LLL Algorithm and it's Application in Diophantine Approximation and Polynomial Factorization

A thesis submitted to

Indian Institute of Science Education and Research Pune
in partial fulfilment of the requirements for the

Mathematics M.Sc Degree Program

under the supervision of

Dr. Baskar Balasubramanyam

by

Balasubramanian S
April 2026

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road, Pashan, Pune India 411008

Certificate

This is to certify that this thesis entitled **"LLL Algorithm and it's Application in Diophantine Approximation and Polynomial Factorization"** submitted towards the partial fulfilment of the Mathematics M.Sc Degree Program at the Indian Institute of Science Education and Research Pune represents work carried out by **Balasubramanian S** under the supervision of **Baskar Balasubramanyam**.

Baskar Balasubramanyam
Master's Thesis Supervisor

Declaration

I hereby declare that the matter embodied in the report entitled LLL Algorithm and it's Application in Diophantine Approximation and Polynomial Factorization are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research (IISER) Pune, under the supervision of Dr. Baskar Balasubramanyam and the same has not been submitted elsewhere for any other degree.

Balasubramanian S

To my Family

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Baskar Balasubramanyam, for valuable guidance and encouragement throughout my thesis work.
I also thank my family − my parents, Sivakumar. B and Muthurathinam. S, my elder sister Gayathri. S for their continuous support during the completion of this work.
Finally, I thank everyone who directly or indirectly contributed to the successful completion of this thesis, especially my friends.

Abstract

This thesis presents an expository study of the Lenstra–Lenstra–Lovász (LLL) algorithm and its important applications in Diophantine approximation and polynomial factorization. The LLL algorithm is a fundamental lattice basis reduction method that produces short and nearly orthogonal basis vectors in polynomial time, making it a powerful tool in computational number theory and algebra. Although the algorithm does not solve the Shortest Vector Problem exactly, it provides an efficient approximation by finding sufficiently short lattice vectors and reduced bases.

Further, the thesis explores applications of the LLL algorithm in Diophantine approximation, particularly in simultaneous rational approximation and problems involving integer relations. Its role in polynomial factorization is also examined, with emphasis on the factorization of polynomials over integers through lattice-based methods. Illustrative examples are included to demonstrate the theoretical and practical effectiveness of the algorithm.

Contents

Chapter 1

Introduction to Lattice Based Problems

A lattice $\mathcal{L}$ in $\mathbb{R}^n$ is defined as the set of all integer linear combinations of linearly independent vectors $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$:

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} z_i b_i \;\middle|\; z_i \in \mathbb{Z} \right\}$$

The set $\{b_1, \ldots, b_n\}$ is called a basis of the lattice. Lattice based problems play an important role in computational number theory and modern cryptography. In theoretical computer science, lattice problems are a class of optimization problems related to mathematical objects called lattices. The conjectured intractability of such problems is central to the construction of secure lattice-based crypto-systems: lattice problems are an example of NP-hard problems which have been shown to be average-case hard, providing one of the best idea for the security of cryptographic algorithms. In addition, some lattice problems which are worst-case hard can be used as a basis for extremely secure cryptographic schemes. The use of worst-case hardness in such schemes makes them among the very few schemes that are very likely secure even against quantum computers.For applications in such cryptosystems, lattices over vector spaces (often $\mathbb{Q}^n$) or free modules (often $\mathbb{Z}^n$) are generally considered.The norm usually considered is the Euclidean norm L_2.

1.1 Shortest Vector Problem (SVP)

1.1.1 What is the Shortest Vector Problem?

Given a lattice $\mathcal{L}$ with basis $B = \{b_1, \ldots, b_n\}$, the **Shortest Vector Problem (SVP)** asks: Find a non-zero vector $v \in \mathcal{L}$ such that $\|v\|$ is minimized. Mathematically,

$$\lambda_1(\mathcal{L}) = \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|$$

where $\lambda_1(\mathcal{L})$ denotes the length of the shortest non-zero vector. In other words, the algorithm should output a non-zero vector $v \in \mathcal{L}$ such that

$$\|v\|_N = \lambda_1(\mathcal{L}),$$

In the following, the size of the problem is specified by n, the dimension of the vector space V. In the γ-approximation version of SVP (denoted SVP_γ), one must find a non-zero lattice vector of length at most

$$\gamma \cdot \lambda_1(\mathcal{L}),$$

for a given parameter $\gamma \geq 1$.The determinant $d(L)$ of L is defined by

$$d(L) = |\det(b_1, b_2, \ldots, b_n)|$$

the b_i being written as column vectors. This is a positive real number that does not depend on the choice of the basis [5]. Now, we will see some useful results which establish information about the basis vectors and elements of the Lattice and how they behave.

Proposition 1.1.1. *Let $b_1, b_2, \ldots, b_n$ be a reduced basis for a lattice L in $\mathbb{R}^n$, and let $b_1^*, b_2^*, \ldots, b_n^*$ be defined as above. Then we have*

$$\|b_j^*\|^2 \leq 2^{i-1} \|b_i^*\|^2 \quad \text{for } 1 \leq j \leq i \leq n, \tag{1.1}$$

$$d(L) \leq \prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} d(L), \tag{1.2}$$

$$\|b_1\| \leq 2^{\frac{n-1}{4}} d(L)^{1/n}. \tag{1.3}$$

Proposition 1.1.2. *Let $L \subset \mathbb{R}^n$ be a lattice with reduced basis $b_1, b_2, \ldots, b_n$. Then*

$$\|b_1\|_2 \leq 2^{\frac{n-1}{2}} \cdot \|x\|_2 \tag{1.4}$$

for every $x \in L, x \neq 0$.

By this preposition from [1] ,LLL is $2^{\frac{n-1}{2}}$-approximation version of SVP.

Proposition 1.1.3. *Let* $L \subset \mathbb{R}^n$ *be a lattice with reduced basis* $b_1, b_2, \ldots, b_n$. *Let* $x_1, \ldots,$ $x_t \in L$ *be linearly independent. Then we have*

$$\|b_j\|^2 \leq 2^{n-1} \max\left\{\|x_1\|^2, \|x_2\|^2, \ldots, \|x_t\|^2\right\} \quad \text{for } j = 1, 2, \ldots, t. \tag{1.5}$$

1.1.2 Hardness Results

The exact version of the problem is known to be NP-hard under randomized reductions. By contrast, the corresponding problem with respect to the uniform norm is known to be NP-hard. Thus, no polynomial time algorithm is known to solve SVP exactly for general lattices. One can suspect that finding shortest vector might be easy since it has to near the origin. But, by seeing the below example one can get the intiution of the NP-Hardness. Consider the lattice $\mathcal{L} \subset \mathbb{R}^2$ generated by the basis vectors

$$b_1 = (1000, 1), \qquad b_2 = (1001, 1).$$

The basis matrix is therefore

$$B = \begin{pmatrix} 1000 & 1001 \\ 1 & 1 \end{pmatrix}.$$

Any lattice vector has the form

$$v = xb_1 + yb_2, \qquad x, y \in \mathbb{Z}.$$

Explicitly,

$$v = x(1000, 1) + y(1001, 1),$$

so

$$v(x, y) = (1000x + 1001y,\ x + y).$$

Using the Euclidean norm, we compute the norm of v :

$$\|v\|^2 = (1000x + 1001y)^2 + (x + y)^2.$$

The SVP asks us to minimize this quantity over all integer pairs $(x, y) \neq (0, 0)$. First compute the lengths of the basis vectors:

$$\|b_1\|^2 = 1000^2 + 1^2 = 1{,}000{,}001,$$

$$\|b_2\|^2 = 1001^2 + 1^2 = 1{,}002{,}002.$$

Thus,

$$\|b_1\| \approx 1000.$$

At first glance, it appears that the shortest vector has length approximately 1000. But, it's not the case. Consider the linear combination

$$v = b_2 - b_1.$$

This corresponds to

$$x = -1, \qquad y = 1.$$

Then

$$v = (1001 - 1000,\ 1 - 1) = (1, 0).$$

Its norm is

$$\|v\| = 1.$$

Thus the shortest non-zero lattice vector is $(1, 0)$, with length 1. The ratio between the basis length and the shortest vector is approximately

$$\frac{\|b_1\|}{\lambda_1(\mathcal{L})} \approx 1000.$$

The shortest vector is therefore hidden inside a near-cancellation:

$$1001 - 1000 = 1.$$

To obtain a short vector, we must choose integers x, y such that

$$1000x + 1001y \approx 0.$$

This is a Diophantine equation. Indeed,

$$1000(-1) + 1001(1) = 1.$$

Thus we are solving an integer optimization problem: finding integer combinations that cause large coefficients to cancel. Which is known to be NP hard. The basis vectors b_1 and b_2 are almost parallel. Their angle is very small. That's a problem.

1.2 Basis Reduction

A lattice can have many different bases. Some bases contain very long and highly non-orthogonal vectors.**Basis reduction** aims to find a "nice" basis with:

- Short vectors
- Nearly orthogonal structure (not nearly dependent)

Reduced bases help in approximating solutions to SVP efficiently.That because, a basis which is nearly dependent (nearly linear) creates problem as we have seen earlier with examples. So, Basis which is not like previous example can help. Which is exactly the short vectors and nearly orthogonal .

Chapter 2

LLL Algorithm

Rather than finding a shortest vector, idea is to find better basis as mentioned above. The Lenstra–Lenstra–Lovász (LLL) algorithm is a polynomial-time lattice basis reduction algorithm.Given a basis $B = \{b_1, \ldots, b_n\}$, LLL produces a reduced basis satisfying:

1. Size reduction condition
2. Lovász condition

The algorithm runs in polynomial time and guarantees:

$$\|b_1\| \leq 2^{\frac{n-1}{2}} \lambda_1(\mathcal{L})$$

Thus, LLL provides an approximate solution to SVP. The LLL algorithm is improved after some results from [10]

2.0.1 Preliminaries

Let $B = \{b_1, b_2, \ldots, b_n\}$ be a basis of a lattice $L \subset \mathbb{R}^n$.We define the Gram–Schmidt orthogonalization (GSO) of B as follows:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$$

where

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$$

and $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product. The vectors $b_1^*, \ldots, b_n^*$ are mutually orthogonal.

2.0.2 Size Reduction

A basis B is said to be **size-reduced** if for all $i > j$:

$$|\mu_{i,j}| \leq \frac{1}{2}$$

Size-Reduction: Why It Reduces the Basis

For each $i > j$, replace

$$b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil \, b_j$$

so that

$$|\mu_{ij}| \leq \tfrac{1}{2}.$$

Geometrically:

- μ_{ij} measures how much of b_j lies inside b_i.If it is large it's nearly dependent and smaller value or 0 means orthogonal or nearly orthogonal.
- If $|\mu_{ij}| > \frac{1}{2}$, then b_i contains a large unnecessary multiple of b_j.

  Subtracting that multiple:

- keeps the lattice unchanged,
- shortens b_i.

Size-reduction removes large parallel components.

2.0.3 Lovász Condition

Let $0 < \delta < 1$ (typically $\delta = 3/4$). A basis satisfies the **Lovász condition** if for all $i \geq 2$:

$$\delta \|b_{i-1}^*\|^2 \leq \|b_i^*\|^2 + \mu_{i,i-1}^2 \|b_{i-1}^*\|^2$$

Equivalently,

$$\|b_i^*\|^2 \geq (\delta - \mu_{i,i-1}^2) \|b_{i-1}^*\|^2$$

The Lovász condition in the definition of an LLL-reduced basis states that

$$\delta \, \|b_{k-1}^*\|^2 \;\leq\; \|b_k^*\|^2 + \mu_{k,k-1}^2 \, \|b_{k-1}^*\|^2, \qquad \text{where } \tfrac{1}{4} < \delta < 1.$$

Equivalent Form

Rewriting the inequality, we obtain

$$\|b_k^*\|^2 \;\geq\; \left(\delta - \mu_{k,k-1}^2\right) \|b_{k-1}^*\|^2.$$

Since the size-reduction condition ensures that

$$|\mu_{k,k-1}| \leq \tfrac{1}{2},$$

it follows that

$$\delta - \mu_{k,k-1}^2 \;\geq\; \delta - \tfrac{1}{4} \;>\; 0.$$

Interpretation

If the Lovász condition fails, i.e.,

$$\|b_k^*\|^2 < \left(\delta - \mu_{k,k-1}^2\right) \|b_{k-1}^*\|^2,$$

then the Gram–Schmidt vector b_k^* is too short relative to b_{k-1}^*, indicating that the basis vectors are badly ordered. So, it order the basis vectors in terms of norm and also makes sure that b_k^* not much shorter than b_{k-1}^*. Thus, the Lovász condition ensures that shorter vectors appear earlier in the basis and serves as the main mechanism driving reduction in the LLL algorithm. The larger the δ value, the stronger the reduction is.

A basis B is called **LLL-reduced** if:

1. It is size-reduced.
2. It satisfies the Lovász condition.

2.0.4 The LLL Algorithm

Input: A lattice basis $B = \{b_1, \ldots, b_n\}$, parameter $1/4 < \delta < 1$.
Output: An LLL-reduced basis.
The idea of the Algorithm as follows : The LLL algorithm maintains a basis

$$B = (b_1, \ldots, b_n)$$

together with its Gram–Schmidt orthogonalization

$$b_1^*, \ldots, b_n^*$$

and coefficients

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}.$$

At each step the algorithm ensures two conditions:

1. Size-Reduction Condition

$$|\mu_{ij}| \leq \tfrac{1}{2} \qquad (1 \leq j < i \leq n).$$

If $|\mu_{k,k-1}| > \frac{1}{2}$, then we replace

$$b_k \leftarrow b_k - r b_{k-1},$$

where r is the nearest integer to $\mu_{k,k-1}$. This removes large parallel components and shortens b_k.

2. Lovász Condition

For a parameter δ with $\frac{1}{4} < \delta < 1$, we require

$$\delta \|b_{k-1}^*\|^2 \leq \|b_k^*\|^2 + \mu_{k,k-1}^2 \|b_{k-1}^*\|^2.$$

If this condition fails, we interchange b_{k-1} and b_k, update the Gram–Schmidt data, and decrease k. If it holds, we increase k and continue. The algorithm alternates between size-reduction and swaps until both conditions hold for all indices. At termination, the basis is called LLL-reduced.

Size-Reduction condition for all n

Suppose either $k = 1$ or the Lovász condition holds:

$$\|b_k^* + \mu_{k,k-1} b_{k-1}^*\|^2 \geq \frac{3}{4} \|b_{k-1}^*\|^2.$$

In this case, we enforce the full size-reduction condition

$$|\mu_{kj}| \leq \tfrac{1}{2} \qquad \text{for } 1 \leq j \leq k-1.$$

If this does not hold, let $\ell < k$ be the largest index such that $|\mu_{k\ell}| > \frac{1}{2}$, and let r be the integer nearest to $\mu_{k\ell}$. We then replace

$$b_k \leftarrow b_k - r b_\ell.$$

The coefficients are updated as follows:

$$\mu_{kj} \leftarrow \mu_{kj} - r\mu_{\ell j} \quad (j < \ell), \qquad \mu_{k\ell} \leftarrow \mu_{k\ell} - r,$$

while all other μ_{ij} and the vectors b_i^* remain unchanged. This process is repeated until

$$|\mu_{kj}| \leq \tfrac{1}{2} \quad \text{for all } 1 \leq j \leq k-1.$$

After size-reduction is achieved, we replace k by $k+1$ and continue the algorithm. In the special case $k = 1$, this step simply replaces k by 2. Here is the Algorithm:

Algorithm 1 LLL Reduction Algorithm

Require: Basis $B = (b_1, \ldots, b_n)$, parameter $\delta \in (\frac{1}{4}, 1)$
Ensure: LLL-reduced basis

Compute Gram–Schmidt vectors b_i^* and coefficients

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}$$

Set $B_i = \|b_i^*\|^2$
$k \leftarrow 2$
while $k \leq n$ **do**
  for $j = k-1$ down to 1 **do**
    if $|\mu_{kj}| > \frac{1}{2}$ **then**
      $r \leftarrow \lfloor \mu_{kj} \rceil$
      $b_k \leftarrow b_k - r b_j$
      for $\ell = 1$ to $j-1$ **do**
        $\mu_{k\ell} \leftarrow \mu_{k\ell} - r\mu_{j\ell}$
      end for
      $\mu_{kj} \leftarrow \mu_{kj} - r$
    end if
  end for
  if $B_k < (\delta - \mu_{k,k-1}^2) B_{k-1}$ **then**
    $\mu := \mu_{k,k-1}$; $B := B_k + \mu^2 B_{k-1}$; $\mu_{k,k-1} := \mu \frac{B_{k-1}}{B}$;
    $B_k := \frac{B_{k-1} B_k}{B}$; $B_{k-1} := B$;
    $\begin{pmatrix} b_{k-1} \\ b_k \end{pmatrix} := \begin{pmatrix} b_k \\ b_{k-1} \end{pmatrix}$;
    $\begin{pmatrix} \mu_{k-1,j} \\ \mu_{k,j} \end{pmatrix} := \begin{pmatrix} \mu_{k,j} \\ \mu_{k-1,j} \end{pmatrix}$ for$j = 1, 2, \ldots, k-2$;
    $\begin{pmatrix} \mu_{i,k-1} \\ \mu_{i,k} \end{pmatrix} := \begin{pmatrix} 1 & \mu_{k,k-1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} \mu_{i,k-1} \\ \mu_{i,k} \end{pmatrix}$ for $i = k+1, k+2, \ldots, n$;
    if k > 2, then k := k-1;
    Update Gram–Schmidt data
    $k \leftarrow \max(k-1, 2)$
  else
    $k \leftarrow k+1$
  end if
end while
return Reduced basis $(b_1, \ldots, b_n)$

2.0.5 Illustration

Consider the lattice basis

$$b_1 = (0, 100), \qquad b_2 = (2, 102)$$

We apply the LLL algorithm with parameter $\delta = \frac{3}{4}$. Fiest, we will calculate Gram–Schmidt Orthogonalization

$$b_1^* = b_1 = (0, 100)$$

Calculating

$$\mu_{21} = \frac{\langle b_2, b_1^* \rangle}{\langle b_1^*, b_1^* \rangle}$$

$$\langle b_2, b_1^* \rangle = 2 \cdot 0 + 102 \cdot 100 = 10200$$

$$\langle b_1^*, b_1^* \rangle = 10000$$

$$\mu_{21} = \frac{10200}{10000} = 1.02$$

Size Reduction as follows. Since

$$|\mu_{21}| > \frac{1}{2}$$

we are calculating size reduction:

$$b_2 = b_2 - \lfloor \mu_{21} \rceil b_1$$

Nearest integer to 1.02 is 1.

$$b_2 = (2, 102) - (0, 100)$$

$$b_2 = (2, 2)$$

We have to recompute Gram–Schmidt

$$b_1^* = (0, 100)$$

$$\mu_{21} = \frac{(2, 2) \cdot (0, 100)}{10000}$$

$$= \frac{200}{10000}$$

$$= 0.02$$

$$b_2^* = b_2 - \mu_{21} b_1^*$$

$$= (2,2) - 0.02(0,100)$$

$$= (2,2) - (0,2)$$

$$= (2,0)$$

Checking the Lovász Condition. The Lovász condition is

$$\delta \|b_1^*\|^2 \leq \|b_2^*\|^2 + \mu_{21}^2 \|b_1^*\|^2$$

Compute norms:

$$\|b_1^*\|^2 = 10000$$

$$\|b_2^*\|^2 = 4$$

By calculation, we will get

$$7500 > 8$$

The Lovász condition fails. Hence we swap the vectors. We have to Swap Basis

$$b_1 = (2,2), \qquad b_2 = (0,100)$$

Doing the Gram–Schmidt again.

$$b_1^* = (2,2)$$

$$\|b_1^*\|^2 = 8$$

$$\mu_{21} = \frac{(0,100)\cdot(2,2)}{8}$$

$$= \frac{200}{8}$$

$$= 25$$

We have to do size reduction again

$$b_2 = b_2 - 25b_1$$

$$= (0,100) - 25(2,2)$$

$$= (0,100) - (50,50)$$

$$= (-50,50)$$

Final Lovász Check.

$$\mu_{21} = 0$$

$$b_2^* = (-50,50)$$

$$\|b_1^*\|^2 = 8$$

$$\|b_2^*\|^2 = 5000$$

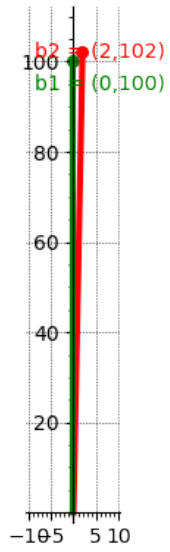
Check:

$$\frac{3}{4} \times 8 = 6$$

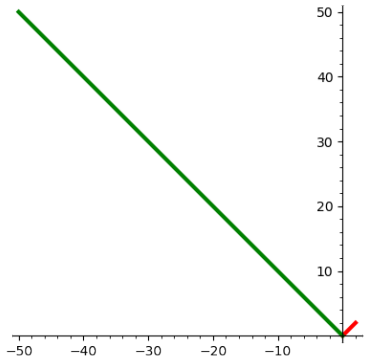
$$6 \leq 5000$$

Thus the Lovász condition is satisfied.Final LLL Reduced Basis is

$$(2,2),\ (-50,50)$$



(a) Image of Input Vectors



(b) Image of Output Vectors

Figure 2.1: LLL algorithm

2.0.6 Complexity

To prove that the algorithm terminates we introduce the quantities

$$d_i = \det\big((b_j, b_\ell)\big)_{1\le j,\ell\le i}$$

for $0 \le i \le n$.This is called Gram matrix with ij th entry as (b_j, b_ℓ). d_i is just determinant of that.
It is easily checked that

$$d_i = \prod_{j=1}^{i} \|b_j^*\|^2 \tag{2.1}$$

for $0 \le i \le n$. Hence the d_i are positive real numbers. Notice that $d_0 = 1$ and $d_n = d(L)^2$. Put

$$D = \prod_{i=1}^{n-1} d_i.$$

Proposition 2.0.1. *Let $L \subset \mathbb{Z}^n$ be a lattice with basis $b_1, b_2, \ldots, b_n$, and let $B \in \mathbb{R}$, $B \geq 2$, be such that $|b_i|^2 \leq B$ for $1 \leq i \leq n$. Then the number of arithmetic operations needed by the basis reduction algorithm described in above is $O(n^4 \log B)$, and the integers on which these operations are performed each have binary length $O(n \log B)$.*

Proof. We first compute the number of times that we pass through cases 1 and 2. In the beginning of the algorithm we have $d_i \leq B^i$. So, $D \leq B^{n(n-1)/2}$.This is because, we have to go add 1 to $n-l$, so its $n(n-1)/2$. Throughout the algorithm we have $D \geq 1$, since $d_i \in \mathbb{Z}$ and $d_i > 0$. And, the number of times that we pass through case 1 is $O(n^2 \log B)$. This is by taking log of D in that inequality . And the same applies to case 2. The initialization of the algorithm takes $O(n^3)$ arithmetic operations with rational numbers. This is nothing but time complexity of Gram-Schmidt orthogonalization process. we need $O(n)$ arithmetic operations for case 1. In case 2 we have to deal with $O(n)$ values of l, that each require $O(n)$ arithmetic operations. Since we pass through these cases $O(n^2 \log B)$ times we arrive at a total of $O(n^4 \log B)$ arithmetic operations. For other details, [1].

2.0.7 Applications

There are two applications of our reduction algorithm. The first is to solve simultaneous diophantine approximation. Let n be a positive integer, $\alpha_1, \alpha_2, \ldots, \alpha_n$ real numbers, and $\varepsilon \in \mathbb{R}, 0 < \varepsilon < 1$. It is a classical theorem [5] that there exist integers $p_1, p_2, \ldots, p_n, q$ satisfying

$$|p_i - q\alpha_i| \leq \varepsilon \quad \text{for} \quad 1 \leq i \leq n,$$
$$1 \leq q \leq \varepsilon^{-n}.$$

We will see that there exists a polynomial-time algorithm to find integers that satisfy a slightly weaker condition using LLL algorithm. The second is to factorize polynomials.

Proposition 2.0.1. *There exists a polynomial-time algorithm that, given a positive integer n and rational numbers $\alpha_1, \alpha_2, \ldots, \alpha_n, \varepsilon$ satisfying $0 < \varepsilon < 1$, finds integers $p_1, p_2, \ldots, p_n, q$ for which*

$$|p_i - q\alpha_i| \leq \varepsilon \qquad \text{for } 1 \leq i \leq n,$$
$$1 \leq q \leq 2^{n(n+1)/4}\varepsilon^{-n}.$$

Proof. Let L be the lattice of rank $n+1$ spanned by the columns of the $(n+1) \times (n+1)$-matrix

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & -\alpha_1 \\ 0 & 1 & \cdots & 0 & -\alpha_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -\alpha_n \\ 0 & 0 & \cdots & 0 & 2^{-n(n+1)/4}\varepsilon^{n+1} \end{pmatrix}.$$

The inner product of any two columns is rational, so by above arguments there is a polynomial-time algorithm to find a reduced basis $b_1, b_2, \ldots, b_{n+1}$ for L. We have

$$\|b_1\| \leq 2^{n/4}\, d(L)^{1/(n+1)} = \varepsilon.$$

This is because, determinant of the above matrix is $2^{-n(n+1)/4}\varepsilon^{n+1}$. Since $b_1 \in L$, we can express b_1 as

$$b_1 = (p_1 - q\alpha_1,\ p_2 - q\alpha_2,\ \ldots,\ p_n - q\alpha_n,\ q \cdot 2^{-n(n+1)/4}\varepsilon^{n+1})^T$$

with $p_1, p_2, \ldots, p_n, q \in \mathbb{Z}$. Now,

$$|p_i - q\alpha_i| \leq \varepsilon \qquad \text{for } 1 \leq i \leq n,$$

$$|q| \leq 2^{n(n+1)/4}\varepsilon^{-n}.$$

From $\varepsilon < 1$ and $b_1 \neq 0$, q is non-zero. We can get $q > 0$ by shifting b_1 by $-b_1$.

Finding Linear Relation Using LLL Construction

Given

$$\alpha = 11.36795, \qquad \varepsilon = 10^{-4}, \qquad n = 1$$

We will now do Lattice Construction. For $n = 1$, the lattice is generated by the columns of

$$\begin{pmatrix} 1 & -\alpha \\ 0 & 2^{-n(n+1)/4}\varepsilon^{n+1} \end{pmatrix}$$

Compute

$$2^{-n(n+1)/4} = 2^{-1/2}$$

$$\varepsilon^{n+1} = (10^{-4})^2 = 10^{-8}$$

Thus

$$2^{-1/2}\varepsilon^2 = \frac{1}{\sqrt{2}} \times 10^{-8} \approx 7.0710678 \times 10^{-9}$$

Hence the lattice basis matrix is

$$B = \begin{pmatrix} 1 & -11.36795 \\ 0 & 7.0710678 \times 10^{-9} \end{pmatrix}$$

Any lattice vector looks like this:

$$b = p \begin{pmatrix} 1 \\ 0 \end{pmatrix} + q \begin{pmatrix} -\alpha \\ 7.0710678 \times 10^{-9} \end{pmatrix}$$

$$b = \begin{pmatrix} p - q\alpha \\ q \cdot 7.0710678 \times 10^{-9} \end{pmatrix}$$

Now, we will find integers p, q. We require

$$|p - q\alpha| \leq \varepsilon$$

A valid solution is

$$q = 5922, \qquad p = 67321$$

The verification as follows:

$$|67321 - 5922 \times 11.36795| = 9.999999 \times 10^{-5}$$

which satisfies

$$|p - q\alpha| \leq 10^{-4}$$

Resulting lattice vector is

$$b_1 = \begin{pmatrix} 9.999999 \times 10^{-5} \\ 4.18748 \times 10^{-5} \end{pmatrix}$$

So, the desired linear relation is

$$67321 - 5922\alpha \approx 0$$

Thus

$$\alpha \approx \frac{67321}{5922} = 11.36795001688$$

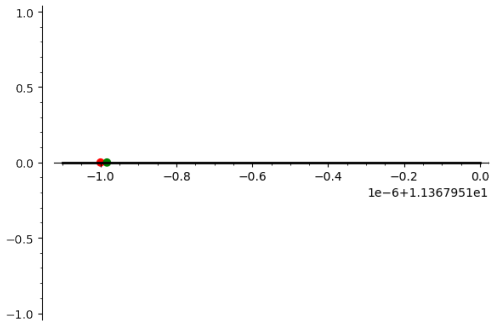


Figure 2.2: Example image of approximation where red color is α and green is our approximation

Algebraicity Test using LLL

We can also use LLL for finding $\mathbb{Q}$- linear relation for given real numbers.So, applying that to α^i for all $\{1, 2, ..n\}$ Let $\alpha = \sqrt{2}$. Our goal is to find integers m_0, m_1, m_2 such that

$$m_0 + m_1\alpha + m_2\alpha^2 \approx 0$$

The lattice construction as follows: Choose $c = 10^6$ and define

$$b_1 = (1, 0, 0, c)$$

$$b_2 = (0, 1, 0, c\alpha)$$

$$b_3 = (0, 0, 1, c\alpha^2)$$

The general vectors looks like this:

$$b = (m_0, m_1, m_2, c(m_0 + m_1\alpha + m_2\alpha^2))$$

We will find (m_0, m_1, m_2) using LLL. Let

$$\alpha = \sqrt{2} \approx 1.41421356$$

We seek integers m_0, m_1, m_2 such that

$$m_0 + m_1\alpha + m_2\alpha^2 \approx 0$$

Choose $c = 10^6$. Define basis vectors:

$$b_1 = (1, 0, 0, c)$$

$$b_2 = (0, 1, 0, c\alpha)$$

$$b_3 = (0, 0, 1, c\alpha^2)$$

Numerically:

$$b_1 = (1, 0, 0, 1000000)$$

$$b_2 = (0, 1, 0, 1414213)$$

$$b_3 = (0, 0, 1, 2000000)$$

We consider the basis

$$B = \begin{pmatrix} 1 & 0 & 0 & 1000000 \\ 0 & 1 & 0 & 1414213 \\ 0 & 0 & 1 & 2000000 \end{pmatrix}$$

$$b_1 = (1, 0, 0, 1000000), \quad b_2 = (0, 1, 0, 1414213), \quad b_3 = (0, 0, 1, 2000000)$$

Fisrt we will compute μ_{21}

$$\mu_{21} \approx 1.414213$$

Then, reduce b_2

$$b_2 = b_2 - b_1 = (-1, 1, 0, 414213)$$

Computing μ_{31}

$$\mu_{31} = 2$$

Reduce b_3

$$b_3 = b_3 - 2b_1 = (-2, 0, 1, 0)$$

Swapping the vectors

$$b_1 \leftrightarrow b_3$$

$$b_1 = (-2, 0, 1, 0)$$

The final basis as follows:

$$(-2, 0, 1, 0),\ (-1, 1, 0, 414213),\ (1, 0, 0, 1000000)$$

The desired results is the numbers in First column.

$$-2 + \alpha^2 = 0$$

$$x^2 - 2 = 0$$

2.0.8 Factoring Polynomials

Another application of LLL is that factoring polynomials with rational co-effients. We introduce two concepts which is useful for our algorithm. First is Berlekamp's Algorithm and another is Hensel's lemma. We will use when we need. For both topics, we refer [2] and [18].

Berlekamp's Algorithm

Berlekamp's Algorithm, which is indeed factoring algorithm for ploynomials in finite field. The idea of the algorithm is same as Lenstra's Algorithm for factorization of natural number which uses elliptic curves. The idea is to construct a situation that we can use GCD of two polynoials, so that we can get a factor. Rather than working with factoring which is NP-hard problem, we are using GCD which is polynomial time. To construct a situation which ensures a given polynomial and another polynomial which has a non-trivial factor is berlekamp's algorithm. So, this algorithm is nothing but a construction of some special polynomial from given polynomial. The Berlekamp factorization algorithm for factoring polynomials in $\mathbb{F}_p[x]$. The time complexity of this algorithm is dependant on the time complexity of computing greatest common divisors in $\mathbb{F}_p[x]$ by the Euclidean Algorithm and on the efficiency of row-reduction matrix algorithms for solving systems of linear equations. Let $f(x) \in \mathbb{F}_p[x]$ be a monic polynomial of degree n and let

$$f(x) = p_1(x)p_2(x)\cdots p_k(x),$$

where $p_1(x), p_2(x), \ldots, p_k(x)$ are powers of distinct monic irreducibles in $\mathbb{F}_p[x]$. To express $f(x)$ as a multiplication of irreducible polynomials in $\mathbb{F}_p[x]$ it enough to find the factors $p_1(x), \ldots, p_k(x)$. If $p(x) = q(x)^N \in \mathbb{F}_p[x]$ with $q(x)$ monic and irreducible, then $q(x)$ can be determined from $p(x)$ by checking for pth powers and by computing greatest common

divisors with derivatives. Let $g(x) \in \mathbb{F}_p[x]$ be any polynomial of degree $< n$. Denote by $R(h(x))$ the remainder of $h(x)$ after division by $f(x)$. Then the following are equivalent:

1. $R(g(x)^p) = g(x)$.
2. $f(x)$ divides $g(x)(g(x)-1)\cdots(g(x)-(p-1))$.
3. $p_i(x)$ divides the product in (b) for $i = 1, 2, \ldots, k$.
4. For each $i = 1, 2, \ldots, k$ there is an $s_i \in \mathbb{F}_p$ such that $p_i(x)$ divides $g(x) - s_i$, i.e., $g(x) \equiv s_i \pmod{p_i(x)}$.

The polynomials $g(x)$ of degree $< n$ satisfying the equivalent conditions form a vector space V over $\mathbb{F}_p$ of dimension k. Let

$$g(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1} \in V.$$

For $j = 0, 1, \ldots, n-1$ let

$$R(x^{p^j}) = a_{0,j} + a_{1,j}x + \cdots + a_{n-1,j}x^{n-1},$$

and let A be the $n \times n$ matrix

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix}. \qquad (*)$$

The condition is equivalent to $R(g(x)^p) = g(x)$

$$(A - I)B = 0 \qquad (**)$$

where B is the column matrix with entries $b_0, b_1, \ldots, b_{n-1}$. The rank of the matrix $A - I$ is $n - k$. This already suffices to determine if $f(x)$ is irreducible, without actually determining the factors. Let $g_1(x), g_2(x), \ldots, g_k(x)$ be a basis of solutions to $(**)$ (so a basis for V), where we may take $g_1(x) = 1$. Beginning with $w(x) = f(x)$, compute the greatest common divisor

$$(w(x),\ g_i(x) - s)$$

for $i = 2, 3, \ldots, k$ and $s \in \mathbb{F}_p$ for every factor of $f(x)$ already computed. By For each $i = 1, 2, \ldots, k$ there is an $s_i \in \mathbb{F}_p$ such that $p_i(x)$ divides $g(x) - s_i$, i.e., $g(x) \equiv s_i \pmod{p_i(x)}$, every factor $p_i(x)$ of $f(x)$ divides such a g.c.d. This process terminates when k relatively prime factors are found.

Hensel's Lemma

Hensel's lemma is also called hensel's lifting lemma, since we are lifting our solutions from mod p^f to mod p^{f+1}. Suppose that for some $f \geq 1$, we have computed an integer b satisfying the congruence

$$b^2 \equiv a \pmod{p^f},$$

and we want to find an integer c satisfying the congruence

$$c^2 \equiv a \pmod{p^{f+1}}.$$

Clearly, if $c^2 \equiv a \pmod{p^{f+1}}$, then

$$c^2 \equiv a \pmod{p^f},$$

and so $c \equiv \pm b \pmod{p^f}$. So let us set $c = b + p^f h$, and solve for h. We have

$$c^2 \equiv (b + p^f h)^2 = b^2 + 2bp^f h + p^{2f} h^2 \equiv b^2 + 2bp^f h \pmod{p^{f+1}}.$$

So we want to find an integer h satisfying the linear congruence

$$2bp^f h \equiv a - b^2 \pmod{p^{f+1}}$$

Since $p \nmid 2b$, we have $\gcd(2bp^f, p^{f+1}) = p^f$. Furthermore, since $b^2 \equiv a \pmod{p^f}$, we have $p^f \mid (a - b^2)$. Therefore, it has a unique solution h modulo p, which we can efficiently computable. By iterating the above procedure, starting with a square root of a modulo p, we can quickly find a square root of a modulo p^e. To know more, [14] , [15] and [16].

Idea of Factoring Algorithm

The Polynomial with integer Coefficients is given as a input. We have no idea what is it's factors are. So, we will start from the known problem of finding divisors, quotients and G.c.d . Suppose we know a given polynomial f and some other polynomial g has common root, then we are done. $gcd(f, g)$ will give you the common factor. But, finding such g is

itself a hard problem which is same as factoring. Instead of creating new polynomial, we can try to create polynomial from f itself. What are the obvious options? It's differentiation of f and integration of f. But, integration of f won't reveal anything about factors of a polynomial since it can be changed upto constants. So, we can take differentiation. So, what will the $gcd(f, f^{'})$ reveals? It exactly means they both have common factors. If $gcd(f, f^{'}) \neq 1$, then they have non-trivial common factors. So, we now have one factor of f. But, it's not always the case. What can we do if $gcd(f, f^{'}) = 1$? That's where the real problem is! We have efficient algorithms for factoring Polynomials over a finite fields. Can we use it somehow to solve this problem? Yes, but that is not straightforward. Suppose we know the Unique factorization of f in a finite field, will it be same in integers? The answer is no. Counterexample is $x^4 + 1$ is irreducible in $\mathbb{Z}[x]$, but reducible in every $\mathbb{Z}_p$. So, we need relations between polynomials in integers and finte fields about reducibility. Now, suppose we the relations and $f = f_1 f_2$, where f_1 and f_2 are polynomials in integers. And we know the complete factorizations of f_1 in $\mathbb{Z}[x]$ and $(f_2 mod p)$ in $\mathbb{Z}_p[x]$. Now, how one can factor the polynomial? We want f to be factored in $\mathbb{Z}[x]$. So, we need our algorithm to start with $f_1 = 1$ and $f_2 = f$.And, end with(output) $f_1 = \pm f$ and $f_2 = \pm 1$.The algorithm stops when this happens. In between, we want the below process happens:

f_1	f_2
1	f
h_0	$\frac{f}{h_0}$
$\vdots$	$\vdots$
$\pm f$	± 1

where h_o is the irreducible factor of f. And, each row of the table is the steps of algorithm with first row of the table as input and last row of the table as output. But, how we can know the factor of f_2 in $\mathbb{Z}_p[x]$ is a factor of f_2 in $\mathbb{Z}[x]$? So, we need to answer this questions below to complete our algorithm. We have to play with polynomilas in $\mathbb{Z}[x]$ and $\mathbb{Z}_p[x]$ and $\mathbb{Z}_{p^k}[x]$ for some prime p and positive integer k. This is because our algorithm depends on these things. So, we have to know answers to few questions regarding reducibility, factors of polynomials in $\mathbb{Z}[x]$ and $\mathbb{Z}_p[x]$ and $\mathbb{Z}_{p^k}[x]$.

(1) If $h \mid f$ and $g \mid f$ in $\mathbb{Z}[x]$, when $h \mid g$ in $\mathbb{Z}[x]$?
(2) If f is reducible in $\mathbb{Z}_p[x]$, when is it reducible in $\mathbb{Z}[x]$?
(3) If $h(x) \mid g(x)$ in $\mathbb{Z}_p[x]$ when $h(x) \mid g(x)$ in $\mathbb{Z}_{p^k}[x]$?

(4) If $h(x) \mid g(x)$ in $\mathbb{Z}_{p^k}[x]$ when $h(x) \mid g(x)$ in $\mathbb{Z}_p[x]$?

We will answer these questions and why it matters below: Let p a prime number and by k a positive integer. We take $\mathbb{Z}/p^k\mathbb{Z}$ for the ring of integers modulo p^k, and $\mathbb{F}_p$ for the field $\mathbb{Z}/p\mathbb{Z}$. For $g = \sum_i a_i X^i \in \mathbb{Z}[X]$ we denote by $(g \bmod p^k)$ the polynomial

$$\sum_i (a_i \bmod p^k) X^i \in (\mathbb{Z}/p^k\mathbb{Z})[X].$$

We fix a polynomial $f \in \mathbb{Z}[X]$ of degree n, with $n > 0$, and a polynomial $h \in \mathbb{Z}[X]$ that has the following properties:

- $(1*)$ h has leading coefficient 1,
- $(2*)$ $(h \bmod p^k)$ divides $(f \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$,
- $(3*)$ $(h \bmod p)$ is irreducible in $\mathbb{F}_p[X]$,
- $(4*)$ $(h \bmod p)^2$ does not divide $(f \bmod p)$ in $\mathbb{F}_p[X]$.

We put $l = \deg(h)$; so $0 < l \leq n$.

Proposition 2.0.1. *The polynomial f has an irreducible factor h_0 in $\mathbb{Z}[X]$ for which $(h \bmod p)$ divides $(h_0 \bmod p)$, and this factor is uniquely determined up to sign. Further, if g divides f in $\mathbb{Z}[X]$, then the following three assertions are equivalent:*

1. *$(h \bmod p)$ divides $(g \bmod p)$ in $\mathbb{F}_p[X]$,*
2. *$(h \bmod p^k)$ divides $(g \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$,*
3. *h_0 divides g in $\mathbb{Z}[X]$.*

In particular $(h \bmod p^k)$ divides $(h_0 \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$.

Proof. First we will prove (iii) $\implies$ (i). Since, h_0 divides g in $\mathbb{Z}_p[X]$. By, transitivity of division (i) is true. Now we will prove (ii) $\implies$ (i).

$$(h \bmod p^k) \mid (g \bmod p^k) \quad \text{in } (\mathbb{Z}/p^k\mathbb{Z})[X].$$

So there exists some polynomial

$$t \in (\mathbb{Z}/p^k\mathbb{Z})[X]$$

such that

$$g \bmod p^k = (h \bmod p^k)\,(t \bmod p^k)$$

Reducing further modulo p, there is a natural ring homomorphism:

$$\mathbb{Z}/p^k\mathbb{Z} \longrightarrow \mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p,$$

which extends to polynomials. Applying this :

$$g \bmod p = (h \bmod p)\,(t \bmod p).$$

.Now assume (i) and prove (iii). From (i) and (4*), we know $(h \bmod p)$ does not divide $(f/g \bmod p)$ in $\mathbb{F}_p[X]$. So, h_0 does not divide f/g in $\mathbb{Z}[X]$, so it has to divide g. We are done.

In this section,we fix an integer m with $m \geq \ell$, and L be the set of all polynomials in $\mathbb{Z}[X]$ of degree $\leq m$ that, if we take modulo p^k,are divisible by $(h \bmod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$. This is a subset of the $(m+1)$-dimensional real vector space

$$\mathbb{R} + \mathbb{R}X + \cdots + \mathbb{R}X^m.$$

This vector space is identified with $\mathbb{R}^{m+1}$ by identifying

$$\sum_{i=0}^{m} a_i X^i \;\longleftrightarrow\; (a_0, a_1, \ldots, a_m).$$

The length of $|\sum_{i=0}^{m} a_i X^i|$ is equal to the length in of $(a_0, a_1, \ldots, a_m)$ in Euclidean metric. One can understand that L is a lattice in $\mathbb{R}^{m+1}$ and that a basis of L is given by

$$\{\, p^k X^i : 0 \leq i < \ell \,\} \;\cup\; \{\, hX^j : 0 \leq j < m - \ell \,\}.$$

The lattice L is defined by:

$$L = \left\{ f \in \mathbb{Z}[X] \;\middle|\; \deg(f) \leq m,\ f \equiv 0 \pmod{h \pmod{p^k}} \right\}.$$

That is same as

$$f \in L \iff f = hq + p^k r, \quad q, r \in \mathbb{Z}[X],\ \deg(f) \leq m.$$

Any $f \in L$ can be written as

$$f = hq + p^k r$$

with

$$\deg(q) \le m - \ell, \quad \deg(r) \le \ell - 1.$$

These bounds is because, hq has degree $\le m$, we must have $\deg(q) \le m - \ell$. The polynomial r accounts for the lower-degree part modulo h, hence $\deg(r) \le \ell - 1$. To understand exactly the Basis elements, We can split into two parts: Multiples of p^k:

$$p^k,\ p^k X,\ p^k X^2,\ \ldots,\ p^k X^{\ell-1}.$$

Multiples of h:

$$h,\ Xh,\ X^2 h,\ \ldots,\ X^{m-\ell} h.$$

Thus, a basis of L is

$$\left\{p^k X^i : 0 \le i < \ell\right\}\ \cup\ \left\{hX^j : 0 \le j \le m - \ell\right\}.$$

We can check the spanning property and linear independence. Every $f \in L$ can be written as

$$f = hq + p^k r.$$

Here, q gives a linear combination of $h, Xh, X^2 h, \ldots$. r gives a linear combination of $p^k, p^k X, p^k X^2, \ldots$ Hence these vectors span L. Consider coefficient vectors in $\mathbb{R}^{m+1}$: $p^k X^i$ are independent which is obvious and hX^j is also. From above arguments,

$$d(L) = p^{k\ell}.$$

We will derive this now. The Structure of the basis matrix: Write the basis vectors as rows (or columns). The matrix has the following form. First ℓ vectors:

$$(p^k, 0, \ldots), \quad (0, p^k, \ldots), \quad \ldots$$

which form the diagonal block

$$p^k I_\ell.$$

Remaining vectors:

$$h,\ Xh,\ X^2 h,\ \ldots$$

Thus, the matrix is triangular, with diagonal entries given by: the first ℓ entries are p^k and the remaining entries are the leading coefficient of h.

$$\begin{pmatrix} p^k & 0 & & 0 & 0 & \cdots & 0 \\ 0 & p^k & & 0 & 0 & \cdots & 0 \\ & & \ddots & \vdots & 0 & \cdots & 0 \\ & & & p^k & 0 & \cdots & 0 \\ a_0 & a_1 & \cdots & a_{\ell-1} & 1 & \cdots & 1 \\ 0 & a_0 & \cdots & a_{\ell-2} & a_{\ell-1} & \cdots & 1 \\ \vdots & \vdots & & \vdots & \vdots & & \\ 0 & 0 & \cdots & a_0 & a_1 & \cdots & 1 \end{pmatrix}$$

The Determinant of the matrix is $p^{k\ell}$.

Next, we need a another condition that ensures relation between divisiblity in $\mathbb{Z}_p[X]$ and $\mathbb{Z}[X]$. One can think why we need more conditions if we already have condtion (2.0.1). So, the below condition is used to determine degree of the polynomials we are playing with. If the following are true,

$$\begin{aligned} &f \text{ has } h_0 \text{ as irreducible factor in } \mathbb{Z}[x], \\ &h \mid h_0 \text{ in } \mathbb{Z}_p[x], \\ &g \mid f \text{ in } \mathbb{Z}[x] \text{ and,} \\ &h \mid b \text{ in } \mathbb{Z}_{p^k}[x]. \end{aligned}$$

When $h_0 \mid b$? The following preposition answers this question also.

Proposition 2.0.2. *Let $b \in L$ satisfy*

$$p^{kl} > |f|^m |b|^n. \tag{2.2}$$

Then b is divisible by h_0 in $\mathbb{Z}[X]$, and in particular $\gcd(f, b) \neq 1$.

Proof. From [1].

We will discuss about partial converse statement of above preposition. We need that to determine the degree of a polynomial in our algorithm.

Proposition 2.0.3. *Let p, k, f, n, h, ℓ be as at the beginning of this section, h_0 as in 2.0.1, and m, L as defined above. Suppose that $b_1, b_2, \ldots, b_{r+\ell}$ is a reduced basis for L, and that*

$$p^{kl} > 2^{\frac{mn}{2}} \binom{2m}{m}^{\frac{n}{2}} |f|^{m+n} \tag{2.3}$$

Then we have $\deg(h_0) \leq m$ *if and only if*

$$|b_1| < \left(\frac{p^{kl}}{|f|^m}\right)^{1/n}. \tag{2.4}$$

Proof. If part is obvious from 2.0.2. Since, $h_0 \mid b$ in Integer Ring. degree of h_0 is less than or equal to degree of b. And, degree of b is less than or equal to l and l is less tahn or equal to m. degree(h_0) is less than or equal to m.Now, the only if part. From 2.0.1, $h_0 \in L$. And, $|h_0| \leq \binom{2m}{m}^{\frac{1}{2}}.|f|$. By using the inequality, our result is true. Now, we will prove the inequality. To prove it, we will use 4 parts of a excercise from [11] and [8].
(i) Let $u(x) = u_n x^n + \cdots + u_0$ be a polynomial over $\mathbb{C}$, and define

$$\|u\| = \left(|u_n|^2 + \cdots + |u_0|^2\right)^{1/2}.$$

Let $u(x) = (x-\alpha)w(x)$ and $v(x) = (\overline{\alpha}x - 1)w(x)$, where $\alpha \in \mathbb{C}$ and $\overline{\alpha}$ is its complex conjugate. Then, $\|u\| = \|v\|$. Let

$$w(x) = w_m x^m + w_{m-1}x^{m-1} + \cdots + w_0.$$

Then

$$u(x) = (x-\alpha)w(x) = w_m x^{m+1} + \cdots + (w_1 - \alpha w_2)x^2 + (w_0 - \alpha w_1)x - \alpha w_0.$$

Similarly,

$$v(x) = (\overline{\alpha}x - 1)w(x) = \overline{\alpha}w_m x^{m+1} + \cdots + (\overline{\alpha}w_1 - w_2)x^2 + (\overline{\alpha}w_0 - w_1)x - w_0.$$

Calculating Norms:

$$\|u\|^2 = \sum |\text{coefficients of } u(x)|^2, \quad \|v\|^2 = \sum |\text{coefficients of } v(x)|^2.$$

Observe that each coefficient of $v(x)$ is obtained from the corresponding coefficient of $u(x)$ by multiplying by $\overline{\alpha}$ and/or rearranging terms. Using the identity

$$|a - \alpha b|^2 = |\overline{\alpha}a - b|^2 \quad \text{for all } a, b \in \mathbb{C},$$

we see that corresponding terms in $\|u\|^2$ and $\|v\|^2$ are equal.Hence,

$$\|u\|^2 = \|v\|^2 \implies \|u\| = \|v\|.$$

(ii) Let

$$u(x) = u_n \prod_{j=1}^{n} (x - \alpha_j)$$

be the complete factorization of a polynomial over $\mathbb{C}$. Define

$$M(u) = |u_n| \prod_{j=1}^{n} \max(1, |\alpha_j|), \quad \|u\| = \left(\sum_{k=0}^{n} |u_k|^2 \right)^{1/2}.$$

Then

$$M(u) \leq \|u\|.$$

Define a new polynomial

$$v(x) = u_n \prod_{j=1}^{n} (x - \beta_j),$$

where

$$\beta_j = \begin{cases} \alpha_j, & \text{if } |\alpha_j| \leq 1, \\ \frac{1}{\overline{\alpha_j}}, & \text{if } |\alpha_j| > 1. \end{cases}$$

By construction, we have $|\beta_j| \leq 1$ for all j. From part (a), replacing a factor $(x - \alpha_j)$ with $(\overline{\alpha_j} x - 1)$ does not change the norm. Hence,

$$\|v\| = \|u\|.$$

Now write

$$v(x) = v_n x^n + \cdots + v_0.$$

Since $|\beta_j| \leq 1$, all roots of v lie in the closed unit disk. Hence, expanding the product gives

$$|v_0| = |u_n| \prod_{j=1}^{n} |\beta_j| \leq |u_n|.$$

But more importantly, from the construction,

$$M(u) = |u_n| \prod_{j=1}^{n} \max(1, |\alpha_j|) = |v_n|.$$

Therefore,

$$M(u) = |v_n|.$$

Finally, since the Euclidean norm dominates any coefficient,

$$|v_n| \leq \left(\sum_{k=0}^{n} |v_k|^2 \right)^{1/2} = \|v\|.$$

Thus,

$$M(u) \leq \|v\| = \|u\|.$$

(iii) Let

$$u(x) = u_n \prod_{k=1}^{n} (x - \alpha_k)$$

be a polynomial over $\mathbb{C}$. Define

$$M(u) = |u_n| \prod_{k=1}^{n} \max(1, |\alpha_k|).$$

Then for $0 \leq j \leq n$,

$$|u_j| \leq \binom{n-1}{j} M(u) + \binom{n-1}{j-1} |u_n|.$$

Expanding the product, the coefficient u_j satisfies

$$u_j = (-1)^{n-j} u_n \sum_{1 \leq i_1 < \cdots < i_{n-j} \leq n} \alpha_{i_1} \cdots \alpha_{i_{n-j}}.$$

Hence,

$$|u_j| \leq |u_n| \sum_{1 \leq i_1 < \cdots < i_{n-j} \leq n} |\alpha_{i_1} \cdots \alpha_{i_{n-j}}|.$$

Fix the root α_n. We split the sum into two parts:

(i) Terms not containing α_n:

$$S_1 = \sum_{1 \leq i_1 < \cdots < i_{n-j} \leq n-1} |\alpha_{i_1} \cdots \alpha_{i_{n-j}}|.$$

(ii) Terms containing α_n:

$$S_2 = \sum_{1 \leq i_1 < \cdots < i_{n-j-1} \leq n-1} |\alpha_{i_1} \cdots \alpha_{i_{n-j-1}}| \, |\alpha_n|.$$

Thus,

$$|u_j| \leq |u_n|(S_1 + S_2).$$

For any k, we have $|\alpha_k| \leq \max(1, |\alpha_k|)$. Hence,

$$|\alpha_{i_1} \cdots \alpha_{i_{n-j}}| \leq \prod_{k=1}^{n} \max(1, |\alpha_k|).$$

Therefore each term in S_1 is bounded by

$$\frac{M(u)}{|u_n|}.$$

The number of such terms is $\binom{n-1}{n-j} = \binom{n-1}{j}$. Hence,

$$|u_n| S_1 \leq \binom{n-1}{j} M(u).$$

Similarly,

$$|\alpha_{i_1} \cdots \alpha_{i_{n-j-1}}| \leq \prod_{k=1}^{n-1} \max(1, |\alpha_k|).$$

Thus each term in S_2 satisfies

$$|u_n| \, |\alpha_n| \, |\alpha_{i_1} \cdots \alpha_{i_{n-j-1}}| \leq |u_n|.$$

The number of such terms is $\binom{n-1}{n-j-1} = \binom{n-1}{j-1}$. Hence,

$$|u_n| S_2 \leq \binom{n-1}{j-1} |u_n|.$$

Adding the two bounds,

$$|u_j| \leq \binom{n-1}{j} M(u) + \binom{n-1}{j-1} |u_n|.$$

(iv) Combine these results to prove that if $u(x) = v(x)w(x)$ and

$$v(x) = v_m x^m + \cdots + v_0,$$

where $u, v, w \in \mathbb{Z}[x]$, then the coefficients of v satisfy

$$|v_j| \leq \binom{m-1}{j} \|u\| + \binom{m-1}{j-1} |u_n|, \qquad 0 \leq j \leq m.$$

Now, we will discuss how above inequality is used to prove our inequality.

Let $f(x) = h_0(x)y(x)$ with $\deg f \leq 2m$ and $\deg h_0 \leq m$. Then

$$\|h\| \leq \binom{2m}{m}^{1/2} \|f\|.$$

From part (c), for $0 \leq j \leq m$,

$$|h_j| \leq \binom{m-1}{j} M(f) + \binom{m-1}{j-1} |f_n|.$$

Since $M(f) \leq \|f\|$ and $|f_n| \leq \|f\|$, we obtain

$$|h_j| \leq \left(\binom{m-1}{j} + \binom{m-1}{j-1} \right) \|f\| = \binom{m}{j} \|f\|.$$

Hence,

$$\|h_0\|^2 = \sum_{j=0}^{m} |h_j|^2 \leq \|f\|^2 \sum_{j=0}^{m} \binom{m}{j}^2.$$

Using the identity

$$\sum_{j=0}^{m} \binom{m}{j}^2 = \binom{2m}{m},$$

we get

$$\|h_0\|^2 \leq \binom{2m}{m} \|f\|^2.$$

Taking square roots gives

$$\|h_0\| \leq \binom{2m}{m}^{1/2} \|f\|.$$

Our next preposition, tell us what is h_0 when it's degree is less than or equal to m under given conditions.

Proposition 2.0.4. *Let the notation and the hypotheses be the same as in 2.0.3, and assume in addition that there exists an index $j \in \{1, 2, \ldots, m+1\}$ for which*

$$\|b_j\| < \left(\frac{p^{kl}}{|f|^m} \right)^{1/n} \tag{2.5}$$

Let t be the largest such j. Then we have

$$\deg(h_0) = m + 1 - t,$$

$$h_0 = \gcd(b_1, b_2, \ldots, b_t),$$

.

Proof. From 2.0.1.

Since, our factoring algorithm needs a relation between polynomials in different rings. We introduced several preposition. Now, we are going to establish 2 algorithm which uses the above relations to help factoring. The first one as follows:

Algorithm 2 LLL Degree Test and Factor Finding

Require: $f \in \mathbb{Z}[X]$, $\deg(f) = n$; prime p; integer k; $h \in \mathbb{Z}[X]$ satisfying (1*)–(4*); integer $m > l = \deg(h)$, and inequality $p^{kl} > 2^{(mn)/2} \cdot \binom{2m}{m}^{n/2} \cdot |f|^{m+n}$ is satisfied

Ensure: Decide whether $\deg(h_0) < m$ and compute h_0 if true

1: Construct lattice L with basis:

$$\{p^k X^i : 0 \le i < l\} \cup \{hX^j : 0 \le j < m - l\}$$

2: Apply LLL reduction to obtain basis $b_1, b_2, \ldots, b_{m+1}$

3: **if** $\|b_1\| > \left(\dfrac{p^{kl}}{\|f\|^m}\right)^{1/m}$ **then**

4: **return** $\deg(h_0) \ge m$

5: **end if**

6: Let t be the largest index such that

$$\|b_t\| < \left(\frac{p^{kl}}{\|f\|^m}\right)^{1/m}$$

7: **return** $\deg(h_0) < m, \quad h_0 = \gcd(b_1, b_2, \ldots, b_t)$

This gcd can be calculated efficiently. The number of arithmetic operations needed by algorithm is $O(m^4 k \log p)$, and the integers on which these operations are performed each have binary length $O(mk \log p)$. It's from [1].

Our Next Algorithm determines polynomial defined in 2.0.1.

Algorithm 3 LLL-factor-2-(f, p, h)

Require: $f \in \mathbb{Z}[X]$, $\deg(f) = n$; prime p; $h \in \mathbb{Z}[X]$ satisfying (1*)–(4*)
Ensure: Irreducible factor h_0 of f corresponding to h

$l \leftarrow \deg(h)$
if $l = n$ **then**
  return f
end if
Compute the smallest integer k such that

$$p^{kl} > 2^{(n-1)n/2} \cdot \binom{2(n-1)}{n-1}^{n/2} \cdot \|f\|^{2n-1}$$

Lift h using Hensel lifting such that

$$h \mid f \pmod{p^k}$$

$u \leftarrow \max\left\{i \in \mathbb{Z} \mid l \leq \frac{n-1}{2^i}\right\}$
for $m = \left\lfloor \frac{n-1}{2^u} \right\rfloor$ **to** $n-1$ **do**
  Construct lattice L using (p^k, h, m)
  Apply LLL reduction to L
  Let b be the shortest vector polynomial
  if $\gcd(f, b) \neq 1$ **then**
    return $\gcd(f, b)$
  end if
end for
return f

Then the number of arithmetic operations needed by this algorithm is $O\big(m_0(n^5+n^4 \log |f|+ n^3 \log p)\big)$, and the integers on which these operations are performed each have binary length $O\big(n^3 + n^2 \log |f| + n \log p\big)$.It follows from [1]

Now, we come back to the original problem which is to factor polynomials over integers.We will explain the algorithm now clearly:

In our algorithm, we will fine gcd and some term called Resultant repeatedly. So we will introduce gcd method and definitions. The resultant of two univariate polynomials over a field or over a commutative ring is defined as the determinant of their Sylvester matrix. More precisely, let

$$A = a_0 x^d + a_1 x^{d-1} + \cdots + a_d$$

and

$$B = b_0 x^e + b_1 x^{e-1} + \cdots + b_e$$

be nonzero polynomials of degrees d and e respectively. Notation as $R(A, B)$. The resultant

of A and B is thus the determinant

$$\begin{vmatrix} a_0 & 0 & \cdots & 0 & b_0 & 0 & \cdots & 0 \\ a_1 & a_0 & \cdots & 0 & b_1 & b_0 & \cdots & 0 \\ a_2 & a_1 & \ddots & 0 & b_2 & b_1 & \ddots & 0 \\ \vdots & \vdots & \ddots & a_0 & \vdots & \vdots & \ddots & b_0 \\ a_d & a_{d-1} & \cdots & \vdots & b_e & b_{e-1} & \cdots & \vdots \\ 0 & a_d & \ddots & \vdots & 0 & b_e & \ddots & \vdots \\ \vdots & \vdots & \ddots & a_{d-1} & \vdots & \vdots & \ddots & b_{e-1} \\ 0 & 0 & \cdots & a_d & 0 & 0 & \cdots & b_e \end{vmatrix},$$

which has e columns of a_i and d columns of b_j. The first column of a's and the first column of b's have the same length, Hence $d = e$, is here only for simplifying the display of the determinant. The other definition of Resultant as follows:
Let

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0, \qquad g(x) = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_0.$$

If $r_1, \ldots, r_n$ are the roots of f, then the resultant is

$$\mathrm{Res}(f, g) = a_n^m \prod_{i=1}^{n} g(r_i).$$

Similarly, using roots $s_1, \ldots, s_m$ of g,

$$\mathrm{Res}(f, g) = (-1)^{nm} b_m^n \prod_{j=1}^{m} f(s_j).$$

For a polynomial

$$f(x) = a_n \prod_{i=1}^{n} (x - r_i),$$

with roots $r_1, r_2, \ldots, r_n$, the discriminant is

$$\mathrm{Disc}(f) = a_n^{2n-2} \prod_{i<j} (r_i - r_j)^2.$$

$\mathrm{Disc}(f) = 0 \iff$ the polynomial has a repeated root. If all roots are distinct, then $\mathrm{Disc}(f) \neq 0$. The discriminant depends only on the coefficients(the roots). $R(f, f')$ is equal

to the product of the leading coefficient of f and the discriminant of f. To compute Gcd and Resultant we subresutant algorithm which reduces unnecessary calculations in Euclidean gcd. For more details, see [8]. Our Algorithm as follows:

Algorithm 4 Factorization in $\mathbb{Z}[X]$

Require: Primitive polynomial $f \in \mathbb{Z}[X]$, $\deg(f) > 0$
Ensure: Irreducible factorization of f
$R \leftarrow \text{Res}(f, f')$
if $R = 0$ **then**
  $g \leftarrow \gcd(f, f')$
  $f_0 \leftarrow f/g$
  Factor f_0 recursively
  Recover factorization of f using g
  return factors
end if
Choose smallest prime p such that $R \not\equiv 0 \pmod{p}$
Compute $f \bmod p$
Factor $f \bmod p$ using Berlekamp algorithm
$f_1 \leftarrow 1$
$f_2 \leftarrow f$
while $f_2 \neq \pm 1$ **do**
  Select irreducible factor h of $f_2 \bmod p$
  Find irreducible faactor $h_0 \in \mathbb{Z}[X]$ of f_2 using last algorithm
  $f_1 \leftarrow f_1 \cdot h_0$
  $f_2 \leftarrow f_2/h_0$
  Remove h from modular factors
end while
return (f_1, f_2)

The above algorithm factors any primitive polynomial $f \in \mathbb{Z}[X]$ of positive degree n into irreducible factors in $\mathbb{Z}[X]$. The number of arithmetic operations required by the algorithm is $O(n^6 + n^5 \log|f|)$, and the integers on which these operations are performed each have binary length $O(n^3 + n^2 \log|f|)$. Using the classical algorithms for the arithmetic operations we have $O(n^{12} + n^9(\log|f|)^3)$ for the number of bit operations. This can be reduced to $O(n^{9+\varepsilon} + n^{7+\varepsilon}(\log|f|)^{2+\varepsilon})$, for every $\varepsilon > 0$, if we do fast multiplication techniques. To prove this complexity, we also used sieving techniques from [12]. For more details [9], [7] and [13].Other irreducibility tests and factoring methods that depend on diophantine approximation (Cantor [4], Ferguson and Forcade [6], Brentjes [3], and Zassenhaus [17]) can also be made into polynomial time algorithms with the help of the basis reduction algorithm, LLL.

Bibliography

[1] Lenstra, A.K., Lenstra, H.W., Jr., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. 261, 515–534 (1982)

[2] Dummit, D.S., Foote, R.M.: Abstract Algebra, 3rd edn. New York: John Wiley & Sons, Inc. (2004)

[3] Brentjes, A.J.: Multi-dimensional continued fraction algorithms. Mathematical Centre Tracts 145. Amsterdam: Mathematisch Centrum 1981

[4] Cantor, D.G.: Irreducible polynomials with integral coefficients have succinct certificates. J. Algorithms 2, 385–392 (1981)

[5] Cassels, J.W.S.: An introduction to the geometry of numbers. Berlin, Heidelberg, New York: Springer 1971

[6] Ferguson, H.R.P., Forcade, R.W.: Generalization of the Euclidean algorithm for real numbers to all dimensions higher than two. Bull. Am. Math. Soc. 1, 912–914 (1979)

[7] Hardy, G.H., Wright, E.M.: An introduction to the theory of numbers. Oxford: Oxford University Press 1979

[8] Knuth, D.E.: The art of computer programming, Vol. 2, Seminumerical algorithms. Reading: Addison-Wesley 1981

[9] Lenstra, A.K.: Lattices and factorization of polynomials, Report IW 190/81. Amsterdam: Mathematisch Centrum 1981

[10] Lenstra, H.W., Jr.: Integer programming with a fixed number of variables. Math. Oper. Res. (to appear)

[11] Mignotte, M.: An inequality about factors of polynomials. Math. Comp. 28, 1153–1157 (1974)

[12] Pritchard, P.: A sublinear additive sieve for finding prime numbers. Comm. ACM 24, 18–23 (1981)

[13] Barkley Rosser, J., Schoenfeld, L.: Approximate formulas for some functions of prime numbers. Ill. J. Math. 6, 64–94 (1962)

[14] Yun, D.Y.Y.: The Hensel lemma in algebraic manipulation. Cambridge: MIT 1974; reprint: New York: Garland 1980

[15] Zassenhaus, H.: On Hensel factorization I. J. Number Theory 1, 291–311 (1969)

[16] Zassenhaus, H.: A remark on the Hensel factorization method. Math. Comp. 32, 287–292 (1978)

[17] Zassenhaus, H.: A new polynomial factorization algorithm (unpublished manuscript, 1981)

[18] Shoup, V.: A Computational Introduction to Number Theory and Algebra, 2nd edn. Cambridge: Cambridge University Press (2009)