El algoritmo LLL

The LLL algorithm

Patricia de Armas González

Trabajo de Fin de Grado

Sección de Matemáticas

Facultad de Ciencias

Universidad de La Laguna

La Laguna, 14 de julio de 2016

Dra. Dña. **Margarita Rivero Álvarez**, con N.I.F. 42773834K profesor Titular de Universidad adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

"*El algoritmo LLL.*"

ha sido realizada bajo su dirección por Dña. **Patricia de Armas González**, con N.I.F. 54114902L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2016.

Agradecimientos

A Margarita por su paciencia, su labor inestimable en el desarrollo de este trabajo y su eterna sonrisa.
A mi familia por su apoyo y su confianza ciega en mí.
A mi amigo David por su ayuda, tanto en la realización de esta memoria como a lo largo de la carrera.
A ti, porque eres la fuerza que me impulsa en cada paso que doy, la cuerda que no me deja caer jamás; porque eres mi vida, mi mundo y mi felicidad.

Resumen

El objetivo de este trabajo es el estudio del algoritmo dado en 1981 por Lenstra-Lenstra y Lovász (LLL) para calcular una base reducida de una red (o retículo), y su aplicación a la factorización de polinomios con coeficientes enteros [4].

En primer lugar se introduce el concepto de red en general y se estudia el caso de las redes de dimensión dos, analizando el algoritmo dado por Gauss para obtener una base minimal, en el que subyace la idea del algoritmo LLL.

Después se analiza el proceso de ortogonalización de Gram-Schmidt ya que nos interesa vectores de componentes enteras. A continuación se estudia el algoritmo LLL demostrando que es polinómico respecto a la dimensión de la red.

Por último se recoge sin demostración el algoritmo dado por Cantor y Zassenhaus [1] para la factorización de polinomios con coeficientes enteros, el cual tiene un coste exponencial en el grado del polinomio, y analizamos cómo el uso del LLL permite reducir los cálculos.

Palabras Clave: Redes, Reducción de bases, Factorización de polinomios enteros.

Abstract

The aim of this work is the study of the algorithm given in 1981 by Lenstra-Lenstra and Lovász (LLL) to calculate a reduced basis in a lattice and its aplication to the integer coefficients polynomial fatorization [4].

First, the concept of lattice is introduced in general and we study the case of two-dimensional lattices, analyzing the algorithm given by Gauss to obtain a minimal basis in which the idea of the algorithm LLL underlays.

Then, the Gram-Schmidt orthogonalization process is analyzed since we are interested in vectors with integer components. After that, the algorithm LLL is studied, demonstrated it is polynomial at the size of the lattice.

Finally, the algorithm to the factorization of polynomials with integer coefficients, which has an exponential time in the polynomial degree, given by Cantor and Zassenhaus [1] is collected without demonstration. And we analyze how the use of LLL reduces the computation time.

Keywords: Lattice, Basis reduction, Factorization of integer polynomial.

Índice general

Introducción

En palabras del Medalla Fields y premio Abel de Matemáticas, Michael Francis Atiyah, muchos problemas que no tienen nada que ver con la Geometría se resuelven cuando uno es capaz de transformarlos en problemas geométricos. Más aún, afirma que en Matemáticas, cuando uno deja de pensar geométricamente, deja uno de entender lo que está haciendo y únicamente hace cálculos.[7]

El campo de las Matemáticas denominado Geometría de los Números fue introducido por Minkowski en la segunda mitad del siglo XIX, y en él se hace uso de la Geometría para resolver, de una forma muy sencilla, ciertos problemas de la teoría de números. El objeto de estudio de esta disciplina son las redes. Una red de dimensión n es el conjunto de todas las combinaciones lineales enteras de una base de $\mathbb{R}^n$. Como quiera que una misma red puede ser generada por un número infinito de bases, un problema fundamental en este campo es el de obtener bases con vectores cortos, es decir, de norma euclídea pequeña. Este problema se conoce como el problema de la Reducción de Base y se planteó en principio al intentar hallar la forma reducida de una forma cuadrática. Fue identificado como "difícil" ya que requería la enumeración de las combinaciones lineales de los vectores de la base para identificar aquellos de más baja norma. De hecho, Emde Boas [3] demostró en 1981 que el problema de encontrar el vector más corto de una red (SVP, Shortest Vector Problem) para la norma del máximo, es NP-duro y por tanto no se espera que pueda existir un algoritmo que lo resuelva en tiempo polinómico. El propio Minkowski demostró que la norma euclídea del vector más corto es menor o igual que $\sqrt{n}|B|^{1/n}$ siendo B la matriz formada por los vectores de la base [2]. En 1982, A.Lenstra, H.Lenstra y L.Lovász, publicaron un algoritmo denominado L^3 o LLL, que es capaz de encontrar en tiempo polinómico un vector de la red con norma euclídea menor que $2^{(n-1)/4}|B|^{1/n}$.

El artículo de Lenstra, Lenstra y Lovász donde aparece el algoritmo antes mencionado lleva por título "Factoring polynomials with rational coefficients", debido a que la primera aplicación del LLL fue la factorización de polinomios en una variable con coeficientes racionales, consiguiendo un algoritmo de coste polinomial en el grado frente al exponecial que se conocía entonces. Ésta será la aplicación del algoritmo LLL que analizaremos en este trabajo, pero las aplicaciones han sido mucho más amplias. Se ha utilizado en criptoanálisis para romper el criptosistema de la mochila, ha permitido atacar de manera muy eficiente el algoritmo de encriptado RSA, resolver ecuaciones diofánticas, para calcular la forma normal de Hermite de una matriz entera, etc.

En el capítulo primero se hace una pequeña introducción a las redes, para a continuación estudiar el método dado por Gauss para encontrar una base reducida de una red de dimensión dos.

En el capítulo segundo se hace una revisión del algoritmo de ortogonalización de Gram-Smidth necesaria para el estudio del algoritmo LLL que se aborda en el capítulo tres. En el cuarto capítulo estudiamos la factorización de polinomios con coeficientes enteros, tanto con el algoritmo exponencial dado por Zassenhaus como por el polinómico dado por Lenstra, Lenstra y Lovász.

CAPÍTULO 1

Redes. Redes de dimensión dos.

1.1. Redes

Definición 1.1.1. Sean $n \geq 1$ y $x_1, x_2, ..., x_n$ una base de $\mathbb{R}^n$. Se llama *red generada por* $x_1, .., x_n$ al conjunto $\mathcal{L}$ de combinaciones lineales de los vectores de la base con coeficientes enteros:

$$\mathcal{L} = \mathbb{Z}x_1 + \mathbb{Z}x_2 + ... + \mathbb{Z}x_n = \left\{ \sum_{i=1}^{n} a_i x_i / a_1, a_2, ..., a_n \in \mathbb{Z} \right\}$$

Decimos que n es la *dimensión* de la red $\mathcal{L}$ y $x_1, x_2, ..., x_n$ una *base*, pues ésta no es única.

Definición 1.1.2. Definimos el *determinante* de una red como

$$d(\mathcal{L}) = |det(x_1, ..., x_n)|$$

que, como veremos más adelante, es independiente de la elección de la base.

Lema 1.1.1. *Sean $x_1, x_2, ..., x_n$ e $y_1, y_2, ..., y_n$ dos bases de la misma red $\mathcal{L} \subset \mathbb{R}^n$. Sea X (respectivamente Y) la matriz $n \times n$ con filas x_i (respectivamente y_i), $i = 1, 2, ..., n$. Entonces $Y = CX$, donde C es una matriz de orden n con coeficientes enteros y determinante ± 1.*

Demostración. Cada y_i pertenece a la red de base $x_1, x_2, ..., x_n$ y cada x_i pertenece a la red de base $y_1, y_2, ..., y_n$, entonces

$$x_i = \sum_{j=1}^{n} b_{ij} y_j, \qquad y_i = \sum_{j=1}^{n} c_{ij} x_j \qquad\qquad i = 1, 2, ..., n,$$

donde $B = (b_{ij})$ y $C = (c_{ij})$ son matrices de orden n con coeficientes enteros. Escribiendo las dos ecuaciones en forma matricial queda, $X = BY$ e $Y = CX$, por lo tanto, $X = BCX$ e $Y = CBY$. Ya que $x_1, x_2, ..., x_n$ e $y_1, y_2, ..., y_n$ son ambas bases de $\mathbb{R}^n$, las correspondientes matrices X e Y son invertibles, luego $XX^{-1} = BCXX^{-1}$ e $YY^{-1} = CBY^{-1}$. Entonces, $BC = I$ y $CB = I$, y así $det(B)det(C) = 1$. Puesto que B y C tienen coeficientes enteros, se sigue que, o bien $det(B) = det(C) = 1$ o bien $det(B) = det(C) = -1$.

□

Teorema 1.1.2. *El determinante de una red no depende de la base.*

Demostración. Supongamos que la red $\mathcal{L} \subset \mathbb{R}^n$ tiene dos bases $x_1, x_2, ..., x_n$ e $y_1, y_2, ..., y_n$. Usando la notación de la demostración del Lema 1.1.1, tenemos que

$$|det(Y)| = |det(CX)| = |det(C)det(X)| = | \pm det(X)| = |det(X)|$$

□

Definición 1.1.3. Sea $\mathcal{L}$ una red de dimensión n en el espacio euclídeo $\mathbb{R}^n$. Se define el *primer mínimo* de la red, y lo denotamos por $\Lambda_1(\mathcal{L})$, a la longitud del menor vector no nulo $x_1 \in \mathcal{L}$. El *segundo menor* de la red, denotado por $\Lambda_2(\mathcal{L})$, es el menor número real r tal que existan dos vectores linealmente independientes $x_1, x_2 \in \mathcal{L}$ tales que $|x_1||x_2| \leq r$. En general, para $i = 1, 2, ..., n$

$$\Lambda_i(\mathcal{L}) = \min_{x_1,...,x_i \in \mathcal{L}} \max(|x_1|, ..., |x_i|)$$

donde el mínimo se toma sobre todos los conjuntos con i vectores independientes de $\mathcal{L}$.

La mejor base posible de una red $\mathcal{L}$ estaría formada por vectores $x_1, ..., x_n \in \mathcal{L}$ tales que $|x_i| = \Lambda_i(\mathcal{L})$ para $i = 1, ..., n$. Pero tal base, en general, es muy difícil de calcular.

Se conoce como problema de reducción de bases al que tiene como objetivo encontrar una base de una red $\mathcal{L}$ formada por vectores de la menor norma posible, y el conocido como Shortest Vector Problem (SVP) trata de encontrar un vector v no nulo tal que $|v| \leq |w|$ para todo $w \in \mathcal{L}, w \neq 0$. En este sentido, Minkowski demostró el siguiente teorema:

Teorema 1.1.3. *Sea $\mathcal{L}$ una red en $\mathbb{R}^n$ y S un conjunto convexo centrado en el origen y simétrico de volumen mayor que $2^n|\mathcal{L}|$. Entonces S contiene un punto no nulo de $\mathcal{L}$.*

Corolario 1.1.4.

$$\Lambda_1(\mathcal{L}) \leq \sqrt{n}|\mathcal{L}|^{1/n}$$

Demostración. La bola $B(0, r)$ contiene al hipercubo $\left[-\frac{r}{\sqrt{n}}, \frac{r}{\sqrt{n}}\right]$. Por tanto, su volumen es mayor que $\left(\frac{2r}{\sqrt{n}}\right)^n$.

Para $r = \sqrt{n}|\mathcal{L}|^{1/2}$, el volumen de la bola $B(0, r)$ es mayor que $2^n|\mathcal{L}|$, y por tanto contiene un vector no nulo de $\mathcal{L}$; y de aquí la longitud del vector más corto es como máximo $\sqrt{n}|\mathcal{L}|^{1/n}$ □

1.2. Redes de dimensión dos

El caso más sencillo para estudiar la reducción de bases es el de las redes de dimensión2. Sea $\{x, y\}$ una base de $\mathbb{R}^2$ y $\mathcal{L}$ la red

$$\mathcal{L} = \{ax + by/a, b \in \mathbb{Z}\}$$

Definición 1.2.1. Se dice que una base $\{x, y\}$ de una red $\mathcal{L}$ de $\mathbb{R}^2$ es *minimal* si x es el vector más corto no nulo en $\mathcal{L}$ e y es el vector más corto en $\mathcal{L}$ que no es múltiplo de x.

1.2.1. El algoritmo de Euclides

El algoritmo que vamos a estudiar para obtener una base minimal de una red de dimensión dos se debe a Gauss, y tiene un gran parecido al algoritmo euclídeo para calcular el máximo común divisor de dos números. Por ello vamos a analizar en primer lugar el algoritmo de Euclides.

Consideremos ahora $a, b \in \mathbb{Z}$ y tomemos el conjunto de todas las combinaciones lineales enteras de a y b:

$$I = \{sa + tb/s, t \in \mathbb{Z}\}$$

En este caso, el menor elemento positivo de I es el máximo común divisor de a y b, $d = \text{mcd}(a, b)$. El algoritmo de Euclides obtiene el máximo común divisor de dos números enteros mediante la aplicación de sucesivas divisiones con resto.

Sean a y b dos enteros cualesquiera con $b \neq 0$, $a \geq b$, hay dos únicos enteros q (cociente) y r (resto) tales que

$$a = qb + r \quad 0 \leq r < |b|.$$

El algoritmo de Euclides realiza repetidamente divisiones con resto, reemplazando el par de enteros (r_{i-1}, r_i) por el par (r_i, r_{i+1}) determinado por la ecuación $r_{i+1} = r_{i-1} - q_{i+1}r_i$, siendo $r_0 = a$, $r_1 = b$ y $r_2 = r$. Al obtener de resto 0 el algoritmo termina y el último resto distinto de cero es el $m.c.d.(a, b)$.

Ejemplo 1.2.1. Si $a = 7854$ y $b = 2145$, tenemos

- $(r_0 = 7854, r_1 = 2145) \quad q_2 = 3 \quad r_2 = 1419 \quad r_2 = r_0 - q_2 r_1$
- $(r_1 = 2145, r_2 = 1419) \quad q_3 = 1 \quad r_3 = 726 \quad r_3 = r_1 - q_3 r_2$
- $(r_2 = 1419, r_3 = 726) \quad q_4 = 1 \quad r_4 = 693 \quad r_4 = r_2 - q_4 r_3$
- $(r_3 = 726, r_4 = 693) \quad q_5 = 1 \quad r_5 = 33 \quad r_5 = r_3 - q_5 r_4$
- $(r_4 = 693, r_5 = 33) \quad q_6 = 21 \quad r_6 = 0 \quad r_6 = r_4 - q_6 r_5$

El último resto distinto de cero, $r_5 = 33$, es el máximo común divisor de $a = 7854$ y $b = 2145$.

Lema 1.2.1. *Sea n el número de divisiones sucesivas del algoritmo de Euclides. Escribiendo log para el logaritmo en base 2, tenemos*

$$n < 1 + 2 \log \min(|a|, |b|).$$

Demostración. Es evidente que $r_0 \geq r_1 > r_2 > ... > r_n > r_{n+1} = 0$, y entonces $q_i \geq 1$ para todo i. De aquí se sigue que

$$r_{i-1} = q_{i+1}r_i + r_{i+1} \geq r_i + r_{i+1} > 2r_{i+1}$$

y por tanto $r_1...r_{n-2} > 2^{n-2}r_3...r_n$, lo que implica $2^{n-2} < \frac{r_1 r_2}{r_{n-1}r_n}$. Pero $r_1 r_2 < r_1^2$ y $r_{n-1}r_n \geq 2$, por lo que se obtiene

$$2^{n-2} < \frac{r_1^2}{2}$$

Tomando logaritmo en base 2 queda $n-2 < -1 + 2\log r_1$, y entonces $n < 1 + 2\log r_1$. Como $r_1 = \min(|a|, |b|)$, se completa la demostración. □

Una variante importante del algoritmo de Euclides usa restos simétricos. Esto significa que modificamos la desigualdad de $0 \leq r_{i+1} < r_i$ por la siguiente:

$$r_{i-1} = q_{i+1}r_i + r_{i+1}, \qquad -\left|\frac{r_i}{2}\right| < r_{i+1} \leq \left|\frac{r_i}{2}\right|$$

Definición 1.2.2. A este algoritmo modificado se le llama *algoritmo de Euclides centrado* y se denota por **CEuclid**(a, b). (También se conoce como *algoritmo de Euclides con restos simétricos*, o *algoritmo de Euclides con restos mínimos absolutos*).

Ejemplo 1.2.2. Realizaremos el Ejemplo 1.2.1 usando el algoritmo de Euclides centrado.

- Para $r_0 = 7854$ y $r_1 = 2145$ se obtiene $7854 = 4 \cdot 2145 - 726$
- Para $r_1 = 2145$ y $r_2 = -726$ se obtiene $2145 = (-3) \cdot (-726) - 33$
- Para $r_2 = -726$ y $r_3 = -33$ se obtiene $-726 = (-22) \cdot (-33) + 0$

El algoritmo de Euclides necesita 5 pasos para calcular el m.c.d., pero el **CEuclid** necesita sólo 3 pasos.

1.2.2. El algoritmo de Gauss

El algoritmo atribuido a Gauss que damos a continuación calcula una base minimal de $\mathcal{L}$ siendo $\mathcal{L} \subset \mathbb{R}^2$ la red generada por la base x e y de $\mathbb{R}^2$:

$$\mathcal{L} = \{ax + by / a, b \in \mathbb{Z}\}$$

Definición 1.2.3. Escribimos $\lceil \mu \rfloor = \lceil \mu - \frac{1}{2} \rceil$ para el *entero más próximo* a $\mu \in \mathbb{R}$. Obsérvese que el entero más próximo a $n + \frac{1}{2} (n \in \mathbb{Z})$ es n, no $n+1$.

El algoritmo

- **Entrada**. Una base x, y de la red $\mathcal{L}$ de $\mathbb{R}^2$ tal que $|x| \leq |y|$.
- **Salida**. Una base minimal v_1, v_2 de la red $\mathcal{L}$.

(1) Set $v_1 \longleftarrow x$ y $v_2 \longleftarrow y$. Set finished $\longleftarrow$ false.

(2) While not finished do:

(a) Set $m \longleftarrow \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor$

(b) Set $v_2 \longleftarrow v_2 - mv_1$

(c) If $|v_1| \leq |v_2|$ then
  (i) set finished $\longleftarrow$ true
  else
  (ii) set $u \longleftarrow v_1$, $v_1 \longleftarrow v_2$, $v_2 \longleftarrow u$ (intercambiar v_1 y v_2).

(3) Return v_1 y v_2.

El paso (2)(a) del algoritmo de Gauss no es otra cosa que un antecedente del proceso de ortogonalización de Gram-Schmidt, pero usa el entero más próximo al coeficiente $\mu = \frac{v_2 \cdot v_1}{v_1 \cdot v_1}$ en lugar del propio μ ya que el vector v_2 debe permanecer en $\mathcal{L}$.

Ejemplo 1.2.3. Sea $\mathcal{L}$ la red en $\mathbb{R}^2$ generada por los vectores

$$v_1 = (-56, 43), \ \ |v_1| \approx 70{,}60, \quad v_2 = (95, -73), \ \ |v_2| \approx 119{,}8$$

La primera iteración calcula

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor = \left\lceil \frac{-8459}{4985} \right\rfloor \approx \lceil -1{,}697 \rfloor = \left\lceil -1{,}697 - \frac{1}{2} \right\rceil = -2$$

Ahora fijamos $v_2 = v_2 + 2v_1 = (-17, 13)$, $|v_2| \approx 21{,}4$. Como $|v_2| < |v_1|$ intercambiamos los vectores y tomamos $v_1 = (-17, 13)$, $v_2 = (-56, 43)$. La segunda iteración calcula

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor = \left\lceil \frac{1511}{458} \right\rfloor \approx \lceil 3{,}299 \rfloor = 3$$

Ahora fijamos $v_2 = v_2 - 3v_1 = (-5, 4)$, $|v_2| \approx 6{,}403$. Intercambiando los vectores queda $v_1 = (-5, 4)$, $v_2 = (-17, 13)$. La tercera iteración calcula

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor = \left\lceil \frac{137}{41} \right\rfloor \approx \lceil 3{,}341 \rfloor = 3$$

Ahora fijamos $v_2 = v_2 - 3v_1 = (-2, 1)$, $|v_2| \approx 2{,}236$. Intercambiando los vectores queda $v_1 = (-2, 1)$, $v_2 = (-5, 4)$. La cuarta iteración calcula

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor = \left\lceil \frac{14}{5} \right\rfloor = \lceil 3{,}8 \rfloor = 3$$

Ahora fijamos $v_2 = v_2 - 3v_1 = (1, 1)$, $|v_2| \approx 1{,}414$. Intercambiando los vectores queda $v_1 = (1, 1)$, $v_2 = (-2, 1)$. La quinta iteración calcula

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor = \left\lceil \frac{-1}{2} \right\rfloor = \lceil 0{,}5 \rfloor = -1$$

Ahora fijamos $v_2 = v_2 + v_1 = (-1, 2)$, $|v_2| \approx 2{,}236$. Como $|v_2| \geq |v_1|$ terminamos con la base minimal

$$v_1 = (1, 1), \ \ |v_1| \approx 1{,}414, \quad v_2 = (-1, 2), \ \ |v_2| \approx 2{,}236$$

El algoritmo de Gauss realiza repetidamente proyecciones ortogonales, reemplazando el par de vectores (v_1, v_2) por el par (v_1, v_2') determinado por la ecuación $v_2' = v_2 - mv_1$, y después intercambiando los vectores (si fuese necesario). La operación aritmética de la división con resto en el algoritmo de Euclides se corresponde con la operación geométrica de proyección ortogonal en el algoritmo de Gauss.

A continuación, probaremos que, en efecto, el algoritmo de Gauss genera una base minimal de la red dada.

Lema 1.2.2. *Después de la ejecución del paso 2(b) del algoritmo se obtiene que*

$$|v_2' \cdot v_1| \leq \frac{1}{2}|v_1|^2,$$

donde v_2' es el nuevo segundo vector de la base.

Demostración. Por la definición de entero más próximo, tenemos

$$m = \left\lceil \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right\rfloor \;\Rightarrow\; m - \frac{1}{2} < \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \leq m + \frac{1}{2}$$

Multiplicando esta desigualdad por $v_1 \cdot v_1$ queda

$$m(v_1 \cdot v_1) - \frac{1}{2}(v_1 \cdot v_1) < v_2 \cdot v_1 \leq m(v_1 \cdot v_1) + \frac{1}{2}(v_1 \cdot v_1)$$

Restando $m(v_1 \cdot v_1)$ a cada miembro queda

$$-\frac{1}{2}(v_1 \cdot v_1) < v_2 \cdot v_1 - m(v_1 \cdot v_1) \leq \frac{1}{2}(v_1 \cdot v_1)$$

y entonces

$$|(v_2 - mv_1) \cdot v_1| \leq \frac{1}{2}|v_1|^2$$

□

Teorema 1.2.3. *El algoritmo de Gauss termina, y cuando lo hace, v_1 es el vector no nulo más corto de la red y v_2 es el vector más corto de la red que no es múltiplo de v_1.*

Demostración. Tomando v_1 y v_2 como vectores fila, podemos expresar el paso (2)(b) en forma matricial como

$$\begin{pmatrix} v_1' \\ v_2' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -m & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

Como la matriz tiene determinante 1, es claro que el paso (2) conserva la propiedad de que v_1, v_2 es una base de la red $\mathcal{L}$. El algoritmo intercambia v_1 y v_2 en el paso (2)(c)(ii) cuando $|v_2| < |v_1|$ por lo que la longitud de v_1 decrece estrictamente de un ejecución del paso (2) a la siguiente. Para cualquier número real $r > 0$, hay solamente un número finito de elementos de la red en el disco $\{u \in \mathbb{R}^2 / |u| \leq r\}$. Se sigue de aquí que el algoritmo

termina después de un número finito de ejecuciones del paso (2). Cuando termina, el paso (2)(c) y el Lema 1.2.2 garantizan que

$$|v_1| \leq |v_2|, \quad -\frac{1}{2}|v_1|^2 \leq v_2 \cdot v_1 \leq \frac{1}{2}|v_1|^2$$

Sea u un vector cualquiera no nulo de $\mathcal{L}$, de modo que $u = av_1 + bv_2$ para algún $a, b \in \mathbb{Z}$, no ambos cero. Tenemos que

$$|u|^2 = (av_1+bv_2)\cdot(av_1+bv_2) = a^2|v_1|^2+2ab(v_1\cdot v_2)+b^2|v_2|^2 \geq a^2|v_1|^2-|ab||v_1|^2+b^2|v_2|^2 \geq$$

$$\geq a^2|v_1|^2 - |ab||v_1|^2 + b^2|v_1|^2 = (a^2 - |ab| + b^2)|v_1|^2$$

Dado que a, b no son ambos cero, tenemos que $a^2b^2 < (a^2+b^2)^2$ y entonces $|ab| < a^2 + b^2$. Por lo tanto, $|u|^2 \geq |v_1|^2$, y entonces v_1 es el vector más corto de $\mathcal{L}$.
Ahora supongamos que $u = av_1 + bv_2$ es linealmente independiente de v_1, o lo que es lo mismo, $b \neq 0$. Tenemos

$$|u|^2 \geq a^2|v_1|^2 - |ab||v_1|^2 + b^2|v_2|^2 = a^2|v_1|^2 - |ab||v_1|^2 + \frac{1}{4}b^2|v_2|^2 + \frac{3}{4}b^2|v_2|^2 =$$

$$= a^2|v_1|^2 - |ab||v_1|^2 + \frac{1}{4}b^2|v_1|^2 + \frac{3}{4}b^2|v_2|^2 = \left(|a| - \frac{1}{2}|b|\right)|v_1|^2 + \frac{3}{4}b^2|v_2|^2$$

Por tanto, $|u|^2 \geq |v_2|^2$ si $|b| \neq 1$. Si $b = \pm 1$ entonces

$$|u|^2 \geq a^2|v_1|^2 - |a||v_1|^2 + |v_2|^2 = |a|(|a|-1)|v_1|^2 + |v_2|^2$$

Como $a \in \mathbb{Z}$ queda $|a|(|a|-1) = 0$ para $|a| \leq 1$ y $|a|(|a|-1) > 0$ para $|a| \geq 2$, y de aquí se sigue que $|u|^2 \geq |v_2|^2$ en este caso también. Por consiguiente, v_2 es el vector linealmente independiente de v_1 más corto de $\mathcal{L}$. □

1.2.3. Análisis de Vallée del algoritmo de Gauss

En esta sección consideraremos redes en $\mathbb{Z}^2$; es decir, suponemos que los vectores tienen componentes enteras. Al igual que el algoritmo de Euclides tiene dos variantes, la original y la versión centrada, el algoritmo de Gauss también las tiene.

Definición 1.2.4. Sea $\alpha \in \mathbb{R}$ definimos $sign(\alpha) = 1$ si $\alpha \geq 0$ y $sign(\alpha) = -1$ si $\alpha < 0$

Algoritmo centrado de Gauss

- **Entrada.** Una base x, y de la red $\mathcal{L} \in \mathbb{Z}^2$ tal que $|x| \leq |y|$.
- **Salida.** Una base minimal v_1, v_2 de la red $\mathcal{L}$.

(1) Set $v_1 \longleftarrow x$ y $v_2 \longleftarrow y$. Set finished $\longleftarrow$ false.
(2) While not finished do:
  (a) Set $\mu \longleftarrow \frac{v_2 \cdot v_1}{v_1 \cdot v_1}$. Set $m \longleftarrow \lceil \mu \rfloor$. Set $\epsilon \longleftarrow$sign $(\mu - m)$

(b) Set $v_2 \longleftarrow \epsilon(v_2 - mv_1)$

(c) If $|v_1| \leq |v_2|$ then

(i) set finished $\longleftarrow$ true

else

(ii) set $u \longleftarrow v_1$, $v_1 \longleftarrow v_2$, $v_2 \longleftarrow u$ (intercambiar v_1 y v_2).

(3) Return v_1 y v_2.

El factor ϵ del paso (2)(b) garantiza que el nuevo v_2 forma un ángulo agudo con el v_1 anterior. Veámoslo, sea $v_2' = \epsilon(v_2 - mv_1)$ el nuevo valor de v_2. Basta ver que $v_2' \cdot v_1 > 0$. Se tiene

$$v_2' \cdot v_1 = \epsilon(v_2 - mv_1) = \epsilon(v_2 \cdot v_1 - mv_1 \cdot v_1) =$$

$$= \epsilon \left(\frac{v_2 \cdot v_1}{v_1 \cdot v_1} - m \right) (v_1 \cdot v_1) = \epsilon(\mu - m)|v_1|^2$$

y la última cantidad es claramente positiva por definición de ϵ.

Teorema 1.2.4. *Los vectores de salida v_1, v_2 del algoritmo de Gauss centrado forman una base minimal de la red generada por los vectores de entrada x, y.*

Demostración. Es claro que v_1 y v_2 satisfacen $|v_1| \leq |v_2|$. Además $0 \leq v_2 \cdot v_1 \leq \frac{1}{2}|v_1|^2$ ya que

$$-\frac{1}{2} < \mu - m \leq \frac{1}{2}, \quad \epsilon(\mu - m) = |\mu - m|$$

Entonces $0 \leq \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \leq \frac{1}{2}$. Sea w la componente de v_2 ortogonal a v_1. Tenemos

$$v_2 = \frac{v_2 \cdot v_1}{v_1 \cdot v_1} v_1 + w, \quad v_1 \cdot w = 0$$

y por tanto

$$|v_2|^2 = \left| \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right|^2 |v_1|^2 + |w|^2$$

Dividiendo todos los términos entre $|v_2|^2$ queda

$$1 = \left| \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \right|^2 \frac{|v_1|^2}{|v_2|^2} + \frac{|w|^2}{|v_2|^2}$$

El primer término del lado derecho es $\leq \frac{1}{4}$, así que

$$\frac{|w|^2}{|v_2|^2} \geq \frac{3}{4} \quad \text{entonces} \quad |w| \geq \frac{\sqrt{3}}{2}|v_2| \tag{1.1}$$

Considerando los tres subconjuntos de la red $\mathcal{L}$ generada por v_1 y v_2:

$$V_+ = \{av_1 + v_2 / a \in \mathbb{Z}\} \quad V_0 = \{av_1 / a \in \mathbb{Z}\} \quad V_- = \{av_1 - v_2 / a \in \mathbb{Z}\}$$

Así V_0 está compuesto por todos los múltiplos enteros de v_1 y V_+ (respectivamente V_-) es el trasladado de V_0 en la dirección v_2 (respectivamente $-v_2$). Se sigue de la desigualdad (1.1) que todos los vectores z de la red $\mathcal{L}$ generada por v_1 y v_2, que no pertenecen al subconjunto $V_+ \cup V_0 \cup V_-$, satisfacen

$$|z| \geq \sqrt{3}|v_2| > |v_2| \geq |v_1|$$

Para estas desigualdades es claro que v_1 y v_2 forman una base minimal de la red $\mathcal{L}$ □

Ejemplo 1.2.4. Consideremos los siguientes vectores de $\mathbb{Z}^2$

$$v_1 = (-67, 16), \quad |v_1| \approx 68{,}88, \quad v_2 = (93, -25), \quad |v_2| \approx 96{,}30$$

La primera iteración calcula

$$\mu = \frac{-6631}{4745} \approx -1{,}397, \quad m = -1, \quad \epsilon = -1$$

Por lo tanto $v_2' = -(v_2 + v_1) = (-26, 9)$, $|v_2'| \approx 27{,}51$. Intercambiando los vectores queda $v_1 = (-26, 9)$, $v_2 = (-67, 16)$. La segunda iteración calcula

$$\mu = \frac{1886}{757} \approx 2{,}491, \quad m = 2, \quad \epsilon = 1$$

Por lo tanto $v_2' = v_2 - 2v_1 = (-15, -2)$, $|v_2'| \approx 15{,}13$. Intercambiando los vectores queda $v_1 = (-15, -2)$, $v_2 = (-26, 9)$. La tercera iteración calcula

$$\mu = \frac{372}{229} \approx 1{,}624, \quad m = 2, \quad \epsilon = -1$$

Por lo tanto $v_2' = -(v_2 - 2v_1) = (-4, -13)$, $|v_2'| \approx 13{,}6$. Intercambiando los vectores queda $v_1 = (-4, -13)$, $v_2 = (-15, -2)$. La cuarta iteración calcula

$$\mu = \frac{86}{185} \approx 0{,}4649, \quad m = 0, \quad \epsilon = 1$$

Por lo tanto $v_2' = v_2$ y el algoritmo termina.

Consideraremos ahora el algoritmo de Gauss parametrizado. Esta tercera versión del algoritmo depende de un parámetro $t \geq 1$: la condición de terminación $|v_1| \leq |v_2|$ se reemplaza por $|v_1| \leq t|v_2|$. El algoritmo parametrizado es un caso debilitado del algoritmo centrado, pero es un ingrediente esencial del algoritmo LLL, ya que en dicho algoritmo basta con utilizar un vector suficientemente pequeño.

Algoritmo de Gauss parametrizado

- **Entrada.** Un base x, y de la red $\mathcal{L}$ de $\mathbb{Z}^2$ tal que $|x| \leq |y|$.
- **Salida.** Una base "minimal débil" v_1, v_2 de la red $\mathcal{L}$.

(1) Set $v_1 \longleftarrow x$ y $v_2 \longleftarrow y$. Set finished $\longleftarrow$ false.

(2) While not finished do:

(a) Set $\mu \longleftarrow \frac{v_2 \cdot v_1}{v_1 \cdot v_1}$. Set $m \longleftarrow \lceil \mu \rfloor$. Set $\epsilon \longleftarrow$sign $(\mu - m)$

(b) Set $v_2 \longleftarrow \epsilon(v_2 - mv_1)$

(c) If $|v_1| \leq t|v_2|$ then

(i) set finished $\longleftarrow$ true

else

(ii) set $u \longleftarrow v_1$, $v_1 \longleftarrow v_2$, $v_2 \longleftarrow u$ (intercambiar v_1 y v_2).

(3) Return v_1 y v_2.

Proposición 1.2.5. *Para la base $x, y \in \mathbb{Z}^2$ de la red, sea n el número de iteraciones realizadas por el algoritmo de Gauss centrado con entrada x, y. Para $1 < t \leq \sqrt{3}$ sea $n[t]$ el número de iteraciones realizadas por el algoritmo de Gauss parametrizado con entrada x, y. Se verifica*

$$n[t] \leq n \leq n[t] + 1$$

Demostración. La primera desigualdad es clara, dado que **CGauss** tiene condición de terminación más fuerte que **PGauss**[t], y dado que el algoritmo centrado no termina más tarde que el algoritmo parametrizado. (Esto se verifica para todo $t \geq 1$).
Para la desigualdad de la derecha, consideramos la última iteración de **PGauss**[t] y asumimos que no es la última iteración de **CGauss**. Basta ver que **CGauss** termina en la siguiente iteración. Tenemos que

$$0 \leq \frac{v_2 \cdot v_1}{v_1 \cdot v_1} \leq \frac{1}{2} \quad \text{entonces} \quad v_1 \cdot v_1 - 2v_2 \cdot v_1 \geq 0$$

Como **CGauss** no termina en esta iteración, se tiene $|v_2| < |v_1|$.
El tercer lado del triángulo construido con v_1 y v_2 es $v_2 - v_1$. Tenemos

$$|v_2 - v_1|^2 = (v_2 - v_1) \cdot (v_2 - v_1) = v_2 \cdot v_2 - 2v_2 \cdot v_1 + v_1 \cdot v_1 \geq v_2 \cdot v_2 = |v_2|^2$$

y entonces v_2 es el lado más corto de este triángulo. El algoritmo intercambia los vectores, y entonces ahora se tiene

$$|v_1| < |v_2|, \quad 0 \leq v_2 \cdot v_1 \leq \frac{1}{2}|v_2|^2$$

donde v_1 es el lado más corto del triángulo construido a partir de v_1 y v_2. Como **PGauss**[t] termina en este punto, tenemos (usando los vectores intercambiados) $|v_2| \leq t|v_1|$.
Por lo tanto

$$0 \leq \frac{v_2 \cdot v_1}{v_1 \cdot v_1} = \left(\frac{v_1 \cdot v_2}{v_2 \cdot v_2}\right)\left(\frac{v_2 \cdot v_2}{v_1 \cdot v_1}\right) \leq \frac{1}{2}t^2 \leq \frac{3}{2}$$

El nuevo valor de v_2 es o bien $v_2' = v_2$ o bien $v_2' = \pm(v_2 - v_1)$. Como v_1 es más corto que ambos, el algoritmo centrado termina en la siguiente iteración. □

CAPÍTULO 2

Proceso de ortogonalización de Gram-Schmidt

El proceso de ortogonalización de Gram-Schmidt permite obtener, a partir de una base arbitraria de $\mathbb{R}^n$, otra que genere el mismo espacio y además sea ortogonal. Es un tema básico del álgebra lineal, pero lo desarrollaremos con intención de ver sus aplicaciones en el algoritmo LLL.

2.1. El teorema de Gram-Schmidt

Definición 2.1.1. Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$. La *ortogonalización de Gram-Schmidt* de $x_1, ..., x_n$ es la base $x_1^*, ..., x_n^*$ tal que

$$x_1^* = x_1,$$

$$x_i^* = x_i - \sum_{j=1}^{i-1} \mu_{ij} x_j^* \quad (2 \leq i \leq n), \qquad \mu_{ij} = \frac{x_i \cdot x_j^*}{x_j^* \cdot x_j^*} \quad (1 \leq j < i \leq n).$$

No normalizamos los vectores. Nótese que si los vectores $x_1, ..., x_n$ están en $\mathbb{Q}^n$, también lo están los vectores $x_1^*, ..., x_n^*$.

Es importante resaltar que los vectores de la base de Gram-Schmidt $x_1^*, ..., x_n^*$ generalmente no están en la red generada por $x_1, ..., x_n$ ya que, en general, $x_1^*, ..., x_n^*$ no son combinaciones lineales enteras de $x_1, ..., x_n$.

Nota 2.1.1. Si hacemos $\mu_{ii} = 1$ para $1 \leq i \leq n$ y tenemos

$$x_i = \sum_{j=1}^{i} \mu_{ij} x_j^* \tag{2.1}$$

Escribimos $x_i = (x_{i1}, ..., x_{in})$ y formamos la matriz $X = (x_{ij})$ en la que la fila i es el vector x_i, y de la misma manera $X^* = (x_{ij}^*)$. Si fijamos $\mu_{ij} = 0$ para $1 \leq i < j \leq n$ entonces (2.1) se puede escribir en forma matricial como $X = MX^*, M = (\mu_{ij})$. La matriz M es triangular inferior con $\mu_{ii} = 1$ para todo i, así que es invertible, y entonces tenemos también $X^* = M^{-1}X$.

Teorema 2.1.1 (**Teorema de Gram-Schmidt**). *Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$ y sea $x_1^*, ..., x_n^*$ su ortogonalización de Gram-Schmidt. Sea X (respectivamente X^*) la matriz $n \times n$ en la que la fila i es el vector x_i (respectivamente x_i^*) para $1 \leq i \leq n$. Tenemos*

(a) $x_i^* \cdot x_j^* = 0$ *para* $1 \leq i < j \leq n$.

(b) $< x_1^*, ..., x_k^* > = < x_1, ..., x_k >$ *para* $1 \leq k \leq n$.

(c) Para $1 \leq k \leq n$, *el vector* x_k^* *es la proyección de* x_k *sobre el complemento ortogonal generado por* $x_1, ..., x_{k-1}$.

(d) $|x_k^*| \leq |x_k|$ *para* $1 \leq k \leq n$.

(e) $\det(X^*) = \det(X)$.

Demostración. (a) Por inducción sobre j. Para $j = 1$ no hay nada que probar. Supongamos que se cumple para j. Veamos que se cumple para $j + 1$

$$x_i^* \cdot x_{j+1}^* = x_i^* \cdot \left(x_{j+1} - \sum_{k=1}^{j} \mu_{j+1,k} x_k^* \right) = x_i^* \cdot x_{j+1} - \sum_{k=1}^{j} \mu_{j+1,k} (x_i^* \cdot x_k^*) =$$

$$= x_i^* \cdot x_{j+1} - \mu_{j+1,i} (x_i^* \cdot x_i^*) = x_i^* \cdot x_{j+1} - \frac{x_{j+1} \cdot x_i^*}{x_i * \cdot x_i^*} (x_i * \cdot x_i^*) = 0$$

(b) Por la Nota 2.1.1, tenemos que $x_i \in < x_1^*, ..., x_n^* >$ para $1 \leq i \leq k$, y entonces $< x_1, ..., x_n > \subseteq < x_1^*, ..., x_n^* >$. La otra inclusión la veremos por inducción sobre k. Para $k = 1$ tenemos $x_1^* = x_1$ y el resultado es evidente. Asumimos que el resultado es cierto para k. Veamos que se cumple para $k+1$. Usando la Definición 3.1 tenemos

$$x_{k+1}^* = x_{k+1} - \sum_{j=1}^{k} \mu_{k+1,j} x_j^* = x_{k+1} + y, \qquad y \in < x_1^*, ..., x_k^* >$$

Por hipótesis de inducción tenemos que $< x_1^*, ..., x_k^* > \subseteq < x_1, ..., x_k >$, y entonces la última ecuación implica $x_{k+1}^* \in < x_1, ..., x_{k+1} >$.

(c) Para simplificar la notación, escribiremos $U = < x_1, ..., x_{k+1} >$; entonces $U^\perp$ es el subespacio de $\mathbb{R}^n$ ortogonal a U. Hay una única descomposición $x_k = x_k' + y$ donde $x_k' \in U^\perp$ e $y \in U$; aquí x_k' es la proyección de x_k sobre el complemento ortogonal de U. Aplicando la Nota 2.1.1 tenemos

$$x_k = x_k^* + \sum_{j=1}^{k-1} \mu_{kj} x_j^*$$

De (b) sacamos que $U = < x_1^*, ..., x_{k-1}^* >$, y por lo tanto $x_k^* = x_k'$.

(d) Usando (a) vemos que $x_k = x_k^* + \sum_{j=1}^{k-1} \mu_{kj} x_j^*$ implica que $|\mathrm{x_k}|^2 = |\mathrm{x_k^*}|^2 + \sum_{\mathrm{j=1}}^{\mathrm{k-1}} \mu_{\mathrm{kj}}^2 |\mathrm{x_j^*}|^2$. Como cada término de la suma es no negativo, queda demostrado.

(e) Por la Nota 2.1.1 tenemos $X = MX^*$ donde $M = (\mu_{ij})$ es una matriz triangular inferior con $\mu_{ij} = 1$ para $1 \leq i \leq n$. Luego $\det(M) = 1$ y por tanto $\det(X) = \det(M)\det(X^*) = \det(X^*)$.

□

Una consecuencia directa del Teorema 3.1 es la siguiente desigualdad para el determinante de una matriz.

Definición 2.1.2. Sea $x_1, ...x_n$ una base de $\mathbb{R}^n$, y sea X la matriz $n \times n$ con x_i en la fila i para $1 \leq i \leq n$. Para $1 \leq k \leq n$, sea X_k la matriz $k \times n$ formada por las k primeras filas de F. La *k-ésima matriz de Gram* de esta base es la matriz simétrica $k \times k$

$$G_k = X_k X_k^t$$

El *k-ésimo determinante de Gram* de la base es $d_k = \det(G_k)$.

Por convenio, fijamos $d_0 = 1$. Si $x_i \in \mathbb{Z}^n$ para todo i entonces $d_k \in \mathbb{Z}$ para $0 \leq k \leq n$. Este hecho será importante más adelante en nuestro análisis del algoritmo LLL.

Proposición 2.1.2. *Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$, y sea $x_1^*, ..., x_n^*$ su ortogonalización de Gram-Schmidt. Para $1 \leq k \leq n$ el k-ésimo determinante de la base es el producto del cuadrado de las longitudes de los vectores de la ortogonalización:*

$$d_k = \prod_{i=1}^{k} |x_i^*|^2$$

Demostración. Por la Nota 2.1.1 podemos expresar la ortogonalización de Gram-Schmidt como la ecuación matricial $X = MX^*$. Sea M_k la submatriz superior izquierda $k \times k$ de M, y sea X^* la matriz $k \times n$ formada por las k primeras filas de X^*. Tenemos la factorización $X_k = M_k X_k^*$ donde $\det(m_k) = 1$. Por lo tanto,

$$d_k = \det(X_k X_k^t) = \det((M_k X_k^*)(M_k X_k^*)^t) = \det(M_k (X_k^* (X_k^*)^t) M_k^t) =$$
$$= \det(M_k)\det(X_k^*(X_k^*)^t)\det(M_k^t) = \det(X_k^*(X_k^*)^t)$$

Ya que las filas $x_1^*, ..., x_k^*$ de la matriz X_k^* son ortogonales por el Teorema 2.1.1(a), vemos que $X_k^*(x_k^*)^t$ es una matriz diagonal cuyos elementos de la diagonal son $|x_1^*|^2, ..., |x_k^*|^2$. □

2.2. Análisis del proceso de Gram-Schmidt

La regla de Cramer establece que si A es una matriz $n \times n$ sobre $\mathbb{R}$ con $\det(A) \neq 0$, sea $y \in \mathbb{R}^n$ un vector columna, y sea $x = (x_1, ..., x_n)^t$ la única solución del sistema lineal $Ax = y$, entonces para $1 \leq i \leq n$ tenemos

$$x_i = \frac{\det(A_i)}{\det(A)}$$

donde A_i es la matriz obtenida de A reemplazando la columna i por y.

Esta regla permite acotar los denominadores de los números racionales que aparecen en los vectores de la ortogonalización de Gram-Schmidt con componentes enteras. Esto será importante en nuestro análisis del algoritmo LLL.

Proposición 2.2.1. *Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$ con $x_i \in \mathbb{Z}^n$ para $1 \leq i \leq n$. Sea $x_1^*, ..., x_n^* \in \mathbb{Q}^n$ su ortogonalización de Gram-Schmidt con coeficientes $\mu_{ij} \in \mathbb{Q}$. Para $1 \leq k \leq n$, sea d_k el determinante de Gram correspondiente. Entonces:*

(a) El vector $d_{k-1}x_k^*$ tiene componentes enteras para $1 \leq k \leq n$.

(b) La cantidad $d_j\mu_{kj}$ es un entero para $1 \leq j \leq k$.

(c) $|\mu_{kj}| \leq d_{j-1}^{1/2}|x_k|$ para $1 \leq j \leq k$.

Demostración. (a) Podemos expresar la ortogonalización de Gram-Schmidt mediante la ecuación matricial $X = MX^*$, o equivalentemente, $X^* = M^{-1}X$, donde M^{-1} (al igual que M) es una matriz triangular inferior con todos elementos de la diagonal iguales a 1. Por lo tanto

$$x_k^* = x_k - \sum_{j=1}^{k-1} \lambda_{kj} x_j, \qquad \lambda_{kj} \in \mathbb{Q} \tag{2.2}$$

Aquí, al contrario que en (2.1), todos los vectores del lado derecho están "sin asterisco"; los λ_{kj} son los "coeficientes inversos" de Gram-Schmidt. Consideramos los vectores x_i para $1 \leq i \leq k-1$. Por el Teorema 2.1.1(a,b) tenemos $x_i \cdot x_k^* = 0$. Multiplicando escalarmente ambos lados de la ecuación (2.2) por x_i obtenemos $0 = x_i \cdot \left(x_k - \sum_{j=1}^{k-1} \lambda_{kj}(x_i \cdot x_j)\right)$ y de aquí que $\sum_{j=1}^{k-1}(x_i \cdot x_j)\lambda_{kj} = x_i \cdot x_k$.
Para un k fijo, hay $k-1$ posibilidades para i, y entonces tenemos un sistema lineal de $k-1$ ecuaciones con $k-1$ incógnitas $\lambda_{k1}, \lambda_{k2}, ..., \lambda_{k,k-1}$. La matriz de coeficientes de este sistema es la matriz de Gram G_{k-1}, y el determinante de G_{k-1} es d_{k-1}. La regla de Cramer implica que $d_{k-1}\lambda_{kj} \in \mathbb{Z}$ para $1 \leq j \leq k-1$. Ahora se tiene

$$d_{k-1}x_k^* = d_{k-1}\left(x_k - \sum_{j=1}^{k-1} \lambda_{kj}x_j\right) = d_{k-1}x_k - \sum_{j=1}^{k-1}(d_{k-1}\lambda_{kj})x_j$$

Luego, para $1 \leq k \leq n$ el vector $d_{k-1}x_k^*$ es una combinación lineal con coeficientes enteros de vectores con componentes enteros.

(b) Por la Proposición 2.1.2 vemos que $|x_j^*|^2 = d_j/d_{j-1}$, y aplicando la fórmula de la Definición 2.1.1 a μ_{kj} tenemos $d_j\mu_{kj} = d_j\frac{x_k \cdot x_j^*}{|x_j^*|^2} = d_{j-1}(x_k \cdot x_j^*) = x_k(d_{j-1}x_j^*)$. Por hipótesis $x_k \in \mathbb{Z}$, y por el apartado (a), $d_{j-1}x_j^* \in \mathbb{Z}^n$; entonces $d_j\mu_{kj} \in \mathbb{Z}$.

(c) Aplicando la fórmula de la Definición 2.1.1 a μ_{kj} y la desigualdad de Cauchy-Schwarz, obtenemos

$$|\mu_{kj}| = \left|\frac{x_k \cdot x_j^*}{x_j^* \cdot x_j^*}\right| = \frac{x_k \cdot x_j^*}{|x_j^*|^2} \leq \frac{|x_k||x_j^*|}{|x_j^*|^2} = \frac{|x_k|}{|x_j^*|}$$

Dado que $x_1, ..., x_j \in \mathbb{Z}^n$, vemos por la Definición 2.1.2 que $d_j \in \mathbb{Z}$, y como $d_j > 0$ entonces $d_j \geq 1$. Usando la Proposición 2.1.2 tenemos $|x_j^*|^2 = \frac{d_j}{d_{j-1}} \geq \frac{1}{d_{j-1}}$, entonces $|x_j^*| \geq \frac{1}{d_{j-1}^{1/2}}$.

□

2.3. Otros resultados del proceso de Gram-Schmidt

Una parte importante del algoritmo LLL será el continuo intercambio de vectores consecutivos de la base. La siguiente proposición, de la que omitimos la demostración, muestra como afectan estos intercambios a la ortogonalización de Gram-Schmidt de la base.

Proposición 2.3.1. *Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$, y sea $x_1^*, ..., x_n^*$ su ortogonalización de Gram-Schmidt. Sea j tal que $1 \leq j \leq n-1$, y sea $\hat{x}_1, ..., \hat{x}_n$ la nueva base obtenida de intercambiar x_j y x_{j+1}:*

$$\hat{x}_j = x_{j+1}, \qquad \hat{x}_{j+1} = x_j, \qquad \hat{x}_i = x_i \quad (i \neq j, j+1)$$

Sea $\hat{x}_1^*, ..., \hat{x}_n^*$ la ortogonalización de Gram-Schmidt de la nueva base. Entonces $\hat{x}_i^* = x_i^*$ para $i \neq j, j+1$ pero

$$\hat{x}_j^* = x_{j+1}^* + \mu_{j+1,j} x_j^*, \qquad \hat{x}_{j+1}^* = \frac{|x_{j+1}^*|^2}{|\hat{x}_j^*|^2} x_j^* - \mu_{j+1,j} \frac{|x_j^*|^2}{|\hat{x}_j^*|^2} x_{j+1}^*$$

Se puede escribir $\hat{x}_{j+1}^*$ enteramente en términos de vectores "sin gorro" en la forma siguiente:

$$\hat{x}_{j+1}^* = \frac{|x_{j+1}^*|^2}{|x_{j+1}^*|^2 + \mu_{j+1,j}^2 |x_j^*|^2} x_j^* - \mu_{j+1,j} \frac{|x_j^*|^2}{|x_{j+1}^*|^2 + \mu_{j+1,j}^2 |x_j^*|^2} x_{j+1}^2$$

El siguiente resultado da una cota inferior de la longitud de un vector no nulo de una red en términos de la ortogonalización de Gram-Schmidt de la base del mismo.

Proposición 2.3.2. *Sea $x_1, ..., x_n$ una base de $\mathbb{R}^n$, y sea $x_1^*, ..., x_n^*$ su ortogonalización de Gram-Schmidt. Sea $\mathcal{L}$ la red generada por $x_1, ..., x_n$. Para todo $y \in \mathcal{L}$ no nulo, se verifica*

$$|y| \geq \min\{|x_1^*|, ..., |x_n^*|\}$$

Esto es, todo vector no nulo de la red es, al menos, tan largo como el vector más corto de la ortogonalización de Gram-Schmidt.

Demostración. Sea y cualquier elemento no nulo de $\mathcal{L}$, $y = \sum_{i=1}^n r_i x_i$, donde $r_i \in \mathbb{Z}$ para $1 \leq i \leq n$. Como $y \neq 0$ tenemos que $r_i \neq 0$ para algún i; sea k el mayor índice tal que $r_k \neq 0$. Usando la Definición 2.1.1, podemos expresar $x_1, ..., x_n$ en términos de $x_1^*, ..., x_n^*$:

$$y = \sum_{i=1}^{k} r_i \sum_{j=1}^{i} \mu_{ij} x_j^* = \sum_{i=1}^{k} \sum_{j=1}^{i} r_i \mu_{ij} x_j^*$$

Cambiando el orden de la suma, y usando que $\mu_{kk} = 1$, obtenemos

$$y = \sum_{j=1}^{k} \left(\sum_{i=j}^{k} r_i \mu_{ij} \right) x_j^* = r_k x_k^* + \sum_{j=1}^{k-1} \nu_j x_j^*,$$

para algún $\nu_1, ..., \nu_{k-1} \in \mathbb{R}$. Dado que $x_1^*, ..., x_n^*$ son ortogonales, queda

$$|y|^2 = r_k^2 |x_k^*|^2 + \sum_{j=1}^{k-1} \nu_j^2 |x_j^*|^2$$

Como r_k es un entero distinto de cero, tenemos $r_k^2 \geq 1$, y entonces

$$|y|^2 \geq |x_k^*|^2 + \sum_{j=1}^{k-1} \nu_j^2 |x_j^*|^2$$

Todos los términos de la suma son no negativos, y por tanto

$$|y|^2 \geq |x_k^*|^2 \geq \min\{|x_1^*|^2, ..., |x_n^*|^2\}$$

Tomando raíces cuadradas se completa la demostración. □

CAPÍTULO 3

El algoritmo LLL

3.1. Base reducida de una red

Sea $x_1, x_2, ..., x_n$ una base ordenada de la red $\mathcal{L}$ de $\mathbb{R}^n$, y sea $x_1^*, x_2^*, ...x_n^*$ su ortogonalización de Gram-Schmidt. Escribimos $X = MX^*$, donde X (respectivamente X^*) es la matriz con x_i (respectivamente x_i^*) en la fila i, y $M = (\mu_{ij})$ es la matriz de coeficientes de la ortogonalización de Gram-Schmidt. Fijemos un parámetro de reducción α, con $\frac{1}{4} < \alpha < 1$. (El *valor estándar* del parámetro es $\alpha = \frac{3}{4}$).

Definición 3.1.1. La base $x_1, x_2, ..., x_n$ se llama α-*reducida* (o *LLL-reducida con parámetro* α) si satisface:

(1) $|\mu_{ij}| \leq \frac{1}{2}$ para $1 \leq j < i \leq n$,

(2) $|x_i^* + \mu_{i,i-1}x_{i-1}^*| \geq \alpha |x_{i-1}^*|^2$ para $2 \leq i \leq n$

La condición (2) se conoce como *condición de intercambio*. Como $x_1^*, x_2^*, ..., x_n^*$ son ortogonales, la condición (2) se puede escribir como

(2') $|x_i^*|^2 \geq (\alpha - \mu_{i,i-1}^2)|x_{i-1}^*|^2$ para $2 \leq i \leq n$

La condición (1) dice que cada vector x_i de la base es "casi ortogonal" al generador de los vectores anteriores, ya que por el Teorema 2.1.1 tenemos $< x_1, ..., x_{i-1} > = < x_1^*, ..., x_{i-1}^* >$. Las condiciones (2) y (2') dicen que intercambiando x_{i-1} y x_i y calculando de nuevo la ortogonalización de Gram-Schmidt se puede generar un nuevo vector más corto $\hat{x}_{i-1}^* = x_i^* + \mu_{i,i-1}x_{i-1}^*$ pero no "mucho más" corto; esto usa la Proposición 2.3.1.

Probaremos que, para cualquier red $\mathcal{L}$ de $\mathbb{R}^n$, cualquier base $x_1, x_2, ..., x_n$ de $\mathcal{L}$ y cualquier $\alpha \in \left(\frac{1}{4}, 1\right)$, el algoritmo LLL genera una base α-reducida de $\mathcal{L}$ en un número de pasos acotado por un polinomio del tamaño de la entrada. También debemos considerar $\alpha = 1$, el "caso límite" de la Definición 3.1.1, pero para este valor del parámetro de reducción no podemos probar que el algoritmo LLL termine en tiempo polinómico.

Definición 3.1.2. Definimos el *parámetro auxiliar* β como:

$$\beta = \frac{4}{4\alpha - 1} \quad \text{asque} \quad \beta > \frac{3}{4} \quad \text{y} \quad \frac{1}{\beta} = \alpha - \frac{1}{4}$$

Para el valor estándar $\alpha = \frac{3}{4}$ se obtiene $\beta = 2$.

Ejemplo 3.1.1. Sea $\mathcal{L}$ la red del Ejemplo ; las filas x_1, x_2, x_3, x_4 de la matriz X forman una base de $\mathcal{L}$. La ortogonalización de Gram-Schmidt se puede escribir como las filas $x_1^*, x_2^*, x_3^*, x_4^*$ de la matriz X^*. Las matrices X y X^* están relacionadas por la ecuación $X = MX^*$ donde $M = (\mu_{ij})$:

$$X = \begin{pmatrix} -2 & 7 & 7 & -5 \\ 3 & -2 & 6 & -1 \\ 2 & -8 & -9 & -7 \\ 8 & -9 & 6 & -4 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{27}{127} & 1 & 0 & 0 \\ -\frac{88}{127} & -\frac{799}{5621} & 1 & 0 \\ -\frac{17}{127} & \frac{10873}{5621} & \frac{350695}{765183} & 1 \end{pmatrix} \begin{pmatrix} -2 & 7 & 7 & -5 \\ \frac{435}{127} & -\frac{443}{127} & \frac{573}{127} & \frac{8}{127} \\ \frac{6189}{5621} & -\frac{20491}{5621} & -\frac{19720}{5621} & -\frac{58771}{5621} \\ \frac{153576}{255061} & \frac{271760}{765183} & -\frac{139672}{765183} & \frac{632}{765183} \end{pmatrix} = MX^*$$

Para la matriz M es claro que la base x_1, x_2, x_3, x_4 no es α-reducida para ningún α, ya que ni siquiera se cumple la primera condición de la Definición 3.1.1:

$$|\mu_{31}|, |\mu_{42}| > \frac{1}{2}$$

Considerando ahora la matriz C cuyos elementos están en $\mathbb{Z}$ y $\det(C) = -1$, y la nueva base y_1, y_2, y_3, y_4 para la red $\mathcal{L}$ dada por las filas de la matriz $Y = CX$

$$Y = CX = \begin{pmatrix} 1 & -8 & -2 & 4 \\ 1 & -6 & -1 & 3 \\ 0 & 4 & 1 & -2 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} -2 & 7 & 7 & -5 \\ 3 & -2 & 6 & -1 \\ 2 & -8 & -9 & -7 \\ 8 & -9 & 6 & -4 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & -1 \\ 2 & 0 & -2 & -4 \\ -2 & 2 & 3 & -3 \\ 3 & -2 & 6 & -1 \end{pmatrix}$$

La ortogonalización de Gram-Schmidt de y_1, y_2, y_3, y_4 se puede escribir como las filas $y_1^*, y_2^*, y_3^*, y_4^*$ de la matriz Y^* que satisface la ecuación matricial $Y = \hat{M}Y^*$ donde $\hat{M} = (\hat{\mu}_{ij})$:

$$Y = \hat{M}Y^* = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{2}{15} & 1 & 0 & 0 \\ \frac{2}{15} & \frac{17}{178} & 1 & 0 \\ \frac{1}{3} & -\frac{5}{89} & \frac{931}{2271} & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 & 1 \\ \frac{34}{15} & \frac{2}{5} & -\frac{28}{15} & -\frac{58}{15} \\ -\frac{221}{89} & \frac{139}{89} & \frac{271}{89} & -\frac{246}{89} \\ \frac{7900}{2271} & -\frac{8216}{2271} & \frac{9796}{2271} & -\frac{316}{757} \end{pmatrix}$$

Usando esta ecuación se puede verificar que la base y_1, y_2, y_3, y_4 es α-reducida para todo $\alpha < 1$. Es claro que la primera condición de la Definición 3.1.1 se cumple, pues $\hat{\mu}_{ij} \leq \frac{1}{2}$ para todo $1 \leq j < i \leq 4$. Para la segunda condición, es suficiente considerar el valor límite $\alpha = 1$, ya que es fácil ver que si la base de la red es α-reducida entonces es α'-reducida para cualquier $\alpha' < \alpha$. Calculamos ahora:

$|y_1^*|^2 = 15$

$|y_2^*|^2 = \frac{356}{15} \approx 27{,}73, \quad \hat{\mu}_{21}^2 = \frac{4}{225}, \quad (1-\hat{\mu}_{21}^2)|y_1^*|^2 = \frac{221}{15} \approx 14{,}73$

$|y_3^*|^2 = \frac{2271}{89} \approx 25{,}52, \quad \hat{\mu}_{32}^2 = \frac{289}{31684}, \quad (1-\hat{\mu}_{32}^2)|y_2^*|^2 = \frac{2093}{89} \approx 23{,}52$

$|y_4^*|^2 = \frac{99856}{2271} \approx 43{,}97, \quad \hat{\mu}_{43}^2 = \frac{866761}{5157441}, \quad (1-\hat{\mu}_{43}^2)|y_3^*|^2 = \frac{4290680}{202119} \approx 21{,}23$

De aquí tenemos que $|y_i^*|^2 \geq (1-\hat{\mu}_{i,i-1}^2)|y_{i-1}^*|^2$ para $i = 1, 2, 3, 4$. Luego la base y_1, y_2, y_3, y_4 es α-reducida para cualquier α tal que $\frac{1}{4} < \alpha < 1$.

El apartado (c) del siguiente resultado (Proposición 3.1.1) da una cota superior para la longitud $|x_1|$ del primer vector de una base α-reducida de una red en términos del determinante de la red. El resultado posterior (Teorema 3.1.2) da una cota superior para la longitud $|x_1|$ en términos del vector más corto (no nulo) en la red.

Proposición 3.1.1. *Si $x_1, x_2, ..., x_n$ es una base α-reducida de la red $\mathcal{L}$ de $\mathbb{R}^n$, y $x_1^*, x_2^*, ..., x_n^*$ es su ortogonalización de Gram-Schmidt, entonces*

(a) $|x_j|^2 \leq \beta^{i-j}|x_i^*|$ *para* $1 \leq j \leq i \leq n$

(b) $\det(\mathcal{L}) \leq |x_1||x_2|...|x_n| \leq \beta^{n(n-1)/4}\det(\mathcal{L})$

(c) $|x_1| \leq \beta^{(n-1)/4}(\det(\mathcal{L}))^{1/n}$

donde β es el parámetro auxiliar de la Definición 3.1.2.

Demostración. Las dos condiciones de la Definición 3.1.1 implican que para $1 < i \leq n$ se tiene $|x_i^*|^2 \geq (\alpha - \mu_{i,i-1}^2)|x_{i-1}^*|^2 \geq \left(\alpha - \frac{1}{4}\right)|x_{i-1}^*|^2 = \frac{1}{\beta}|x_{i-1}^*|^2$
Por lo tanto $|x_{i-1}^*|^2 \leq \beta|x_i^*|^2$, y por inducción de obtiene

$$|x_j^*| \leq \beta^{i-j}|x_i^*|^2 \quad (1 \leq j \leq i \leq n) \tag{3.1}$$

La definición de x_i^* en la ortogonalización de Gram-Schmidt se puede reescribir como $x_i = x_i^* + \sum_{j=1}^{i-1}\mu_{ij}x_j^*$, y como $x_1^*, ..., x_n^*$ son ortogonales, $|x_i|^2 = |x_i^*|^2 + \sum_{j=1}^{i-1}\mu_{ij}^2|x_j^*|^2$. De la Definición 3.1.1 y la ecuación (3.1)

$$|x_i|^2 = |x_i^*|^2 + \sum_{j=1}^{i-1}\frac{1}{4}\beta^{i-j}|x_i^*|^2 = \left(1 + \frac{1}{4}\sum_{j=1}^{i-1}\beta^{i-j}\right)|x_i^*|^2$$

Usando la fórmula de la suma de una progresión geométrica, se obtiene

$$|x_i|^2 \leq \left(1 + \frac{1}{4}\cdot\frac{\beta^i - \beta}{\beta - 1}\right)|x_i^*|^2$$

Por inducción sobre i se demuestra que

$$1 + \frac{1}{4}\cdot\frac{\beta^i - \beta}{\beta - 1} \leq \beta^{i-1}$$

Para $i = 1$ es trivial. Para el paso de inducción es suficiente probar que

$$1 + \frac{1}{4} \cdot \frac{\beta^{i+1} - \beta}{\beta - 1} \leq \beta \left(1 + \frac{\beta^i - \beta}{4(\beta - 1)}\right)$$

Como $\beta > \frac{4}{3}$ multiplicando por $4(\beta - 1)$ se tiene una desigualdad equivalente, que simplificada queda $(\beta - 1)(3\beta - 4) \geq 0$. Ahora tenemos

$$|x_i|^2 \leq \beta^{i-1}|x_i^*|^2 \tag{3.2}$$

Usando la ecuación (3.1)

$$|x_j|^2 \leq \beta^{j-1}|x_j^*|^2 \leq \beta^{i-1}|x_i^*|^2 \quad (1 \leq j \leq i \leq n)$$

lo que prueba (a). Por el Teorema 2.1.1 sabemos que

$$\det(\mathcal{L}) = |x_1^*||x_2^*|...|x_n^*| \leq |x_1||x_2|...|x_n|$$

lo que prueba la desigualdad de la izquierda de (b). La ecuación (3.2) implica

$$|x_1|^2|x_2|^2...|x_n|^2 \leq \beta^{0+1+2+...+(n-1)}|x_1^*|^2|x_2^*|^2...|x_n^*|^2$$

y por lo tanto

$$|x_1||x_2|...|x_n| \leq \beta^{n(n-1)/4}|x_1^*||x_2^*|...|x_n^*| = \beta^{n(n-1)/4}\det(\mathcal{L})$$

lo que prueba la desigualdad de la derecha de (b). Fijando $j = 1$ en (a) se tiene

$$|x_1|^2 \leq \beta^{i-1}|x_i^*| \quad (1 \leq i \leq n)$$

y tomando el producto $i = 1, 2, ..., n$

$$|x_1|^{2n} \leq \beta^{0+1+2+...+(n-1)}|x_1^*|^2|x_2^*|^2...|x_n^*|^2 = \beta^{n(n-1)/2}(\det(\mathcal{L}))^2$$

Tomando la raíz $2n$ se prueba (c). □

Teorema 3.1.2 (Teorema LLL). *Si $x_1, x_2, ..., x_n$ es una base α-reducida de la red $\mathcal{L}$ de $\mathbb{R}^n$, e $y \in \mathcal{L}$ es cualquier vector no nulo de la red, entonces*

$$|x_1| \leq \beta^{(n-1)/2}|y|$$

En particular, el primer vector de la base α-reducida no es más largo que $\beta^{(n-1)/2}$ veces el vector más corto no nulo de $\mathcal{L}$.

Demostración. Sea $x_1^*, x_2^*, ..., x_n^*$ la ortogonalización de Gram-Schmidt de $x_1, x_2, ..., x_n$. Por la definición de base α-reducida, para $2 \leq i \leq n$ se tiene

$$|x_i^*|^2 \geq (\alpha - \mu_{i,i-1}^2)|x_{i-1}^*|^2 \geq \left(\alpha - \frac{1}{4}\right)|x_{i-1}^*|^2 = \frac{1}{\beta}|x_{i-1}^*|^2$$

Como $x_1^* = x_1$, $|x_1|^2 = |x_1^*|^2 \leq \beta|x_2^*|^2 \leq \beta^2|x_3^*|^2 \leq ... \leq \beta^{n-1}|x_n^*|^2$, y entonces para $1 \leq i \leq n$ se tiene $|x_i^*|^2 \geq \beta^{-(i-1)}|x_1|^2$. Por la Proposición 2.3.2 se demuestra que para cualquier vector $g \in \mathcal{L}$ no nulo $|y| \geq \min\{|x_1^*|, ..., |x_n^*|\} \geq \beta^{-(n-1)/2}|x_1|^2$, y esto completa la prueba. □

El resultado anterior es el caso $m = 1$ del siguiente resultado, que da una cota superior para la longitud de todos los vectores en una base α-reducida, del que obviamos la demostración

Teorema 3.1.3. *Si $x_1, x_2, ..., x_n$ es una base α-reducida de la red $\mathcal{L}$ de $\mathbb{R}^n$, e $y_1, y_2, ..., y_m \in \mathcal{L}$, m vectores linealmente independientes de la red cualesquiera, entonces para $1 \leq j \leq m$*

$$|x_j| \leq \beta^{(n-1)/2} \max\{|y_1|, ..., |y_m|\}$$

3.2. El algoritmo LLL original

Se muestra el algoritmo LLL original, escrito de una forma más "estructurada". La entrada consiste en una base $x_1, x_2, ..., x_n$ de la red $\mathcal{L} \subset \mathbb{R}^n$, y un parámetro de reducción $\alpha \in \mathbb{R}$ tal que $\frac{1}{4} < \alpha < 1$. La salida consiste en una base α-reducida $y_1, y_2, ..., y_n$ de la red $\mathcal{L}$.

- **Entrada.** Una base $x_1, x_2, .., x_n$ de la red $\mathcal{L} \subset \mathbb{R}^n$, y un parámetro de reducción $\alpha \in \mathbb{R}$ tal que $\frac{1}{4} < \alpha < 1$.
- **Salida.** Una base α-reducida $y_1, y_2, ..., y_n$ de la red $\mathbb{L}$.
- **Procedure** `reduce`(k, l):
  If $|\mu_{kl}| > \frac{1}{2}$ then
  (1) Set $y_k \longleftarrow y_k - \lceil \mu_{kl} \rfloor y_l$.
  (2) For $j = 1, 2, ..., l-1$ do: Set $\mu_{kj} \longleftarrow \mu_{kj} - \lceil \mu_{kl} \rfloor \mu_{lj}$.
  (3) Set $\mu_{kl} \longleftarrow \mu_{kl} - \lceil \mu_{kl} \rfloor$.
- **Procedure** `exchange`(k):
  (1) Set $z \longleftarrow y_{k-1}, y_{k-1} \longleftarrow y_k, y_k \longleftarrow z$ (intercambiar y_{k-1} e y_k).
  (2) Set $\nu \longleftarrow \mu_{k,k-1}$. Set $\delta \longleftarrow \gamma_k^* + \nu^2 \gamma_{k-1}^*$.
  (3) Set $\mu_{k,k-1} \longleftarrow \nu \gamma_{k-1}^* / \delta$. Set $\gamma_k^* \longleftarrow \gamma_k^* \gamma_{k-1}^* / \delta$. Set $\gamma_{k-1}^* \longleftarrow \delta$.
  (4) For $j = 1, 2, ..., k-2$ do: Set $t \longleftarrow \mu_{k-1,j}, \mu_{k-1,j} \longleftarrow \mu_{kj}, \mu_{kj} \longleftarrow t$ (intercambiar $\mu_{k-1,j}$ y μ_{kj}).
  (5) For $i = k+1, ..., n$ do:
    (a) Set $\psi \longleftarrow \mu_{ik}$. Set $\mu_{ik} \longleftarrow \mu_{i,k-1} - \nu \mu_{ik}$.
    (b) Set $\mu_{i,k-1} \longleftarrow \mu_{k,k-1} \mu_{ik} + \psi$.
- **Main loop:**
  (1) For $i = 1, 2, ..., n$ do: Set $y_i \longleftarrow x_i$.
  (2) For $i = 1, 2, ..., n$ do:
    (a) Set $y_i^* \longleftarrow y_i$.

(b) For $j = 1, 2, ..., i-1$ do:
Set $\mu_{ij} \longleftarrow (y_i \cdot y_j^*)/\gamma_j^*$ e $y_i^* \longleftarrow y_i^* - \mu_{ij} y_j^*$.
(c) Set $\gamma_i^* \longleftarrow y_i^* \cdot y_i^*$.

(3) Set $k \longleftarrow 2$.

(4) While $k \leq n$ do:

(a) Call `reduce`$(k, k-1)$.
(b) If $\gamma_k^* \geq (\alpha - \mu_{k,k-1}^2)\gamma_{k-1}^*$ then
(i) For $l = k-2, ..., 2, 1$ do: Call `reduce`(k, l).
(ii) Set $k \longleftarrow k+1$.
else
(iii) Call `exchange`(k).
(iv) If $k > 2$ then set $k \longleftarrow k-1$.

El paso (1) de **Main loop** simplemente realiza una copia $y_1, y_2, ..., y_n$ de los vectores de entrada $x_1, x_2, ..., x_n$. El paso (2) calcula los vectores de la ortogonalización de Gram-Schmidt $y_1, y_2, ..., y_n$. El paso (3) inicializa el índice k del vector y_k que se está procesando. El paso (4) ejecuta la reducción de base; llama repetidamente a dos **Procedure** que reducen e intercambian los vectores $y_1, y_2, ..., y_n$.

Procedure `reduce`(k, l) hace a y_k casi ortogonal a y_l. Si $|\mu_{kl}| \leq \frac{1}{2}$ entonces **Procedure** no hace nada; de lo contrario, reduce y_k restando el múltiplo entero $\lceil \mu_{kl} \rfloor$ de y_l. Como $\lceil \mu_{kl} \rfloor$ es el entero más cercano al coeficiente de Gram-Schmidt μ_{kl} , ésta es la mejor reducción posible que se puede realizar, dado que y_k debe permanecer en la red generada por $x_1, x_2, ..., x_n$. Después **Procedure** actualiza la base y los coeficientes de la ortogonalización de Gram-Schmidt.

Procedure `exchange`(k) intercambia los vectores y_{k-1} e y_k, y luego actualiza la base y los coeficientes de la ortogonalización de Gram-Schmidt de acuerdo con la Proposición 2.3.1.

Los vectores $y_1, y_2, ..., y_n$ se modifican continuamente a lo largo del algoritmo, pero de tal manera que siempre formen parte de una base de red $\mathcal{L}$. El paso (4)(a) de **Main loop** reduce y_k usando y_{k-1}; esto se hace antes de comprobar la condición de reducción del paso (4)(b) ya que esta condición depende solamente de esos dos vectores. En el paso (4)(b), si se satisface la condición de la reducción, y_k queda completamente reducido usando $y_{k-2}, ..., y_2, y_1$ y entonces se aumenta el índice k; en caso contrario, se realiza un intercambio, y en índice k disminuye (si no tiene ya el mínimo valor $k = 2$).

Ejemplo 3.2.1. Empezamos con los vectores x_1, x_2, x_3, x_4 del Ejemplo 3.1.1 que forman una base de la red $\mathcal{L}$ de $\mathbb{R}^4$. Implementaremos el algoritmo LLL usando para el parámetro de reducción el valor límite $\alpha = 1$.

El paso (1) realiza una copia y_1, y_2, y_3, y_4 de los vectores de la red, y el paso (2) calcula su ortogonalización de Gram-Schmidt $y_1^*, y_2^*, y_3^*, y_4^*$. En todo este ejemplo escribiremos el estado actual del algoritmo como la terna $(Y,\ M,\ \gamma^*)$, donde Y es la matriz con filas y_1, y_2, y_3, y_4, $M = (\mu_{ij})$ es la matriz de coeficientes de Gram-Schmidt, y γ^* es el vector

columna del cuadrado de las longitudes de los vectores de la ortogonalización de Gram-Schmidt $y_1^*, y_2^*, y_3^*, y_4^*$. El estado inicial (iteración 0) es

$$\begin{pmatrix} -2 & 7 & 7 & -5 \\ 3 & -2 & 6 & -1 \\ 2 & -8 & -9 & -7 \\ 8 & -9 & 6 & -4 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{27}{127} & 1 & 0 & 0 \\ -\frac{88}{127} & -\frac{799}{5621} & 1 & 0 \\ -\frac{17}{127} & \frac{10873}{5621} & \frac{350695}{765183} & 1 \end{pmatrix}, \begin{pmatrix} 127 \\ \frac{5621}{127} \\ \frac{765183}{5621} \\ \frac{399424}{765183} \end{pmatrix}$$

El paso (3) fija $k = 2$. El paso 4(a) llama a `reduce`$(2, 1)$; como $|\mu_{21}| = 27/127 < 1/2$, **Procedure** no realiza ninguna acción. El paso (4)(b) comprueba la condición de intercambio; tenemos

$$\gamma_2^* = \frac{5621}{127} < \frac{15400}{127} = (1 - \mu_{21}^2)\gamma_1^*$$

Luego, el paso (4)(b)(iii) llama a `exchange`(2); el estado actual (iteración 1) es

$$\begin{pmatrix} 3 & -2 & 6 & -1 \\ -2 & 7 & 7 & -5 \\ 2 & -8 & -9 & -7 \\ 8 & -9 & 6 & -4 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{27}{50} & 1 & 0 & 0 \\ -\frac{1}{2} & -\frac{3725}{5621} & 1 & 0 \\ -\frac{41}{25} & -\frac{3064}{5621} & \frac{350695}{765183} & 1 \end{pmatrix}, \begin{pmatrix} 50 \\ \frac{5621}{50} \\ \frac{765183}{5621} \\ \frac{399424}{765183} \end{pmatrix}$$

Como $k = 2$ no disminuimos k en el paso (4)(b)(iv); regresamos al principio del bucle con $k = 2$.

El paso (4)(a) llama a `reduce`$(2, 1)$; como $|\mu_{21}| = 27/50 > 1/2$ y $\lceil \mu_{21} \rfloor = 1$, el **Procedure** reduce y_2 restando y_1. El estado actual (iteración 2) es

$$\begin{pmatrix} 3 & -2 & 6 & -1 \\ -5 & 9 & 1 & -4 \\ 2 & -8 & -9 & -7 \\ 8 & -9 & 6 & -4 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{23}{50} & 1 & 0 & 0 \\ -\frac{1}{2} & -\frac{3725}{5621} & 1 & 0 \\ \frac{41}{25} & -\frac{3064}{5621} & \frac{350695}{765183} & 1 \end{pmatrix}, \begin{pmatrix} 50 \\ \frac{5621}{50} \\ \frac{765183}{5621} \\ \frac{399424}{765183} \end{pmatrix}$$

El paso (4)(b) comprueba la condición de intercambio; tenemos

$$\gamma_2^* = \frac{5621}{50} \geq \frac{1971}{50} = (1 - \mu_{21}^2)\gamma_1^*$$

Como $k = 2$, el paso (4)(b)(i) no hace nada, y el paso (4)(b)(ii) incrementa k; ahora regresamos al principio del bucle con $k = 3$.

El paso (4)(a) llama a `reduce`$(3, 2)$; como $|\mu_{32}| = 3725/5621 > 1/2$ y $\lceil \mu_{32} \rfloor = -1$, **Procedure** reduce y_3 sumando y_2. El estado actual (iteración 3) es

$$\begin{pmatrix} 3 & -2 & 6 & -1 \\ -5 & 9 & 1 & -4 \\ -3 & 1 & -8 & -11 \\ 8 & -9 & 6 & -4 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{23}{50} & 1 & 0 & 0 \\ -\frac{24}{25} & \frac{1896}{5621} & 1 & 0 \\ \frac{41}{25} & -\frac{3064}{5621} & \frac{350695}{765183} & 1 \end{pmatrix}, \begin{pmatrix} 50 \\ \frac{5621}{50} \\ \frac{765183}{5621} \\ \frac{399424}{765183} \end{pmatrix}$$

El paso (4)(b) comprueba la condición de intercambio; tenemos

$$\gamma_3^* = \frac{765183}{5621} \geq \frac{1120033}{11242} = (1 - \mu_{32}^2)\gamma_2^*$$

El paso (4)(b)(i) llama a `reduce`$(3,1)$; como $|\mu_{31}| = 24/25 > 1/2$ y $\lceil \mu_{31} \rfloor = -1$, **Procedure** reduce y_3 sumando y_1. El estado actual (iteración 4) es

$$\begin{pmatrix} 3 & -2 & 6 & -1 \\ -5 & 9 & 1 & -4 \\ 0 & -1 & -2 & -12 \\ 8 & -9 & 6 & -4 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{23}{50} & 1 & 0 & 0 \\ \frac{1}{25} & \frac{1896}{5621} & 1 & 0 \\ \frac{41}{25} & -\frac{3064}{5621} & \frac{350695}{765183} & 1 \end{pmatrix}, \begin{pmatrix} 50 \\ \frac{5621}{50} \\ \frac{765183}{5621} \\ \frac{399424}{765183} \end{pmatrix}$$

El paso (4)(b)(ii) incrementa k, y regresamos al principio del bucle con $k = 4$.

Después de otras 19 iteraciones, el algoritmo termina. La tabla siguiente resume el resto de las reducciones e intercambios realizados.

iteración 5	intercambiar	$k = 4$		
iteración 6	reducir	$k = 3$	$l = 2$	$\lceil \mu_{kl} \rfloor = -1$
iteración 7	intercambiar	$k = 3$		
iteración 8	reducir	$k = 2$	$l = 1$	$\lceil \mu_{kl} \rfloor = 1$
iteración 9	reducir	$k = 3$	$l = 2$	$\lceil \mu_{kl} \rfloor = 1$
iteración 10	reducir	$k = 3$	$l = 1$	$\lceil \mu_{kl} \rfloor = -1$
iteración 11	reducir	$k = 4$	$l = 3$	$\lceil \mu_{kl} \rfloor = -1$
iteración 12	intercambiar	$k = 4$		
iteración 13	reducir	$k = 3$	$l = 2$	$\lceil \mu_{kl} \rfloor = 1$
iteración 14	intercambiar	$k = 3$		
iteración 15	intercambiar	$k = 2$		
iteración 16	intercambiar	$k = 3$		
iteración 17	reducir	$k = 2$	$l = 1$	$\lceil \mu_{kl} \rfloor = 1$
iteración 18	intercambiar	$k = 2$		
iteración 19	intercambiar	$k = 4$		
iteración 20	reducir	$k = 3$	$l = 2$	$\lceil \mu_{kl} \rfloor = 1$
iteración 21	intercambiar	$k = 3$		
iteración 22	reducir	$k = 2$	$l = 1$	$\lceil \mu_{kl} \rfloor = -1$
iteración 23	intercambiar	$k = 2$		

El estado final del algoritmo es

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 2 & 0 & -2 & -4 \\ -2 & 2 & 3 & -3 \\ 3 & -2 & 6 & -1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{2}{15} & 1 & 0 & 0 \\ \frac{2}{15} & \frac{17}{178} & 1 & 0 \\ \frac{1}{3} & -\frac{5}{89} & \frac{931}{2271} & 1 \end{pmatrix}, \begin{pmatrix} 15 \\ \frac{356}{15} \\ \frac{2271}{89} \\ \frac{99856}{2271} \end{pmatrix}$$

En este ejemplo con cuatro vectores con componentes de una cifra, el algoritmo LLL ha mejorado sustancialmente los vectores de la base: el mayor vector reducido tiene la misma longitud que el menor vector original.

3.3. Análisis del algoritmo LLL

Para estudiar la complejidad del algoritmo LLL, debemos determinar cómo cambian la base de Gram-Schmidt y los coeficientes durante la reducción en el paso (4)(b)(i) y durante el intercambio en el paso (4)(b)(iii).

Lema 3.3.1 (**Lema de Reducción**). *Consideramos una llamada a* `reduce` *en el paso (4)(b)(i) con k y l dados, y escribimos $\nu = \lceil \mu_{kl} \rfloor$. Sean $Y = MY^*$ y $Z = NZ^*$ las ecuaciones matriciales para la ortogonalización de Gram-Schmidt antes y después de la llamada a* `reduce`(k,l). *Sea $E = I_n - \nu E_{kl}$ la matriz elemental que representa restar ν veces la fila l a la fila k (así $E_{ii} = 1$ para $1 \leq i \leq n$, $E_{kl} = -\nu, E_{ij} = 0$, en otro caso). Tenemos*

$$Z = EY, \quad N = EM, \quad Z^* = Y^*$$

En particular, la base ortogonal de Gram-Schmidt no cambia. Antes de la llamada a `reduce`(k,l) *tenemos $|\mu_{kj}| \leq \frac{1}{2} \quad (l < j < k)$. Después de la ejecución del bucle en l en el paso (4)(b)(i) tenemos $|\mu_{kj}| \leq \frac{1}{2} \quad (1 \leq j < k)$*

Demostración. Como Z se obtiene de Y restando ν veces la fila l a la fila k, se tiene que $Z = EY$. Dado que $l < k$, el conjunto generado por $y_1, y_2, ..., y_i$ permanece igual para todo i, y por lo tanto la base ortogonal $y_1^*, y_2^*, ..., y_n^*$ no cambia. Por consiguiente $Z^* = Y^*$, y esto da

$$(EM)Y^* = E(MY^*) = EY = Z = NZ^* = NY^*$$

y ya que Y^* es invertible obtenemos $EM = N$. Al inicio del bucle en el paso (4)(a)(i) tenemos $l = k - 1$, así que no hay valores de j tales que $l < j < k$; luego la condición $|\mu_{kj}| \leq \frac{1}{2}$ para $(l < j < k)$ se satisface trivialmente. Supongamos que la condición se verifica antes de la llamada a `reduce`(k,l). La operación $Z = EY$ sólo cambia la fila k de Y, y entonces para $l < j < k$ el coeficiente

$$\mu_{kj} = \frac{y_k \cdot y_j^*}{y_j^* \cdot y_j^*}$$

se convierte en

$$\mu'_{kj} = \frac{(y_k - \nu y_l) \cdot y_j^*}{y_j^* \cdot y_j^*} = \frac{y_k \cdot y_j^*}{y_j^* \cdot y_j^*} - \nu \frac{y_l \cdot y_j^*}{y_j^* \cdot y_j^*} = \mu_{kj} - \nu \mu_{lj} = \mu_{kj}$$

puesto que $j > l$ implica $\mu_{lj} = 0$. Para μ_{kl}, queda $\mu_{kl} - \nu\mu_{ll} = \mu_{kl} - \lceil \mu_{kl} \rfloor$ ya que $\mu_{ll} = 1$, y por definición de $\lceil \mu_{kl} \rfloor$ tenemos $|\mu_{kl} - \lceil \mu_{kl} \rfloor| \leq \frac{1}{2}$ □

El siguiente lema muestra que el producto de los cuadrados de las longitudes de los vectores de la base ortogonal disminuye en un factor de al menos el parámetro de reducción α durante cada ejecución del paso (4)(b)(iii). Este será el hecho clave en la prueba de que el algoritmo LLL termina, y explicará, además, porqué necesitamos asumir que α es estrictamente menor que 1.

Lema 3.3.2 (**Lema de intercambio**). *Consideramos la llamada a* `exchange` *con un k dado en el paso (4)(b)(iii). Sean $Y = MY^*$ y $Z = NZ^*$ las ecuaciones matriciales para la ortogonalización de Gram-Schmidt antes y después del intercambio. Tenemos*

$$z_i^* = y_i^* \ \ (i \neq k-1, k), \quad |z_{k-1}^*|^2 < \alpha |y_{k-1}^*|^2, \quad |z_k^*| \leq |y_{k-1}^*|$$

Demostración. Para todo i excepto $i = k-1, k$ tenemos $y_i = z_i$ y el conjunto generado por $y_1, y_2, ..., y_{i-1}$ es igual al conjunto generado por $z_1, z_2, ..., z_{i-1}$. Como y_i^* (respectivamente z_i^*) es la proyección de y_i (respectivamente z_i) sobre el complemento ortogonal del conjunto generado por $y_1, y_2, ..., y_{i-1}$ (respectivamente $z_1, z_2, ..., z_{i-1}$), la primera igualdad se cumple.

Como $z_k = y_{k-1}$ y $y_k = z_{k-1}$, el vector z_{k-1}^* es la componente de y_k ortogonal al conjunto generado por $y_1, y_2, ..., y_{k-2}$. Por la definición de los coeficientes μ_{ij} tenemos

$$y_k = y_k^* + \sum_{l=1}^{k-1} \mu_{kl} y_l^*$$

por lo tanto $z_{k-1}^* = y_k^* + \mu_{k,k-1} y_{k-1}^*$. Dado que y_k^* e y_{k-1}^* son ortogonales, obtenemos

$$|z_{k-1}^*|^2 = |y_k^*|^2 + \mu_{k,k-1}^2 |y_{k-1}^*|^2$$

Como hemos supuesto que la condición de intercambio $\gamma_k^* \geq (\alpha - \mu_{k,k-1}^2)\gamma_{k-1}^*$ en el paso (4)(b) es falsa, tenemos $|y_k^*|^2 < (\alpha - \mu_{k,k-1}^2)|y_{k-1}^*|^2$ y por lo tanto $|z_{k-1}^*|^2 < \alpha |y_{k-1}^*|^2$. Esto prueba la segunda afirmación. En cuanto a z_k^*, es la componente de y_{k-1} ortogonal al conjunto generado por $y_1, ..., y_{k-2}, y_k$. Llamamos U al conjunto generado por $y_1, ..., y_{k-2}$. Tenemos $y_{k-1} = y_{k-1}^* + \sum_{l=1}^{k-2} \mu_{k-1,l} y_l^* = y_{k-1}^* + u$ donde $u = \sum_{l=1}^{k-2} \mu_{k-1,l} y_l^*$. Por lo tanto z_k^* es la componente de $y_{k-1}^* + u$ ortogonal al subespacio $U + \mathbb{R}y_k$. Usando el Teorema 2.1.1, obtenemos

$$u \in < y_1^*, ..., y_{k-2}^* > = < y_1, ..., y_{k-2} > = U \subset U + \mathbb{R}y_k$$

De aquí, z_k^* es la componente de y_{k-1}^* ortogonal al subespacio $U + \mathbb{R}y_k$. □

Lema 3.3.3. *Al inicio de cada iteración del paso (4)(b), se verifica la siguiente condición*

$$|\mu_{ij}| \leq \frac{1}{2} (1 \leq j < i < k), \quad |y_i^* + \mu_{i,i-1} y_{i-1}^*|^2 \geq \alpha |y_{i-1}^*|^2 (2 \leq i < k)$$

Si el algoritmo termina, entonces la salida $y_1, y_2, ..., y_n$ es una base reducida de la red.

Nuestro siguiente objetivo será encontrar una cota superior para el número de iteraciones del bucle del paso (4) del algoritmo LLL. Para simplificar, asumiremos que la base original $x_1, x_2, ..., x_n$ tiene vectores de componentes enteras.

Lema 3.3.4. *Durante las llamadas a* `reduce`(k, l) *en el algoritmo LLL, el determinante de Gram d_i no cambia. Durante las llamadas a* `exchange`(k)*, el determinante de Gram d_i no cambia para $i \neq k-1$, pero d_{k-1} cambia a un nuevo valor $d'_{k-1} \leq \alpha d_{k-1}$, donde α es el parámetro de reducción.*

Demostración. El Lema 3.3.1 muestra que la base ortogonal $y_1^*, y_2^*, ..., y_n^*$ no cambia durante las llamadas a `reduce`, así que la primera afirmación sigue la Proposición 2.1.2. Para $i < k-1$, la llamada a `exchange`(k) no tiene efecto en la matriz de Gram G_i, y por consiguiente, no causa efecto en d_i. Para $i > k-1$, la llama a `exchange`(k) transpone dos filas de G_i y dos columnas de G_i^t, multiplicando el determinante $\det(G_i G_i^t)$ por $(-1)^2$, así

que de nuevo, no hay efecto en d_i. Para $i = k-1$ escribimos y_l^* y z_l^* para los vectores antes y después de a llamada a `exchange`(k). Tenemos

$$d'_{k-1} = \prod_{l=1}^{k-1} |z_l^*|^2 = |z_{k-1}^*|^2 \prod_{l=1}^{k-2} |z_l^*|^2 \leq \alpha |y_{k-1}^*|^2 \prod_{l=1}^{k-2} |z_l^*|^2 =$$

$$= \alpha |y_{k-1}^*|^2 \prod_{l=1}^{k-2} |y_l^*|^2 = \alpha \prod_{l=1}^{k-1} |y_l^*|^2 = \alpha d_{k-1}$$

□

Definición 3.3.1. El *invariante del bucle* es la cantidad $D = \prod_{k=1}^{n-1} d_k$

Escribimos D_0 para el valor de D al inicio del algoritmo. La asunción de que los vectores de la base original $x_1, .., x_n$ están en $\mathbb{Z}^n$ implica que D es un entero positivo en todo el algoritmo. (El término "invariante" no es del todo correcto, pues nuestro objetivo es probar que esta cantidad decrece estrictamente durante la ejecución del algoritmo LLL).

Lema 3.3.5. *Si* $B = \max(|x_1|, |x_2|, ..., |x_n|)$ *se verifica que*

$$D_0 \leq B^{n(n-1)}$$

Demostración. Tenemos

$$D_0 = \prod_{k=1}^{n-1} d_k = \prod_{k=1}^{n-1} \prod_{l=1}^{k} |x_l^*|^2 = (|x_1^*|^2)(|x_1^*|^2 |x_2^*|^2)...(|x_1^*|^2 |x_2^*|^2 ... |x_{n-1}^*|^2) =$$

$$\prod_{k=1}^{n-1} |x_k^*|^{2(n-k)} \leq \prod_{k=1}^{n-1} |x_k|^{2(n-k)} \leq \prod_{k=1}^{n-1} B^{2(n-k)} = \prod_{k=1}^{n-1} B^{2k} = B^{n(n-1)}$$

□

Lema 3.3.6. *Si* E *denota al número total de llamadas a* `exchange` *realizadas a lo largo de todo el algoritmo LLL, se tiene que*

$$E \leq -\frac{\log B}{\log \alpha} n(n-1)$$

Demostración. El Lema 3.3.4 implica que D disminuye como máximo hasta αD después de cada llamada a `exchange`. Ya que D es un entero positivo desde el inicio hasta el final de algoritmo, tenemos $1 \leq \alpha^E D_0$ o, equivalentemente, $\alpha^{-E} \leq D_0$. El Lema 3.3.5 implica que $\alpha^{-E} \leq B^{n(n-1)}$ o, equivalentemente, $-E \log \alpha \leq n(n-1) \log B$. Dividiendo entre $-\log \alpha$ (que es > 0 pues $\alpha < 1$) se completa la demostración. □

De este último lema se deduce que el algoritmo LLL termina.

Teorema 3.3.7 (**Teorema de terminación**). *El número total de ejecuciones del bucle del paso (4) del algoritmo LLL es a lo sumo*

$$-\frac{2\log B}{\log \alpha} n(n-1) + (n-1)$$

Demostración. Por definición, E es el número de veces que el algoritmo ejecuta los pass (4)(b)(iii) y (4)(b)(iv). Llamamos E' al número de veces que el algoritmo ejecuta los pasos (4)(b)(i) y (4)(b)(ii). Así $E + E'$ es el número total de ejecuciones del bucle del paso (4). Cada vez que E aumenta en 1, el índice k disminuye en 1; cada vez que E' aumenta en 1, el índice k aumenta en 1. Se sigue que el entero $k + E - E'$ permanece constante a lo largo de todo el algoritmo. Al inicio, tenemos $k = 2$ y $E = E' = 0$, entonces $k + E - E' = 2$. Al final, tenemos $k = n + 1$, luego $n + 1 + E - E' = 2$, y por consiguiente $E' - E = n - 1$. Esto nos da que $E' + E = 2E + n - 1$, y por el Lema 3.3.6 completamos la prueba. □

CAPÍTULO 4

Factorización de polinomios con coeficientes enteros

En este capítulo estudiaremos la primera aplicación importante del algoritmo LLL, que dio título al artículo original de Lenstra, Lenstra y Lovasz (1982): "Un algoritmo polinomial para factorizar polinomios en una variable con coeficientes en el cuerpo de los números racionales" [4]. Hasta el momento de aparición de este artículo el mejor algoritmo para factorizar polinomios con coeficientes enteros era el denominado Berlekamp-Zassenhaus [1] el cual, en el peor de los casos, el número de iteraciones que realizaba es exponencial en el grado de f.

Los dos algoritmos mencionados usan la misma estrategia. Un polinomio $f \in \mathbb{Z}[x]$ se factoriza módulo un primo p adecuado y después usa la elevación de Hensel para obtener la factorización módulo p^2, p^4, p^8,..., hasta una cota p^n. Los factores así obtenidos se usan para obtener los factores enteros de f en $\mathbb{Z}[x]$. Hacemos a continuación un resumen de este proceso de factorización sin adjuntar las demostraciones que pueden verse en [5]. Analizaremos con detalle la mejora que introduce el algoritmo LLL para conseguir un algoritmo polinomial en lugar del exponencial de Zassenhaus.

4.1. Factorización sobre cuerpos finitos

Consideramos un polinomio mónico no constante libre de cuadrados $f \in \mathbb{F}_q[x]$, donde $\mathbb{F}_q$ es cualquier cuerpo finito. Dado que $\mathbb{F}_q[x]$ es un dominio de factorización única, sabemos que

$$f = \prod_{i=1}^{k} g_i,$$

donde los factores $g_i \in \mathbb{F}_q[x]$ son distintos, mónicos e irreducibles. Sea δ el máximo de los grados de los factores irreducibles:

$$\delta = \max\{\deg(g_1), \deg(g_2), ..., \deg(g_k)\}$$

Para cada grado $d = 1, 2, ..., \delta$ escribimos l_d para el número de factores irreducibles distintos de grado d, y reunimos estos factores atendiendo al grado:

$$f = \prod_{d=1}^{\delta} h_d, \qquad h_d = \prod_{j=1}^{l_d} h_{dj},$$

donde $\deg(h_{dj}) = d$. Así h_d es el producto de los factores irreducibles de f de grado d; si $l_d = 0$ entonces el producto es vacío y $h_d = 1$.

Definición 4.1.1. La *descomposición de distinto grado* de f es la sucesión

$$ddd(f) = [h_1, h_2, ..., h_\delta]$$

El algoritmo que proporciona esta factorización de un polinomio sobre un cuerpo finito $f \in \mathbb{F}_q[x]$ lo denominaremos **DDD** (Véase Anexo I), en él se hace uso del algoritmo de Euclides (**Euclid**) para calcular el máximo común divisor de dos polinomios.

A continuación se descompone cada uno de los polinomios h_d en sus factores irreducibles. Sea por tanto un polinomio mónico no constante y libre de cuadrados $h_d \in \mathbb{F}_q[x]$, y suponemos que cada factor de h_d tiene grado d:

$$h_d = \prod_{j=1}^{l_d} h_{dj}, \qquad \deg(h_{dj}) = d,$$

donde $l_d \geq 1$ es el número de factores mónicos irreducibles distintos.

Definición 4.1.2. La *descomposición de igual grado* de h_d es la sucesión

$$edd(h_d) = [h_{d1}(x), ..., h_{dl_d}]$$

Para calcular la descomposición de igual grado usamos algoritmos probabilísticos y suponemos que $q = p^n$, $p \neq 2$ (Véase Anexo II). Los algoritmos **TrialSplit** y **Split** simplemente descomponen el polinomio en dos factores. La probabilidad de encontrar con **Split** un factor propio de h_d haciendo s llamadas al algoritmo **TrialSplit** es $\geq 1 - 2^{-s}$, que converge a 1 tan rápido como s aumenta. Para aumentar la probabilidad de encontrar un factor propio de h, usamos el algoritmo **Split**(h, s) cuya entrada de **Split**(h, s) incluye un parámetro de terminación $s \geq 1$; ésta es la cota superior del número de procesos que **Split** realizará. Para encontrar la descomposición de igual grado completa, usamos el algoritmo **EDD**(h) que llama a **Split** sucesivamente hasta que no se pueda continuar dividiendo.

La factorización completa del polinomio f (no necesariamente libre de cuadrados) en una variable x con coeficientes en el cuerpo finito $\mathbb{F}_q$ donde $q = p^n$, $(p \neq 2)$ queda recogida en el algoritmo **Factor** (Véase Anexo III). Para calcular la multiplicidad de cada factor irreducible se usan repetidas divisiones para eliminar la potencia correcta de cada uno de estos factores irreducibles de f.

4.2. Elevación de Hensel para factorización de polinomios

Sea $f \in \mathbb{Z}[x]$ y p es un número primo que no divide al coeficiente del término de mayor grado $l(f)$. Sea $\overline{f}$ el polinomio de $\mathbb{F}_p[x]$ obtenido reduciendo los coeficientes de f módulo p. Supongamos que tenemos $g_1, h_1 \in \mathbb{Z}[x]$ polinomios tales que $\overline{f} = \overline{g_1}\overline{h_1}$ en $\mathbb{F}_p[x]$ y esta factorización es propia en el sentido que ni $\overline{g_1}$ ni $\overline{h_1}$ son constantes y que $\overline{g_1}$ y $\overline{h_1}$ son primos entre si en $\mathbb{F}_p[x]$. Queremos elevar esta factorización de "módulo p" a "módulo p^2", esto es, encontrar $g_2, h_2 \in \mathbb{Z}[x]$ polinomios tales que $\deg(g_1) = \deg(g_2)$, $\deg(h_1) = \deg(h_2)$, y $f \equiv g_2 h_2 \pmod{p^2}$. En otras palabras, queremos elevar la factorización del anillo de polinomios $(\mathbb{Z}/p\mathbb{Z})[x]$ al anillo de polinomios $(\mathbb{Z}/p^2\mathbb{Z})[x]$.

Más en general, el algoritmo de elevación de Hensel permite pasar de una factorización de f módulo $m = p^n$ a una factorización de f módulo m^2. (Véase Anexo IV)

En principio la elevación de Hensel se aplica a la factorización de f como producto de dos factores g y h, pero puede extenderse a cualquier número de factores mediante sucesivas llamadas al algoritmo para dos factores como sigue:

Supongamos que $p \in \mathbb{Z}$ es un número primo, y que $f \in \mathbb{Z}[x]$ es un polinomio no constante con $l(f) \not\equiv 0 \pmod{p}$. Sea $\overline{f} \in \mathbb{F}_p[x]$ el polinomio obtenido al reducir los coeficientes de f módulo p. Supongamos que tenemos una factorización de $\overline{f}$ módulo p; esto es, tenemos polinomios mónicos no constantes $f_1, f_2, ..., f_r \in \mathbb{Z}[x]$ para los cuales

$$\overline{f} = \overline{l(f)}\overline{f_1}...\overline{f_r} \text{ en } \mathbb{F}_p[x]$$

Asumimos además que para todo i, j con $1 \leq i < j \leq r$ tenemos los polinomios $u_{ij}, v_{ij} \in \mathbb{Z}[x]$ para los cuales

$$\overline{u_{ij}}\overline{f_i} + \overline{v_{ij}}\overline{f_j} = 1 \text{ en } \mathbb{F}_p[x]$$

Dado $n \geq 2$, queremos reducir esta fatorización de f a módulo p^n.

Primero dividimos la factorización los más uniformemente posible definiendo

$$k = \left\lfloor \frac{r}{2} \right\rfloor, \quad g_0 = l(f)f_1...f_k, \quad h_0 = f_{k+1}...f_r$$

Usando el algoritmo extendido de Euclides calculamos $s_0, t_0 \in \mathbb{Z}[x]$ tales que

$$\overline{s_0 g_0} + \overline{t_0}\overline{h_0} = 1 \text{ en } \mathbb{F}_p[x]$$

Aplicamos la elevación de Hensel d veces para $d = \lceil \log n \rceil$, de modo que p^n es divisor de p^{2^d}. Obtenemos los polinomios $g_d, h_d, s_d, t_d \in \mathbb{Z}[x]$ tales que

$$f \equiv g_d h_d (\text{mod } p^{2^d}), \qquad s_d g_d + t_d h_d \equiv 1 (\text{mod } p^{2^d})$$

Hacemos ahora dos llamadas a este algoritmo: primero, para descomponer y elevar la factorización de g_d; y segundo, para descomponer y elevar la factorización de h_d. Estas llamadas recursivas terminan cuando ya no es posible dividir las factorizaciones; habremos obtenido entonces los polinomios $F_1, ..., F_r \in \mathbb{Z}[x]$ para los que

$$F_i \equiv f_i (\text{mod } p), \qquad f \equiv F_1...F_r (\text{mod } p^{2d}),$$

junto con los polinomios $U_{ij}, V_{ij} \in \mathbb{Z}[x]$ tales que

$$U_{ij} \equiv u_{ij} (\text{mod } p), \quad V_{ij} \equiv v_{ij} (\text{mod } p), \quad \overline{U_{ij}F_i} + \overline{V_{ij}F_i} \equiv 1 (\text{mod } p^{2^d})$$

Esto nos da una elevación de la factorización original en r factores módulo p a una factorización en los r factores correspondientes módulo p^{2^d} (y por consiguiente módulo p^n).

4.3. Factorización de polinomios con coeficientes enteros

Consideramos un polinomio arbitrario con coeficientes enteros,

$$f = a_n x^n + ... + a_1 x + a_0 \in \mathbb{Z}[x]$$

Definición 4.3.1. El *contenido* de f es el máximo común divisor de sus coeficientes, $cont(f) = mcd(a_n, ..., a_1, a_0)$. Decimos que f es *primitivo* si $cont(f) = 1$. Si dividimos f por su contenido, obtenemos un polinomio primitivo, llamado la *parte primitiva* de f, $prim(f) = \frac{1}{cont(f)} f$

Necesitaremos en lo que sigue algunas medidas, cotas y resultados relativos a los polinomios con coeficientes enteros que recogemos a continuación (Véase [8])

Definición 4.3.2. Para $s > 0$, la *norma* s de un polinomio $f \in \mathbb{Z}[x]$ es

$$|f|_s = \left(\sum_{i=0}^{n} |a_i|^s \right)^{1/s} \quad para\ f = \sum_{i=0}^{n} a_i x^i$$

En particular, la *norma 1* y *norma 2* son

$$|f|_1 = |a_0| + |a_1| + ... + |a_n|, \qquad |f|_2 = \sqrt{a_0^2 + a_1^2 + ... + a_n^2}$$

Lema 4.3.1 (Cota de Mignotte). *Si $f, g, h \in \mathbb{Z}[x]$ satisfacen $f = gh$ entonces*

$$|g|_1 |h|_1 \leq (n+1)^{1/2} 2^n |f|_\infty, \qquad n = \deg(f)$$

siendo $|f|_\infty = max\{|a_0|, |a_1|, .., |a_n|\}$

Definición 4.3.3. El *discriminante* de un polinomio $f \in \mathbb{Z}[x]$ es la resultante de f y su derivada formal f', es decir disc(f)=res(f, f'), siendo la resultante el determinante de la matriz de Silvester (Véase [8]).

Proposición 4.3.2. *Sea $f \in \mathbb{Z}[x]$ ($f \neq 0$) libre de cuadrados , y sea $p \in \mathbb{Z}$ un primo tal que $l(f) \not\equiv 0$ (mod p). Entonces el polinomio reducido $\bar{f} \in \mathbb{F}_p[x]$ es libre de cuadrados si y sólo si p no divide al discriminante de f.*

Dado un polinomio primitivo libre de cuadrados $f \in \mathbb{Z}[x]$, primero necesitamos encontrar un primo p que no divida al coeficiente del término de mayor grado de f, es decir que $l(f) \not\equiv 0 \pmod{p}$ y para el que el polinomio reducido $\bar{f} \in \mathbb{F}_p[x]$, también libre de cuadrados, es decir que $\text{disc}(f) \not\equiv 0 \pmod{p}$. Si $n = \deg(f)$ entonces fijamos

$$B = (n+1)^{1/2} 2^n |f|_\infty |l(f)|$$

Elegimos un entero positivo k para el que $p^k > 2B$ y supongamos que tenemos una factorización de f módulo p^k:

$$f \equiv l(f) g_1 g_2 ... g_r (\text{mod } p^k), \qquad g_1, g_2, ..., g_r \in \mathbb{Z}[x]$$

con $g_1, g_2, ..., g_r$ mónicos. Usando los representantes simétricos de las clases de congruencia módulo p^k, podemos suponer que $|g_i|_\infty \leq p^k/2$ para todo i. Sea S un subconjunto no vacío de $\{1, 2, ..., r\}$, y escribimos $\bar{S} = \{1, 2, ..., r\}/S$. Elegimos $G, H \in \mathbb{Z}[x]$ tales que

$$G \equiv l(f) \prod_{i \in S} g_i (\text{mod } p^k), \quad H \equiv l(f) \prod_{i \in \bar{S}} g_i (\text{mod } p^k), \quad |G|_\infty, |H|_\infty < \frac{p^k}{2}$$

G y H contienen al coeficiente principal de f. Supongamos que

$$|G|_1 |H|_1 \leq B$$

Se pretende que esta desigualdad se cumpla si y sólo si

$$l(f) f = GH$$

esto es, la factorización con representantes simétricos módulo p^k es en realidad una factorización sobre $\mathbb{Z}$. Si $l(f)f = GH$, entonces la cota de Mignotte, reemplazando f por $l(f)f$, implica que

$$|G|_1 |H|_1 \leq |l(f)| (n+1)^{1/2} 2^n |f|_\infty = B$$

Recíprocamente, asumiendo que $|G|_1 |H|_1 \leq B$. Tenemos

$$l(f) f \equiv GH (\text{mod } p^k) \tag{4.1}$$

y las desigualdades que relacionan las diferentes normas implican que

$$|GH|_\infty \leq |GH|_1 \leq |G|_1 |H|_1 \leq B < \frac{p^k}{2}$$

De este modo ambos lados de la congruencia (4.1) son polinomios con coeficientes $< p^k/2$ en valor absoluto, y entonces son iguales en $\mathbb{Z}[x]$.

Podemos entonces abordar la factorización de polinomios primitivos y libres de cuadrados en $\mathbb{Z}[x]$ dada por Zassenhaus.

4.3.1. Algoritmo de factorización de Zassenhaus

- **Entrada.** Un polinomio primitivo no constante $f \in \mathbb{Z}[x]$ libre de cuadrados.
- **Salida.** Los factores irreducibles $f_1, ..., f_r \in \mathbb{Z}[x]$ de f.

(1) Set $n \longleftarrow \deg(f)$

(2) If $n = 1$ then
Set `result` $\longleftarrow [f]$
else

(a) Set $C \longleftarrow (n+1)^{2n}|f|_\infty^{2n-1}$, set $r \longleftarrow \lceil 2\log C \rceil$.

(b) Set $B \longleftarrow (n+1)^{1/2}2^n|f|_\infty|l(f)|$.

(c) Encontrar un primo $p < 2r\ln r$ tal que $p \nmid l(f)$ y $p \nmid \text{disc}(f)$.

(d) Set $k \longleftarrow \lceil \log_p(2B+1) \rceil$.

(e) Call **Factor** para encontrar $h_1, h_2, ..., h_r \in \mathbb{Z}[x]$ polinomios mónicos irreducibles no constantes para los cuales $\overline{f} = \overline{l(f)}\overline{h_1}\overline{h_2}...\overline{h_r}$ en $\mathbb{F}_p[x]$. Usando representantes simétricos, $|h_i|_\infty < p/2$ para todo i.

(f) Call **Hensel** repetidamente para encontrar $g_1, g_2, ..., g_r \in \mathbb{Z}[x]$ polinomios mónicos irreducibles no constantes para los cuales $f \equiv l(f)g_1g_2...g_r \pmod{p^k}$ y $g_i \equiv h_i \pmod{p}$ para todo i. Usando representantes simétricos, $|g_i|_\infty < p^k/2$ para todo i.

(g) Set $T \longleftarrow \{1, 2, ...r\}$, set $F \longleftarrow f$.

(h) Set `result` $\longleftarrow$ [] (*empty list*), set $s \longleftarrow 1$.

(i) While $2s \leq |T|$ do

- Set `done` $\longleftarrow$ false, set `list` $\longleftarrow \{s\text{-element subsets of } T\}$.
- Set $j \longleftarrow 0$.
- While $j <$|`list`| and not `done` do
  - Set $j \longleftarrow j+1$, set $S \longleftarrow$ `list`$[j]$.
  - Encontrar $G, H \in \mathbb{Z}[x]$ tal que $|G|_\infty, |H|_\infty < p^k/2$ y

$$G \equiv l(F)\prod_{i\in S} g_i (\text{mod } p^k), H \equiv l(F)\prod_{i\in T\setminus S} g_i (\text{mod } p^k).$$

  - If $|G|_1|F|_1 \leq B$ then
    * Append prim (G) to `result`.
    * Set $T \longleftarrow T\setminus S$, set $F \longleftarrow \text{prim}(H)$, set `done` $\longleftarrow$ true.
- Set $s \longleftarrow s+1$

(h) Append F to `result`.

(3) Return `result`.

El paso (2)(g) inicializa el conjunto T de los índices de los factores que quedan por considerar, y el producto F de esos factores restantes. El bucle en los pasos (2)(h)-(i) es el corazón del algoritmo. Se trata de combinar los factores módulo p^k en un subconjunto que

provenga de un factor irreducible en $\mathbb{Z}$, ya que un factor irreducible de f sobre $\mathbb{Z}$ podría dividirse en el producto de más de un factor irreducible sobre $\mathbb{F}_p$. El algoritmo busca de forma exhaustiva sobre todos los subconjuntos $\{1, 2, ..., r\}$ para reconstruir los factores irreducibles sobre $\mathbb{Z}$, empezando con los conjuntos de tamaño $s = 1$ e incrementando s. Cada vez que se encuentra un nuevo factor que es irreducible sobre $\mathbb{Z}$, el algoritmo borra los factores correspondientes a considerar. Ya que hay $2^r - 1$ subconjuntos no vacíos de $\{1, 2, ..., r\}$, en el peor de los casos el número de iteraciones del bucle es una función exponencial de $n = \deg(f)$.

Ejemplo 4.3.1. Sea $f(x) = x^3 + 5x^2 + 2x + 6$ con $\deg(f) = 3$; $\text{cont}(f) = 1$, entonces f es primitivo. Además $\text{mcd}(f, f') = 1$ así que f es libre de cuadrados. Calculamos: $res(f, f') = 2824$, $C = (n+1)^{2n}|f|_\infty^{2n-1} = 4^6 \cdot 6^5 = 31850496$, $r = \lceil 2 \log C \rceil = 50$, $B = (n+1)^{1/2} 2^n |f|_\infty |l(f)| = \sqrt{4} \cdot 2^3 \cdot 6 = 96$.
Ahora buscamos un primo p tal que $p \not| \ l(f)$, $p \not| \ disc(f)$ y $p < 2r \ln r = 391'2$, luego $p = 3$. Y entonces $k = \lceil \log_p(2B+1) \rceil = 5$.
Tenemos ahora que buscar la factorización de f en $\mathbb{Z}_3[x]$ usando representantes simétricos y para ello hacemos uso del algoritmo **Factor** (ver Anexo III).

$$\overline{f} = x^3 - x^2 - x = x(x^2 - x - 1) \quad \text{en } \mathbb{Z}_3[x]$$

Llega ahora el momento de aplicar el algoritmo de elevación de Hensel para calcular la factorización de f módulo p^k, es decir, módulo 3^5.
Sabemos que $f \equiv x(x^2 - x - 1) \pmod 3$ y como $\text{mcd}(x, x^2 - x - 1) = 1$ entonces
$1 \equiv s_1 x + t_1(x^2 - x - 1) \pmod 3$, luego $1 \equiv (x-1)x - (x^2 - x - 1) \pmod 3$.
Calculamos ahora $e \equiv f - g_1 h_1 \pmod{3^2}$, esto es $e \equiv x^3 + 5x^2 + 2x + 6 - x(x^2 - x - 1) \pmod{3^2}$ entonces $e \equiv -3x^2 + 3x - 3 \pmod{3^2}$. Y buscamos q y r tales que $s_1 e \equiv q h_1 + r \pmod{3^2}$ tal que $\deg(r) < \deg(h_1)$, esto es $-3x^3 - 3x^2 + 3x + 3 \equiv q(x^2 - x - 1) + r \pmod{3^2}$, luego $q = -3x + 3$ y $r = 3x - 3$.
Tenemos así que $g_2 \equiv g_1 + t_1 e + q g_1 \equiv x + (-1) \cdot (-3x^2 + 3x - 3) + (-3x + 3)x \equiv x + 3 \pmod{3^2}$ y que $h_2 \equiv h_1 + r \equiv x^2 - x - 1 + 3x - 3 \equiv x^2 + 2x - 4 \pmod{3^2}$. Luego,

$$f \equiv (x+3)(x^2 + 2x - 4) \pmod{3^2}$$

Como queremos la factorización de f módulo 3^5 debemos aplicar de nuevo el algoritmo de Hensel. Ahora como $\text{mcd}(x+3, x^2 + 2x - 4) = 1$ entonces $1 \equiv s_2(x+3) + t_2(x^2 + 2x - 4) \pmod{3^2}$, luego $1 \equiv (x-1)(x+3) - (x^2 + 2x - 4) \pmod{3^2}$.
Calculamos $e \equiv f - g_2 h_2 \pmod{3^4}$, esto es $e \equiv x^3 + 5x^2 + 2x + 6 - (x+3)(x^2 + 2x - 4) \pmod{3^4}$ entonces $e \equiv 18 \pmod{3^4}$. Y buscamos q y r tales que $s_2 e \equiv q h_2 + r \pmod{3^4}$ tal que $\deg(r) < \deg(h_2)$, esto es $18x - 18 \equiv q(x^2 + 2x - 4) + r \pmod{3^4}$, luego $q = 0$ y $r = 18x - 18$.
Tenemos así que $g_3 \equiv g_2 + t_2 e + q g_2 \equiv (x+3) + (-1) \cdot (18) + 0(x+3) \equiv x - 15 \pmod{3^4}$ y que $h_3 \equiv h_2 + r \equiv (x^2 + 2x - 4) + (18x - 18) \equiv x^2 + 20x - 22 \pmod{3^4}$. Luego,

$$f \equiv (x - 15)(x^2 + 20x - 22) \pmod{3^4}$$

Aplicamos una vez más el algoritmo de Hensel. Ahora como $\text{mcd}(x-15, x^2+20x-22) = 1$ entonces $1 \equiv s_3(x - 15) + t_3(x^2 + 20x - 22)$

$(\text{mod } 3^4)$, luego $1 \equiv (19x+17)(x-15) - 19(x^2+20x-22) \ (\text{mod } 3^4)$.
Calculamos $e \equiv f - g_3h_3 \ (\text{mod } 3^8)$, esto es $e \equiv x^3+5x^2+2x+6-(x-15)(x^2+20x-22)$ $(\text{mod } 3^8)$ entonces $e \equiv 324x - 324 \ (\text{mod } 3^8)$. Y buscamos q y r tales que $s_3e \equiv qh_3 + r$ $(\text{mod } 3^8)$ tal que $\deg(r) < \deg(h_3)$, esto es $-405x^2 - 648x + 1053 \equiv q(x^2+20x-22) + r$ $(\text{mod } 3^8)$, luego $q = -405$ y $r = 891x - 1296$.
Tenemos así que $g_4 \equiv g_3 + t_3e + qg_3 \equiv (x-15) - 19 \cdot (324x - 324) - 405(x-15) \equiv x - 906$ $(\text{mod } 3^8)$ y que $h_4 \equiv h_3 + r \equiv (x^2+20x-22) + 891x - 1296 \equiv x^2 + 911x - 1318$ $(\text{mod } 3^8)$. Luego,

$$f \equiv (x-906)(x^2+911x-1318) \quad (\text{mod } 3^8)$$

Y por consiguiente,

$$f \equiv (x+66)(x^2-61x-103) \quad (\text{mod } 3^5)$$

Debemos ahora comprobar todas las combinaciones posibles de estos últimos factores irreducibles de f en $\mathbb{Z}_{3^5}[x]$ y ver si alguna de ellas divide a f en $\mathbb{Z}[x]$. En este caso, $(x+66) \nmid f$, $(x^2-61x-103) \nmid f$ y $(x+66)(x^2-61x-103) \nmid f$, por lo tanto f es irreducible.

4.4. Factorización de polinomios usando LLL

Es esta sección estudiaremos como el algoritmo LLL se puede usar para proporcionar un algoritmo de tiempo polinomial para factorizar polinomios primitivos libres de cuadrados con coeficientes enteros.
Primero haremos unos pequeños cambios es los pasos (2)(b) y (2)(d) del algoritmo de Zassenhaus; redefinimos B y k como sigue:

$$B \leftarrow (n+1)^{1/2}2^n|f|_\infty, \qquad k \leftarrow \lceil \log_p(2^{n^2/2}B^{2n}) \rceil$$

Para este primer valor de k, tenemos

$$p^k \geq 2^{n^2/2}B^n = 2^{n^2/2}((n+1)^{1/2}2^n|f|_\infty)^{2n} = 2^{n^2/2}(n+1)^n 2^{2n^2}|f|_\infty^{2n}$$

El mayor cambio del algoritmo está en los pasos (2)(h)-(i): reemplazamos la búsqueda sobre todos los subconjuntos del conjunto de factores por un cálculo que usa el algoritmo de reducción de base. Necesitamos los siguientes lemas (véase [8]).

Lema 4.4.1. *Sean $f, g \in \mathbb{Z}[x]$ no nulos con $\deg(f) + \deg(g) \geq 1$ (al menos uno de los dos, f o g, no constante). Entonces existen $s, t \in \mathbb{Z}[x]$ tales que $sf + tg = res(f,g)$ con $\deg(s) < \deg(g)$ y $\deg(t) < \deg(f)$.*

Lema 4.4.2. *Sean $f, g \in \mathbb{Z}[x]$ con $n = \deg(f)$ y $m = \deg(g)$. Entonces*

$$|res(f,g)| \leq |f|_2^m |g|_2^n \leq (n+1)^{m/2}(m+1)^{n/2}|f|_\infty^m |g|_\infty^n$$

donde $|f|_\infty$ es el máximo de los valores absolutos de los coeficientes de f.

Usaremos estos lemas para probar el siguiente resultado.

Lema 4.4.3. *Sean $f, g \in \mathbb{Z}[x]$ con $n = \deg(f) > 0$ y $t = \deg(g) > 0$. Suponiendo que $u \in \mathbb{Z}[x]$ es mónico y no constante, y que $f \equiv uv_1 \pmod m$ y $g \equiv uv_2 \pmod m$ para algún $v_1, v_2 \in \mathbb{Z}[x]$ y algún $m > |f|_2^t |g|_2^n$. Entonces mcd$(f, g) \in \mathbb{Z}[x]$ es no constante.*

Demostración. Veremos por contradicción que $\mathrm{mcd}(f, g) \in \mathbb{Q}[x]$, el MCD con coeficientes racionales, es no constante. Supongamos que $\mathrm{mcd}(f, g) = 1$ en $\mathbb{Z}[x]$. Por el Lema 4.4.1 (mod m). Como $f \equiv uv_1 \pmod m$ y $g \equiv uv_2 \pmod m$ tenemos

$$res(f, g) = u(sv_1 + tv_2) \ (mod \ \ m)$$

así que u es divisor de $\mathrm{res}(f, g)$ módulo m. Pero u es mónico y no constante, y $\mathrm{res}(f, g) \in \mathbb{Z}$, así que $\mathrm{res}(f, g) \equiv 0 \pmod m$. El Lema 4.4.2 implica

$$m > |f|_2^t |g|_2^n \geq |res(f, g)|$$

y por tanto $\mathrm{res}(f, g) \equiv 0$. Pero esto implica que $\mathrm{mcd}(f, g) \neq 1$, una contradicción. Luego $\mathrm{mcd}(f, g) \in \mathbb{Q}[x]$ es no constante, y por consiguiente también lo es $\mathrm{mcd}(f, g) \in \mathbb{Z}[x]$. □

Supongamos que $f \in \mathbb{Z}[x]$ es libre de cuadrados, primitivo y sea $n = \deg(f)$. Supongamos que $u \in \mathbb{Z}[x]$ es mónico y no constante con $d = \deg(u) < n$, $u = u_0 + u_1 x + ... + u_d x^d$, y que u es divisor de f módulo m; esto es, $f \equiv uv \pmod m$ para algún $v \in \mathbb{Z}[x]$ (donde $m = p^k$ siendo k la fijada al principio de esta sección).

Fijado un $j \in \{d + 1, ..., n\}$, definimos $\mathcal{L} \subset \mathbb{Z}^j$ la red cuya base viene dada por los vectores formados por los coeficientes de los polinomios

$$\{u, xu, ..., x^{j-1-d}u\} \cup \{m, mx, ..., mx^{d-1}\}$$

es decir, por los $(j - d) + d = j$ vectores:

$$\{(u_0, ..., u_d, 0, .., 0), (0, u_0, ..., u_d, 0, .., 0), .., (0, ..., 0, u_0, ..., u_d),$$
$$(m, 0, ..., 0), (0, m, 0, ..., 0), ..., (0, ..., 0, m, 0, ..., 0)\}$$

A cada elemento de la red $(a_0, ..., a_{j-1}) \in \mathcal{L}$, le asociamos un polinomio

$$g = a_0 + a_1 x + ... + a_{j-1} x^{j-1}$$

y por abuso de lenguaje diremos que $g \in \mathcal{L}$.

Proposición 4.4.4. *En las condiciones anteriores*

$$g \in \mathcal{L} \Longleftrightarrow \deg(g) < j \quad y \quad u | g \pmod m$$

Demostración. Si $g \in \mathcal{L}$ se tiene que

$$g = \sum_{i=0}^{j-d-1} q_i x^i u + \sum_{i=0}^{d-1} r_i m x^i = \left(\sum_{i=0}^{j-d-1} q_i x^i \right) u + m \left(\sum_{i=0}^{d-1} r_i x^i \right) = qu + mr$$

donde $q = \sum_{i=0}^{j-d-1} q_i x^i$, $r = \sum_{i=0}^{d-1} r_i x^i$. Luego $g \equiv qu \pmod m$ y se tiene $\deg(g) < j$ y $u|g \pmod m$.

Recíprocamente, supongamos que $g \in \mathbb{Z}[x]$ con $\deg(g) < j$ y $u|g \pmod m$. Como u es divisor de g módulo m, tenemos que $g = q_1 u + m r_1$ para algún $q_1, r_1 \in \mathbb{Z}[x]$. Recordando que u es mónico, debemos dividir r_1 entre u en $\mathbb{Z}[x]$, obteniendo $r_1 = q_2 u + r_2$ con $\deg(r_2) < \deg(u)$. Tenemos ahora

$$(q_1 + m q_2)u + m r_2 = q_1 u + m(q_2 u + r_2) = q_1 u + r_1 m = g$$

Luego $g = qu + mr$ para $q = q_1 + m q_2$ y $r = r_2$. Es claro que $\deg(r) < \deg(u)$; y además $\deg(q_1) \leq \deg(g) - \deg(u) < j - d$ lo que implica $\deg(q_2) \leq \deg(r_1) - \deg(u) < j - d$. Por lo tanto $g \in \mathcal{L}$ □

Usando el algoritmo LLL con $\alpha = \frac{3}{4}$ podemos encontrar una base reducida de $\mathcal{L}$. Si g_1 es el primer vector de la base reducida

$$|g_1|_2 \leq 2^{(j-1)/2} |g|_2, \quad \forall g \in \mathcal{L} - \{0\}$$

Dado que $\dim(\mathcal{L}) = j \leq n$, se tiene $|g_1|_2 \leq 2^{n/2}|g|_2$.

Si $g \in \mathcal{L}$ un factor irreducible de f, por la cota de Mignotte del Lema 4.3.1, se tiene

$$|g|_\infty \leq |g|_2 \leq (n+1)^{1/2} 2^n A, \qquad A = \max(|f|_\infty, |g|_\infty)$$

Por lo tanto

$$|g_1|_2 \leq 2^{n/2} B, \qquad B = (n+1)^{1/2} 2^n A$$

Ahora se obtiene

$$|g_1|_2^{j-1} |g|_2^{\deg(g_1)} < (2^{n/2} B)^n B^n = 2^{n^2/2} (n+1)^{1/2} 2^{2n^2} A^{2n} \leq p^k = m$$

con la elección original de k.

Por el Lema 4.4.3, se sigue que el $\mathrm{mcd}(g, g_1)$ es no constante en $\mathbb{Z}[x]$ y por tanto también $\mathrm{mcd}(f, g_1)$ es no constante y un factor de f.

De este modo, podemos sustituir el tiempo exponencial de los pasos (2)(h)-(i) en el algoritmo de Zassenhaus por tiempo polinomial con el algoritmo LLL.

Ejemplo 4.4.1. Tomamos el mismo polinomio que en el Ejemplo 4.3.1 para comparar el algoritmo de Zassenhaus y con el algoritmo LLL. Sea $f(x) = x^3 + 5x^2 + 2x + 6$ tenemos los mismos valores para C, r, B y p, y por tanto la factorización de f en $\mathbb{Z}_3[x]$ usando representantes simétricos es

$$\overline{f} = x^3 - x^2 - x = x(x^2 - x - 1) \quad \text{en } \mathbb{Z}_3[x]$$

Y aplicamos, igual que antes, la elevación de Hensel para calcular la factorización de f módulo p^k, es decir, módulo 3^5. Ya tenemos que

$$f \equiv (x + 66)(x^2 - 61x - 103) \pmod{3^5}$$

Aplicamos el algoritmo LLL. Tomamos $u = (x+66) \in \mathbb{Z}[x]$ que es mónico, no constante con d =deg$(u) < n$ y que es divisor de f módulo $m = 3^5$. Tenemos que tomar $j \in \{d+1, ..., n\}$, en este caso $j = \{2, 3\}$, y definir $\mathcal{L} \subset \mathbb{Z}^2$ la red cuya base es de la forma $\{(66, 1), (243, 0)\}$, y la base reducida queda $\{(-21, -4), (66, 1)\}$. Luego $g = -21x^2 - 4x + 66$ y como mcd(f, g_1)=1 que es constante entonces f es irreducible.

Conclusiones

En este trabajo hemos estudiado esencialmente el algoritmo LLL que permite obtener una base reducida de una red $\mathcal{L}$ dada. Después hemos utilizado dicho algoritmo para factorizar polinomios con coeficientes enteros.

Podemos considerar este contenido como la base para poder abordar muchos y muy diversos tipos de cuestiones tanto dentro de las propias Matemáticas, en particular en la teoría de números, como en Ciencias de la Computación.

Sin ánimo de ser exhaustivos, destaquemos, por ejemplo, que con el algoritmo LLL se obtiene una buena base para la red, pero dicha base no tiene porqué contener el vector más corto de $\mathcal{L}$. El algoritmo dado por Kannan [9] en 1987 permite obtener una base que contiene el vector más corto de L. Por otra parte, como ya dijimos en la introducción, el algoritmo LLL es la base para el estudio de muchos problemas en criptoanálisis y teoría de códigos, y ha permitido romper varias variantes del sistema de cifrado RSA y el algoritmo de firma digital DSA encontrando pequeñas soluciones de ecuaciones modulares.[6] [10]

Por tanto consideramos que el campo de estudio que nos abre este trabajo va mucho más allá de la mera factorización de polinomios.

APÉNDICE A

Anexos

A.1. Anexo I

A.1.1. Algoritmo de descomposición de distinto grado DDD

- **Entrada.** Un polinomio mónico no constante y libre de cuadrados $f \in \mathbb{F}_q[x]$.
- **Salida.** La descomposición de distinto grado $\mathrm{ddd}(f)$.

(1) Set $f_0 \longleftarrow f$, set $g_0 \longleftarrow x$, set $d \longleftarrow 0$

(2) While $f_d \neq 1$ do:

(a) Set $d \longleftarrow d + 1$,

(b) Set $g_d \longleftarrow g_{d-1}^q$

(c) Set $h_d \longleftarrow \mathbf{Euclid}(f_{d-1}, g_d - x)$

(d) Set $f_d \longleftarrow f_{d-1}/h_d$

(3) Return $h_1, h_2, ..., h_d$.

A.2. Anexo II

A.2.1. Algoritmo TrialSplit

- **Entrada.** Un polinomio mónico no constante y libre de cuadrados $h \in \mathbb{F}_q[x]$ de grado dl que es el producto de l factores irreducibles mónicos de grado d.
- **Salida.** Si $l \geq 2$ entonces, con probabilidad $\geq \frac{1}{2}$ el algoritmo devuelve un factor propio g de f (en caso de fallar, el algoritmo devuelve $g = 0$).

(1) If $\deg(h) = 1$ then

(a) Set $g \longleftarrow 0$

else

(b) Set $g_1 \longleftarrow 0$

(c) While $g_1 \in \mathbb{F}_q$ do:
Generar pseudorandom $g_1 \in \mathbb{F}_q[x]$ con $\deg(g_1) < \deg(h)$

(d) Set $g_2 \longleftarrow \text{mcd}(g_1, h)$

(e) If $g_2 \neq 1$ then

Set $g \longleftarrow g_2$

else

Set $e \longleftarrow (q^d - 1)/2$

Set $g_3 \longleftarrow \overline{g_1^e}$ donde la barra denota al resto módulo h

Set $g_4 \longleftarrow \text{mcd}(g_3 - 1, h)$

If $0 < \deg(g_4) < \deg(h)$ then set $g \longleftarrow g_4$ else set $g \longleftarrow 0$

(2) Return g

A.2.2. Algoritmo Split

- **Entrada.** Un polinomio mónico no constante y libre de cuadrados $h \in \mathbb{F}_q[x]$ de grado dl que es el producto de l factores irreducibles mónicos de grado d, y un parámetro entero de terminación $s \geq 1$.
- **Salida.** Con probabilidad $\geq 1 - 2^{-s}$, un factor propio g de h

(1) Set $g \longleftarrow 0$, set $k \longleftarrow 0$.

(2) While $g = 0$ y $k < s$ do

(a) Set $g \longleftarrow$ **TrialSplit**(h)

(b) Set $k \longleftarrow k + 1$

(3) Return g

A.2.3. Algortimo EDD

- **Entrada.** Un polinomio mónico no constante y libre de cuadrados $h \in \mathbb{F}_q[x]$ de grado dl que es el producto de $l \geq 2$ factores irreducibles mónicos de grado d, y un parámetro entero de terminación $s \geq 1$.
- **Salida.** La descomposición de igual grado de h
- **Nota.** Antes de llamar a este algoritmo, la variable global `factorlist` se inicializa como la lista vacía.

(1) Set $g_1 \longleftarrow$ **Split**(h, s)

(2) If $g_1 = 0$ then

(a) Append h to `factorlist`

else

(b) Recursively call **EDD**(g_1, s)

(c) Recursively call **EDD**$(h/g_1, s)$

La probabilidad de encontrar un factor propio de h con el algoritmo **TrialSplit** es $\geq 1-2^{-s}$, que converge a 1 tan rápido como s aumenta. Para aumentar la probabilidad de encontrar un factor propio de h, usamos el algoritmo **Split**(h, s). La entrada de **Split**(h, s) incluye un parámetro de terminación $s \geq 1$; esta es la cota superior del número de procesos que **Split** realizará antes de concluir que h es irreducible. El algoritmo **Split** no proporciona la descomposición de igual grado completa de h, simplemente divide h en el producto de dos factores propios. Para encontrar la descomposición de igual grado completa, usamos el algoritmo **EDD**(h) que llama a **Split** sucesivamente hasta que no se pueda continuar dividiendo.

A.3. Anexo III

A.3.1. Algoritmo Factor

- **Entrada.** Un polinomio mónico no constante $f \in \mathbb{F}_q[x]$, y parámetro entero de terminación s para controlar las llamadas a **Split**(h, s).
- **Salida.** La factorización completa de f en factores mónicos irreducibles de $\mathbb{F}_q[x]$.

(1) Set `complete` $\longleftarrow$ [ ]
(2) Set $f_0 \longleftarrow f$, set $g_0 \longleftarrow x$, set $d \longleftarrow 0$
(3) While $f_d \neq 1$ do:
  (a) Set $d \longleftarrow d+1$
  (b) Set $g_d \longleftarrow g_{d-1}^q$
  (c) Set $h_d \longleftarrow$ **Euclid**$(f_{d-1}, g_d - x)$
  (d) If $h_d \neq 1$ then
    - Set `factorlist` $\longleftarrow$ [ ] (*empty list*)
    - Call **EDD**(h_d, s)
    - For k in `factorlist` do
      Set $m \longleftarrow 0$
      While $k|f$ do: Set $m \longleftarrow m+1$, set $f \longleftarrow f/k$
      Append $[k, m]$ to `complete`
(4) Return `complete`

A.4. Anexo IV

A.4.1. Algoritmo de elevación de Hensel

- **Entrada.**
  - El módulo $m \in \mathbb{Z}(m \geq 2)$

- Polinomios $f, g_1, h_1 \in \mathbb{Z}[x]$ tales que h_1 es mónico y

$$f \equiv g_1 h_1 (\text{mod } m), \quad \deg(f) = \deg(g_1) + \deg(h_1)$$

- Polinomios $s_1, t_1 \in \mathbb{Z}[x]$ tales que

$$s_1 g_1 + t_1 h_1 \equiv 1 (\text{mod } m), \quad \deg(s_1) < \deg(h_1) \quad \deg(t_1) < \deg(g_1)$$

- **Salida.**

- Polinomios $g_2, h_2 \in \mathbb{Z}[x]$ tales que h_2 es mónico y

$$\begin{array}{lll} f \equiv g_2 h_2 (\text{mod } m^2) & g_2 \equiv g_1 (\text{mod } m) & \deg(g_2) = \deg(g_1) \\ & h_2 \equiv h_1 (\text{mod } m) & \deg(h_2) = \deg(h_1) \end{array}$$

- Polinomios $s_2, t_2 \in \mathbb{Z}[x]$ tales que

$$\begin{array}{lll} s_2 g_2 + t_2 h_2 \equiv 1 (\text{mod } m^2) & s_2 \equiv s_1 (\text{mod } m) & \deg(s_2) < \deg(h_2) \\ & t_2 \equiv t_1 (\text{mod } m) & \deg(t_2) < \deg(g_2) \end{array}$$

- **Algoritmo.**

(1) Set $e \longleftarrow f - g_1 h_1 \ (\text{mod } m^2)$.

(2) Compute $q, r \in \mathbb{Z}[x]$ such that

$$s_1 e \equiv q h_1 + r (\text{mod } m^2), \quad \deg(r) < \deg(h_1)$$

(3) Set $g_2 \longleftarrow g_1 + t_1 e + q g_1 \ (\text{mod } m^2)$

(4) Set $h_2 \longleftarrow h_1 + r \ (\text{mod } m^2)$

(5) Set $e^* \longleftarrow s_1 g_2 + t_1 h_2 - 1 \ (\text{mod } m^2)$

(6) Compute $q^*, r^* \in \mathbb{Z}[x]$ such that

$$s_1 e^* \equiv q^* h_2 + r^* (\text{mod } m^2), \quad \deg(r^*) < \deg(h_2)$$

(7) Set $s_2 \longleftarrow s_1 + r^* \ (\text{mod } m^2)$

(8) Set $t_2 \longleftarrow t_1 - t_1 e^* - q^* g_2 \ (\text{mod } m^2)$

(9) Return g_2, h_2, s_2, t_2

Bibliografía

[1] D. G. Cantor and H. Zassenhaus: A new algorithm for factoring polynomialss over finite fields, *Mathematics of Computation* (1981), 587–592.

[2] J.W.S. Cassels: *An introduction to the Geometry of numbers.* Springer-Verlag Berlin Heidelberg (1997).

[3] E. Boas: *Another NP-complete Partition Problem and the Complexity of Computing Short Vectors in a Lattice.* Technical report. Universiteit van Amsterdam. Mathematisch Instituut (1981).

[4] A.K. Lenstra, H.W. Lenstra, and L. Lovász: Factoring polinomials with rational coefficients, *Mathematische Annalen* **261(4)** (1982), 515–534.

[5] M. Bremner: *Lattice Basis Reduction. An Introduction to the LLL Algorithm and Its Applications.* Boca Raton, FL : CRC Press,(2012).

[6] A. Joux and J. Stern: Lattice reduction: A toolbox for the cryptanalyst, *Journal of Cryptology* **11** (1998), 161—185.

[7] M. Atiyah: Mathematics in the 20th century. *Bull. London Math. Soc. 34* (2002).

[8] M. Mignotte and Stefanescu: *Polynomials:An algorithmic approach.* Springer-Verlag, Singapore (1999).

[9] R. Kannan: Reducibility among combinatorial problems. Complexity of Computer Computations, *Mathematics of Operations Research* **12** (1987), 415–440.

[10] P. Nguyen and J. Stern: The two faces of lattices in cryptology, in *Cryptography and lattices conference – CaLC 2001, J. Silverman, ed., Lecture Notes in Computer Science* **2146** (2001), Springer-Verlag, 146—180.

Póster

The LLL algorithm

Patricia de Armas González
Universidad de La Laguna

Objectives

1. To study the case of two-dimensional lattices, analyzing the algorithm given by Gauss to obtain a minimal basis.
2. To analyze the Gram-Schmidt orthogonalization process since we are interested in vectors with integer components.
3. To study the algorithm to the factorization of polynomials with integer coefficients, which has an exponential time in the polynomial degree, given by Cantor and Zassenhaus.
4. To analyze the algorithm LLL, which is polynomial at the size of the lattice, to calculate a reduced basis in a lattice and its aplication to the integer coefficients polynomial fatorization.

Introduction

In words of Abel Prize and Fields Medal, Michael Francis Atiyah, many problems that have nothing to do with geometry are solved when one is able to transform them into geometric problems. Moreover, he says that in mathematics, when you stop thinking geometrically, you stop understanding what you are doing and you only make calculations.
Minkowski introduced the Geometry of Numbers in the XIX century, and it makes use of Geometry to solve problems of Theory of Numbers in a very simple way.
The subject of study of that field are the lattices. A lattice can be spanned by an infinite number of basis, and a key problem is to obtain basis with short vectors, that is, with a small Euclidean norm. This problem is known as Basis Reduction problem.
Emde Boas proved in 1981 that the problem of finding the shortest vector of a lattice (SVP, Shortest Vector Problem) to the maximum norm, is NP-hard. Minkowski showed that the shortest Euclidean norm of a vector is less than or equal

$$\sqrt{n}|B|^{1/n}$$

where B is the matrix of the basis vectors.
In 1982 , A.Lenstra , H.Lenstra and L.Lovasz, published an algorithm called LLL , which is able to find in polynomial time a lattice vector with Euclidean norm less than

$$2^{(n-1)/4}|B|^{1/n}$$

This article is titled "Factoring polynomials with rational coefficients" because its first application was the polynomial with rational coefficients factorizaction in polynomial time.

Lattices. Two-dimensional lattices

Let $n \geq 1$ and let $x_1, x_2, ..., x_n$ be a basis of $\mathbb{R}^n$. The *lattice* with dimension n and basis $x_1, x_2, ..., x_n$ is the set $\mathcal{L}$ of all linear combinations of the basis vectors with integral coefficients:

$$\mathcal{L} = \mathbb{Z}x_1 + \mathbb{Z}x_2 + ... + \mathbb{Z}x_n = \left\{ \sum_{i=1}^{n} a_i x_i / a_1, a_2, ..., a_n \in \mathbb{Z} \right\}$$

Let x and y form a basis of $\mathbb{R}^2$. The lattice $\mathcal{L} \subset \mathbb{R}^2$ generated by x and y is the set of all integral linear combinations of x and y:

$$\mathcal{L} = \{ax + by / a, b \in \mathbb{Z}\}$$

We say that a basis x, y of a lattice $\mathcal{L}$ in $\mathbb{R}^2$ is *minimal* if x is a shortest nonzero vector in $\mathcal{L}$ and y is a shortest vector in $\mathcal{L}$ which is not a multiple of x.
A basic problem is to find a shortest (nonzero) vector in this lattice; that is, a vector v for which $|v| \geq |w|$ for all $w \in \mathcal{L}, w \neq 0$. This is achieved by the "Gaussian algorithm".

References

[1] E. Boas: Another NP-complete Partition Problem and the Complexity of Computing Short Vectors in a Lattice. *Universiteit van Amsterdam. Mathematisch Instituut* (1981).
[2] M. Bremner: Lattice Basis Reduction An Introduction to the LLL Algorithm and Its Applications. *Boca Raton, FL : CRC Press, c2012* (2012).
[3] J.W.S. Cassels: An introduction to the Geometry of numbers. *Springer-Verlag Berlin Heidelberg* (1997).

Gram-Schmidt orthogonalization process

We review the classical Gram-Schmidt algorithm for converting an arbitrary basis of $\mathbb{R}^n$ into an orthogonal basis. This is a standard topic in elementary linear algebra, but we develop this material with a view to its application to the LLL algorithm.
Theorem Let $x_1, x_2, ..., x_n$ be a basis of $\mathbb{R}^n$, and let $x_1^*, x_2^*, ..., x_n^*$ be its Gram-Schmidt orthogonalization. Let $\mathcal{L}$ be the lattice generated by $x_1, x_2, ..., x_n$. For any nonzero $y \in \mathbb{L}$ we have

$$|y| \geq \min\{|x_1^*|, |x_2^*|, ..., |x_n^*|\}$$

Through the **Cramer's Rule** we will be able to give bounds on the denominators of the rational numbers that appear in the Gram-Schmidt orthogonalization of vectors with integral components. This will be important in our analysis of the LLL algorithm.

The LLL algorithm

Theorem. If $x_1, x_2, ..., x_n$ is an α-reduced basis of the lattice $\mathcal{L}$ in $\mathbb{R}^n$, and $y_1, y_2, ..., y_m \in \mathcal{L}$ are any m linearly independent lattice vectors, then for $1 \leq j \leq m$ we have

$$|x_j| \leq \beta^{(n-1)/2} \max\{|y_1|, ..., |y_m|\}$$

The original LLL algorithm
The input consists of a basis $x_1, x_2, ..., x_n$ of the lattice $\mathcal{L} \subset \mathbb{R}^n$, and a reduction parameter $\alpha \in \mathbb{R}$ in the range $\frac{1}{4} < \alpha < 1$. The output consists of an α-reduced basis $y_1, y_2, ..., y_n$ of the lattice $\mathbb{L}$.

- Computes the Gram-Schmidt orthogonalization of the vectors $y_1, y_2, ..., y_n$
- Performs the basis reduction; it repeatedly calls two procedures which reduce and exchange the vectors $y_1, y_2, ..., y_n$
- Procedure `reduce`(k, l) makes y_k almost orthogonal to y_l. If $|\nu_{kl}| \leq \frac{1}{2}$ then the procedure does nothing; otherwise, it reduces y_k by subtracting the integral multiple $\lceil \nu_{kl} \rfloor$ of y_l
- Procedure `exchange`(k) interchanges the vectors y_{k-1} and y_k, and then updates the GSO basis and coefficients

The vectors $y_1, y_2, ..., y_n$ are modified continually throughout the algorithm, but in such a way that they always form a basis for the lattice $\mathcal{L}$.

Polynomial factorization

The Zassenhaus algorithm and the LLL algorithm both use the same strategy. A polynomial $f \in \mathbb{Z}[x]$ is factored modulo a suitable prime p and then use the Hensel lifting to obtain the factorization modulo $p^2, p^4, p^8, ..., p^n$.
The major changes to the algorithm occur when we replace the search over all subsets of the set of modular factors by a computation which uses lattice basis reduction.
In this way, we can replace the exponential-time in Zassenhaus algorithm by polynomial-time calls to the LLL algorithm.

Conclusion

- This content can be the basis to attack many and different types of problems within Mathematics, and particularly in the Theory of Numbers, and Computer Science.
- With the LLL algorithm a good basis is obtained, but it does not have to contain the shortest vector of $\mathcal{L}$.
- The LLL algorithm has allowed to break several variants of RSA cryptosystem and the digital signature algorithm DSA finfing small modular equation solutions.

About

Departament: Mathematics, Statistics and Operative Investigation
Faculty: Science, Mathematics Section **University:** La Laguna
Email: alu0100602210@ull.edu.es
Final Year Project in Mathematics done under the supervision of Dr. Margarita Rivero Álvarez during the academic year 2015/2016