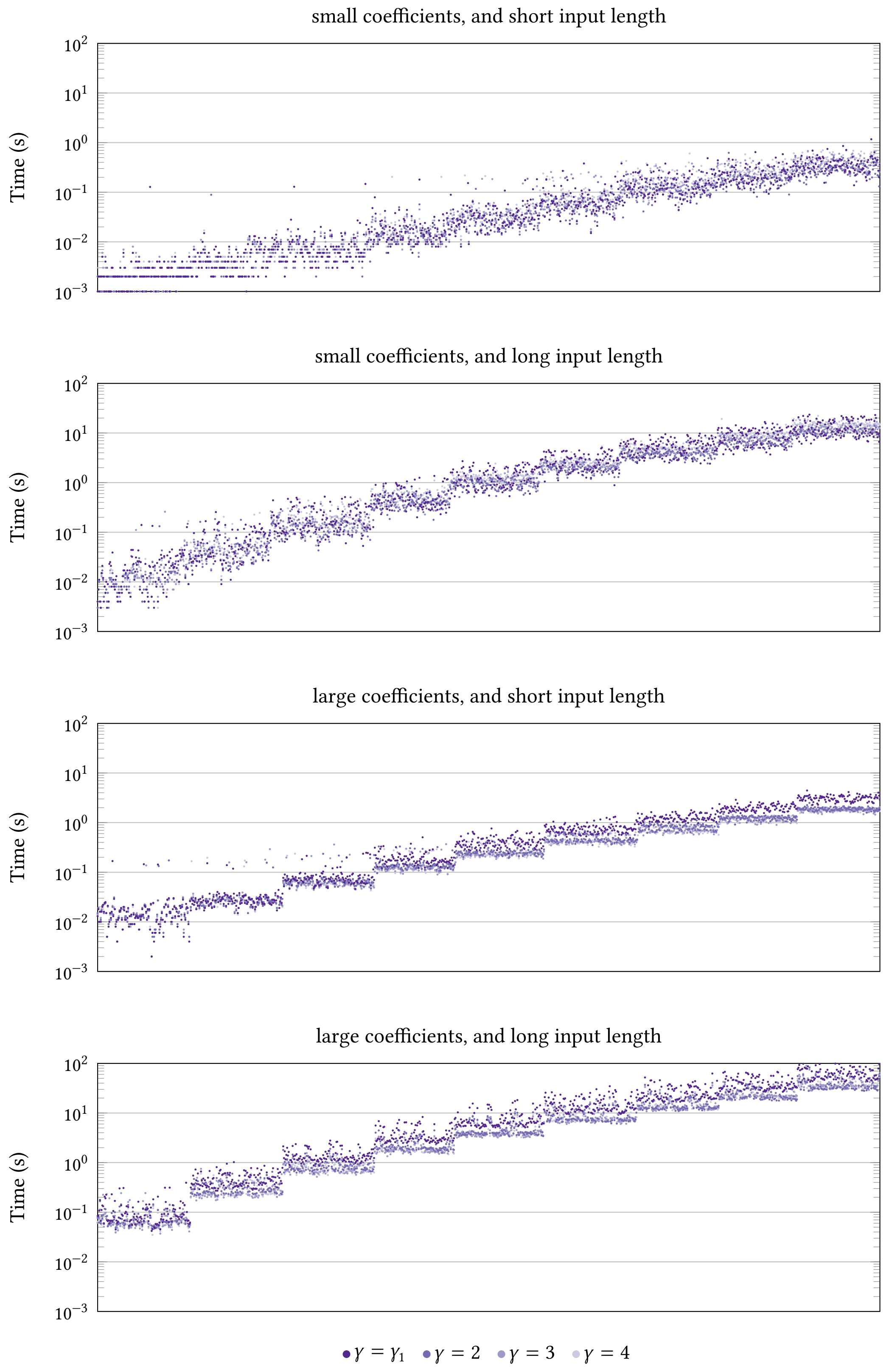


Figure B.58: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{R}}$ (APSLQ only).

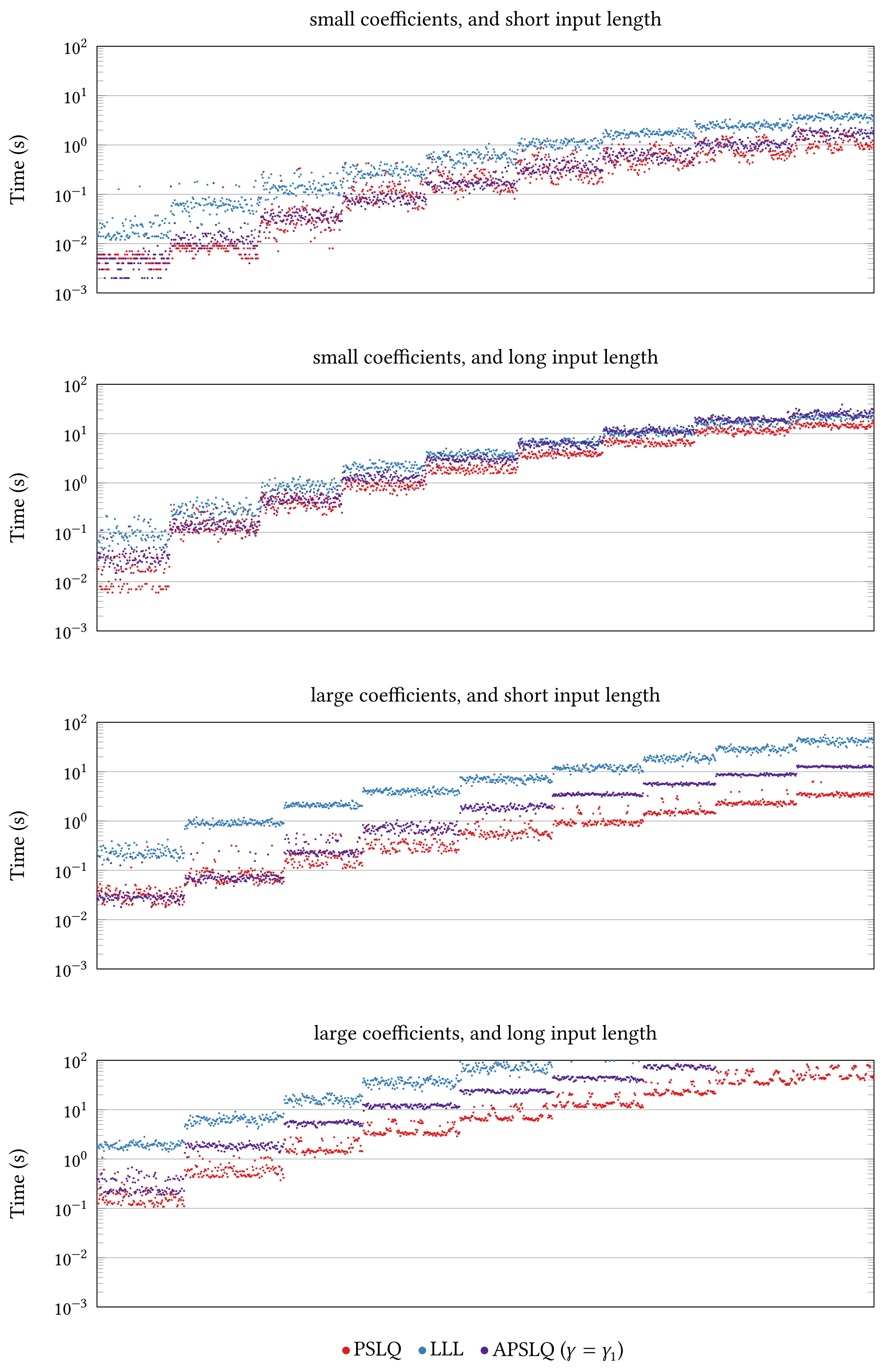


Figure B.59: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{C}}$.

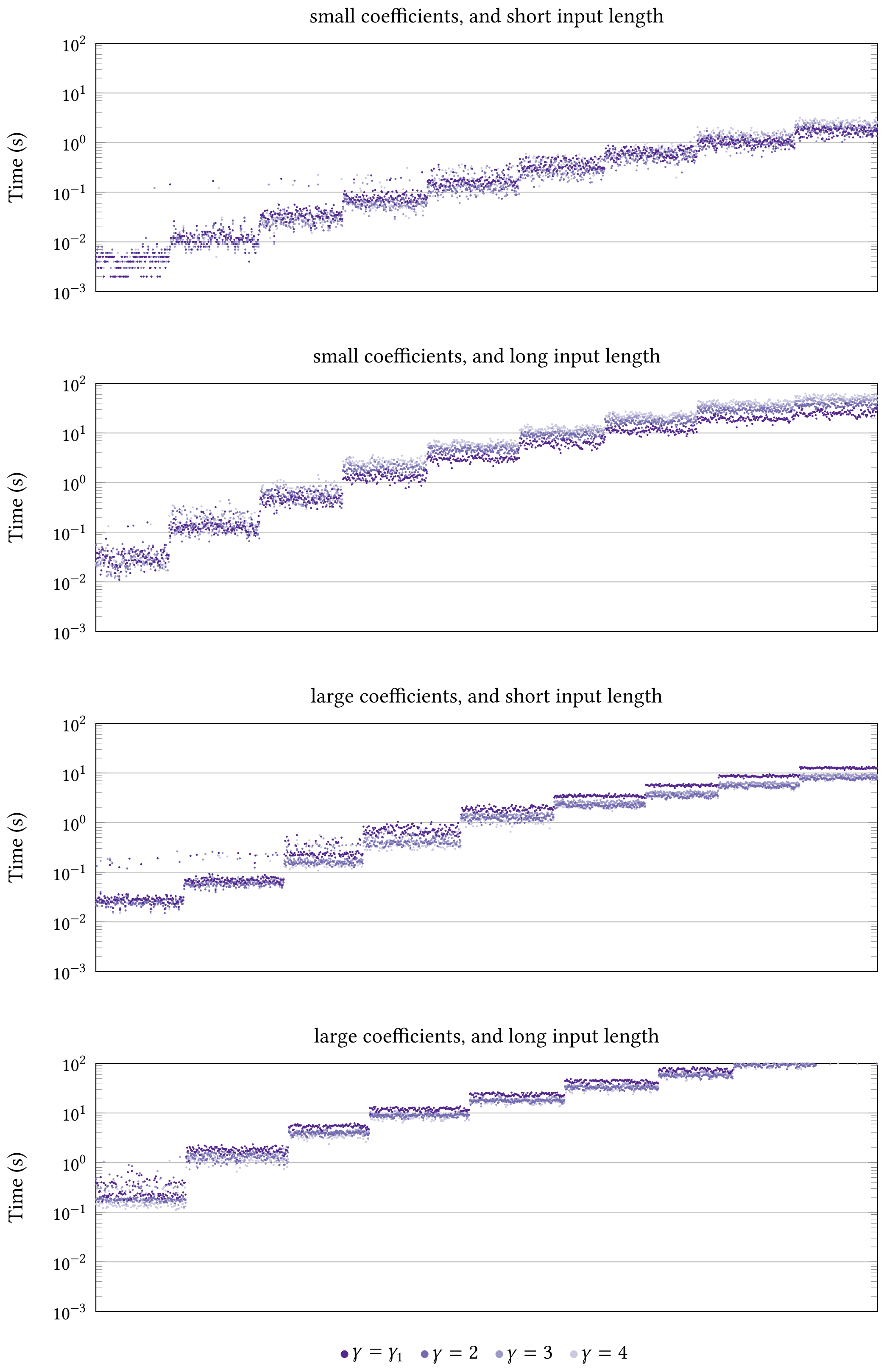


Figure B.60: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-3}), C = C_{\mathbb{C}}$ (APSLQ only).

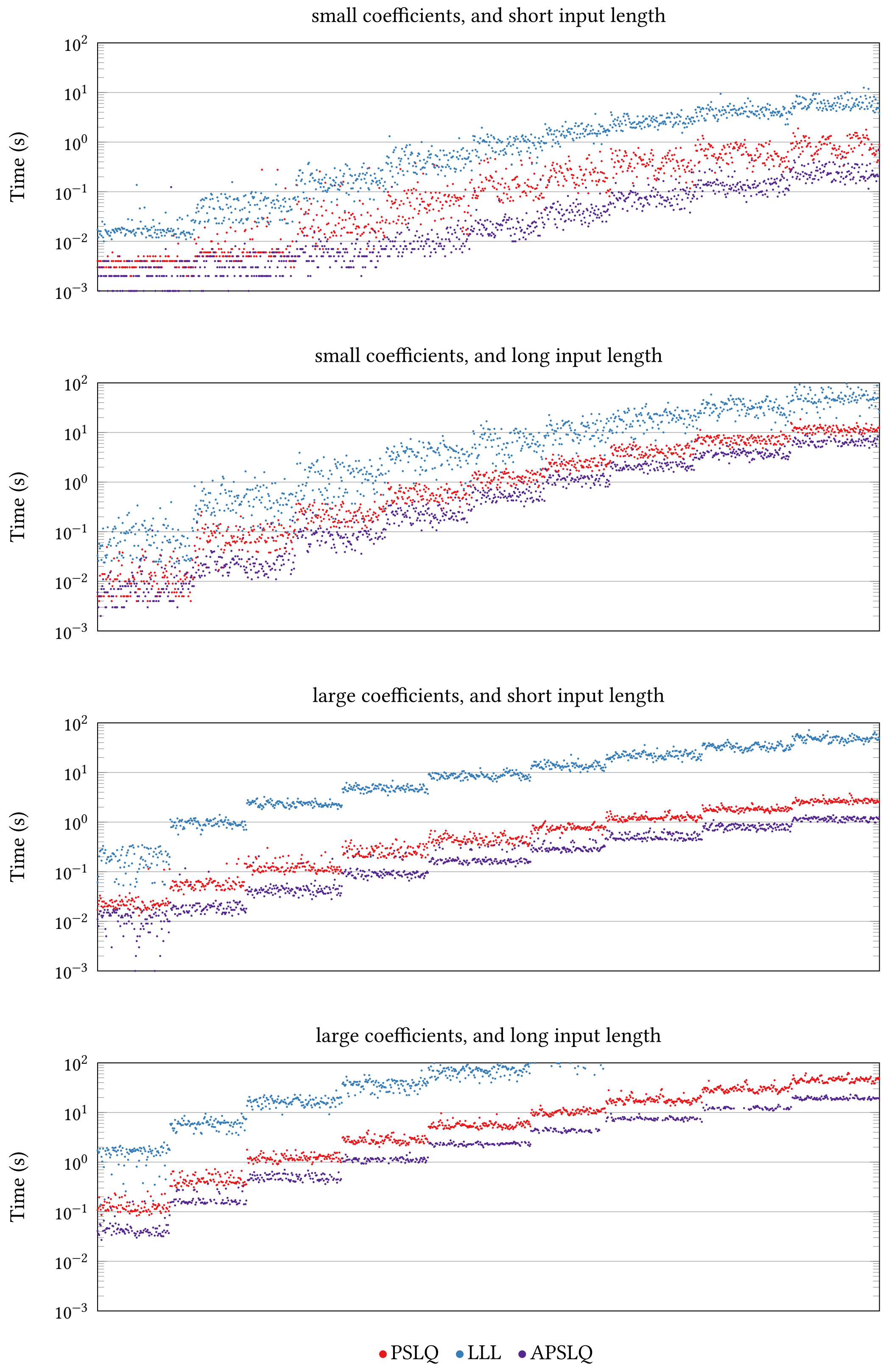


Figure B.61: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{R}}$.

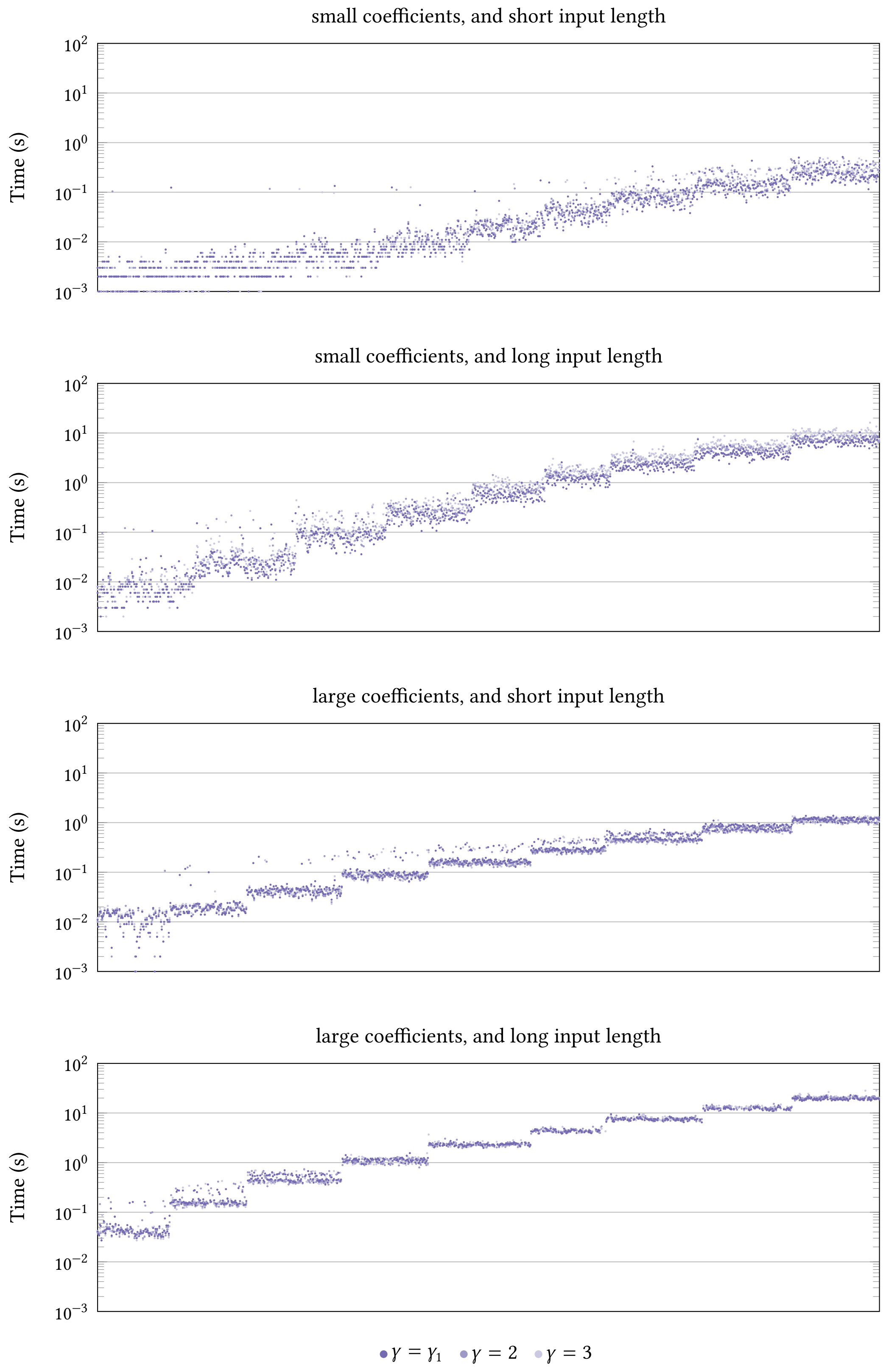


Figure B.62: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{R}}$ (APSLQ only).

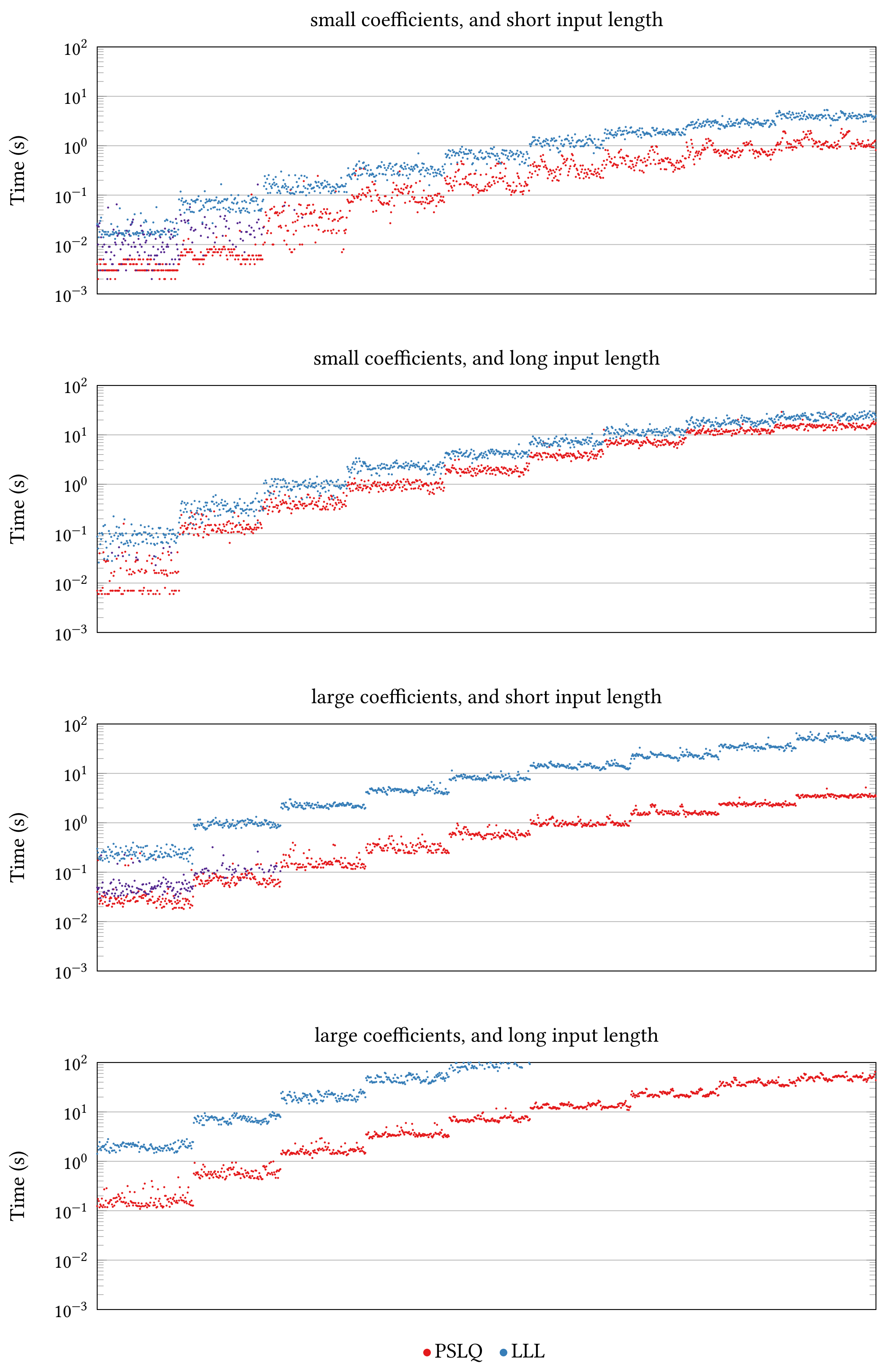


Figure B.63: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{C}}$.

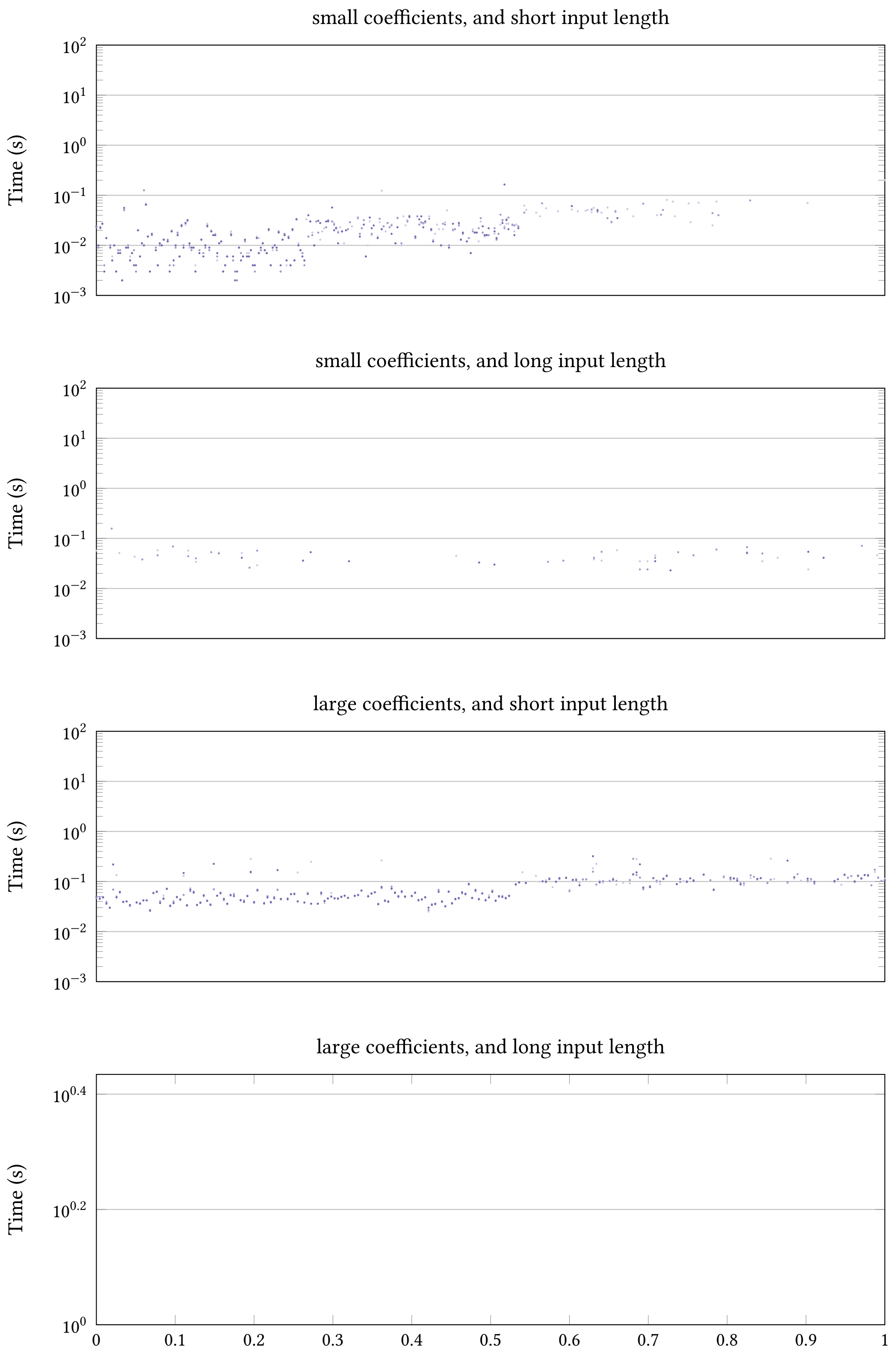


Figure B.64: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-5}), C = C_{\mathbb{C}}$ (APSLQ only).

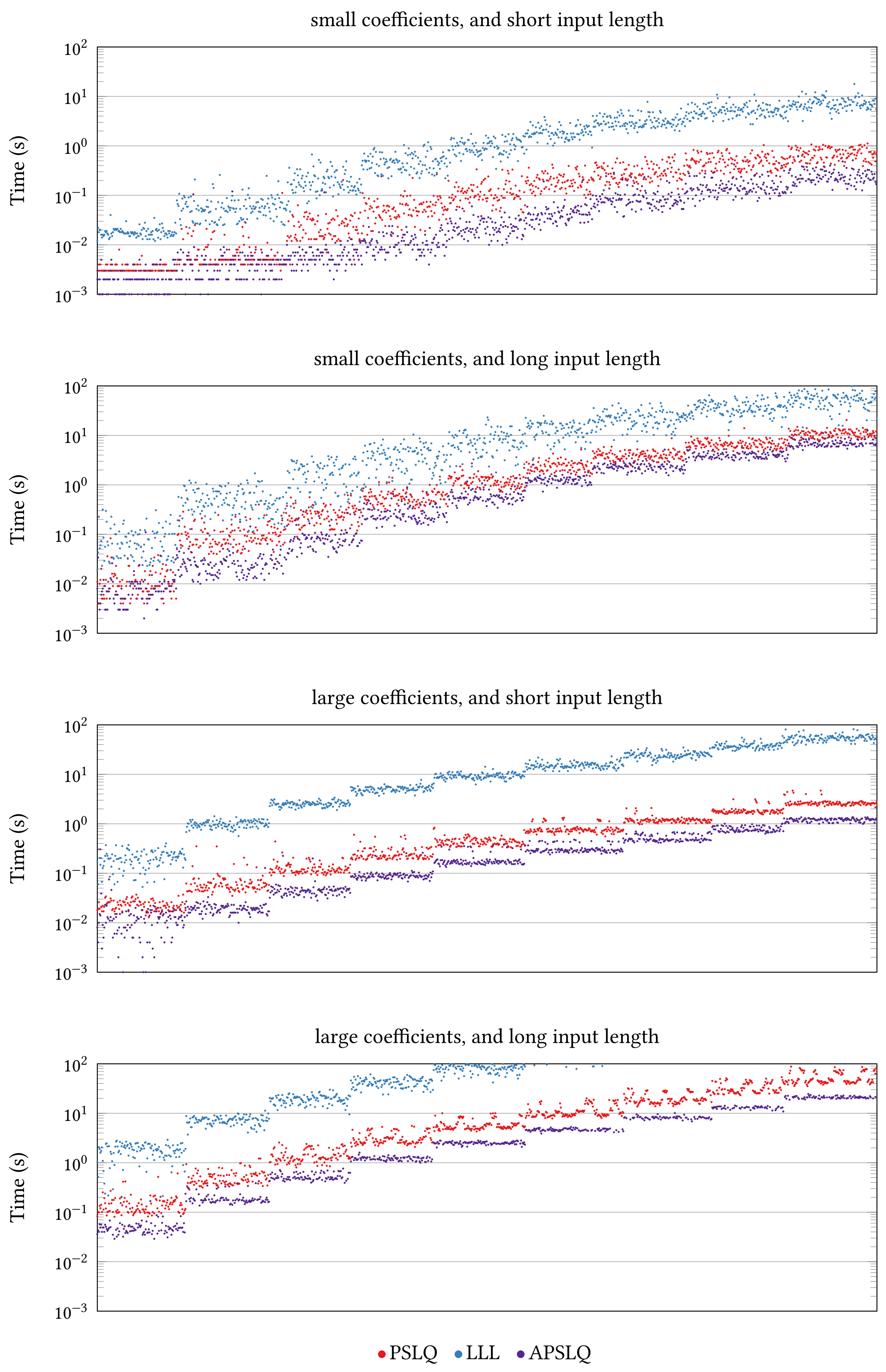


Figure B.65: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{R}}$.

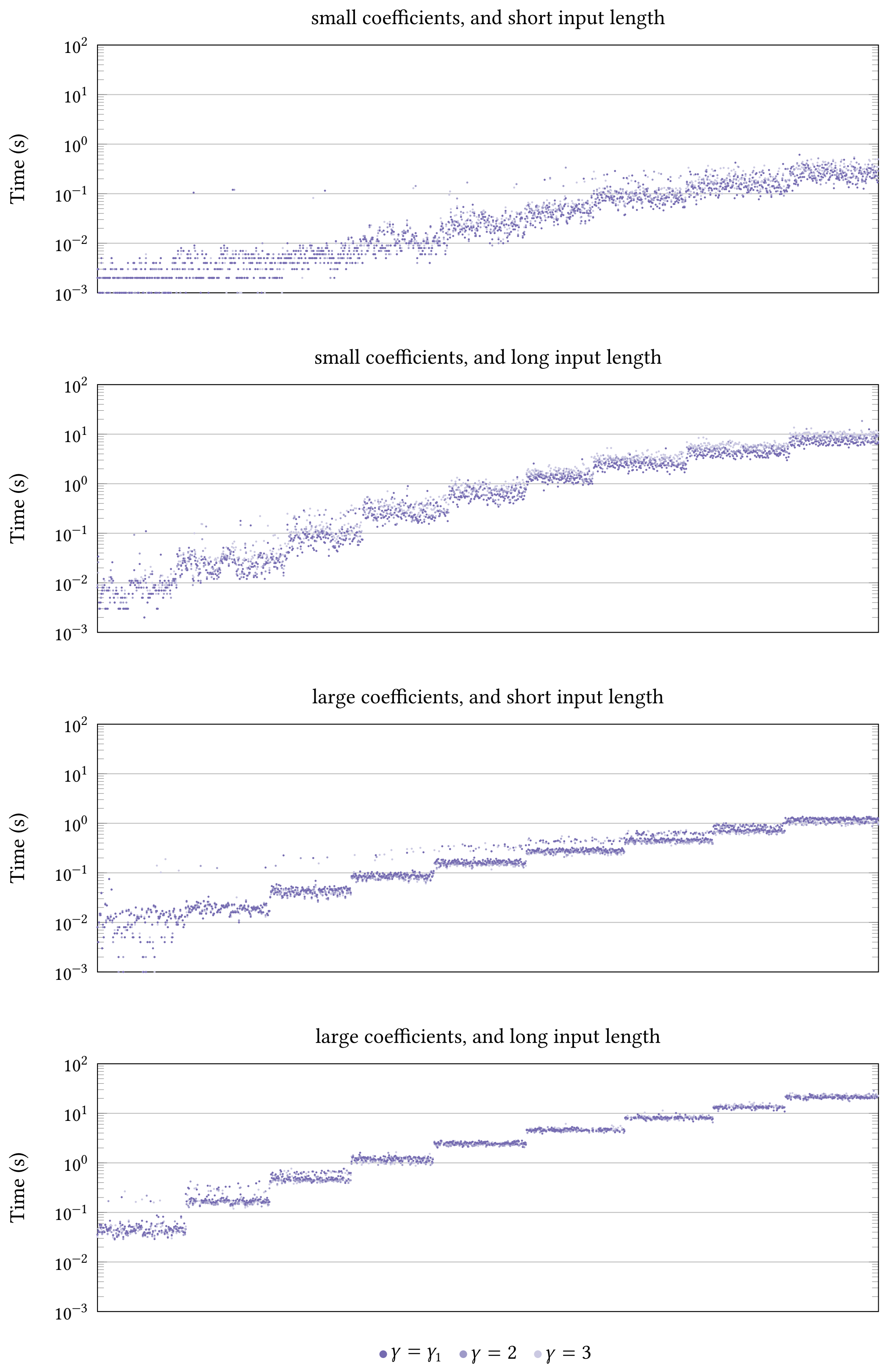


Figure B.66: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_\mathbb{R}$ (APSLQ only).

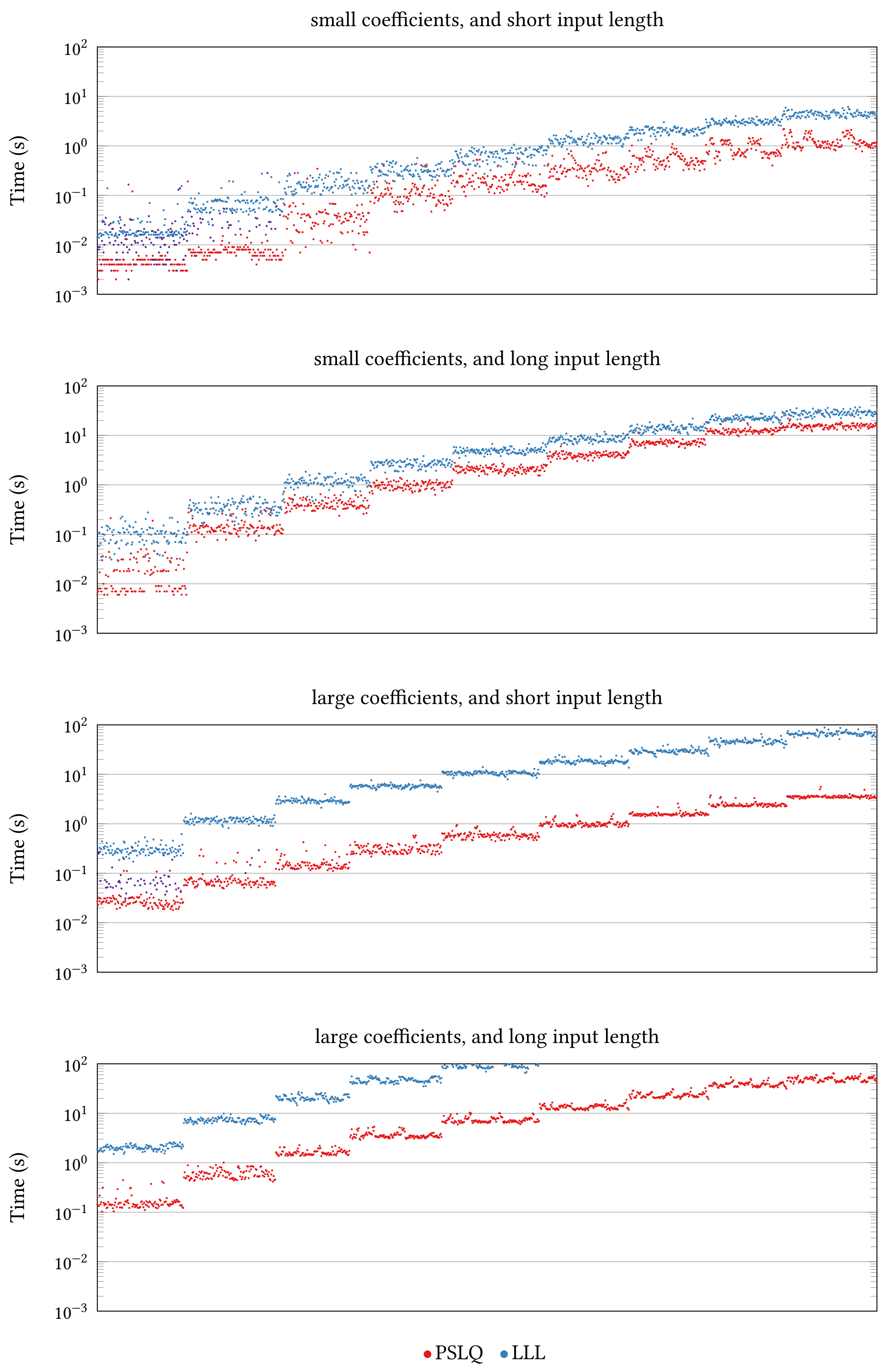


Figure B.67: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{C}}$.

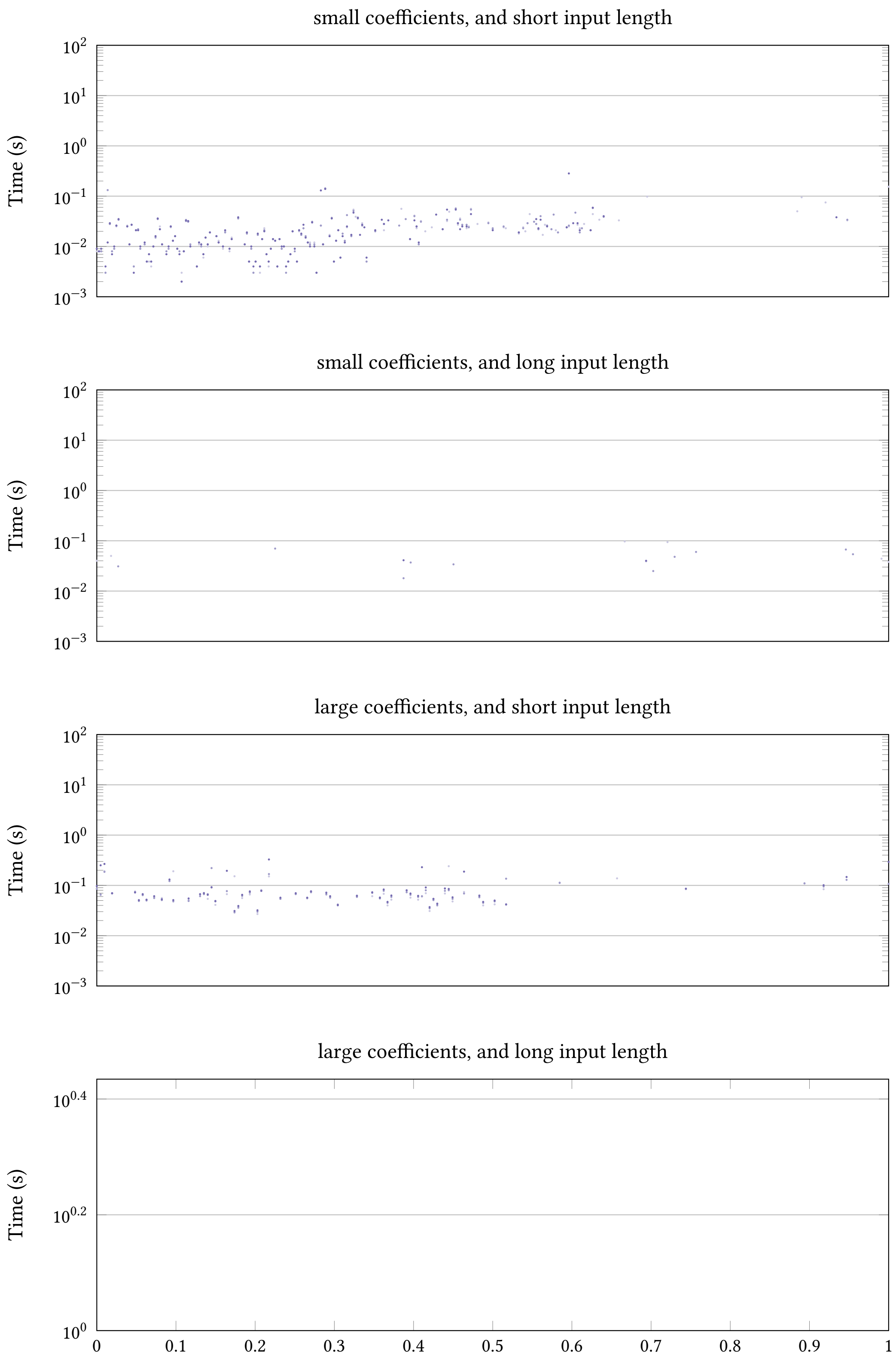


Figure B.68: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-6}), C = C_{\mathbb{C}}$ (APSLQ only).

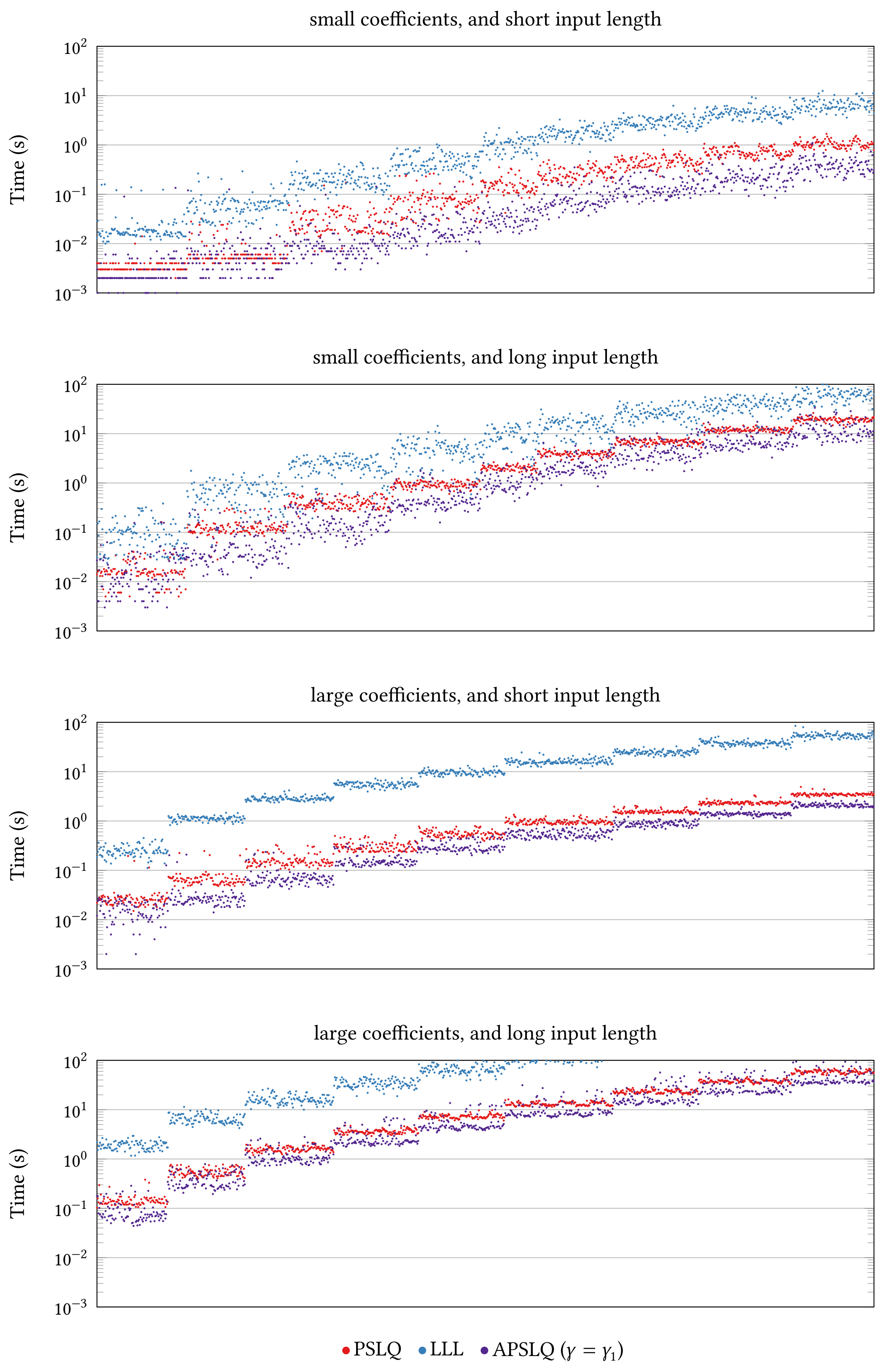


Figure B.69: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{R}}$.

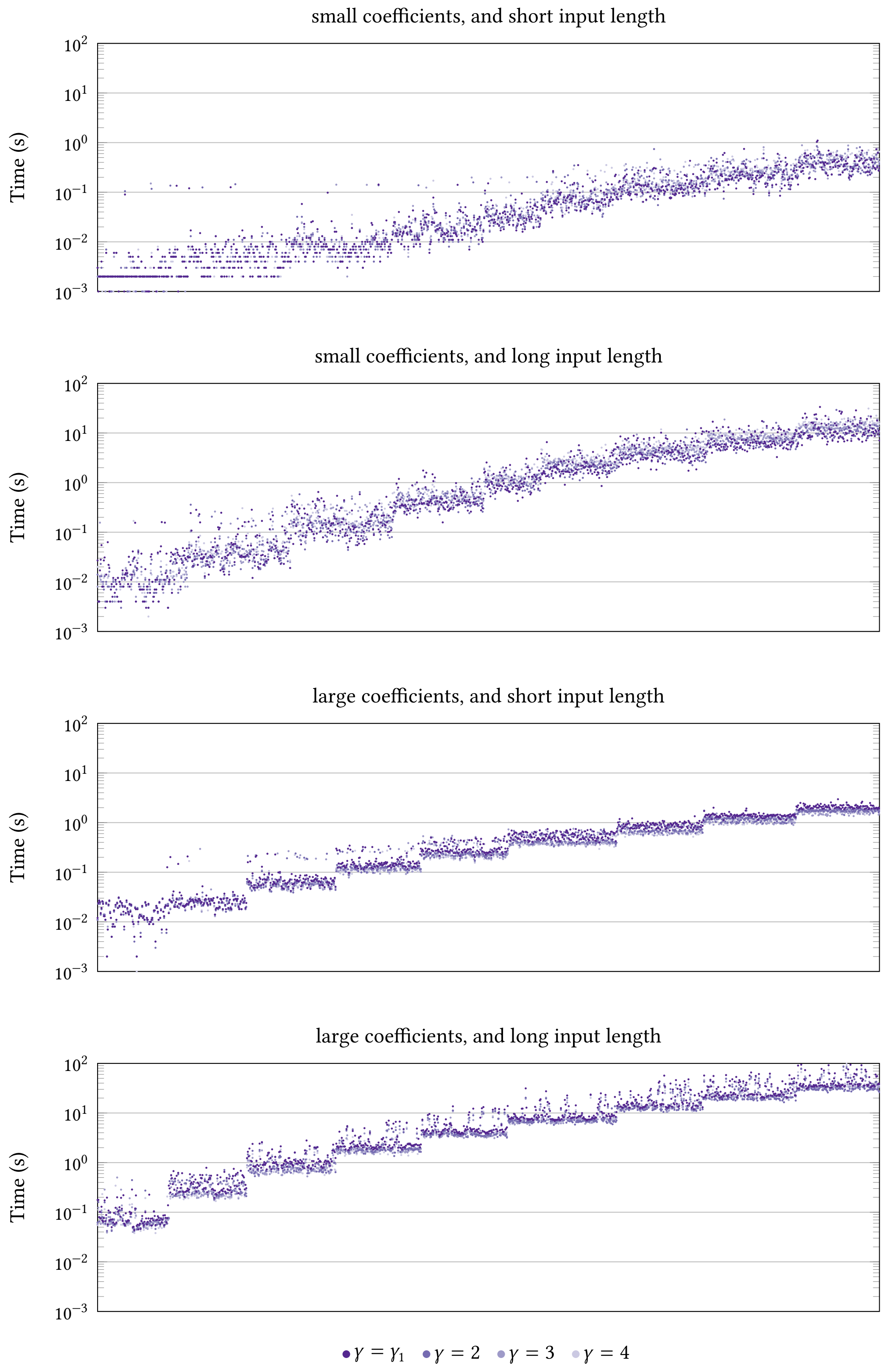


Figure B.70: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{R}}$ (APSLQ only).

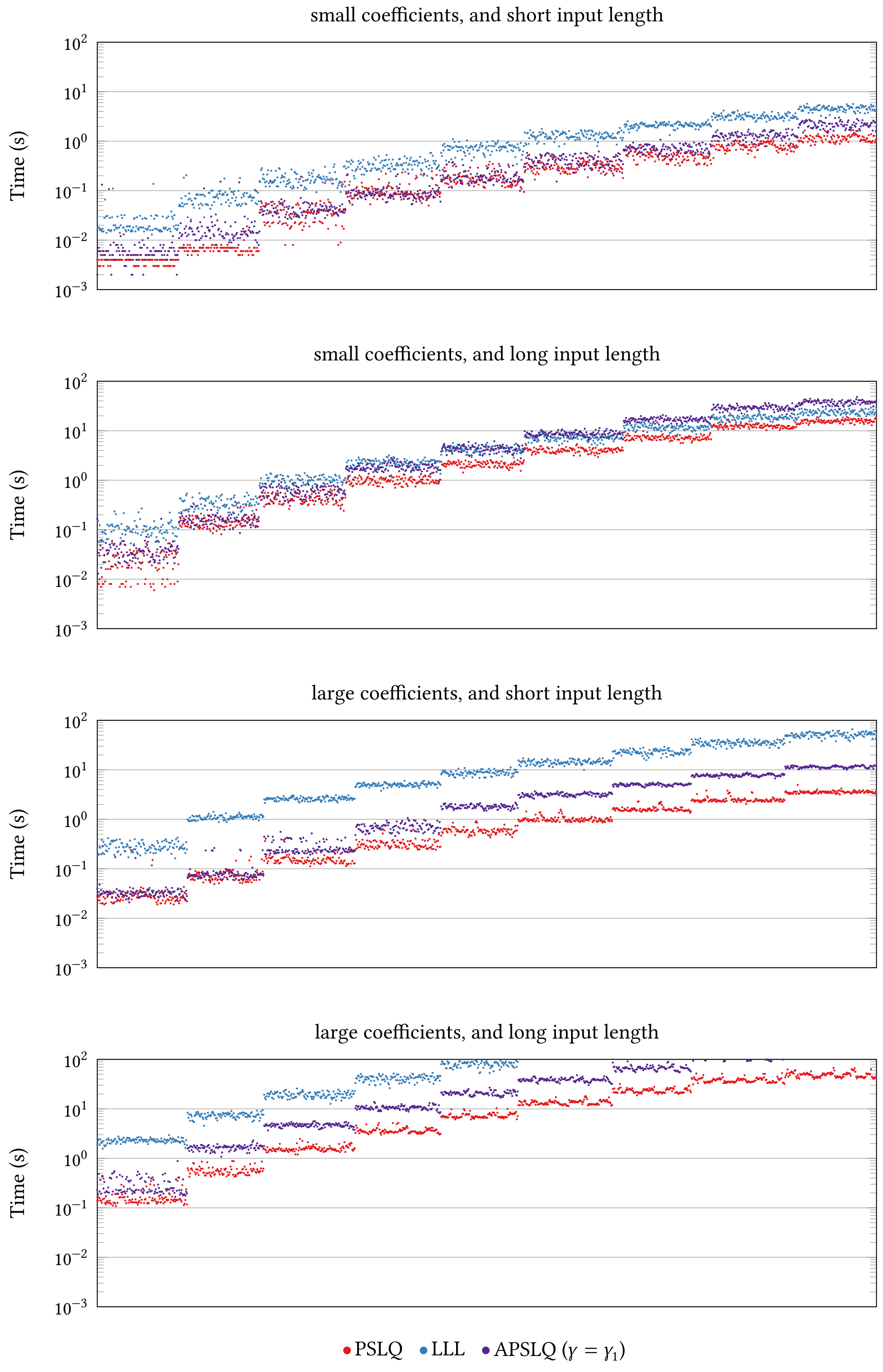


Figure B.71: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{C}}$.

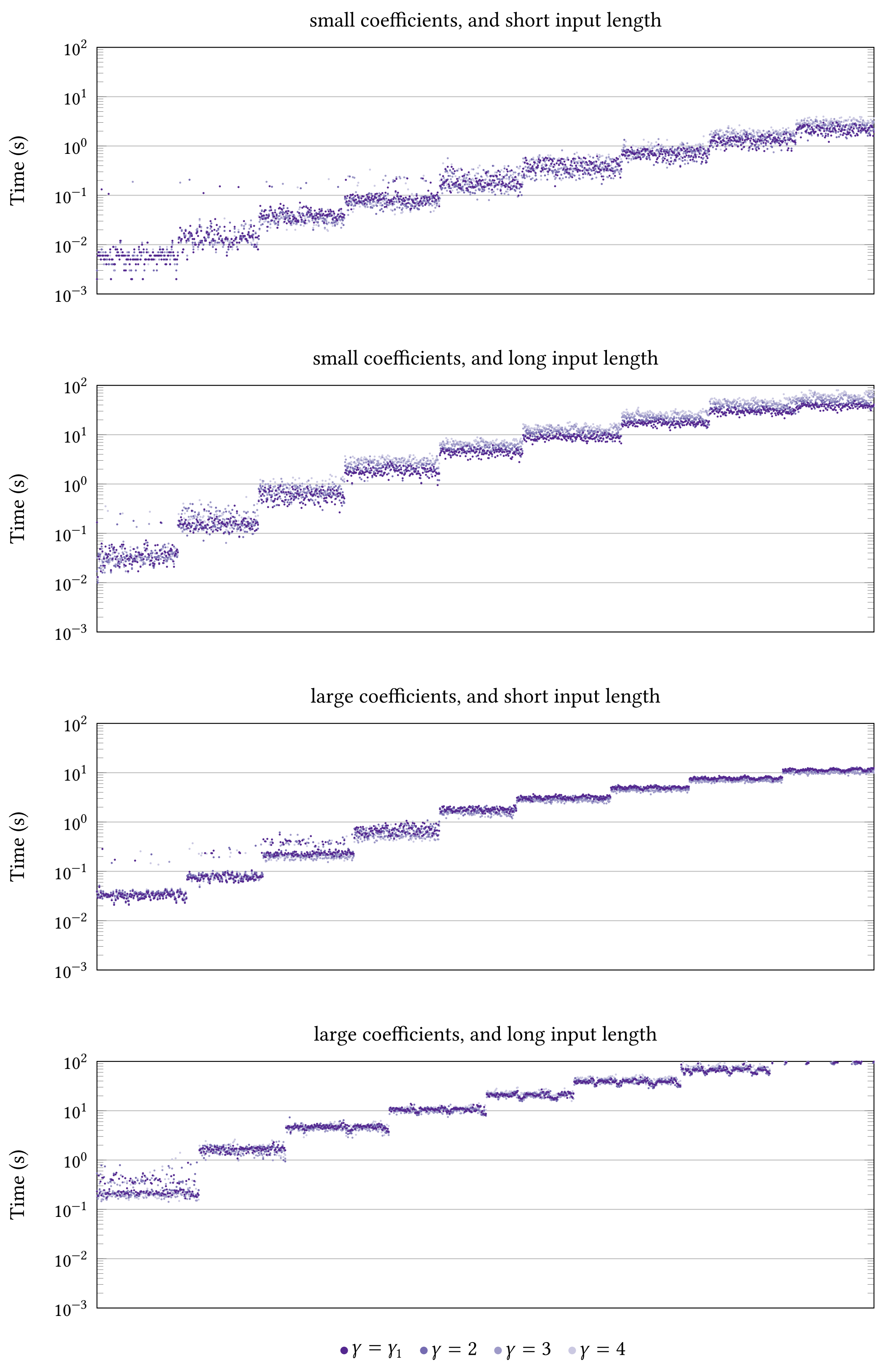


Figure B.72: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-7}), C = C_{\mathbb{C}}$ (APSLQ only).

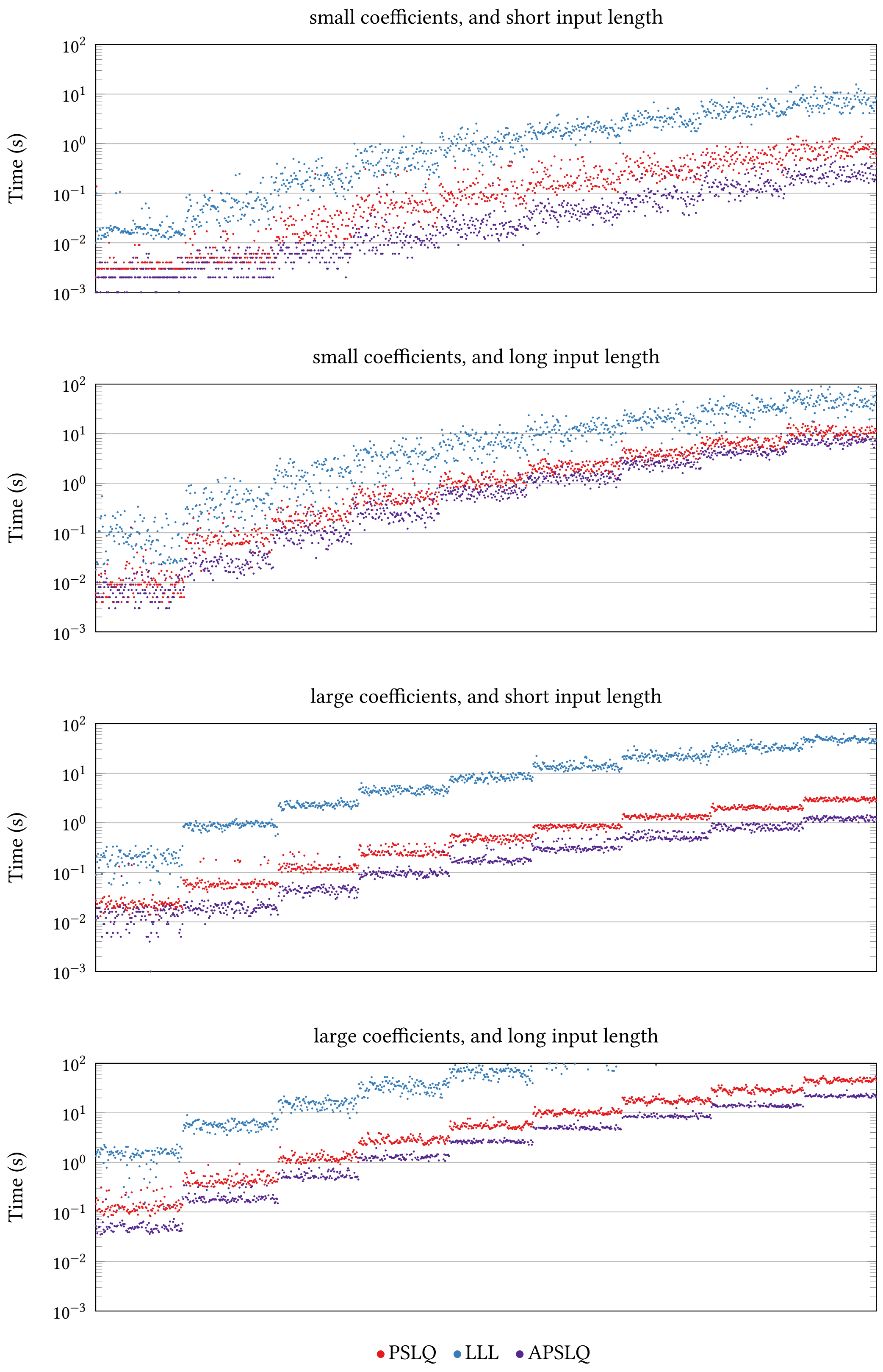


Figure B.73: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{R}}$.

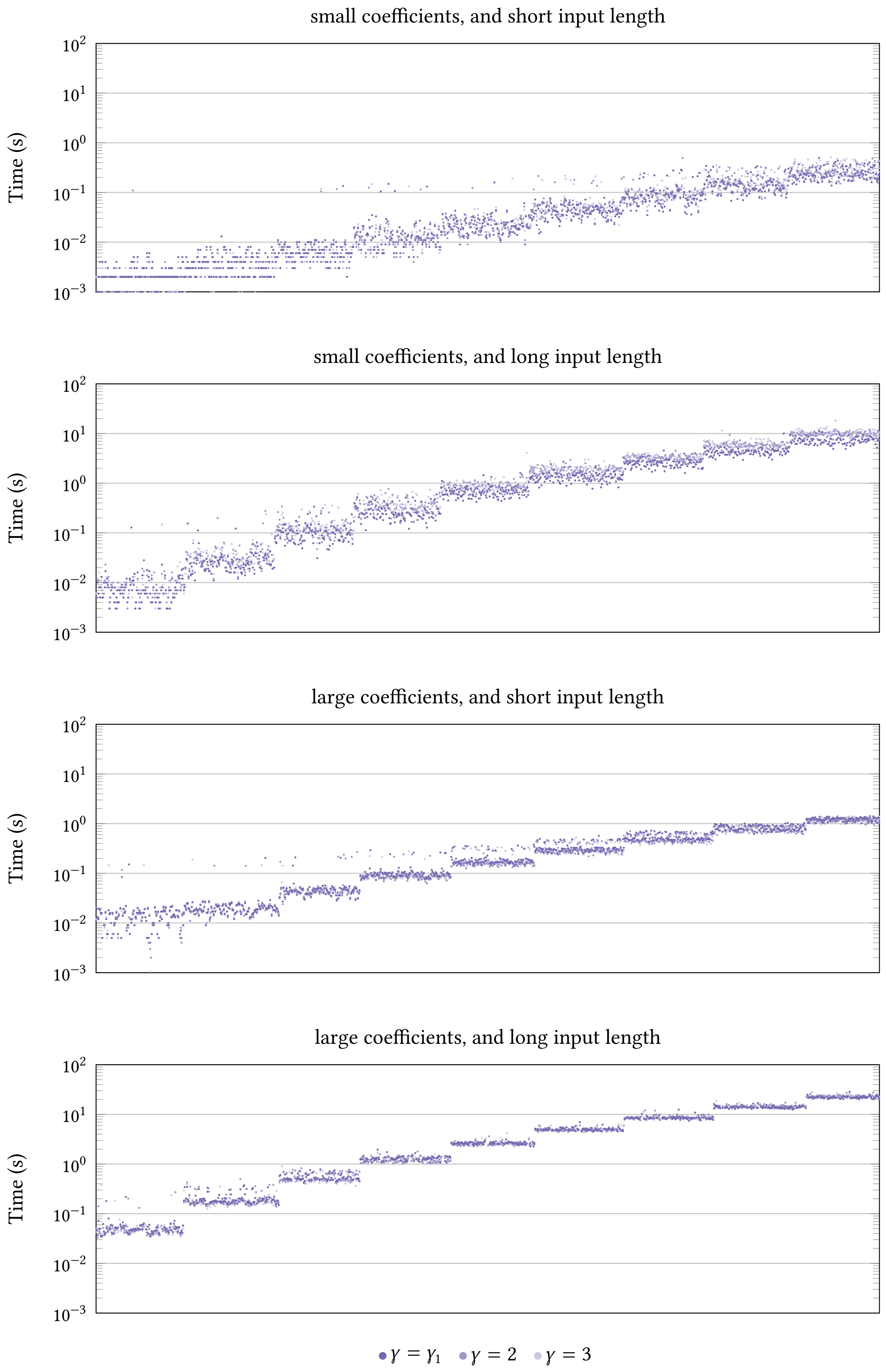


Figure B.74: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{R}}$ (APSLQ only).

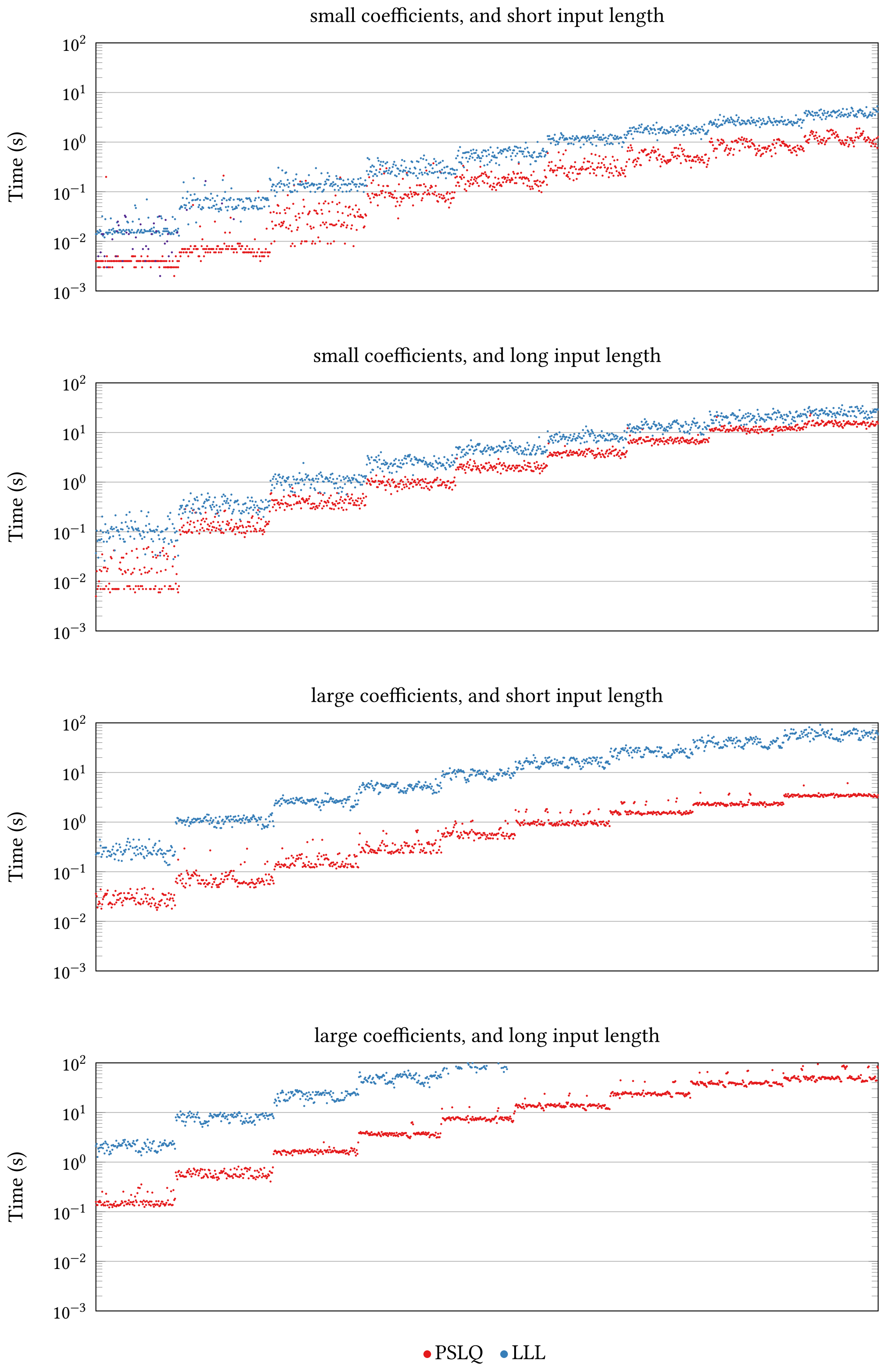


Figure B.75: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{C}}$.

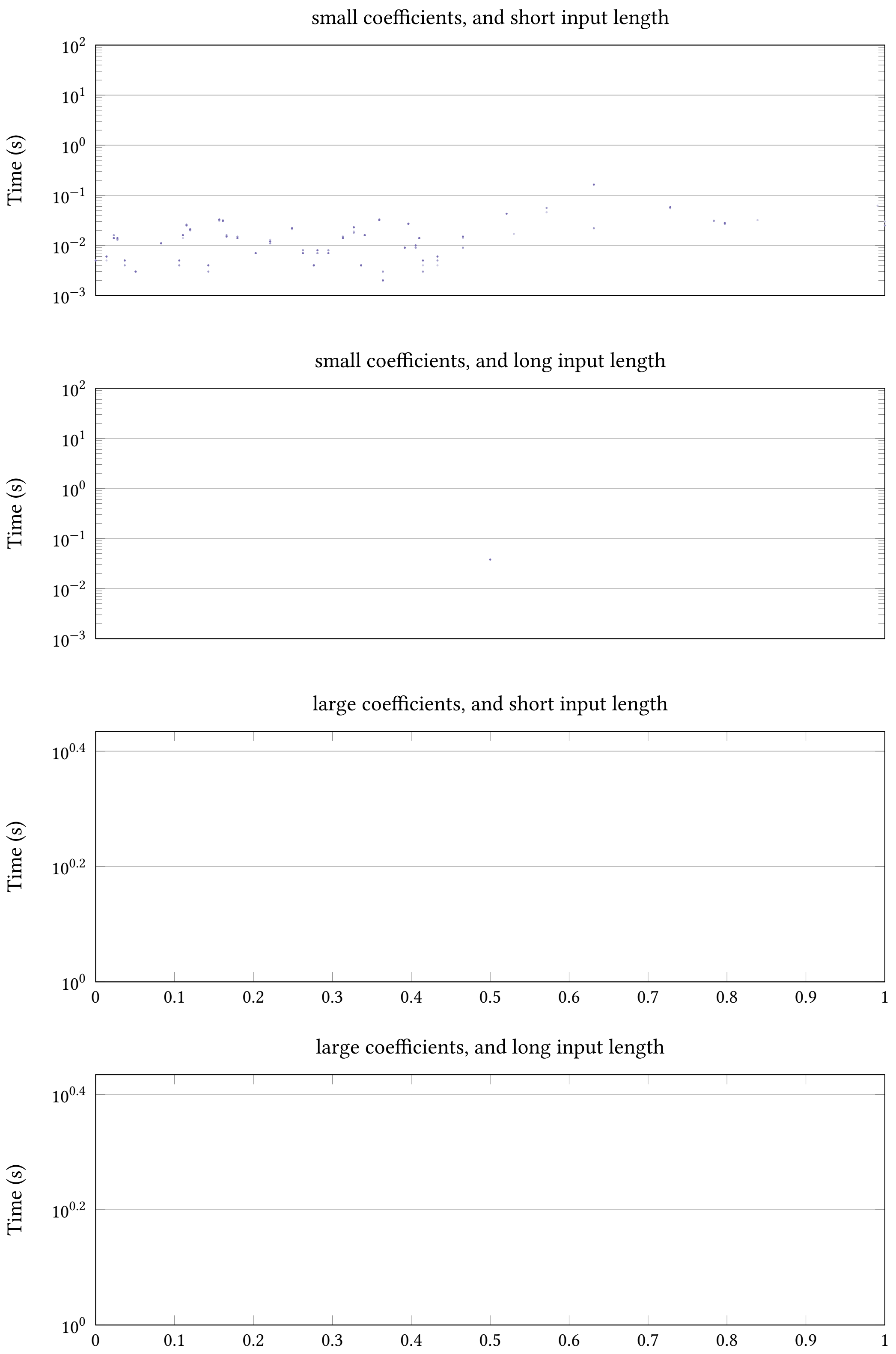


Figure B.76: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-10}), C = C_{\mathbb{C}}$ (APSLQ only).

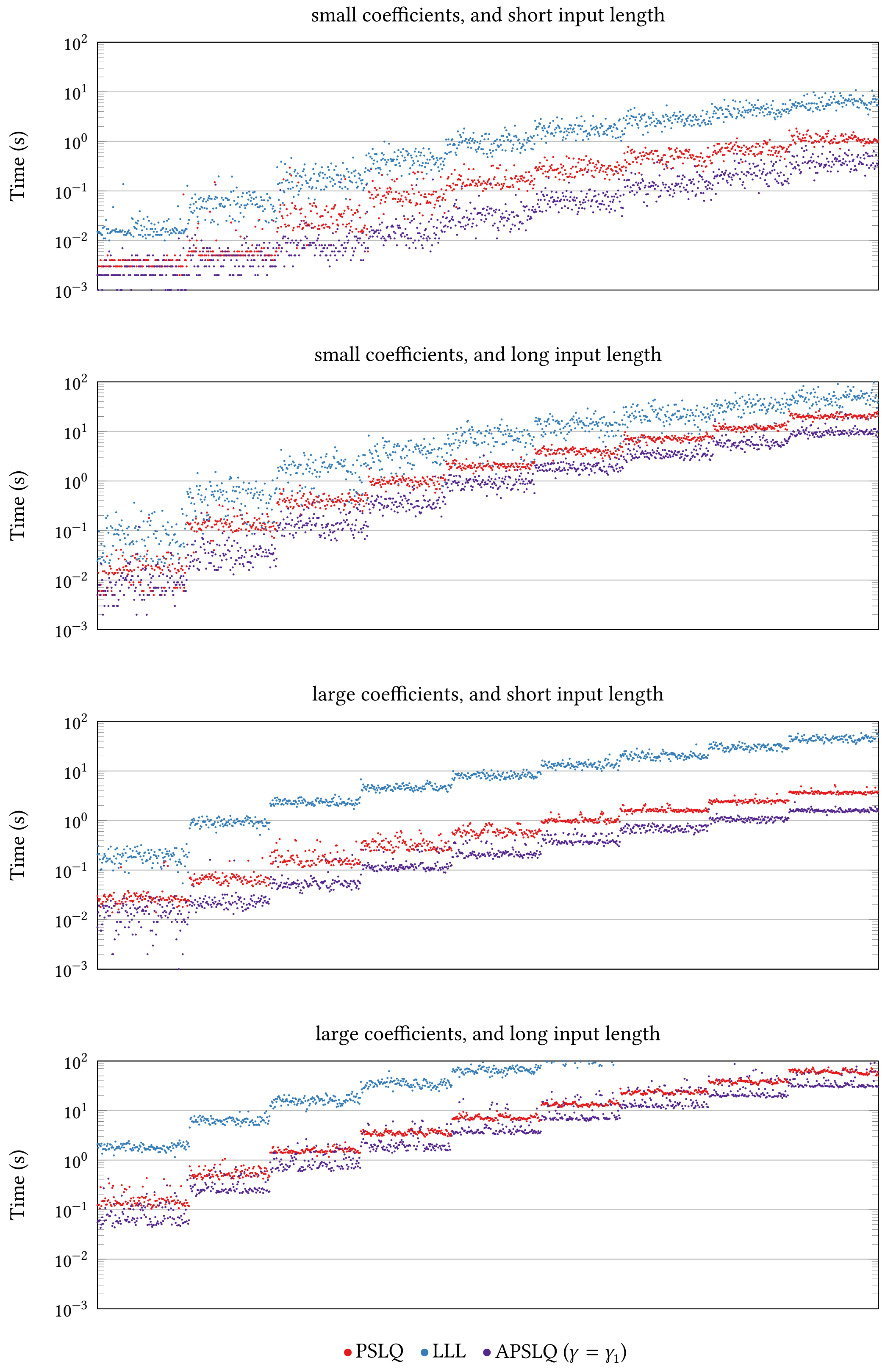


Figure B.77: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{R}}$.

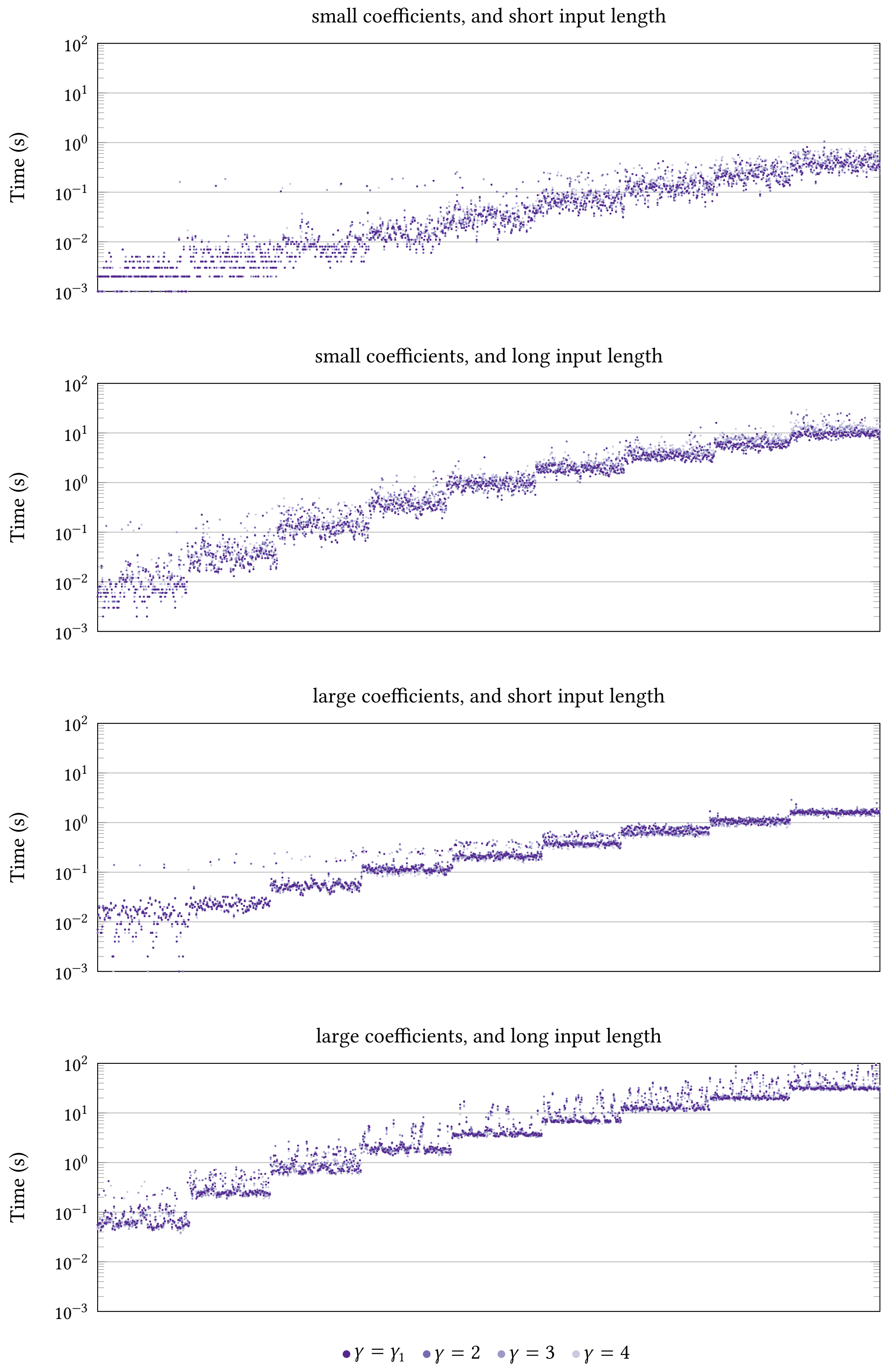


Figure B.78: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{R}}$ (APSLQ only).

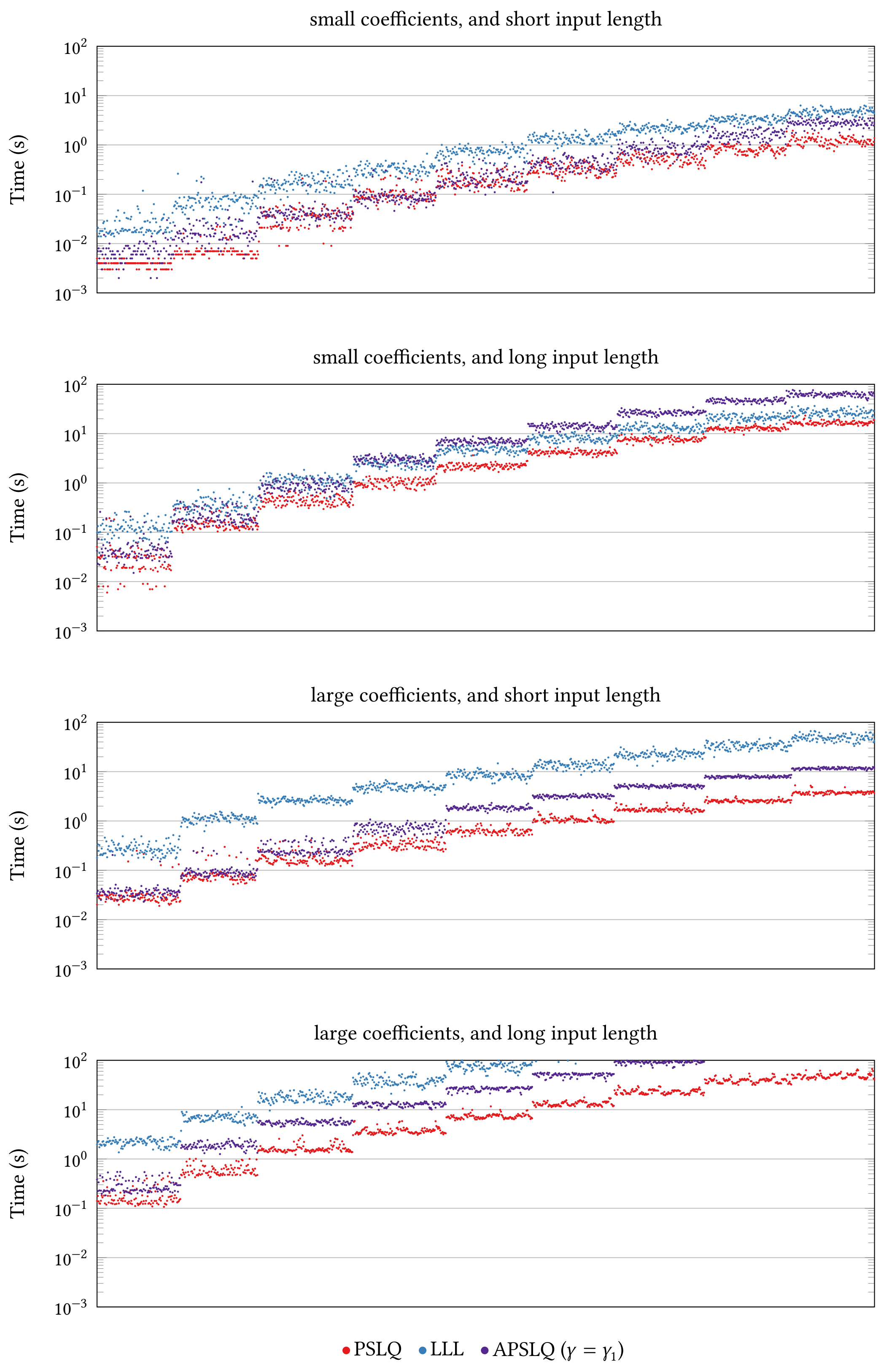


Figure B.79: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{C}}$.

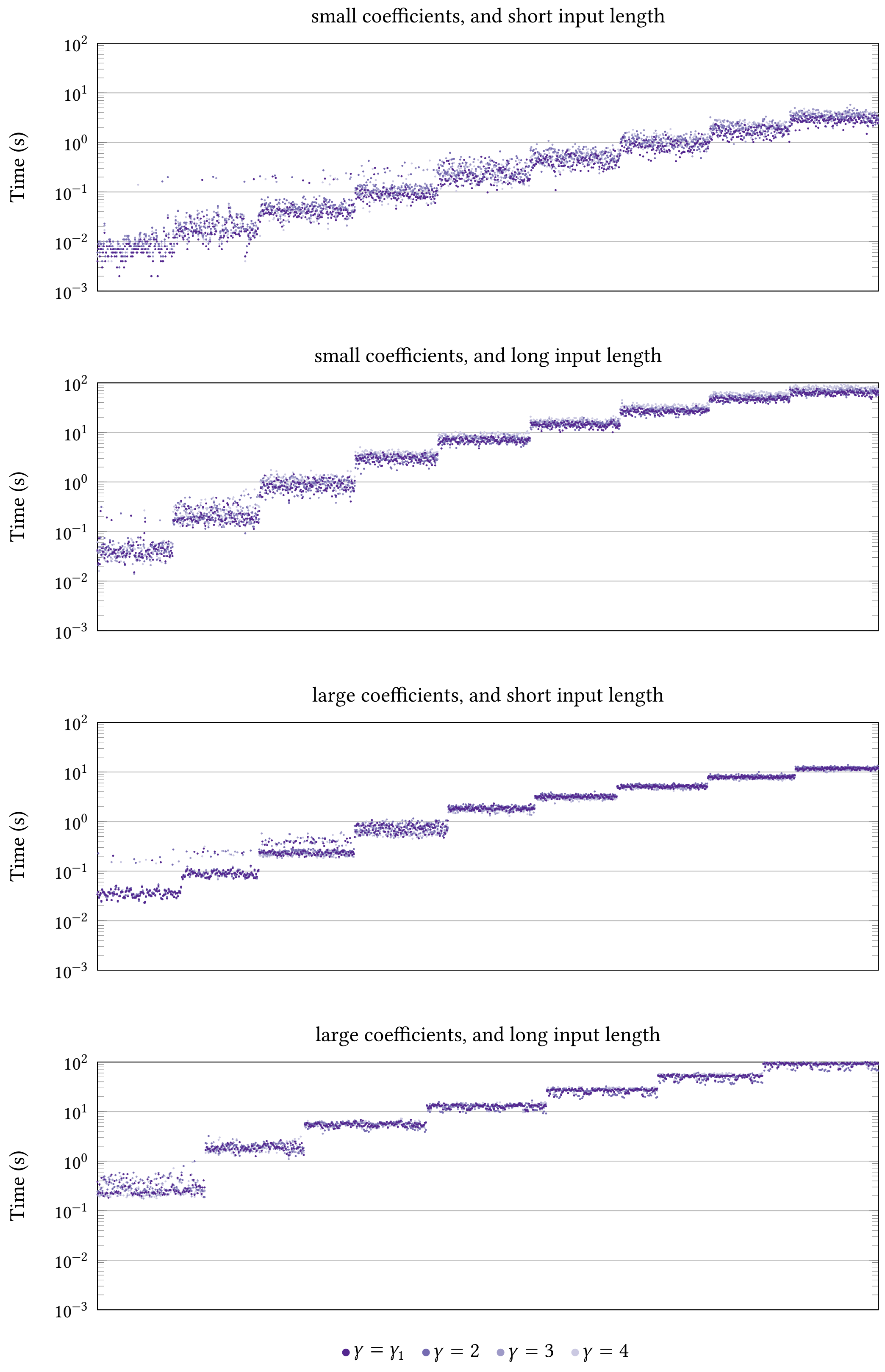


Figure B.80: Computation time for $\mathbb{K} = \mathbb{Q}(\sqrt{-11}), C = C_{\mathbb{C}}$ (APSLQ only).

Bibliography

[1] A. G. Ağargün and R. Fletcher C. “Euclidean Rings”. In: *Turkish Journal of Mathematics* 19.3 (1995), pp. 291–299.

[2] F. J. Aragón Artacho and J. M. Borwein. “Global convergence of a non-convex Douglas-Rachford iteration”. In: *Journal of Global Optimization* 57.3 (Nov. 2013), pp. 753–769. DOI: 10.1007/s10898-012-9958-4.

[3] F. J. Aragón Artacho, J. M. Borwein, and M. K. Tam. “Douglas-Rachford feasibility methods for matrix completion Problems”. In: *The ANZIAM Journal* 55.4 (Apr. 2014), pp. 299–326. DOI: 10.1017/S1446181114000145.

[4] F. J. Aragón Artacho, J. M. Borwein, and M. K. Tam. “Recent results on Douglas-Rachford methods for combinatorial optimization problems”. In: *Journal of Optimization Theory and Applications* 163.1 (Oct. 2014), pp. 1–30. DOI: 10.1007/s10957-013-0488-0.

[5] M. Baake and U. Grimm. *Aperiodic Order*. Vol. 1: *A Mathematical Invitation*. Encyclopedia of Mathematics and Its Applications 149. Cambridge University Press. DOI: 10.1017/CBO9781139025256.

[6] D. H. Bailey and J. M. Borwein. “Experimental computation as an ontological game changer. The impact of modern mathematical computation tools on the ontology of mathematics”. In: *Mathematics, Substance and Surmise. Views on the Meaning and Ontology of Mathematics*. Ed. by E. Davis and P. J. Davis. Springer International Publishing, 2015, pp. 25–67. DOI: 10.1007/978-3-319-21473-3_3.

[7] D. H. Bailey, J. M. Borwein, J. S. Kimberley, and W. Ladd. “Computer Discovery and Analysis of Large Poisson Polynomials”. In: *Experimental Mathematics* 26.3 (2017), pp. 349–363. DOI: 10.1080/10586458.2016.1180565.

[8] D. H. Bailey, P. Borwein, and S. Plouffe. “On the Rapid Computation of Various Polylogarithmic Constants”. In: *Mathematics of Computation* 66.218 (Apr. 1997), pp. 903–913. DOI: 10.1090/S0025-5718-97-00856-9.

[9] D. H. Bailey and D. J. Broadhurst. “Parallel Integer Relation Detection: Techniques and Applications”. In: *Mathematics of Computation* 70.236 (2001), pp. 1719–1736. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-00-01278-3.

[10] D. H. Bailey and H. R. P. Ferguson. “Numerical Results on Relations Between Fundamental Constants Using a New Algorithm”. In: *Mathematics of Computation* 53.188 (1989), pp. 649–656.

[11] D. H. Bailey and S. Plouffe. “Recognizing Numerical Constants”. In: *Organic Mathematics: Proceedings of the Organic Mathematics Workshop*. Canadian Mathematical Society, 1997, pp. 73–88. URL: http://www.cecm.sfu.ca/organics.

[12] H. H. Bauschke and J. M. Borwein. “On Projection Algorithms for Solving Convex Feasibility Problems”. In: *SIAM Review* 38.3 (1996), pp. 367–426. DOI: 10.1137/S0036144593251710.

[13] H. H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. 1st ed. New York, NY: Springer, 2011. DOI: 10.1007/978-1-4419-9467-7.

[14] H. H. Bauschke, P. L. Combettes, and D. R. Luke. “Phase retrieval, error reduction algorithm, and Fienup variants: a view from convex optimization”. In: *Journal of the Optical Society of America* 19.7 (July 2002), pp. 1334–1345. DOI: 10.1364/JOSAA.19.001334.

[15] H. H. Bauschke, P. L. Combettes, and D. R. Luke. “Finding best approximation pairs relative to two closed convex sets in Hilbert spaces”. In: *Journal of Approximation Theory* 127.2 (2004), pp. 178–192. DOI: 10.1016/j.jat.2004.02.006.

[16] H. H. Bauschke, P. L. Combettes, and D. R. Luke. “A strongly convergent reflection method for finding the projection onto the intersection of two closed convex sets in a Hilbert space”. In: *Journal of Approximation Theory* 141.1 (2006), pp. 63–69. DOI: 10.1016/j.jat.2006.01.003.

[17] H. H. Bauschke and W. M. Moursi. “On the Douglas-Rachford algorithm”. In: *Mathematical Programming* 164.1 (July 2017), pp. 263–284. DOI: 10.1007/s10107-016-1086-3.

[18] H. H. Bauschke et al. “The rate of linear convergence of the Douglas-Rachford algorithm for subspaces is the cosine of the Friedrichs angle”. In: *Journal of Approximation Theory* 185 (Sept. 2014), pp. 63–79. DOI: 10.1016/j.jat.2014.06.002.

[19] J. Benoist. “The Douglas-Rachford Algorithm for the Case of the Sphere and the Line”. In: *Journal of Global Optimization* 63.2 (Oct. 2015), pp. 363–380. DOI: 10.1007/s10898-015-0296-1.

[20] G. M. Bergman. “Notes on Ferguson and Forcade’s Generalized Euclidean Algorithm”. Unpublished notes. 1980.

[21] J. Borwein and K. Devlin. *The Computer as Crucible. An Introduction to Experimental Mathematics*. A K Peters, 2009. DOI: 10.1201/b10684.

[22] J. M. Borwein and P. Lisoněk. “Applications of Integer Relation Algorithms”. In: *Discrete Mathematics* 217.1-3 (Apr. 2000), pp. 65–82. DOI: 10.1016/S0012-365X(99)00256-3.

[23] J. M. Borwein and M. K. Tam. “A cyclic Douglas–Rachford iteration scheme”. In: *Journal of Optimization Theory and Applications* 160.1 (Jan. 2014), pp. 1–29. DOI: 10.1007/s10957-013-0381-x.

[24] J. M. Borwein. “The Life of Modern Homo Habilis Mathematicus. Experimental Computation and Visual Theorems”. In: *Tools and Mathematics*. Mathematics Education Library 110. Cham: Springer International Publishing, 2016, pp. 23–90. DOI: 10.1007/978-3-319-02396-0_3.

[25] J. M. Borwein and B. Sims. “The Douglas-Rachford algorithm in the absence of convexity”. In: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. Ed. by H. H. Bauschke et al. Springer Optimization and Its Applications 49. New York, NY: Springer, 2011, pp. 93–109. DOI: 10.1007/978-1-4419-9569-8_6.

[26] J. M. Borwein and M. K. Tam. “Reflection Methods for Inverse Problems with Applications to Protein Conformation Determination”. In: *Generalized Nash Equilibrium Problems, Bilevel Programming and MPEC*. Ed. by D. Aussel and C. Lalitha. Forum for Interdisciplinary Mathematics. Singapore: Springer, 2017, pp. 83–100. DOI: 10.1007/978-981-10-4774-9_5.

[27] J. M. Borwein et al. *Appendix to dynamics of the Douglas-Rachford method for ellipses and p-spheres*. 2017. URL: http://hdl.handle.net/1959.13/1330341.

[28] J. M. Borwein et al. “Dynamics of the Douglas Rachford Method for Ellipses and p-Spheres”. In: *Set-Valued and Variational Analysis* 26.2 (June 2018), pp. 385–403. DOI: 10.1007/s11228-017-0457-0.

[29] P. Borwein. *Computational Excursions in Analysis and Number Theory*. CMS Books in Mathematics. New York: Springer-Verlag, 2002. ISBN: 978-0-387-95444-8. DOI: 10.1007/978-0-387-21652-2.

[30] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. New York, NY: Springer, 2011. DOI: 10.1007/978-0-387-70914-7.

[31] M. Chamberland. “Using Integer Relations Algorithms for Finding Relationships Among Functions”. In: *Tapas in Experimental Mathematics*. AMS Special Session on Experimental Mathematics. Ed. by T. Amdeberhan and V. Moll. Vol. 457. Contemporary Mathematics. New Orleans, Louisiana: American Mathematical Society, 2008, pp. 127–133.

[32] D. A. Clark. “A quadratic field which is Euclidean but not norm-Euclidean”. In: *manuscripta mathematica* 83.1 (Dec. 1994), pp. 327–330. DOI: 10.1007/BF02567617.

[33] P. J. Davis. *The Schwarz function and its applications*. 1st ed. Carus Mathematical Monographs 17. Mathematical Association of America, 1974. URL: https://www.jstor.org/stable/10.4169/j.ctt5hh99x.

[34] F. Deutsch. "Rate of Convergence of the Method of Alternating Projections". In: Parametric Optimization and Approximation (Mathematisches Forschungsinstitut, Oberwolfach, Oct. 16, 1983–Oct. 22, 1983). Ed. by B. Brosowski and F. Deutsch. International Series of Numerical Mathematics / Internationale Schriftenreihe zur Numerischen Mathematik / Série internationale d'Analyse numérique book 72. Basel: Birkhäuser, 1985, pp. 96–107. DOI: 10.1007/978-3-0348-6253-0_7.

[35] A. Dontchev. private communication.

[36] J. Douglas and H. H. Rachford. "On the numerical solution of the heat conduction problem in 2 and 3 space variables". In: *Transactions of the American Mathematical Society* 82.2 (May 1956), pp. 421–43. DOI: 10.2307/1993056.

[37] V. Elser, I. Rankenburg, and P. Thibault. "Searching with iterated maps". In: *Proceedings of the National Academy of Sciences* 104.2 (2007), pp. 418–423. DOI: 10.1073/pnas.0606359104.

[38] H. R. P. Ferguson. "A Short Proof of the Existence of Vector Euclidean Algorithms". In: *Proceedings of the American Mathematical Society* 97.1 (1986), pp. 8–10. DOI: 10.2307/2046068.

[39] H. R. P. Ferguson. "A Noninductive GL(n, Z) Algorithm That Constructs Integral Linear Relations for n Z-Linearly Dependent Real Numbers". In: *Journal of Algorithms* 8.1 (Mar. 1987), pp. 131–145.

[40] H. R. P. Ferguson. "PSOS. A new integral relation finding algorithm involving partial sums of squares and no square roots". In: *Abstracts of Papers Presented to the American Mathematical Society* 9 (Mar. 1988), p. 214.

[41] H. R. P. Ferguson and D. H. Bailey. *A Polynomial Time, Numerically Stable Integer Relation Algorithm*. RNR Technical Report RNR-91-032. July 1992. URL: https://www.nas.nasa.gov/assets/pdf/techreports/1991/rnr-91-032.pdf.

[42] H. R. P. Ferguson, D. H. Bailey, and S. Arno. "Analysis of PSLQ, an Integer Relation Finding Algorithm". In: *Mathematics of Computation* 68.225 (1999), pp. 351–369. DOI: 10.1090/S0025-5718-99-00995-3.

[43] H. R. P. Ferguson and R. W. Forcade. "Generalization of the Euclidean Algorithm for Real Numbers to All Dimensions Higher Than Two". In: *Bulletin (New Series) of the American Mathematical Society* 1.6 (Nov. 1979), pp. 912–914. DOI: 10.1090/S0273-0979-1979-14691-3.

[44] H. R. P. Ferguson and R. W. Forcade. "Multidimensional Euclidean Algorithms". In: *Journal für die reine und angewandte Mathematik (Crelle's Journal)* 1982.334 (1982), pp. 171–181.

[45] Y. H. Gan, C. Ling, and W. H. Mow. “Complex lattice reduction algorithm for low-complexity full-diversity MIMO detection”. In: *IEEE Transactions on Signal Processing* 57.7 (July 2009), pp. 2701–2710. DOI: 10.1109/TSP.2009.2016267.

[46] S. Gravel and V. Elser. “Divide and concur: A general approach to constraint satisfaction”. In: *Physical Review E* 78.3 (Sept. 2008), p. 036706. DOI: 10.1103/PhysRevE.78.036706.

[47] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. 5th ed. Oxford University Press, 1979. ISBN: 978-0-19-853171-5; 978 0-19-853170-8.

[48] J. Hastad, B. Just, J. C. Lagarias, and C. P. Schnorr. “Polynomial Time Algorithms for Finding Integer Relations Among Real Numbers”. In: *SIAM Journal on Computing* 18.5 (1989), pp. 859–881. DOI: 10.1137/0218059.

[49] J. Hastad, B. Just, J. C. Lagarias, and C. P. Schnorr. “Erratum: Polynomial Time Algorithms for Finding Integer Relations Among Real Numbers”. In: *SIAM Journal on Computing* 43.1 (2014), p. 254. DOI: 10.1137/130947799.

[50] T. L. Heath. *The Thirteen Books of Euclid’s Elements*. Translated from the text of Heiberg. Vol. 3: *Books X–XIII and Appendix*. Cambridge University Press, 1908.

[51] V. Lakshmikantham and D. Trigiante. *Theory of Difference Equations. Numerical Methods and Applications*. 2nd ed. Monographs and textbooks in pure and applied mathematics 251. Marcel Dekker, Inc, 2002. ISBN: 978-0-8247-0803-0.

[52] A. K. Lenstra, H. W. Lenstra Jr, and L. Lovász. “Factoring Polynomials with Rational Coefficients”. In: *Mathematische Annalen* 261.4 (Dec. 1982), pp. 515–534. DOI: 10.1007/BF01457454.

[53] S. B. Lindstrom, B. Sims, and M. P. Skerritt. “Computing intersections of implicitly specified plane curves”. In: *Journal of Nonlinear and Convex Analysis* 18.3 (2017), pp. 347–359.

[54] P. Lions and B. Mercier. “Splitting algorithms for the sum of two nonlinear operators”. In: *SIAM Journal on Numerical Analysis* 16.6 (1979), pp. 964–979. DOI: 10.1137/0716071.

[55] J. E. Littlewood. *Littlewood’s Miscellany*. Ed. by B. Bollobás. Cambridge University Press, 1986. ISBN: 978-0521337021.

[56] Y. Luo and S. Qiao. “A Parallel LLL Algorithm”. In: *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering*. C3S2E ’11. Montreal, Quebec, Canada: Association for Computing Machinery, 2011, pp. 93–101. DOI: 10.1145/1992896.1992908.

[57] S. Lyu, C. Porter, and C. Ling. “Performance Limits of Lattice Reduction over Imaginary Quadratic Fields with Applications to Compute-and-Forward”. In: *2018 IEEE Information Theory Workshop (ITW)*. 2018, pp. 1–5. DOI: 10.1109/ITW.2018.8613476.

[58] A. Meichsner. “Integer Relation Algorithms and the Identification of Numerical Constants”. M.Sc thesis. Simon Fraser University, 2001.

[59] A. Meichsner. “The Integer Chebyshev Problem: Computational Explorations”. PhD thesis. Simon Fraser University, 2009.

[60] J. S. Milne. *Algebraic Number Theory*. Course notes. 2017. URL: `http://www.jmilne.org/math/CourseNotes/ant.html`.

[61] T. Needham. *Visual complex analysis*. Oxford: Clarendon Press, 1997. ISBN: 978-0-19-853446-4.

[62] G. Pierra. “Eclatement de contraintes en parallele pour la minimisation d’une forme quadratique”. In: *Optimization Techniques Modeling and Optimization in the Service of Man Part 2*. Ed. by J. Cea. Berlin, Heidelberg: Springer, 1976, pp. 200–218. ISBN: 978-3-540-38150-1. DOI: `10.1007/3-540-07623-9_288`.

[63] G. Pierra. “Decomposition Through Formalization in a Product Space”. In: *Mathematical Programming* 28 (Jan. 1984), pp. 96–115. DOI: `10.1007/BF02612715`.

[64] H. L. Resnikoff. *On Curves and Surfaces of Constant Width*. 2015. arXiv: `1504.06733 [math.DG]`.

[65] H. S. Shapiro. *The Schwarz Function and Its Generalization to Higher Dimensions*. John Wiley and Sons, 1992. ISBN: 978-0-471-57127-8.

[66] M. P. Skerritt. *Algebraic-PSLQ: Testing and Results*. Version Thesis/Submission. 2020. DOI: `10.5281/zenodo.3900580`.

[67] M. P. Skerritt and P. Vrbik. “Extending the PSLQ Algorithm to Algebraic Integer Relations”. In: *From Analysis to Visualization*. JBCC 2017. Ed. by B. Sims et al. Springer Proceedings in Mathematics & Statistics. Cham: Springer International Publishing, 2020, pp. 407–421. DOI: `10.1007/978-3-030-36568-4_26`.

[68] I. Stewart and D. Tall. *Algebraic Number Theory and Fermat’s Last Theorem*. 3rd ed. A K Peters, 2001. ISBN: 978-1-56881-119-2.

[69] A. Straub. *A Gentle Introduction to PSLQ*. 2010. URL: `http://arminstraub.com/math/pslq-intro`.