From Measurement to Symbolic Expression: GeoGebra as an Aid to Scientific Observations

Kai Chung Tam

December 15, 2012

Acknowledgments

The research study in Linz was a memorable experience in my life. It is a rewarding to work with experts and enthusiasts in the fields of mathematics, computer science, and mathematics education, which have long been my endeavors.

I would like to give special thanks to Professor Markus Hohenwarter, who have made this project possible and gave me a lot of advices within or beyond the work. For more than 10 years, GeoGebra has been helping the teachers and students communities. I so much appreciate your contribution by having this software initiated at first, and subsequently gather developers and interested people from all over the world.

I would like to thank the Austrian Marshall Plan Foundation for your enabling this honorable opportunity to further my small ideas and initial projects.

I would like to thank Mr. y Mahringer for all of your instructions and suggestions from the very beginning.

I would like to thank Zolt‘’an Kov‘’acs for your guidance from math and programming to life and spiritual thoughts.

I would also like to thank Michael, Philip, Simon, Alexander, and all colleagues. It was nice to meeting you all and working together as a team.

I would like to thank Martina and your family. You are very nice hosts and I have learned a lot about the country.

I would like to thank Dr Bruce Vogeli, Dr Henry Pollak of my home institution, Teachers College, Columbia Unviersity. My interests on mathematics education have been furthering in these years. Your expertise and experiences have had a large impact on my development of ideas on education.

I would like to also thank Dr Nii Nartey, who introduces GeoGebra along with other mathematics software in my first year in TC.

Last but not the least, I would like to thank my family and Haoqi, who have been supporting me wholeheartedly.

Abstract

This document is a report on my work at the Johannes Kepler Universität in Linz from May 26th, 2012 to August 25th, 2012, sponsored by Marshall Plan Scholarship. All of my work is related to the development of GeoGebra software, which is led by Professor Markus Hohenwarter at JKU.

This document demonstrates how the '`SurdText`' command was developed. This command is able to convert a decimal to a radical expression, using the PSLQ method originated from computational mathematics. The motivation of its development, and the implication on teaching and learning will also be discussed.

Contents

Chapter 1

SurdText Command

1.1 Motivation

In school mathematics, converting fraction to a decimal or doing its inverse is a standard topic. It is totally natural to ask for a way to convert a radical to a decimal or doing its inverse. For example, student might calculate the value of sine fifteen degrees using a scientific calculator and get:

$$0.2588190451$$

as a ten-digit approximation of the value; however, it is also worth knowing that:

$$\sin(15^\circ) = \frac{\sqrt{6}-\sqrt{2}}{4},$$

which is hardly implemented in any calculators. We may think of two possible ways to achieve this functionality. First, if we could have a specific collection of functions (such as trigonometric functions) and special arguments that we would like to obtain radical expressions as values instead of decimal expressions, we could try to get the answer in radical expression, symbolically. This will give us an exact result, but it does not work if we only are given the decimal value, not knowing from what function it is calculated. The second way is to "convert a decimal to a radical", or to be more precise, given a decimal expression x, and a tolerance of error $\epsilon > 0$, number of terms n, decide if there exists $r_0, r_1, r_2, \cdots r_n \in \mathbb{Q}$ and $k_1, k_2, \cdots k_n$ such that $|x_0 - x| < \epsilon$, where

$$x_0 = r_0 + r_1\sqrt{k_1} + r_2\sqrt{k_2} + \cdots + r_n\sqrt{k_n}.$$

Also, if it exists, find a minimal solution (in a certain sense). It turns out that this problem can be fitted into a more general problem, the problem of detecting integer relations. A vector $(x_1, x_2, \cdots, x_n)$ possesses an integer relation if and only if there exists n integers $a_1, a_2, \cdots a_n$, not all being zero, such that

$$a_1x_1 + a_2x_2 + \cdots a_nx_n = 0.$$

In principle, if this problem is solved, then one can detect whether x is an algebraic number of degree no larger than k by letting $n = k + 1$ and that $x_j = x^j$ from $j = 0, 1, \cdots k$. However, since there is no radical formula for any polynomial of degree higher than 4, other remedial operations should be adopted in order to get a satisfiable expression.

There are famous algorithms to solve the problem of detecting integer relations, such as the LLL algoritm and the PSLQ algorithm. In the following sections, I will introduce the functionality of `SurdText` Command, the PSLQ algorithm, the GeoGebra implementation, and some applications and educational implications of the `SurdText` command.

1.2 What SurdText does

In GeoGebra, the `SurdText` command accepts three types of argument(s)[1]:

- `SurdText[<Number x>]`
- `SurdText[<Number x>,<List L>]`
- `SurdText[<Point P>]`

The first one searches for integers $a, b, c, d \in [-100, 100]$ such that $x = \frac{a+b\sqrt{c}}{d}$. For example, `SurdText[1.618033988749895]` returns a textbox which displays $\frac{1+\sqrt{5}}{2}$ as its content. If such integers are not found, x will be returned unchanged. The second one also reads a list of given numbers $L = (L_1, L_2, \cdots, L_k)$, and searches for integers $a_1, a_2, \cdots, a_k$ such that $x = \sum_{1 \le j \le k} a_j L_j$. However, when L is an empty list, it is defined by default a list of common constants, i.e. $\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{6}, \pi$. The last one applies `SurdText` command to each coordinate of P.

Note that there is a fixed precision requirement for the input. For example, `SurdText[1.61803398874]` gives $\frac{1+\sqrt{5}}{2}$ but `SurdText[1.6180339887]` will

[1]Documentation on the GeoGebra wiki site: `http://wiki.geogebra.org/en/SurdText_Command`

leave the expression unchanged. One should also note that the `SurdText` command without a list is faster, not only because the former has only 1 square root expression, but also because the latter requires an exact method of computation, which will be discussed in Section 1.4.

The most important part in the implementation of `SurdText` command is the *PSLQ algorithm*, which finds an integer relationship of n real numbers.

1.3 PSLQ Algorithm, and mPSLQ

The heart of converting a raw decimal to any kind of symbolic, exact expression of numbers is to find an integer relation. For example, to prove that y is an algebraic number of degree k, it is to find a polynomial $p(t)$ of integer coefficient such that $p(y) = 0$. A prototype of integer relation algorithm is the Euclidean algorithm, since for any two real numbers x, y, the Euclidean algorithm being applied to them terminates in a finite number of steps if and only if x/y is rational. *LLL algorithm* (1986) is a classical algorithm of finding integer relations and also solves problems in a general lattice. For integer relations, however, a significantly faster and numerically stabler algorithm is the PSLQ, developed by the mathematician-sculptor Helaman Ferguson.

PSLQ stands for a "Polynomial-time algorithm, involving the Sum-of-square and LQ decomposition schemes". This was devised by mathematician-sculptor H. Ferguson (1991), and subsequently a modified version was developed (1999).

In the implementation of `SurdText` command, the key algorithm is PSLQ and *multiple PSLQ*, which will be explained as follows.

Given a real parameter τ such that $1 < \tau < 2$, PSLQ (of parameter τ) takes the following input:

- n, an integer ≥ 2;
- n real numbers: $x_1, x_2, \cdots, x_n \in \mathbb{R}$;
- $M > 0$, a positive real number which indicates the desired upper bound of the norm of an integer relation;
- A binary function $\zeta : \mathbb{R} \to \{0, 1\}$ that checks whether a number is zero.

And it provides the following output:

- First case: an $n \times n$ integer matrix B such that there exists j, $1 \leq j \leq n$ such that $\zeta((xB)_j) = 0$.

- Second case: an integer relation with norm no larger than M does not exist.

It is proven that if there exists an integer relation of norm less than M, then the algorithm finds an integer relation (whose norm is not necessarily less than M) in less than $k(n, \tau, M) = \binom{n}{2} \log_\tau(\gamma^{n-1}M)$ iterations, where γ is the positive real number such that $1/\tau = \sqrt{1/4 + 1/\gamma^2}$. Moreover, each iteration contains $O(n)$ exact arithmetic operations.

Notice that when PSLQ terminates, there might still be other integer relations exist that have norm less than M. A correct interpretation is the following: if no relation is found in $k(n, \tau, M)$ iterations, it is certain that there is no integer relation of x with norm less than M. If a relation m is found, it is not guaranteed that its norm is less than M[2]. If we are dedicated to find a relation with norm less than M, we may have to apply PSLQ multiple times. Also, for the purpose of implementation, we might not be satisfied to find just one integer relation, for we might need to choose the

[2]Nonetheless, it is guaranteed that $|m| \leq \gamma^{n-2} M_x$, where M_x is the smallest possible norm for an integer relation of x.

best relation in some sense (cf. Subsection 1.4.4). A note in section 6 of [4] briefly described how one can find all possible relations by applying PSLQ algorithm multiple times, but it was not detailed enough and also with some minor flaws [3]. In order to apply PSLQ multiple times and to get different integer relations each time, some careful adjustments to PSLQ are necessary. Also, in our implementation, the choice of τ was not very important so we may just take $\tau = 1.5$ in our convenience. In this case, $\gamma = 2.2678$ and

$$k(n, M) := k(n, 1.5, M) \approx n(n-1)((1.0097)(n-1) + \ln(M)(1.2332)).$$

The adjusted PSLQ takes $\tau = 1.5$ and provides the following output:

- An integer matrix B.
- An n dimensional, binary vector $u \in \{0, 1\}^n$, called the indicator vector, such that the j-th column of B is an integer relation with norm no larger than M if and only if $u_j = 1$, otherwise $u_j = 0$.

The multiple-PSLQ algorithm takes the same input as PSLQ, and provides as many integer relations as possible. First, apply PSLQ once and we get an $n \times n$ integer matrix B, and the indicator vector u. Suppose there are q ones

[3]For example, it mentioned that PSLQ would find one relation each time but actually there might be $q \geq 1$ relations found at one application of PSLQ, so one would need to take the $n - q$ instead of $n - 1$ non-zero coordinates in order to applied PSLQ once more.

and $n-q$ zeros in u. If $q=n$, then there are no desired integer relation; if $q<n$, let B_{n} be the $n \times q$ submatrix of B so that it contains all the q columns in B which correspond to 1 in u, and let B_{r} be the $n \times (n-q)$ submatrix of B that consists of the other columns. The subscripts n and r stand for *normal* and *the rest*, respectively. Then $xB_{\mathrm{n}} = 0_{1\times q}$. Let $y = xB_{\mathrm{r}}$, and we apply PSLQ for y (which is q dimensional). If we can find an integer relation $m \in \mathbb{Z}^q$ for y, then we have $0 = ym = xB_{\mathrm{r}}m$, thus $B_{\mathrm{r}}m$ is an integer relation for x. We will store all the column vectors to a list L, which is initially empty.

Iteratively, suppose we have applied PSLQ for a p dimensional vector y and we get the $p \times p$ matrix B and the p dimensional vector Ind, we let $q \leftarrow$ the number of ones in u, and arrange B to the two new matrices $\tilde{B}_{\mathrm{n}}$ ($p \times q$ dimensional), $\tilde{B}_{\mathrm{r}}$ ($p \times (p-q)$ dimensional) in the same way described above. Add to L all the column vectors of $B_{\mathrm{r}}\tilde{B}_{\mathrm{n}}$, which is a product of a $n \times p$ matrix and a $p \times q$ matrix. Then let $B_{\mathrm{r}} \leftarrow \tilde{B}_{\mathrm{r}}$, $p \leftarrow q$. The algorithm terminates when there is no new integer relation found. The following pseudo code summarizes mPSLQ:

Algorithm 1 mPSLQ

```
Require: a positive integer $n$, an $n$ dimensional real vector $x$; a positive
  number $M$; a binary-valued function $\zeta$
  $T \leftarrow \emptyset$, $y \leftarrow x$, $B_{\mathrm{r}} \leftarrow I_n$
  $p \leftarrow n$
  if $p > 0$ then
      $\tilde{B}_{\mathrm{r}} \leftarrow I_p$, $\tilde{B}_{\mathrm{n}} \leftarrow I_p$
      Apply PSLQ(1.5) to the $p$ dimensional vector $y$, and the bound $M$,
      and function $\zeta$ of checking zero, getting a $p \times p$ matrix $B$ and a $p$
      dimensional vector $u$.
      if $u = 0_p$ then
          return $T$
      end if
      $q \leftarrow 0$, $q' \leftarrow 0$ $i \leftarrow 0$
      for $i = 1..p$ do
          if $u_j = 1$ then
              $q \leftarrow q + 1$
              replace the $q$-th column of $\tilde{B}_{\mathrm{n}}$ with the $i$-th column of $B$
          else
              $q' \leftarrow q' + 1$
              replace the $q'$-th column of $\tilde{B}_{\mathrm{r}}$ with the $i$-th column of $B$
          end if
      end for
      $\tilde{B}_{\mathrm{n}} \leftarrow$ the first $q$ columns of $\tilde{B}_{\mathrm{n}}$
      Add all the columns of $B_{\mathrm{r}}\tilde{B}_{\mathrm{n}}$ to $T$
      $\tilde{B}_{\mathrm{r}} \leftarrow$ the first $q'$ columns of $\tilde{B}_{\mathrm{r}}$
      $B_{\mathrm{r}} \leftarrow B_{\mathrm{r}}\tilde{B}_{\mathrm{r}}$
  else
      return $T$
  end if
```

Figure 1.1: Helaman Ferguson at the Clay Mathematics Institute opening ceremony on May 10, 1999

1.4 Implementation

1.4.1 Types of expressions to be fit

It will be impractical to consider all possible algebraic expressions of real numbers. Even restricted in standard mathematics curricula, irrational expressions such as radical expressions, exponential and logarithmic expressions (in base 10, 2, or e), and trigonometric functions (with a heavy usage of the constant π) are common objects to be explored. Although there is no general method to handle all of these expressions, there are some special identities between these expressions that are commonly seen:

- $c = \sqrt{a^2 + b^2}$ in the Pythagorean Theorem;
- The quadratic formula, $\dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$;
- Rationalization, $\dfrac{1}{\sqrt{2} + \sqrt{3}} = \sqrt{3} - \sqrt{2}$;
- $\sin(60^\circ) = \cos(30^\circ) = \dfrac{\sqrt{3}}{2}$;
- $\arcsin(0.5) = \dfrac{\pi}{6}$;
- $\log_{10} 5 + \log_{10} 2 = 1$.

Note that logarithm is less “geometric”, so it is less used in GeoGebra. Unless we are going to handle a product of powers (cf. Section 1.6), we will not consider logarithm in the implementation. Nonetheless, quadratic roots and constant π are very geometric concepts and therefore we shall bring them into the implementation of `SurdText` command.

By an inspection of the listing above, we are going to consider the following types of expressions: (1) Quadratic, i.e. $x = (a + b\sqrt{c})/d$ with c, d that are not large in magnitude; (2) A linear combination of square roots of integers; (3) $a + b\pi$; (4) Linear combination of a given list of constants.

Quadratics.

Suppose $t \in \mathbb{R}$ and that we look for parameters $a, b, c, d \in \mathbb{Z}$ such that $t = (a + b\sqrt{c})/d$. Then we have

$$(dt - a)^2 = b^2c$$

$$d^2t^2 - 2adt + a^2 - b^2c = 0$$

Therefore $(t^2, t, 1)$ has an integer relation $(d^2, -2ad, a^2 - b^2c)$. Conversely, if (A, B, C) is an integer relation of $(t^2, t, 1)$, then $t = (-B \pm \sqrt{B^2 - 4AC})/(2A)$, so a,b,c,d can be found. We may also reduce the radical according to simplification rules. In GeoGebra, the CAS (computer algebra system) is able to simplify radical expressions, therefore it is used in the implementation.

Linear Combination of surds.

Combination of square roots is tricky. Suppose $t = \sqrt{A} + \sqrt{B}$, and we look for the parameters A, B. It is possible to treat t as algebraic number which is a root of minimal polynomial $p(y)$ has a factor $(y - t)$ and thus a factor $(y - \bar{t})$, where $\bar{t}$ is any of t's conjugates. One can deduce that $p(y) = (y^2 - A - B)^2 - 4AB = y^4 - 2(A + B)y^2 + (A - B)^2$, which is quartic but not a general one, since there are no odd powers of y. Therefore one may first find a relation (m_1, m_2, m_3) for $x = (t^4, -2t^2, 1)$. If m_3 is a perfect square, then we have revealed that

$$t = \sqrt{\frac{m_1 + \sqrt{m_3}}{2}} + \sqrt{\frac{m_1 - \sqrt{m_3}}{2}}.$$

In general, if t is a combination of k (irreducible) radicals, the polynomial $p(y)$ will be a 2^k-degree polynomial without odd terms, or $n = 2^{k-1} + 1$ coefficients to be determined. Recall that the PSLQ algorithm requires nd digits in solving integer relations of n dimension and size of d digits. If we set the bound as $M = 100$ (so $d \leq 2$), we need $2n$ digits in the input. Since GeoGebra inputs have around 12 to 15 digits, only $n \leq 6$ or 7 would be practical. Therefore if t is a combination of $k \geq 4$ square roots than this method will not work. Moreover, when $k = 3$, even if one has found $p(y)$ of degree 8, it is quite hard[4] to find a general way to reveal the expression of a root t. For example, if $t = \sqrt{2} + \sqrt{3} + \sqrt{5}$ then $p(y) = y^8 - 40y^6 + 352y^4 - 960y^2 + 576$. There could be some number-theoretic method in solving it[5], but it is beyond of scope for the project.

At this stage, linear combination of surds is not implemented. Nonetheless, it is partially fulfilled by the next type of expression: linear combination of given constants.

[4]Actually $p(y)$ is a quartic of y^2, but a quartic formula does not solve the problem at once because it will be another time-consuming task to simplify!

[5]e.g. noticing that if $A = 2$, $B = 3$, and $C = 5$, then $(A^2+B^2+C^2-2AB-2BC-2CA)^2$ must be equal to the constant term, which is $576 = 24^2$

Linear Combination of Given Constants.

Suppose that t is a linear combination of a given set of constants $\{C_1 = 1, C_2, \cdots, C_k\}$, and the coefficients are rational. Then $a_0 t + a_1 C_1 + a_2 C_2 + \cdots + a_k C_k = 0$, where $a_0, a_1, \cdots, a_k$ are some integer parameters. Therefore, if we can find a relation for $(t, C_1, C_2, \cdots, C_k)$, then we have revealed that

$$t = -\frac{a_1}{a_0} C_1 - \cdots - \frac{a_k}{a_0} C_k.$$

In this report, only the first and the third type are considered. As there could be many other types of expressions to be fitted, I would let each case above an instance of (parametric) algebraic fitting.

Definition. Given a computable function F which is defined on (part of) $\mathbb{Z}^k$ and takes value in $\mathbb{R}$. The problem of *parametric algebraic fitting* is to construct an algorithm such that, for any given input $\epsilon > 0$, $m > 0$, and $t \in \mathbb{R}$, it returns as many k-tuples $\alpha = (a_1, a_2, \cdots, a_k)$ as possible such that $|a_j| \leq m$, and that $|t - F(\alpha)| < \epsilon$.

Under this definition, the first and the third cases are named and specified by a certain function as follows.

Quadratic fitting:

$$F_1(a,b,c,d) = \frac{a+b\sqrt{c}}{d};$$

Constants fitting:

$$F_3(a_0,a_1,\cdots,a_k) = \frac{a_1C_1+\cdots a_kC_k}{a_0}.$$

1.4.2 Bounds of Parameters

Recall that when the PSLQ algorithm terminates in $k(n,M)$ steps, it means that any relation with norm less than M is found. However, the bound M is quite different from the bound L for the parameters. For quadratic fitting, suppose that we have found a relation for $(1,t,t^2)$ with norm less than M. Then we have

$$d^2 \le M, |2ad| \le M, |a^2-b^2c| \le M.$$

In order that *all* possible fittings within the bound L of parameters can be found, one needs to guarantee that the largest possible of $d^2, |2ad|$, and $|a^2-b^2c|$ are not greater than M. It can be the case that $a=0$ and $b=c=L$, so $L^3 \leq M$. To guarantee that all quadratic fittings within the bound $L=10$ are found, one should set $M=1000$. In this case $k(n,M)=k(3,1000)=63.23$. Even if we want $L=100$ and need $M=10^6$, the number of steps will only be $k(3,10^6)=114.2$.

For constants fitting, M is just the largest possible a_j. Therefore if we want $L=100$ than we may just take $M=100$.

1.4.3 Accuracy and Precision

There are three (types of) questions about accuracy: Precision of the input, accuracy of the integer relations, and working precision.

Precision of input, or ϵ.

First, what ϵ should we choose for the problem? In other words, what is the acceptable "error" of $F(\alpha)$? Since the amount of error is not customized, it is determined by the input. If the input has d digits after the decimal point, the answer $F(\alpha)$ should have the same value if rounded to d digits, therefore $|F(\alpha)-d| \leq 0.5\times 10^{-d}$. However, one should notice that GeoGebra does not keep track of the number of digits actually typed in by the user. The internal precision is about 16 digits, so, for example, when a user typed `SurdText[1.4142135624]`, the number `1.4142135624` will be equivalent to `1.4142135624000000`, which is obviously not equal to the rounded value of $\sqrt{2}$.

Practically, we accept the accuracy of the display of a scientific calculator, which has at least 10 digits, thus we choose $\epsilon = 0.5 \times 10^{-10}$ in our implementation, and `SurdText[1.4142135624]` returns $\sqrt{2}$.

Working precision.

As mentioned in Ferguson (1999), the properties of PSLQ hold was proven in the assumption that all the arithmetic operations in the iterative steps are exact, including addition, subtraction, multiplication, integer part, and comparison of real numbers. Therefore, the implementation also ideally does the exact arithmetic in the iterative steps. Division and square root operations are involved in the initialization steps, but since they are to be done before any iterative steps, they are only done once and so therefore will not affect the performance significantly.

For our purpose, exact arithmetic was not used, and there are two different implementations: one for quadratic fitting and another for constants fitting. Double-precision (64-bit) floating point was used for both initialization and iterations. This implementation is fast, and quite sufficient for revealing quadratic expressions of small parameters (less than 20) because there are only $n = 4$ dimensions. For constants fitting, we use double-precision for initialization and multi-precision or iterative steps (i.e. more than nd digits, where n is the dimension of the input x, and d is the precision of the input).

Accuracy of the integer relations, related to ϵ.

Last but not the least, each time when an integer relation m is found for x, we may check if xm^T is small enough, namely $|xm^T| < \delta$ for a given $\delta > 0$. In any algebraic fitting problem, this value δ should be chosen so that we will not miss any $F(\alpha)$ which satisfies $|F(\alpha) - t| < \epsilon$. Of course it is inevitable to include some *false* relations m, in the sense that $|xm^T| < \delta$ holds but

$|F(\alpha)-t| \geq \epsilon$. The best choice here is the minimal δ such that $|F(\alpha)-t| < \epsilon$ implies $|xm^T| < \delta$. We have the following results:

Lemma.

1. For quadratic fitting, $2L^{5/2} + L^2\epsilon$ and $3M\epsilon$ are upper bounds for δ.

2. For constants fitting, $L\epsilon$ is an upper bound for δ.

Proof. For quadratic fitting, suppose that $0 < \epsilon < 1$, and that $t_0 = (a_0 + b_0\sqrt{c_0})/d_0$ is the *true* expression, where a_0, b_0, c_0, d_0 are integers, and $c, d > 0$. Suppose also that $t_0 = (a+b\sqrt{c})/d$ satisfies $|t-t_0| < \epsilon$, where $-L \leq a, b \leq L$, $0 \leq c, d \leq L$, $L \geq 1$. Then,

$$\begin{aligned}
\left|\frac{a+b\sqrt{c}}{d} - t\right| < \epsilon \quad &\Rightarrow \quad -d\epsilon < a + b\sqrt{c} - dt < d\epsilon \\
&\Rightarrow \quad -b\sqrt{c} - d\epsilon < a - dt < -b\sqrt{c} + d\epsilon \\
&\Rightarrow \quad b\sqrt{c} - d\epsilon < dt - a < b\sqrt{c} + d\epsilon
\end{aligned}$$

Therefore, no matter $a > dt$ or $a < dt$, we have $|a - dt| \leq |b|\sqrt{c} + d\epsilon$, or squaring it to get $(a-dt)^2 \le b^2c + |2bd\sqrt{c}|\epsilon + d^2\epsilon^2$. So,

$$\begin{aligned}
|xm^T| &= \left| \begin{pmatrix} t^2 & t & 1 \end{pmatrix} \begin{pmatrix} d^2 & -2ad & a^2-b^2c \end{pmatrix}^T \right| \\
&= |d^2t^2 - 2adt + a^2 - b^2c| = |(dt-a)^2 - b^2c| \\
&\le |2bd\sqrt{c}|\epsilon + |d^2\epsilon^2| \\
&\le (2L^{5/2} + L^2)\epsilon \le 3L^3\epsilon \le 3M\epsilon.
\end{aligned}$$

We used $L^3 \le M$ as indicated in Subsection 1.4.2. For constants fitting, suppose that $t = \sum a_jC_j/a_0$, $t' = (\sum a'_jC_j)/a'_0$, and that $|t-t'| < \epsilon$. Then

$$\begin{aligned}
|xm^T| &= \left| \begin{pmatrix} t' & C_1 & \cdots & C_k \end{pmatrix} \begin{pmatrix} a'_0 & -a'_1 & \cdots & -a'_k \end{pmatrix}^T \right| \\
&= \left| a'_0t - \sum a'_jC_j \right| = |a'_0t - a'_0t'| = |a'_0||t-t'| < L\epsilon \quad \square
\end{aligned}$$

Again, one should take these results in Subsection 1.4.2 and this subsection with care that, when we choose the specified bound L and accuracy δ, all solutions $F(\alpha) = t'$ satisfying $|\alpha_j| \le L$ and $|t-t'| < \epsilon$ will be found, but there could also be other undesired relations. In the following subsection, we discuss the possible cases when there are more than one integer relations, and explain by examples constructed during the project.

1.4.4 More than one integer relations

For a list of real numbers $x \in \mathbb{R}^n$, it is possible that there exist more than one integer relations $m \in \mathbb{Z}^n$ that satisfy the norm condition ($mm^T \leq M$) and the accuracy condition ($|xm^T| \leq \delta$). There may be two main reasons for this to happen. First, it may be symbolically true that $x^\perp$ contains more than one independent vectors in integers, e.g. $x = (\sqrt{2}, 2\sqrt{2}, -\sqrt{2})$ has independent integer relations $(0, 1, 2)$ and $(1, 0, 1)$. Another possibility is that a relation (usually of a large norm) is not symbolically correct but satisfies the accuracy condition. For example, $x = (1, \pi^3, \pi/500)$ "almost" has an integer relation $(-31, 1, -1)$, for $-31 + \pi^3 - \pi/500 = -6.5 \times 10^{-6}$. Another example generated from convergents of $\sqrt{2}$ is $x = (1970/1393, \sqrt{2})$, which "almost" has the relation $(1, -1)$, since $|1970/1393 - \sqrt{2}| < 10^{-6}$. Although these seems quite rare in practice, it may be more often for a larger n. Suppose $n = 6$ and $x = (1, \sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{6}, \pi)$. One may check the following:

$$\alpha = 29\sqrt{2} - 7\sqrt{3} + 14\sqrt{6} - \pi = 60.039101401\ldots$$

$$\beta = -5 + 8\sqrt{2} - 2\sqrt{3} + 10\sqrt{5} + 45\sqrt{6} - 24\pi = 60.039101397\ldots$$

Therefore $m = (5, 21, -5, -10, -31, 23)$ is an integer vector such that $|xm^T| = |\alpha - \beta| = 4 \times 10^{-9}$.

Given $\epsilon > 0$, if we would like to find a symbolic expression $t' = F(\alpha)$ that represents $t \in \mathbb{R}$, and that $|t' - t| < \epsilon$, we should specify the norm of any integer relation to be found and the precision requirement during the computation. The norm condition is important because it is always possible to find convergents $(p_k/q_k)_{k\leq 1}$ of a real number α such that $p_k/q_k \to \alpha$.

Efficiency issues in avoiding undesired integer relations.

As mentioned before, when PSLQ terminates, the resulting matrix might not contain any desirable integer relation. Recall that ϵ is the precision of the input t, L is the bound of parameters (t and L are the *direct* requirements in the problem of SurdText), M depending on L is the bound of coefficients for mPSLQ, δ depending on ϵ and L is the accuracy requirement for mPSLQ. In a more rigorous treatment, one should use large enough M and δ to get a collection of relations which contains all desirable ones, then use the direct requirements ϵ and L to rule out all the undesired integer relations. This will be slow in general. Although theoretically there will be no more than $n-1$ integer relations for an n dimensional vector x, it is different in the implementation because we allow certain error, as shown in the examples above. There can be a lot more than $n-1$ tentative solutions to the problem, and that is why it may be slow if mPSLQ is dedicated to find all those solutions.

A balance between correctness and efficiency can be obtained by setting a smaller M and/or δ in the mPSLQ. Also, if one needs even faster implementation, we can use PSLQ only once, hoping that what it finds is the desired one. For example, although we proved previously that we should choose $M = 1000$ and $\delta = 3M\epsilon$ in quadratic fitting that requires $m = 10$, one should realize that $\delta = 3 \times 10^{-7}$ when $\epsilon = 10^{-10}$, and a one-time PSLQ would probably not provide the most desired integer relation. We may then choose $\delta = 10\epsilon = 10^{-9}$ to get a better chance of getting the desired relation, at the cost of the higher possibility that no desired relation is found even if there is one.

This more efficient but less correct implementation is used in quadratic fitting only. For constants fitting, since there are even more possible relations to be found, a one-time PSLQ will not work. Instead, the mPSLQ is applied to get a collection of relations, and then we apply certain rules to choose the best solution. This will be explained in the next section.

Choosing the "best" out of the "nice" solutions.

We have explained in the mPSLQ algorithm how to apply PSLQ multiple times to get as many integer relations as possible, and those relations are stored in a list T. It is also easy to rule out all relations m not satisfying $mm^T \leq M$ and $|xm^T| \leq \delta$. There can still be many solutions[6].

Although there is a concept called *significance*, introduced in Ferguson (1999), to describe the "fitness" of a relation m, it is not suitable for our purpose. In particular, the significance is defined by "the ratio between the multiprecision epsilon and the largest entry of the updated x vector when a relation is recovered". This can be used as a rule to avoid unacceptable relations, but when two relations are significant, we can not judge that one is better than the other.

Similarly, we cannot use the error $|t' - t|$ as a measure of the goodness of fit. After all, when the input is $t = 1.4142135624$, any answer that has value in

[6]As observed, there can be hundreds of "tentative solutions" for n just equal to six.

the interval $[t - 0.5 \times 10^{-10}, t + 0.5 \times 10^{-10}] = [1.41421356235, 1.41421356245]$ is equally probable. Indeed, we need measures that depend on the several features of the relation, for example, the number of non-zero terms. I use the *penalty* approach to rank the solutions.

For constants fitting, we define seven characteristics and the corresponding penalty score of a relation m. Each penalty score is a function p_j, taking a relation m as input and an integer as output. We compare the scores using a dictionary order, namely, a relation m_1 is ranked better than m_2 if and only if there exists j_0, such that $p_j(m_1) = p_j(m_2)$ for all $j < j_0$, and that $p_{j_0}(m_1) < p_{j_0}(m_2)$. Here are the definitions of p_j in constants fitting:

1. Zero-denominator: Is a_0 equal to zero? If so, set $p_1(m) = 1$; otherwise $p_1(m) = 0$.

2. Out-of-bound: If there is a j such that $|a_j| > L$, then $p_2(m) = 1$, otherwise $p_2(m) = 0$.

3. Number of irrational constants used: $p_3(m)$ is equal to the number of j's such that $1 \leq j < k$, $a_j \neq 0$.

4. Sum of coefficients: $p_4(m) = \max\{\ \sum |a_j|, L + 1\ \}$.

5. Non-algebraic: If there is a j such that $a_j \neq 0$ and that C_j is not algebraic (this is indicated manually), then $p_5(m) = 1$, otherwise $p_5(m) = 0$.

6. Non-homogeneity: If $a_k \neq 0$ then $p_6(m) = 1$, otherwise $p_6(m) = 0$.

7. Index: $p_7(m)$ is equal to the opposite of the position of m in T.

p_1 and p_2 are merely the basic requirements of the parameters. Theoretically they should all be 0 but they are defined for the purpose of testing.

Now, suppose that $k = 6$ and $(C_1, \cdots, C_6) = (\pi, \sqrt{6}, \sqrt{5}, \sqrt{3}, \sqrt{2}, 1)$. p_3 essentially looks at the number of irrational terms, so $a_1\pi$ is better than $a_4\sqrt{3} + a_5\sqrt{2}$, $(a_6 + a_2\sqrt{6})/a_0$.

If two relations have the same number of irrational terms, p_4 looks at the absolute sum of the two relations. It is quite rare to have two relations that have equal absolute sum. If it happens, then p_5, p_6 looks at the use of π and 1, respectively. Lastly, if p_1 up to p_6 can not distinguish two relations (this is possible theoretically, but really rare), then we take the one which comes earlier in the list T.

We may take the examples mentioned earlier in this subsection to see how they are ranked:

$(\sqrt{2}, 2\sqrt{2}, -\sqrt{2})$ has relation $m_1 = (0, 1, 2)$ and $m_2 = (1, 0, 1)$. One can see that p_1 up to p_3 are the same, but $p_4(m_1) = 3 > 2 = p_4(m_2)$, so m_2 is ranked as the "better" relation here.

Let $t = 0.6003910140$. We know that

$$\frac{\alpha}{100} = \frac{29\sqrt{2} - 7\sqrt{3} + 14\sqrt{6} - \pi}{100} \approx t,$$

and that

$$\frac{\beta}{100} = \frac{-5 + 8\sqrt{2} - 2\sqrt{3} + 10\sqrt{5} + 45\sqrt{6} - 24\pi}{100} \approx t.$$

Furthermore, $|\frac{\alpha}{100} - t|$, $|\frac{\beta}{100} - t|$ are both less than $\epsilon = 10^{-10}$, so both are acceptable expressions of t. In terms of integer relations, let $m_1 = (100, 1, -14, 0, 7, -29, 0)$ and $m_2 = (100, 24, -45, -10, 2, -8, 5)$. The penalty p_3 distinguishes them because m_1 uses only three irrational numbers but m_2 uses all five. Therefore m_1 is better.

Good side of type I error.

Although there might be different possible way to fit a number t, we have already chosen the "best" one according to the penalty rules. Also, if a relation is not found, then the original number will be returned (in decimal form). As a simplistic example, assume that $\epsilon = 10^{-5}$. What is the "correct" return value of `SurdText[0.32222]`? Both $\frac{29}{90}$ and $\frac{17-4\sqrt{13}}{8}$ have the same value when truncated to five decimal places, do don't be over-surprised that `SurdText[(17-4sqrt(13))/8]` returns $\frac{29}{90}$, according to the penalty rules.

In a specific problem of algebraic fitting (specified by $F(\alpha)$ where α is the parameter), we have got a function σ_F from finite decimals to the set of all possible α's union the set of all possible decimals. Note that this is not yet a function defined on all real numbers, although it may be very promising that this function can be extended to a function from $\mathbb{R}$ to $\mathbb{R}$.

For any integer d, We may also have the truncation function τ_d defined on the set of real numbers, such that $\tau_d(t)$ is equal to the largest number that is a multiple of 10^{-d} and not greater than t. For example, $\tau_0(t)$ is just the integer part of t. Of course the values of τ_d are in the set of all decimals.

It is true that $\tau_d(\sigma_F(t)) = \tau_d(t)$ for all decimal numbers t because $|\sigma_F(t)-t| < 0.5 \times 10^{-d}$. In particular, if t is a decimal number of no more than d digits, then $\tau_d(\sigma_F(t)) = t$. So, in some sense, τ_d is an inverse of σ_F, and it might also be desirable that $\sigma_F(\tau_d(t)) = t$. Indeed, when the algebraic fitting *failed* to find the parameters, since in this case it returns the same number as the input: $\sigma_F(\tau_d(t)) = \tau_d(t)$, which is equal to t only when it has less than d decimal digits. When it is not equal to t, we say that a type II error (i.e. false negative) occurs, because it is failed to find a relation when there is one. On the other hand, when some parameters are found, it may be a "correct" result (e.g. `SurdText[sqrt(2)]` returns $\sqrt{2}$), or it can be a type one error (i.e. false positive). A "false positive" is not bad, since we are really looking for such positives (e.g. `SurdText[1.6180339887]` returns $\frac{1+\sqrt{5}}{2}$). Even if we get something different from the input, the returned result is quite educational. For example, `SurdText[1/(sqrt(2)-1)]` returns $1 + \sqrt{2}$, which is a well-known result of rationalization. Another example is

```
SurdText[(-5+8sqrt(2)-2sqrt(3)+10sqrt(5)+45sqrt(6)-24pi)/10,{}],
```

which returns

$$\frac{20\sqrt{5} - 57\sqrt{6} - 12\sqrt{2} + 92\pi + 63}{40}.$$

They are both equal to 6.0039101398 when truncated at $d = 10$ digits, but the latter one contains only five terms instead of six.

1.4.5 Summary: objects related to SurdText command

The issues discussed above are all considered in the actual program written in Java. The program mostly in the file `AlgoSurdText.java`. For more information about how to add a command in GeoGebra, one may visit the development website of GeoGeobra, `http://dev.geogebra.org`.

The file describes the `AlgoSurdText` class, some methods of it, and some subclasses. The major properties and methods of `AlgoSurdText` class are listed below:

- Properties: `GeoNumeric num`, `GeoList list`, `GeoText text`
- Constructor methods
- A `compute()` method
- `int[] PSLQ(double[], double, int)` method
- `int[] PSLQ(int, double[], double, int, int[][], double[], int[])` method
- `int[][] mPSLQ(int, double[], double, int)` method

- `void PSLQappendQuadratic(StringBuilder, double, StringTemplate)` method
- `void PSLQappendGeneral(StringBuilder, double, StringTemplate)` method

And the following are two subclasses of `AlgoSurdText`:

- `IntRelationFinder` subclass
  - Constructor methods
  - `IntRelation` subclass
  - `void initialize_full()` method
  - `void hermiteReduction()` method
- `AlgebraicFit` subclass
  - Constructor methods
  - `void computeConstant(double)` method
  - `void computeRational(double)` method

  – `void computeQuadratic(double)` method

These methods and subclasses supports algebraic fitting in two different fashion, which will be described below.

A quick implementation.

This is applied to `SurdText[ number ]`. When there is an input t (e.g. $t = 11.61803398874989$), the kernel of GeoGebra initializes the `AlgoSurdText` object, with `num` assigned as t in double-precision. Then the `compute()` method is invoked. First, it is checked if it is equal to zero according to the system's precisin (`10E-12`). If it is, then the number t will be returned as an output text. If not, then `PSLQappendQuadratic` method is called.

The `PSLQappendQuadratic` method computes $x = (t^2, t, 1)$ using double-precision and send x to the `PSLQ` method, using the following call:

```
PSLQ(x, 1E-10 * 30, 1000),
```

where `1000` is the bound M, and `1E-10 *30` is the δ which we only take 30ϵ instead of $3M\epsilon$. The PSLQ method is implemented again in double-precision, and it returns a list of coefficients (`coeff[0]`, `coeff[1]`, `coeff[2]`) that is a possible integer relation, or returns `null` if no relation is found. In the latter case, the original number t will be returned; in the former case, we check if:

- All coefficients are zero. In this case it is not considered as a legitimate relation, so t is returned.
- Some coefficients are greater than 100. We also return t.
- `coeff[0] == 0`. In this case t is equal to the fraction $-\frac{\text{coeff[2]}}{\text{coeff[1]}}$.
- None of the above happens, so `coeff[0] != 0`, and t can be expressed by the quadratic formula.

In the last case, `PSLQappendQuadratic` method also reduces the radicals by trying to divide squares from `coeff[1]*coeff[1] - 4*coeff[0]*coeff[2]`, and to reduce the fractions using euclidean algorithm. Finally, it returns the latex from of the result. In our example, the result will be

```
\frac{21+\sqrt{5}}{2},
```

which will be displayed as a text label in the Graphics view of GeoGebra. Note that one may also type the command in the CAS view and receive the output from the CAS.

SurdText with a List.

This is applied to `SurdText[ number, list ]`. Again, the kernel of GeoGebra initializes the `AlgoSurdText` object, with `num` assigned as t in double-precision, and the `compute()` method is invoked, which checks zero first. If t is not zero, the `PSLQappendGeneral` method is called, which is much more complicated than `PSLQappendQuadratic`.

First, it will test if it is a fraction with numerator and denominator less than 1000. For this purpose, an `AlgebraicFitter` object is initialized with the type `AlgebraicFittingType.RATIONAL_NUMBER` and the bound set to be `1000`. Then call the `compute(t)` for this object. It will send t to `AlgebraicFittingType.computeRationalNumber(t)`. Then `mPSLQ` is called to work on $x = (t, 1)$ with bound 1000 and precision 10^{-12} (the internal precision of GeoGebra). If the rational number fitting succeed, it will just return the fraction. Otherwise, it considers the constants fitting.

For example, suppose user inputs `SurdText[3.1415926535898,{}]`, so $t = 3.1415926535898$. Then `mPSLQ` in this step returns `null`, since it cannot find integers $a, b \in [-1000, 1000]$ such that $|at + b| < \delta$. So the rational number fitting was failed, and it sends the number t, a list of number values `testValues`, and the "names" of those values `testNames`, to the method `AlgebraicFittingType.fitLinearComb`. If a list was not provided, then a default list will be generated:

```
testValues = new double[] {Math.sqrt(2.0), Math.sqrt(3.0), Math.sqrt(5.0), Math.sqrt(6.0), Math.PI};

testNames = new String[] {"sqrt(2)", "sqrt(3)","sqrt(5)", "sqrt(6)", "pi"};
```

If the list is provided, then it checks if they are within the following:

If not, then they are omitted. (Thank Michael Borcherds for implementing this part.)

An `AlgebraicFit` object is initialized, with this value set and name set, and bound $L = 100$. Note that the type of fitting is `LINEAR_COMBINATION`. After that `AlgebraicFit.compute()` is called, and the number t is sent to `AlgebraicFit.computeConstant(t)`. This time, `mPSLQ` method is called.

Since we need to store the integer relations and compare them, `IntRelationFinder` and `IntRelation` are the objects that deal with these issues. To deal with higher precision or even exact arithmetic, we adapted the `java.math.BigDecimal` package and extended it to two classes `MyDecimal` and `MyDecimalMatrix` to deal with decimal number type with arbitrarily defined precision, and matrices that are defined and calculated with entries of `MyDecimal` type. The steps mentioned in the PSLQ algorithm are done by higher precision arithmetic provided by these two classes. The basic difference between the fast PSLQ and the more accurate version here is not just the precision, but also the fact that the PSLQ algorithm in this case do NOT terminate even if a relation is found, since the found relation can be false and thus has a large norm. It terminates after $k(n, M, \tau)$ steps.

After mPSLQ is done, a two dimensional array will be returned; each column is a possible relation for x. The `computeConstant(t)` method continue to rank the solutions and finds the best one, if there are solutions. A formal solution will be generated, which is in the format that is acceptable by CAS. We borrow the `evaluateGeoGebraCAS` method from the GeoGebraCAS to simplify the result, then use `GeoText.setLaTeX` method to convert it into LaTeX format.

Table 1.1: Examples of SurdText command

`SurdText[number]`	result	`SurdText[number, list]`	result
`Surdtext[10.2]`	$\frac{51}{5}$	`Surdtext[10.1]`	$\frac{101}{10}$
`Surdtext[30.1]`	30.1	`SurdText[30.0000001,{}]`	$60\sqrt{5}+30\sqrt{3}+\sqrt{2}-25\pi-79$
`Surdtext[sqrt(2)]`	$\sqrt{2}$	`Surdtext[sqrt(2),{}]`	$\sqrt{2}$
`Surdtext[ (1+2sqrt(3)) /(2-sqrt(3))]`	$8+5\sqrt{3}$	`Surdtext[ 1/(sqrt(3)+sqrt(7)), {sqrt(3),sqrt(7)} ]`	$\frac{\sqrt{7}-\sqrt{3}}{4}$
`Surdtext[6.283185307]`	6.28319	`Surdtext[ 6.283185307,{}]`	6.28319
		`Surdtext[ 6.2831853072, {}]`	2π
`Surdtext[2^(1/3)]`	1.25992	`Surdtext[2^(1/3),{}]`	1.25992
`Surdtext[ sqrt(3+sqrt(8))]`	$1+\sqrt{2}$	`surdText[ sqrt(5+2sqrt(6)),{}]`	$\sqrt{3}+\sqrt{2}$
		`Surdtext[ 1/(sqrt(2)+sqrt(3)-1) ,{}]`	$\frac{\sqrt{6}+\sqrt{2}-2}{4}$
`Surdtext[sqrt(1682)]`	41.401219	`Surdtext[sqrt(1682),{}]`	$29\sqrt{2}$
`surdtext[ sqrt(2)+sqrt(3)]`	3.14626	`surdtext[ (sqrt(2)+sqrt(3))^3,{}]`	$9\sqrt{3}+11\sqrt{2}$

A list of Examples.

1.4.6 Problems and Limitations

Return Type.

In our implementation, the return type is `GeoText`, which is a text box floating on the graphic view, and we cannot do any further manipulation.

We can also use the `SurdText` command in the CAS view, but still cannot do anything further from the returned expression. This is the part that should be easily fixed and should be very useful. For example, one may want to multiply the expression $(\sqrt{2}+1)$ to itself to see that it is still in the form of $a+b\sqrt{2}$, and see how a and b change.

Problems with constants fitting.

In the constants fitting, we wish to recover any possible relation, namely, we should find one if there exists one. However, this is sometimes not true (false negative), especially for larger coefficients. The problem lies in the theoretic property that we can exhaust the *exact* relations, i.e. $\delta = 0$. There are at most $n-1$ of them. What is practically implemented here is not for exact relations but approximate relations. There are a lot more of them (but not infinite, since we always restrict the coefficients to be integer between $-M$ and M.)

We already use a remedy to decrease the frequency of false negative, i.e. even we have got a relation by PSLQ, we still let the iteration continue. In this way we can collect much more relations. However, their might still be overlooked ones, because we can only choose one matrix B to proceed to another PSLQ.

Customizable Precision

As mentioned, we can only handle a fixed precision ϵ. In practice, we may need a customizable precision. For example, mathematics teacher might use 1.4142 as a satisfiable approximation of $\sqrt{2}$, and 3.1416 for π, and so forth. Four digits are usually enough in most usage in high school. After all, one will be confident to say that, if an answer is evaluated as 6.2832, it is over

95 percent sure that it is actually the truncated 2π. This is just a scenario that a precision $\epsilon = 0.5 \times 10^{-4}$ may be enough, and so are other choices of ϵ.

However, this will generate other problems because a larger ϵ implies a larger δ, and as mentioned before, there will be a lot more tentative relations to be chosen. Therefore it might not be the wisest solution to dig into PSLQ for a relatively large ϵ. It is possible to do some "pre-screening" such as what we have done in the implementation (i.e. test if it is a fraction of small denominator). So, we might restrict our search to $a + b\sqrt{c}$ for small a, b, and very commonly used c's such as $c = 2, 3, 5, 6$.

Correctness Proofs.

Since the proofs in the classical paper are for $\delta = 0$ cases, we need to have proofs for the approximations ($\delta > 0$) if we are to prove that the algorithm is correct (or not). It seems from the experiments that it is not completely sure that PSLQ can find all the solutions, but a probabilistic statement may also be useful. What is the probability that all solutions are found, if there is one? (This is one minus the conditional probability of false negative given there is an integer relation.)

1.5 Applications and Implications

1.5.1 Exploring special values on the arc.

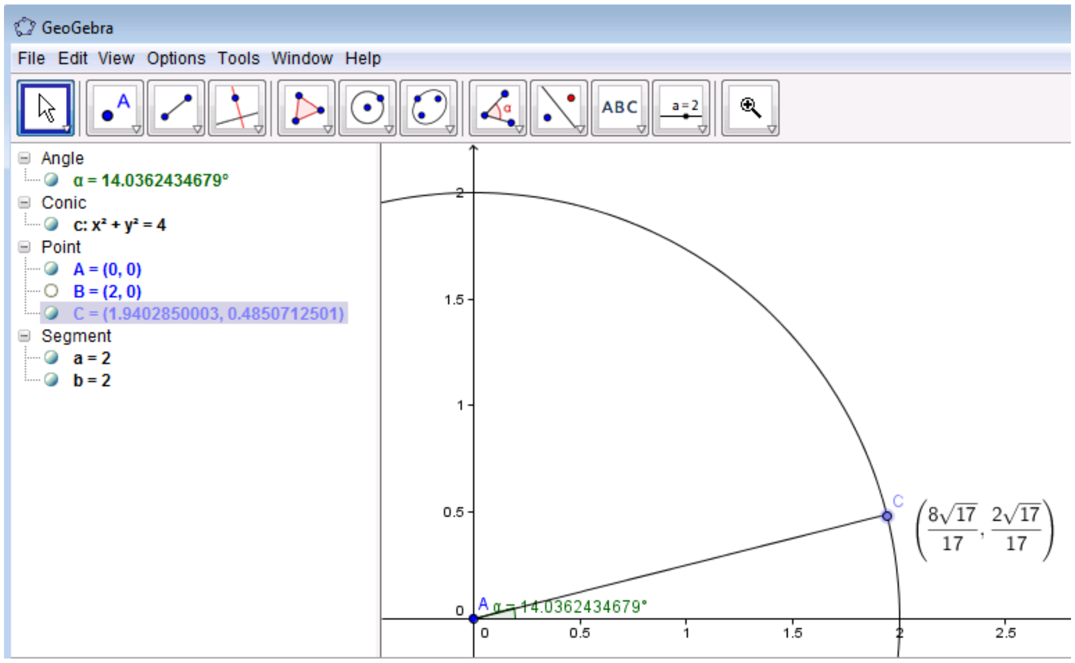
Let O be a circle described by $x^2 + y^2 = 4$, and C is a point on its circumference, and one may move this point along the arc. In the input bar, type the command `SurdText[C]` to get possible radical expression of point C.

```
Input: surdtext[C]
```

We are going to explore when both coordinates of C are radical expressions. (See the figures on the next page.)

In GeoGebra, when the user drags a point, the mouse location can be "snapped" to the lattice points (i.e. points with integer coordinates). Students may find that when this happens, both coordinates of the point becomes radical expressions.

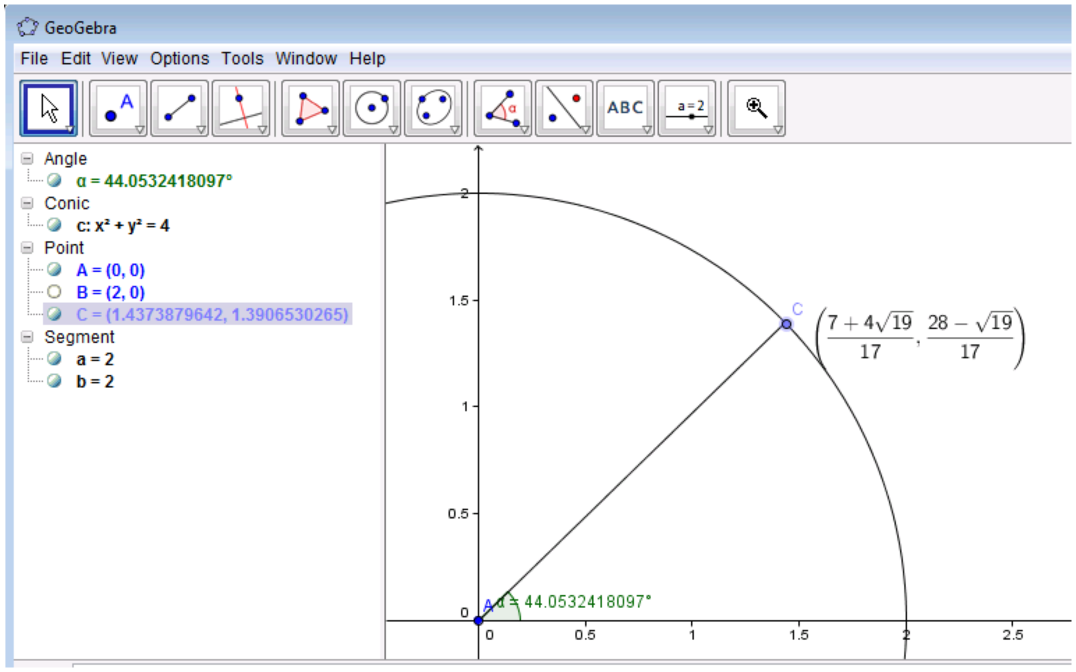
Is it plausible to conjecture that this condition is necessary and sufficient, or you might have found some counter-examples?



Angle
α = 14.0362434679°
Conic
c: x² + y² = 4
Point
A = (0, 0)
B = (2, 0)
C = (1.9402850003, 0.4850712501)
Segment
a = 2
b = 2

$\left(\frac{8\sqrt{17}}{17}, \frac{2\sqrt{17}}{17}\right)$

First found point



Angle
α = 44.0532418097°
Conic
c: x² + y² = 4
Point
A = (0, 0)
B = (2, 0)
C = (1.4373879642, 1.3906530265)
Segment
a = 2
b = 2

$\left(\frac{7+4\sqrt{19}}{17}, \frac{28-\sqrt{19}}{17}\right)$

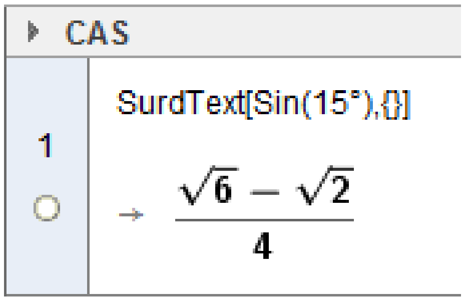
Another finding

Figure 1.2: Exploring the "radical points" on the arc.

1.5.2 A better calculator.

Student might try to calculate special values of trigonometric functions and wrap it up with `SurdText` command, with or without an empty list.

Now we may solve the problem stated in the beginning: Find sin(15°) by typing `SurdText[15*2pi/360,{}]` in the CAS view. Students may use the CAS view to explore more possibilities. Isn't it a better calculator?



```
CAS
1  SurdText[Sin(15°),{}]
```

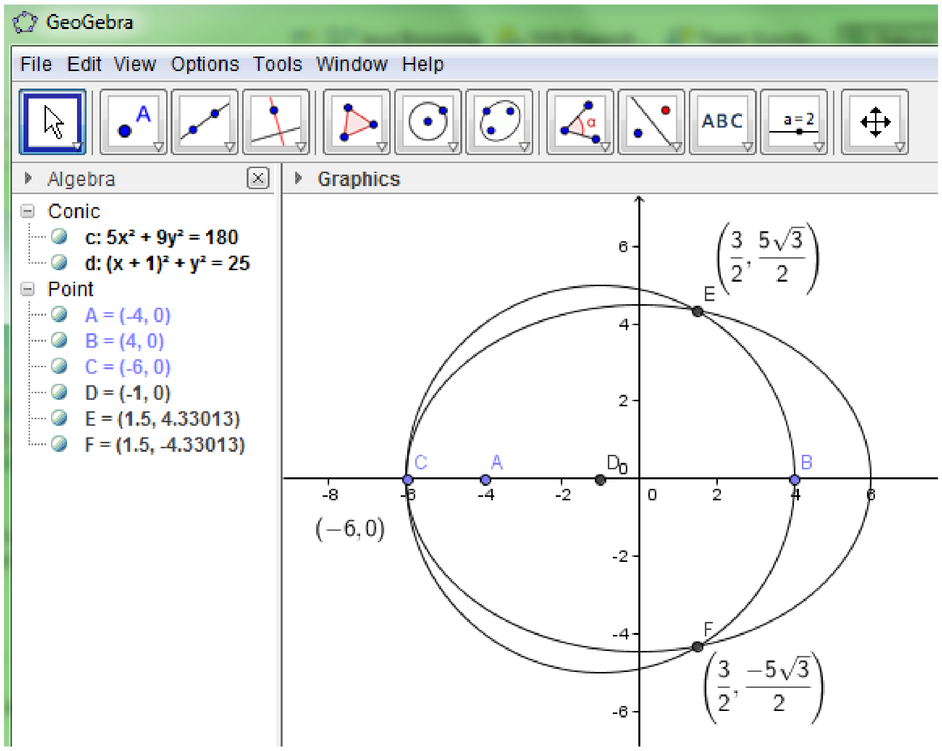
$$\to \frac{\sqrt{6}-\sqrt{2}}{4}$$

Figure 1.3: Find the value of sine 15 degrees in GeoGebra CAS

1.5.3 Teaching and learning

Suppose we have this problem of analytic geometry in high school: Given an ellipse c whose foci are $A = (-4, 0)$, $B = (4, 0)$ and passes through $C = (-6, 0)$, and a circle d such that BC is its diameter. Find all the intersections of c and d.

With GeoGebra and SurdText command serving as a “powerful calculator”, it is done in an instinct as the picture has shown below:


GeoGebra
File Edit View Options Tools Window Help
ABC
a=2
Algebra
Graphics
$\left(\frac{3}{2}, \frac{5\sqrt{3}}{2}\right)$
E
C
A
D
B
$(-6, 0)$
F
$\left(\frac{3}{2}, \frac{-5\sqrt{3}}{2}\right)$

```
Conic
    c: 5x² + 9y² = 180
    d: (x + 1)² + y² = 25
Point
    A = (-4, 0)
    B = (4, 0)
    C = (-6, 0)
    D = (-1, 0)
    E = (1.5, 4.33013)
    F = (1.5, -4.33013)
```

Figure 1.4: Find the intersection of c and d

We we just read the coordinate of E and F, i.e. $(1.5, 4.33013)$ and $(1.5, -4.33013)$, it is not very interesting, except we know that $x(E) = x(F)$ and $y(E) = -y(F)$. However, we can apply the SurdText command to these two points and see that $E = (3/2, 5\sqrt{3}/2)$ and $F = (3/2, 5\sqrt{3}/2)$. These nice numbers suggest more properties from the figure. For example, it can be easily shown by these coordinates that $\triangle BED$, $\triangle BDF$ and $\triangle CEF$ are equilateral, because $x(E) = x(F)$ are just the average of $x(B)$ and $x(D)$, and $y(F)$ is just $|BD|$ times $\sqrt{3}/2$. The radicals tell a lot more than the decimals!

In the past, when calculators are not very popular, mathematics teachers would have instructed their students that one should not rely on calculators because they are not readily available. During the past 10 or 20 years, usual calculators and scientific calculators were wide-spread, but teachers would still insist that this may not be helpful, for at times one may encounter irrationals which are not represented exactly or nicely on calculators. But what they are nicely represented?

We should admit that calculators cannot replace the hand calculation process because human brain needs that kind of training to learn about the concepts, but with more powerful computation tools at hand, they can only boost the conceptualization and ease the burdens of calculation, and provide another type of intuition.

1.6 Possible Extension

Integer relation is defined in terms of linear combinations, and therefore deals with a sum. We may extend the functionality to identifying a product, since $\log(ab) = \log(a) + \log(b)$. If we have a "table of logarithm" of common constants such as small integers, π, e, and so on, then we may apply the integer relation algorithms to recognize numbers in the form of

$$x = C_1^{r_1} C_2^{r_2} \cdots C_k^{r_k},$$

where C_j are those constants, and r_k are positive rational numbers.

Therefore we may extend the `AlgebraicFit` object by adding some more new types of fitting.

Chapter 2

How PSLQ works: Intuitive observations

Although only the results of PSLQ is important in the `SurdText` command, it is worth understanding how PSLQ itself works, as it was selected the "Top Ten Algorithm of the Century" in 2000 [1], and has many nice applications [1].

Given $(x_1, x_2, \cdots, x_n) \in \mathbb{R}^n$ where $|x|^2 = x_1^2 + \cdots x_n^2 = 1$. Let $x^\perp = \{w \in \mathbb{R}^n \mid w^T x = w_1 x_1 + \cdots w_n x_n = 0\}$ be the set of orthogonal vectors of x.

[1]cf. Wikipedia, "Integer Relation Algorithm" at `http://en.wikipedia.org/wiki/Integer_relation_algorithm`

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

Figure 2.1: Formula found by the usage of PSLQ (Bailey, Borwein, & Plouffe, 1997)[2]

The PSLQ algorithm was introduced and described in detail in two classical papers [3] [4]. It returns a vector $m \in \mathbb{Z}^n \cap x^{\perp}$, if exists, such that its norm $\sqrt{m_1^2 + \cdots m_n^2}$ is the smallest. In that case, denote M_x as this norm.

Let us further assume that all x_j are not equal to zero (if some $x_j = 0$, there is an obvious integer relation, namely $m_j = 1$ and $m'_j = 0$ for all $j' \neq j$).

Intuitively, recall the problem of orthogonalization (a systematic way to look for orthogonal vectors): Given the standard orthonormal basis $(e_j)_{1 \leq j \leq n}$ of $\mathbb{R}^n$, find another orthogonal basis $(v_j)_{1 \leq j \leq n}$ such that $v_1 = x = x_1 e_1 + \cdots + x_n e_n$. To proceed, let $\tilde{v_2} = e_1 - (e_1 \cdot v_1) v_1 = e_1 - x_1 v_1$, which is just the vector obtained by subtracting from x the projection of itself to e_1, and it must be a non-zero vector that is perpendicular to v_1. Indeed, $\tilde{v_2} \cdot v_1 = e_1 \cdot v_1 - x_1 v_1 \cdot v_1 = x_1 - x_1 = 0$, and $|\tilde{v_2}|^2 = 1 + x_1^2 - 2x_1(e_1 \cdot v_1) = 1 - x_1^2 > 0$. Now let $v_2 = \tilde{v_2}/|\tilde{v_2}|$. We have got a unit vector v_2 that is orthogonal to v_1.

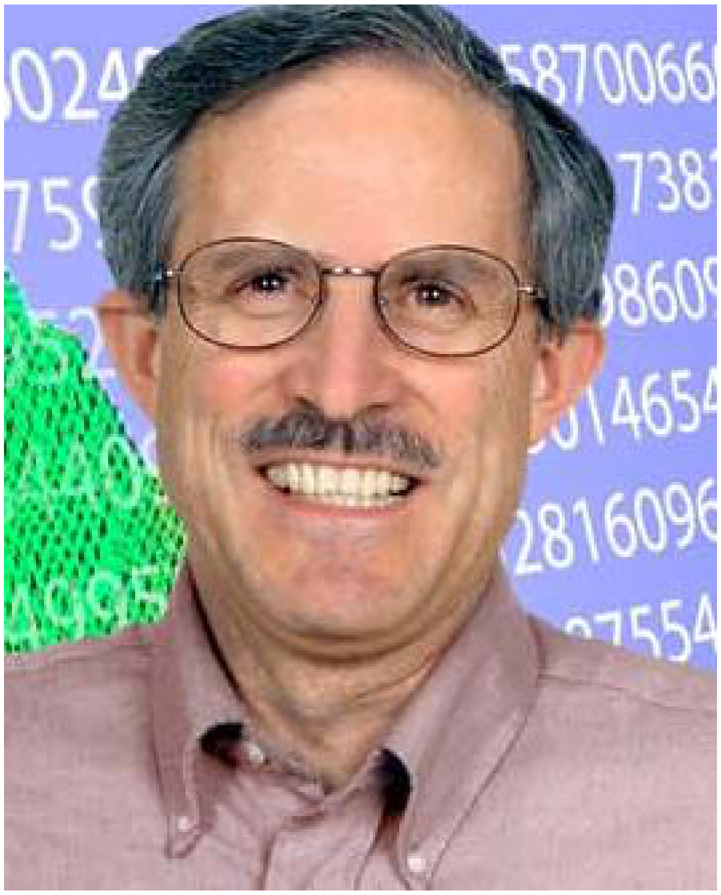


Figure 2.2: David H. Bailey, co-discoverer of the Bailey-Borwein-Plouffe formula of π.

This process can be carried out iteratively, i.e. for any $k < n$, after we get $v_2, \cdots, v_k$ such that $\{v_1, \cdots v_k\}$ is a set of orthonormal vectors, define

$$\tilde{v}_{k+1} = e_k - (e_k \cdot v_1)v_1 - (e_k \cdot v_2)v_2 - \cdots - (e_k \cdot v_k)v_k$$

and $v_{k+1} = \tilde{v}_{k+1}/|\tilde{v}_{k+1}|$. Then v_{k+1} is orthogonal to $v_1, \cdots, v_k$. Thus a set of orthogonal basis $\{v_1 = x, v_2, \cdots, v_n\}$ is inductively constructed, and $x^\perp$ is equal to the subspace spanned by $v_2, \cdots, v_n$. It is evident that if x has some integer relation $m \in x^\perp \cap \mathbb{Z}^n$, m must be a linear combination of $v_2, \cdots, v_n$.

Putting everything in terms of matrices, we write x as a $1 \times n$ row vector, and $H_x = [v_2 \cdots v_n]$ as a $n \times (n-1)$ matrix. We have $xH_x = 0_{1\times(n-1)}$.

2.1 A lucky case solved in one-step.

Although a natural way to pose the problem is to search for real numbers $y_2, \cdots y_n$ such that $y_2v_2 + \cdots y_nv_n$ becomes an integer vector (and thus an integer relation of x), another perspective, which turns out to be more suitable for computation, is to "reduce" the matrix H_x to one that has a column that contains a single non-zero element. To illustrate this, suppose that $x = (1, \sqrt{3}, 1/\sqrt{3})$ and imagine (ideally) that we have got orthogonal basis of $x^\perp$ as follows:

$$H = \begin{pmatrix} 0.5 & 0 \\ * & 0.3 \\ * & 0.9 \end{pmatrix}$$

Notice that if we add (-3) times the second row to the third row, we will get $\begin{pmatrix} * & 0 \end{pmatrix}$ in the third row, and $\begin{pmatrix} 0 & 0.3 & 0 \end{pmatrix}^T$ in the second column. Equivalently, what we did was to multiply some elementary transformation matrix D to the left of H:

$$\tilde{H} = DH = \begin{pmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & -3 & 1 \end{pmatrix} \begin{pmatrix} 0.5 & 0 \\ * & 0.3 \\ * & 0.9 \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ * & 0.3 \\ * & 0 \end{pmatrix}$$

Since we also have $(xD^{-1})(DH) = xH = 0_{1\times(n-1)}$, we know that e_2 is an integer relation for xD^{-1}, so we have found an integer relation for x, which is e_2D, the second column of D. Indeed, $\begin{pmatrix} 1 & \sqrt{3} & 1/\sqrt{3} \end{pmatrix} \begin{pmatrix} 0 & 1 & -3 \end{pmatrix}^T = \sqrt{3} - 3/\sqrt{3} = 0$.

In this illustration, it is just a coincidence that 0.9 is three times 0.3 so we can get the integer relation in just one step. This is not the case in general, but we may perform the elementary row-transformation in a way similar to Euclidean algorithm. Given two real numbers a and b that has integer relation, we may perform the Euclidean algorithm on a and b, and the algorithm must get to a halt in a finite number of steps. In the following, we illustrate a non-trivial example for $n = 3$: Find integer relations for $(2, 2 - \sqrt{2}, 2 + \sqrt{2})$.

Of course, this input is somewhat misleading, since we actually DON'T know the radical form of these numbers, but only decimal form. During the calculation, the input would be given as something like $(2, 0.5858, 3.4142)$, where precision is important. However, for the purpose of illustration, we only keep 4 decimal places here.

2.2 Initial Step.

Initially, let x be the normalized vector for this input, namely:

$$x \leftarrow \frac{1}{|x|}\, x = \frac{1}{4}\, x = \begin{pmatrix} 0.5000 & 0.1464 & 0.8536 \end{pmatrix}$$

Then we are to find $n-1$ column vectors that combine the lower trapezoidal H_x, which is the result of orthogonalization. Instead of performing the orthogonalization in $n-1$ steps, it turns out that there is a closed form for H_x, according to a formula in [3] [4]: $h_{jj} = s_{j+1}/s_j$ and $h_{ij} = -x_i x_j/(s_j s_{j+1})$ for all $i > j$, where s_j^2 are the sum of squares:

$$s_j^2 = x_j^2 + x_{j+1}^2 + \cdots + x_n^2,$$

which is what "S" stands for in PSLQ. ("P" stands for polynomial-time, and "LQ" stands for LgQ-decomposition, which we will see later in this illustration.)

In our example, $s_1 = 1$, $s_2 = \sqrt{((2-\sqrt{2})^2 + (2+\sqrt{2})^2)/16} = \sqrt{3}/2 = 0.866$, and $s_3 = x_3 = 0.8536$, and we get H from the above formula:

$$H = \begin{pmatrix} 0.8660 & 0 \\ -0.0846 & 0.9856 \\ -0.4928 & -0.1691 \end{pmatrix}$$

As mentioned before, we want some row transformation matrix D such that DH has some column that contains a single non-zero element. However, to get an integer relation, we also want to ensure that D contains integer entries. This adds a restriction that we can only multiply an integer to a row and add it to another. In general, it is not possible to perform such transformation to get a column that contains a single non-zero element. The most we can do is to make it as close to this as possible: to find an integer matrix D such that every column of DH has a single element that "dominates" the other, i.e. is at least two times in magnitude than all the other elements.

2.3 Reduction

In our example, the second column of H is already dominated ($0.9856 > 2 \times 0.1691$), but the first is not. So, we add one times the first row to the third:

$$\tilde{H} = D_1 H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.8660 & 0 \\ -0.0846 & 0.9856 \\ -0.4928 & -0.1691 \end{pmatrix} = \begin{pmatrix} 0.8660 & 0 \\ -0.0846 & 0.9856 \\ 0.3732 & -0.1691 \end{pmatrix}$$

Let $H \leftarrow \tilde{H}$, and let $A \leftarrow D_1$, $B \leftarrow D_1^{-1}$ in order to keep track of the transformations, and let $x[1] \leftarrow xB = xD_1^{-1}$:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}, x[1] = \begin{pmatrix} -0.3536 & 0.1464 & 0.8536 \end{pmatrix}.$$

This is the first iteration.

2.4 Exchange

After the reduction, we found that $\tilde{H}$ has no column that has a single dominating entry. By the inspiration of Euclidean algorithm, we may swap the row that contains the "largest" element (in some sense) with some other row. In our case, the second row contains 0.9856, which is the largest element, so we swap the second row with the third row:

$$\tilde{H} \leftarrow R_2\tilde{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0.8660 & 0 \\ -0.0846 & 0.9856 \\ 0.3732 & -0.1691 \end{pmatrix} = \begin{pmatrix} 0.8660 & 0 \\ 0.3732 & -0.1691 \\ -0.0846 & 0.9856 \end{pmatrix}$$

where $R_2 = (e_1, e_3, e_2)$ is the row transformation matrix that in effect swaps the second and the third row. Now $\tilde{H}$ is still lower-trapezoidal, a reduction is possible: add 6 times of the second row to the third row, reducing $H_{32} = 0.9856$; but at the same time H_{31} will become 2.1546, so we also add (-2) times of the first row to the third row, getting $H_{31} = 0.4226$. Therefore,

$$\tilde{H} \leftarrow D_2\tilde{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 6 & 1 \end{pmatrix} \begin{pmatrix} 0.8660 & 0 \\ 0.3732 & -0.1691 \\ -0.0846 & 0.9856 \end{pmatrix} = \begin{pmatrix} 0.8660 & 0 \\ 0.3732 & -0.1691 \\ 0.4228 & -0.0290 \end{pmatrix}$$

Now, reassign $H \leftarrow \tilde{H}$, and let $A \leftarrow (D_2R_2)A$, $B \leftarrow B(D_2R_2)^{-1} = BR_2D_2^{-1}$, $x[2] \leftarrow x[1]B$. We have now:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 4 & 1 & 6 \end{pmatrix}, B = \begin{pmatrix} 0 & 1 & 0 \\ -6 & 2 & 1 \\ -1 & 1 & 0 \end{pmatrix}, x[2] = \begin{pmatrix} -1.7322 & 1.6464 & 0.1464 \end{pmatrix}.$$

This is the second iteration in PSLQ, which contains a reduction operation (called *Hermite Reduction* in [4]), and a swap operation (called *Exchange*). In any step k where the swapping is not needed, we just let $R_k = I_n$ be the $n \times n$ identity matrix. In our example, let $R_1 = I_n$.

2.5 Corner

This time the first row contains the largest element, so we consider swapping the first and the second rows. However, after that, H will become a matrix that is not lower-trapezoidal. Therefore we need to perform one more operation, which is explained as follows. In general terms, if h_{jj} is the largest element where $j < n-1$, then we focus on the 2×2 matrix $h_{jj}, h_{j,j+1}(=0), h_{j+1,j}, h_{j+1,j+1}(\neq 0)$, which was denoted in [4] as

$$\begin{pmatrix} \alpha & 0 \\ \beta & \lambda \end{pmatrix}.$$

We also suppose that all elements have absolute value less than 1, and that α is large enough so that $|\alpha| > 2|\beta|$ and that $|\alpha| \leq \gamma|\lambda|$ for some constant $\gamma > 1$. Also, let $\delta = \sqrt{\beta^2 + \lambda^2}$. After swapping, the non-zero number λ goes to the upper-right corner. We will "rotate" this matrix so that the upper-right corner becomes 0, while ensuring that the left-upper corner is large enough. This is done by multiplying the following unitary matrix to the right:

$$\tilde{Q} = \begin{pmatrix} \beta/\delta & -\lambda/\delta \\ \lambda/\delta & \beta/\delta \end{pmatrix}.$$

Indeed,

$$\begin{pmatrix} \beta & \lambda \\ \alpha & 0 \end{pmatrix} \cdot \begin{pmatrix} \beta/\delta & -\lambda/\delta \\ \lambda/\delta & \beta/\delta \end{pmatrix} = \begin{pmatrix} \delta & 0 \\ \alpha\beta/\delta & -\alpha\lambda/\delta \end{pmatrix}.$$

To ensure that $|\delta| < 1$, a sufficient condition is that $\beta^2 + \lambda^2 \leq \alpha^2/4 + \alpha^2/\gamma^2 < \alpha^2 < 1$, therefore we may choose γ so that $1/4 + 1/\gamma^2 < 1$, or $\gamma > 2/\sqrt{3}$. The lower-right element still have absolute value less than 1, for $|\alpha\lambda/\delta| = |\alpha||\lambda/\sqrt{\beta^2 + \lambda^2}| \leq |\alpha| < 1$. Let Q_3 be a unitary $n-1 \times n-1$ matrix such that, when it is multiplied to the right of H, it keeps track of the above operation. This is called the *Corner* operation in [4]. In any step k when the Corner operation is not needed, just let $Q_k = I_{n-1}$ be the $(n-1) \times (n-1)$ identity matrix. In our example, let $Q_1, Q_2 = I_{n-1}$.

In our example, the corner in question is:

$$\begin{pmatrix} \alpha = 0.8660 & 0 \\ \beta = 0.3732 & \lambda = -0.1691 \end{pmatrix},$$

and $\delta = 0.4097$, $\beta/\delta = 0.9109, \lambda/\delta = -0.4126$. So,

$$\tilde{H} \leftarrow R_3 \tilde{H} Q_3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot H \cdot \begin{pmatrix} 0.9109 & 0.4126 \\ -0.4126 & 0.9109 \end{pmatrix} = \begin{pmatrix} 0.4097 & 0 \\ 0.7888 & 0.3574 \\ 0.3970 & 0.1480 \end{pmatrix}$$

We may now reduce this by using the following matrix D_3:

$$\tilde{H} \leftarrow D_3\tilde{H} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.4097 & 0 \\ 0.7888 & 0.3574 \\ 0.3970 & 0.1480 \end{pmatrix} = \begin{pmatrix} 0.4097 & 0 \\ -0.0127 & 0.1480 - 0.0305 & 0.3574 \end{pmatrix}$$

Let $H \leftarrow \tilde{H}$, $A \leftarrow D_3R_3A$, $B \leftarrow BR_3D_3^{-1}$, $x[3] \leftarrow x[2]B$, and this is the third iteration. We have now

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 3 & 1 & 5 \\ -1 & 0 & -2 \end{pmatrix}, B = \begin{pmatrix} 2 & 0 & 1 \\ -1 & 1 & 2 \\ -1 & 0 & -1 \end{pmatrix}, x[3] = \begin{pmatrix} 0.0000 & 0.1464 & -0.0607 \end{pmatrix}.$$

2.6 Termination, and beyond

We see that in each iteration of PSLQ, we do necessary exchanging and cornering, and then reduce. As we can see, the first coordinate of $x[3]$ is zero (under our precision requirement), therefore the first column of B must be an integer relation. Indeed,

$$2 \cdot 2 + (-1) \cdot (2 - \sqrt{2}) + (-1) \cdot (2 + \sqrt{2}) = 0$$

.

This is one of the two halt conditions of PSLQ, where one of the coordinates of $x[k]$ is zero at step k. Another condition is when $h_{ii} = 0$ for some $1 \leq i \leq n-1$. Alyson Reeves proved that this condition also guarantees an integer relation as some column of B ([4], Lemma 5). Moreover, if an integer relation exists for x with norm M_x, the PSLQ algorithm can find it within $2(n^3 + n^2 \log M_x)$ steps (Corollary 2). This is the exact meaning of how the algorithm works in polynomial time.

The essentials of PSLQ algorithm was shown above: First. normalize x and get a real $n \times (n-1)$ column-orthonormal, lower-trapezoidal matrix H_x by orthogonalization. Perform so-called Hermite reduction to get an integer matrix D, such that the entries in the diagonal of DH is greater than all other entries by a factor of at least 2, in magnitude. If there is a column j in which a single element is non-negative then we are done: multiplying all the D^{-1}'s we have obtained, then we will see the integer relation at the column j of it. Otherwise, swap a pair of rows and "rotate" the corner according to some rule so that the Hermite reduction could be done again.

Bibliography

[1] D. H. Bailey and J. M. Borwein. PSLQ: an algorithm to discover integer relations. *LBNL Paper LBNL-2144E*, 2009.

[2] D. H. Bailey, Borwein P., and S. Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66:903–913, 1997.

[3] H. R. P. Ferguson and D. H. Bailey. A polynomial time, numerically stable integer relation algorithm. *NASA Technical Report RNR-91-032*, 1991.

[4] H. R. P. Ferguson, D. H. Bailey, and S. Arno. Analysis of PSLQ, an integer relation finding algorithm. *Mathematics of Computation*, 68(225):351–370, 1999.

List of Figures

List of Tables