Seminario de Tesis II: Análisis de distribución de los ceros de las sumas parciales de la función zeta de Riemann y funciones asociadas

Facultad de Ciencias - Escuela Profesional de Ciencia de la Computación

Tesista: Víctor Racsó Galván Oyola

Asesor: Dr. Oswaldo José Velásquez Castañón

Código de estudiante: 20140097A

27 de agosto de 2021

Índice general

1. Introducción **3**

1.1. Objeto de investigación 3

1.2. Objetivos 4

1.2.1. Objetivo general 4

1.2.2. Objetivo inmediato 4

1.3. Antecedentes de la investigación 4

2. Análisis teórico **7**

2.1. Reducción de bases de retículos 7

2.1.1. Retículos 7

2.1.2. Bases de retículos LLL-reducidas 14

2.2. Algoritmo Lenstra-Lenstra-Lovász (LLL) 20

2.3. Aplicación del algoritmo LLL 24

3. Análisis implementativo **28**

3.1. Lenguaje y librerías 28

3.2. Implementación del algoritmo LLL 29

3.3. Obtención de T 31

3.4. Cálculo de ceros especiales 32

4. Resultados **35**

4.1. Tabla resumen de los resultados 35

5. Conclusiones **37**

5.1. Conclusión inmediata 37

5.2. Discusión 37

Bibliografía **40**

Capítulo 1

Introducción

1.1. Objeto de investigación

El objeto de estudio de esta investigación es la función zeta de Riemann, función de variable compleja definida por la serie

$$\zeta(s) = \sum_{i=1}^{\infty} i^{-s}, s \in \mathbb{C} \text{ y } \operatorname{Re}(s) > 1$$

que tiene una prolongación analítica a todo el plano complejo $\mathbb{C}$, como una función meromorfa con un único polo simple en $s = 1$, cuyo residuo es 1.
Se realizará el estudio de esta función, basándonos en la información que nos brindan las llamadas sumas parciales de la función zeta de Riemann, definidas de la siguiente forma:

$$\zeta_n(s) = \sum_{i=1}^{n} i^{-s}$$

Los llamados ceros especiales de la función zeta de Riemann, los cuales pertenecen a la franja:

$$\{s \in \mathbb{C} : 1 < \operatorname{Re}(s)\}$$

Comenzaron a tomar importancia luego del artículo de Turán [16], pues demostró que:
Si existen n_0, $K > 0$ y $0 \leq \varepsilon < \frac{1}{2}$ tales que:

$$\forall n > n_0, \zeta_n(s) \text{ no tiene ceros en } \operatorname{Re}(s) \geq 1 + Kn^{-\frac{1}{2}+\varepsilon}$$

entonces la hipótesis de Riemann es cierta.
En este segundo seminario, estableceremos un método general para el cálculo de ceros especiales de las sumas parciales de la función zeta de Riemann aprovechando la naturaleza de la función para poder aplicar una aproximación diofántica usando el algoritmo LLL.

1.2. Objetivos

1.2.1. Objetivo general

Estudiar la distribución de los ceros especiales de las sumas parciales de la función zeta de Riemann, considerando las primeras instancias $n = 23, 24, 25, 26, 31$.

1.2.2. Objetivo inmediato

Implementar un algoritmo que pueda calcular ceros especiales de las sumas parciales de la función zeta de Riemann partiendo de un cero especial cualquiera.

1.3. Antecedentes de la investigación

La conexión entre la función zeta de Riemann y los números primos, se debe al producto que Euler demostró para s real mayor que 1:

$$\sum_{i=1}^{\infty} i^{-s} = \prod_{p} \left(1 - \frac{1}{p^s}\right)^{-1}$$

el mismo Euler usó esta identidad para dar una prueba analítica de la infinitud de los números primos. Riemann fue el primero en considerar esta identidad en términos de variable compleja, se demostró la ecuación funcional

$$\zeta(s) = 2^s \pi^{s-1} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s)$$

donde Γ es la función gamma de Euler.

El problema de corroborar o descartar la hipótesis de Riemann, planteado por él en 1859 y propuesto formalmente por David Hilbert el año 1900, como el octavo problema de su lista de 23 problemas abiertos hasta entonces, indican la dificultad y trascendencia de resolver esta conjetura. Transcurridos ya más de cien años, con muchos intentos infructuosos llegamos a mayo del 2000, fecha en la que el instituto norteamericano Clay relanza el reto, dentro de su lista de los siete problemas del milenio, ofreciendo un millón de dólares al que resuelve. La sucesión de funciones que se analizará en este trabajo son las sumas parciales de la funcion zeta de Riemann, esto es

$$\zeta_n(s) = \sum_{i=1}^{n} i^{-s}, \text{ con } \operatorname{Re}(s) > 1$$

estas son funciones casi-periódicas (Hernández, 2009).

Paul Turán (Turán, 1948) demostró que la hipótesis de Riemann será cierta, si para todo n suficientemente grande, $\zeta_n(s)$ no tendría ceros en $\operatorname{Re}(s) > 1$, Turán también demostró que $\zeta_n(s)$ en $\operatorname{Re}(s) > 1$ no tiene ceros para $1 \leq n \leq 5$. Robert Spira demuestra que $\zeta_n(s)$ en $\operatorname{Re}(s) > 1$ no tiene ceros para $6 \leq n \leq 9$ (Spira, 1966), y en un trabajo posterior Spira demuestra que si $\zeta_n(s)$ tiene un cero en $\operatorname{Re}(s) > 1$, entonces tiene infinitos ceros.

Si

$$b_n = \sup\{\operatorname{Re}(s) : \zeta_n(s) = 0\}$$

H.L.Montgomery (Montgomery, 1983) demostró que

$$b_n = 1 + \left(\frac{4}{\pi} - 1 + o(1)\right)\frac{\log(\log n)}{\log n}$$

Si

$$a_n = \text{ínf}\,\{\text{Re}(s) : \zeta_n(s) = 0\}$$

M. Balazard y O. Velásquez Castañon (Balazard and Castanón, 2009) demostraron que

$$a_n = -n\log 2 + o(n)$$

Un resultado clave en el estudio tanto de a_n como de b_n es el teorema de equivalencia de Bohr, que permite torcer los coeficientes de un polinomio de Dirichlet tal como $\zeta_n(s)$ (pero incluso de una serie) para estudiar de manera más sencilla sus propiedades (Dubon, 2015).

En un trabajo posterior Spira demuestra que $\zeta_n(s)$ en $\text{Re}(s) > 1$ (Spira, 1968) tiene ceros para:

$$n = 19, 22 \leq n \leq 27, 29 \leq n \leq 50$$

Por su parte, W.R Monach (Monach, 1980) demuestra que para todo $n > 50$ existen ceros de $\zeta_n(s)$ en $\text{Re}(s) > 1$. Cerrando este enfoque D.J.Platt y T.S. Trudgian en el año 2016 demuestran que $\zeta_n(s)$ en $\text{Re}(s) > 1$ no tiene ceros para

$$10 \leq n \leq 18, \quad n = 20, 21, 28.$$

Teorema 1.1. *Para $1 \leq n \leq 18$, $n = 20, 21, 28$ no hay ceros de $\zeta_n(s)$ en $\text{Re}(s) > 1$. Para los demás enteros positivos n existen infinitos ceros en dicha región.*

Capítulo 2

Análisis teórico

2.1. Reducción de bases de retículos

2.1.1. Retículos

Definición 2.1 (Retículo)**.** *Dados n vectores linealmente independientes $b_1, \ldots, b_n \in \mathbb{R}^m$, el retículo generado por ellos se define como:*

$$\mathcal{L}(b_1, b_2, \ldots, b_n) = \left\{ \sum x_i b_i | x_i \in \mathbb{Z} \right\}$$

Y se refiere a dichos vectores b_i como la base del retículo. De manera equivalente, si se define a B como la matriz de $m \times n$ cuyas columnas son los vectores b_i, entonces el retículo generado por B es:

$$\mathcal{L}(B) = \{Bx | x \in \mathbb{Z}^n\}$$

Por último, se dice que el retículo es de rango n y dimensión m. Si $n = m$, entonces el retículo es un retículo de rango completo.

Definición 2.2 (Espacio generado)**.** *El espacio generado por un retículo $\mathcal{L}(B)$, denotado por* $\text{span}(\mathcal{L}(B))$*, es el espacio lineal expresado como:*

$$\text{span}(\mathcal{L}(B)) = \text{span}(B) = \{By | y \in \mathbb{Z}^n\}$$

Definición 2.3 (Paralelepípedo fundamental)**.** *Para cualquier base de retículo B, se define el paralelepípedo fundamental como:*

$$\mathcal{P}(B) = \{Bx | x \in \mathbb{R}^n, \forall i : 0 \leq x_i < 1\}$$

Además, por la definición del conjunto, si es que se coloca un paralelepípedo fundamental sobre cada punto del retículo, se estaría rellenando el espacio generado $\text{span}(B)$ *por completo como si los paralelepípedos fundamentales fueran baldosas.*

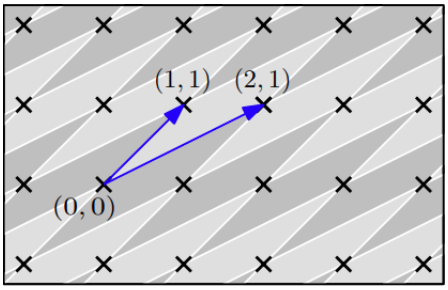

Figura 2.1: Paralelepípedo fundamental de la base $B = \{(1,1),(2,1)\}$. Imagen extraida del curso *Lattices in Computer Science* de la Universidad Tel-Aviv

Lema 2.4. *Sea Λ un retículo de rango n y sean $b_1, b_2, \ldots, b_n \in \Lambda$ vectores linealmente independientes del retículo; entonces, $b_1, b_2, \ldots, b_n$ es una base de Λ si y solo si $\mathcal{P}(b_1, \ldots, b_n) \cap \Lambda = \{0\}$.*

Demostración. Se considerará que $b_1, b_2, \ldots, b_n$ es una base de Λ. Por definición, Λ es el conjunto de todas las combinaciones lineales de $b_1, b_2, \ldots, b_n$ con coeficientes enteros. Dado que el único coeficiente entero x_i en el paralelepípedo fundamental es $x_i = 0$, el único elemento en común que tendrán Λ y $\mathcal{P}(b_1, b_2, \ldots, b_n)$ será 0.
Para probar la dirección contraria, se considerarará que $\mathcal{P}(b_1, b_2, \ldots, b_n) \cap \Lambda = \{0\}$. Ya que $b_1, b_2, \ldots, b_n$ son vectores linealmente independientes y Λ es de rango n, podemos expresar cualquier vector $x \in \Lambda$ como una combinación lineal:

$$x = \sum_{i=1}^{n} y_i b_i, \forall i = 1, \ldots, n : y_i \in \mathbb{R}$$

Como un retículo cumple con la clausura en la adición y $b_1, b_2, \ldots, b_n \in \Lambda$, se tendrá que el vector:

$$w = \sum_{i=1}^{n} (y_i - \lfloor y_i \rfloor) b_i, \forall i = 1, \ldots, n : y_i \in \mathbb{R}$$

pertenece a Λ. Sin embargo, es importante recordar que:

$$0 \leq y_i - \lfloor y_i \rfloor < 1$$

Así que w también pertenece al paralelepípedo fundamental $\mathcal{P}(b_1, b_2, \ldots, b_n)$. Dado que la intersección es el vector 0, tendremos que:

$$y_i - \lfloor y_i \rfloor = 0, \forall i = 1, \ldots, n$$

Finalmente, $y_i \in \mathbb{Z}, \forall i = 1, \ldots, n$, así que los vectores $b_1, b_2, \ldots, b_n$ generan el retículo Λ y además son linealmente independientes (definición de base). □

Definición 2.5 (Matriz Unimodular)**.** *Una matriz $U \in \mathbb{Z}^{n \times n}$ es unimodular si* $\det(U) = \pm 1$.

Lema 2.6. *Si una matriz U es unimodular, entonces su inversa U^{-1} también lo es. Además, $U^{-1} \in \mathbb{Z}^{n \times n}$.*

Lema 2.7. *Dos bases $B_1, B_2 \in \mathbb{R}^{n \times n}$ son equivalentes si y solo si $B_2 = B_1 U$ para alguna matriz unimodular U.*

Demostración. Se considerará que $\mathcal{L}(B_1) = \mathcal{L}(B_2)$; entonces para cada columna b_i de B_2 se da que $b_i \in \mathcal{L}(B_1)$, por lo tanto existe una matriz $U \in \mathbb{Z}^{n \times n}$ tal que:

$$B_2 = B_1 U$$

De manera análoga, se concluye que existe otra matriz $V \in \mathbb{Z}^{n\times n}$ tal que:

$$B_1 = B_2 V \rightarrow B_2 = B_2 VU$$

Se puede tomar:

$$B_2^T B_2 = (VU)^T B_2^T B_2 VU$$

Y al obtener los determinantes:

$$\det(B_2^T B_2) = \det((VU)^T B_2^T B_2 VU) = \det(VU)^2 \det(B_2^T B_2)$$

$$1 = (\det(U)\det(V))^2$$

Pero dado que $U, V \in \mathbb{Z}^{n\times n}$, se debe dar necesariamente que:

$$\det(U) = \pm 1$$

Para probar la dirección contraria, se considerará que $B_2 = B_1 U$, así que cada vector b_i de B_2 está en $\mathcal{L}(B_1)$; por lo tanto, $\mathcal{L}(B_2) \subseteq \mathcal{L}(B_1)$. Se puede plantear el mismo argumento para obtener $\mathcal{L}(B_1) \subseteq \mathcal{L}(B_2)$, pues U^{-1} también es unimodular y $B_1 = B_2 U^{-1}$. Finalmente, se concluye que $\mathcal{L}(B_1) = \mathcal{L}(B_2)$. □

Definición 2.8 (Determinante)**.** *Sea $\Lambda = \mathcal{L}(B)$ un retículo de rango n. Se define la determinante de Λ, denotada por $\det(\Lambda)$, como el volumen n-dimensional de $\mathcal{P}(B)$. Se puede expresar en función a B y reformular la definición a:*

$$\det(\Lambda) = \sqrt{\det(B^T B)}$$

Y cuando B es una matriz cuadrada (y por lo tanto, Λ sea de rango completo), se tendrá que:

$$\det(\Lambda) = |\det(B)|$$

Teorema 2.9. *El determinante de un retículo* $\det(\Lambda)$ *no depende de la base elegida.*

Demostración. Sean B_1, B_2 bases de Λ, entonces se debe cumplir que:

$$B_2 = B_1 U$$

Para alguna matriz unimodular U.

Al tomar la determinante del retículo:

$$\det(\Lambda) = \sqrt{\det(B_2^T B_2)} = \sqrt{\det(U^T B_1^T B_1 U)} = \sqrt{\det(U)^2 \det(B_1^T B_1)} = \sqrt{\det(B_1^T B_1)}$$

Donde la última igualdad es debido a que $\det(U) = \pm 1$, por lo tanto $\det(U)^2 = 1$. □

Definición 2.10 (Ortogonalización Gram-Schmidt)**.** *Para una secuencia de n vectores linealmente independientes* $b_1, b_2, \ldots, b_n$ *se define su ortogonalización Gram-Schmidt como la secuencia de vectores* $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n$*, los cuales son obtenidos de la siguiente manera:*

$$\widetilde{b}_i = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j, \ \ con\ \mu_{i,j} = \frac{\langle b_i, \widetilde{b}_j \rangle}{\langle \widetilde{b}_j, \widetilde{b}_j \rangle}$$

Donde se dice que $\widetilde{b}_i$ *es el componente de* b_i *ortogonal a* $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_{i-1}$.

Algunas propiedades importantes de la ortogonalización Gram-Schmidt son:

1. Debido a la ortogonalidad de las bases, para todo $i \neq j$ se cumple que $\left\langle \widetilde{b}_i, \widetilde{b}_j \right\rangle = 0$.

2. Para todo $i = 1, \ldots, n$, se cumple que:

$$\text{span}(b_1, b_2, \ldots, b_i) = \text{span}(\widetilde{b}_1, \widetilde{b}_2, \ldots, b_i)$$

3. Los vectores $\widetilde{b}_1, \ldots, \widetilde{b}_n$ no necesariamente son una base de $\mathcal{L}(b_1, \ldots, b_n)$.

4. Por la condición de los espacios generados de los primeros i elementos, el orden de los vectores iniciales $b_1, \ldots, b_n$ importa, por lo tanto se considera a la ortogonalización $\widetilde{b}_1, \ldots \widetilde{b}_n$ como una secuencia en vez de un conjunto.

5. Se puede reordenar la expresión:

$$\widetilde{b}_i = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j$$

$$b_i = \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j$$

Si se toma producto interno con $\widetilde{b}_i$:

$$\left\langle b_i, \widetilde{b}_i \right\rangle = \left\langle \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j, \widetilde{b}_i \right\rangle = \left\langle \widetilde{b}_i, \widetilde{b}_i \right\rangle + \sum_{j=1}^{i-1} \mu_{i,j} \left\langle \widetilde{b}_i, \widetilde{b}_j \right\rangle = \left\langle \widetilde{b}_i, \widetilde{b}_i \right\rangle$$

Donde la última igualdad es debido a que $\left\langle \widetilde{b}_i, \widetilde{b}_j \right\rangle = 0$ para todo $i \neq j$. Finalmente:

$$\left\langle b_i, \widetilde{b}_i \right\rangle = \left\langle \widetilde{b}_i, \widetilde{b}_i \right\rangle$$

Además, si es que consideramos un $k > i$, se puede tomar el producto interno:

$$\left\langle b_i, \widetilde{b}_k \right\rangle = \left\langle \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j, \widetilde{b}_k \right\rangle = 0$$

Donde la última igualdad es debido a que $\widetilde{b}_k$ es ortogonal a todas las bases $\widetilde{b}_1, \ldots \widetilde{b}_{k-1}$.

Se puede usar el siguiente algoritmo para procesar la ortogonalización:

Algoritmo 1: Algoritmo Gram-Schmidt

Entrada: $b_1, b_2, \ldots, b_n \in \mathbb{R}^m$ linealmente independientes

Salida: $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n \in \mathbb{R}^m$

$\widetilde{b}_1 \leftarrow b_1$

para $i = 2 \rightarrow n$ **hacer**

  $v \leftarrow b_i$

  para $j = i - 1 \rightarrow 1$ **hacer**

    $\mu_{i,j} \leftarrow \frac{\langle b_i, \widetilde{b}_j \rangle}{\langle \widetilde{b}_j, \widetilde{b}_j \rangle}$

    $v \leftarrow v - \mu_{i,j} \widetilde{b}_j$

  $\widetilde{b}_i \leftarrow v$

Devolver $\{\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n\}$

Lema 2.11. *Sea $b_1, b_2, \ldots, b_n$ una base de un retículo $\Lambda \in \mathbb{R}^m$ y sea $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n$ su ortogonalización Gram-Schmidt, entonces* $\det(\Lambda) = \prod_{i=1}^{n} ||\widetilde{b}_i||$.

Definición 2.12 (Mínimo sucesivo)**.** *Sea Λ un retículo de rango n, para cada $i = 1, \ldots, n$ se define el i-ésimo mínimo sucesivo a:*

$$\lambda_i(\Lambda) = \inf \{r | \dim(\text{span}(\Lambda \cap \overline{B}(0, r))) \geq i\}$$

Donde $\overline{B}(0, r) = \{x \in \mathbb{R}^m : ||x|| \leq r\}$ es la bola cerrada de radio r alrededor de 0.

Teorema 2.13. *Sea B un retículo de rango n y $\widetilde{B}$ su ortogonalización Gram-Schmidt. Si se denota por $\lambda_1, \lambda_2, \ldots, \lambda_n$ a sus mínimos sucesivos, entonces:*

$$\lambda_1 \geq \min_{i=1,\ldots,n} ||\widetilde{b}_i|| > 0$$

Además, sean $w_1, \ldots, w_i \in \Lambda$ vectores linealmente independientes tales que $\max\{||w_1||, \ldots, ||w_i||\} = \lambda_i$ *y* $w_j = \sum_{k=1}^{n} z_{j,k} b_k$. *Para todo $1 \leq j \leq i$ se denota por $k(j)$ al máximo k tal que $1 \leq k \leq n$ y $z_{j,k} \neq 0$. Entonces $||w_j|| \geq ||\widetilde{b}_{k(j)}||$*

Demostración. Sea $x \in \mathbb{Z}^n$ un vector arbitrario diferente de 0, entonces sea $j \in \{1, 2, \ldots, n\}$ el máximo tal que $x_j \neq 0$:

$$\left|\left\langle Bx, \widetilde{b}_j \right\rangle\right| = \left|\left\langle \sum_{i=1}^{j} x_i b_i, \widetilde{b}_j \right\rangle\right| = |x_j| \left\langle b_j, \widetilde{b}_j \right\rangle = |x_j| \cdot ||\widetilde{b}_j||^2$$

Donde la segunda igualdad aprovecha que $\left\langle b_i, \widetilde{b}_j \right\rangle = 0$ para $i < j$ y la tercera considera que $\left\langle b_i, \widetilde{b}_i \right\rangle = \left\langle \widetilde{b}_i, \widetilde{b}_i \right\rangle$.
Por otro lado, se tiene que:

$$\left|\left\langle Bx, \widetilde{b}_j \right\rangle\right| \leq ||Bx|| \cdot ||\widetilde{b}_j||$$

Reemplazando:

$$||Bx|| \cdot ||\widetilde{b}_j|| \geq |x_j| \cdot ||\widetilde{b}_j||^2 \geq ||\widetilde{b}_j||^2$$

Ya que $\widetilde{b}_j \neq 0$:

$$||Bx|| \geq ||\widetilde{b}_j|| \geq \min_{i=1,\ldots,n} ||\widetilde{b}_i|| > 0$$

Ya que $x \in \Lambda$ es arbitrario, se puede tomar $||x|| = \lambda_1$ y se cumple el resultado. En el caso de los vectores w_i, se puede tomar $x = z$ y también se contempla dicha situación.

□

2.1.2. Bases de retículos LLL-reducidas

Definición 2.14 (Base LLL-reducida)**.** *Sea $b_1, b_2, \ldots, b_n$ la base de un retículo Λ y sea $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n$ su ortogonalización Gram-Schmidt. Considerando:*

$$\mu_{i,j} = \frac{\left\langle b_i, \widetilde{b}_j \right\rangle}{\left\langle \widetilde{b}_j, \widetilde{b}_j \right\rangle}, \forall 1 \leq j < i \leq n$$

Y sea $\frac{1}{4} < \delta < 1$ fijo, entonces la base es LLL-reducida (con factor δ) si las siguientes condiciones se cumplen:

- *(Reducidas en tamaño)*

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

- *(Condición de Lovász)*

$$||\widetilde{b}_i||^2 \geq \left(\delta - \mu_{i,i-1}^2\right) ||\widetilde{b}_{i-1}||^2, \forall i = 2, \ldots, n$$

Y comúnmente se usa $\delta = \frac{3}{4}$.

Lema 2.15. *Sea* $b_1, b_2, \ldots, b_n$ *una base LLL-reducida con factor* $\delta = \frac{3}{4}$ *para un retículo* $\Lambda \subset \mathbb{R}^m$ *y* $\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_n$ *su ortogonalización Gram-Schmidt, se cumple:*

1. $2^{i-j}||\widetilde{b}_i||^2 \geq ||\widetilde{b}_j||^2$, *para todo* $1 \leq j \leq i \leq n$.
2. $||\widetilde{b}_i||^2 \leq ||b_i||^2 \leq \left(\frac{1}{2} + 2^{i-2}\right) ||\widetilde{b}_i||^2$, *para todo* $i = 1, \ldots, n$.
3. $||b_j|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i||$, *para todo* $1 \leq j \leq i \leq n$.

Demostración. 1. La condición de Lovász con el factor $\delta = \frac{3}{4}$ implica que:

$$||\widetilde{b}_i||^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) ||\widetilde{b}_{i-1}||^2, \forall i = 2, \ldots, n$$

Ya que $|\mu_{i,j}| < \frac{1}{2}$, se tiene que $\frac{3}{4} - \mu_{i,j}^2 \geq \frac{1}{2}$:

$$||\widetilde{b}_i||^2 \geq \frac{1}{2} ||\widetilde{b}_{i-1}||^2, \forall i = 2, \ldots, n$$

Por lo tanto, se puede concluir que:

$$||\widetilde{b}_i||^2 \geq \frac{1}{2^{i-j}} ||\widetilde{b}_j||^2 \rightarrow 2^{i-j} ||\widetilde{b}_i||^2 \geq ||\widetilde{b}_j||^2$$

Para todo $1 \leq j \leq i \leq n$.

2. Como se cumple que:

$$b_i = \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j$$

entonces tomando producto interno con el mismo vector:

$$\langle b_i, b_i \rangle = \left\langle \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j, \widetilde{b}_i + \sum_{j=1}^{i-1} \mu_{i,j} \widetilde{b}_j \right\rangle$$

$$||b_i||^2 = ||\widetilde{b}_i||^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 ||\widetilde{b}_j||^2$$

Dado que los elementos de la sumatoria son no negativos, es evidente que:

$$||b_i||^2 \geq ||\widetilde{b}_i||^2$$

Si se analiza la cota superior de los sumandos, entonces:

$$||b_i||^2 = ||\widetilde{b}_i||^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 ||\widetilde{b}_j||^2 \leq ||\widetilde{b}_i||^2 + \sum_{j=1}^{i-1} \frac{1}{4} ||\widetilde{b}_j||^2$$

Usando la propiedad 1. del lema, se tiene que:

$$||b_i||^2 \leq ||\widetilde{b}_i||^2 + \sum_{j=1}^{i-1} \frac{1}{4} ||\widetilde{b}_j||^2 \leq ||\widetilde{b}_i||^2 + \frac{1}{4} \sum_{j=1}^{i-1} 2^{i-j} ||\widetilde{b}_i||^2 = ||\widetilde{b}_i||^2 \left(1 + \frac{1}{4} \sum_{j=1}^{i-1} 2^{i-j} \right)$$

Al analizar el coeficiente que acompaña al $||\widetilde{b}_i||^2$:

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

Reemplazando:

$$||b_i||^2 \leq \left(\frac{1}{2} + 2^{i-2} \right) ||\widetilde{b}_i||^2$$

3. Dado que $j \geq 1$, se cumple que:

$$\frac{1}{2} + 2^{j-2} \leq 2^{j-1}$$

Por lo tanto, se puede reescribir la propiedad 2 como:

$$||b_j||^2 \leq 2^{j-1}||\widetilde{b}_j||^2$$

Usando la propiedad 1, se tiene que:

$$||\widetilde{b}_j||^2 \leq 2^{i-j}||\widetilde{b}_i||^2, 1 \leq j \leq i \leq n$$

Combinando ambas desigualdades:

$$||b_j||^2 \leq 2^{j-1}||\widetilde{b}_j||^2 \leq 2^{j-1}2^{i-j}||\widetilde{b}_i||^2 = 2^{i-1}||\widetilde{b}_i||^2$$

Ya que tanto $||b_j||^2$ como $||\widetilde{b}_i||^2$ son no negativos, si se saca la raiz cuadrada a ambos términos de la desigualdad, esta se va a mantener:

$$||b_j|| \leq 2^{\frac{i-1}{2}}||\widetilde{b}_i||$$

Para todo $1 \leq j \leq i \leq n$.

□

Además se pueden deducir otras propiedades importantes cuando se tiene una base LLL-reducida:

Teorema 2.16. *Sea $b_1, b_2, \ldots, b_n$ una base LLL-reducida con factor $\delta = \frac{3}{4}$ para un retículo $\Lambda \subset \mathbb{R}^m$ y sean $\lambda_1, \lambda_2, \ldots, \lambda_n$ sus mínimos sucesivos, así como $\widetilde{b}_1, \widetilde{b}_2, \ldots \widetilde{b}_n$ su ortogonalización Gram-Schmidt, se cumple:*

1. $\|b_1\| \leq 2^{\frac{n-1}{2}} \lambda_1$.
2. $\|b_j\| \leq 2^{\frac{n-1}{2}} \lambda_i$ *para todo* $1 \leq j \leq i \leq n$.
3. $2^{\frac{1-i}{2}} \lambda_i \leq \|b_i\| \leq 2^{\frac{n-1}{2}} \lambda_i$.
4. $\det(\Lambda) \leq \prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(\Lambda)$
5. $\|b_1\| \leq 2^{\frac{n-1}{4}} \det(\Lambda)^{\frac{1}{n}}$.

Demostración. 1. Se puede deducir de la primera propiedad de 2.15 que $2^{\frac{i-1}{2}} \|\widetilde{b}_i\| \geq \|\widetilde{b}_1\|$. Además, debido a 2.13, se tiene que:

$$\lambda_1 \geq \min_{i=1,\ldots,n} \|\widetilde{b}_i\| \geq \min_{i=1,\ldots,n} 2^{\frac{1-i}{2}} \|\widetilde{b}_1\| = 2^{\frac{1-n}{2}} \|\widetilde{b}_1\|$$

Tomando en cuenta que $b_1 = \widetilde{b}_1$, se concluye que:

$$2^{\frac{n-1}{2}} \lambda_1 \geq \|b_1\|$$

2. Sean $w_1, w_2, \ldots, w_i \in \Lambda$ vectores linealmente independientes del retículo tales que $\max\{\|w_1\|, \ldots, \|w_n\|\} = \lambda_i$, además sea la notación $k(j)$ definida en 2.13.

   Si se reordenan los vectores de forma que:

$$k(1) \leq k(2) \leq \ldots \leq k(i)$$

   Afirmamos que $j \leq k(j)$, pues de lo contrario $w_1, \ldots, w_j$ pertenecerían a $\text{span}(b_1, \ldots, b_{j-1})$ y no serían linealmente independientes.

   Finalmente:

$$\|b_j\| \leq 2^{\frac{k(j)-1}{2}} \|\widetilde{b}_{k(j)}\| \leq 2^{\frac{n-1}{2}} \|\widetilde{b}_{k(j)}\| \leq 2^{\frac{n-1}{2}} \|w_j\| \leq 2^{\frac{n-1}{2}} \lambda_i$$

3. La cota superior de la desigualdad está dada por la propiedad 2, así que se debe probar la cota inferior:

   Ya que $b_1, \ldots, b_i$ son linealmente independientes, se tiene que:

   $$\lambda_i \leq \max_{1 \leq j \leq i} ||b_j||$$

   Además, por 2.15, se cumple que:

   $$||b_j|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i||, \forall 1 \leq j \leq i \rightarrow \max_{1 \leq j \leq i} ||b_j|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i||$$

   Ya que $||\widetilde{b}_i|| \leq ||b_i||$:

   $$\max_{1 \leq j \leq i} ||b_j|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i|| \leq 2^{\frac{i-1}{2}} ||b_i||$$

   Por lo tanto, se obtiene el resultado:

   $$\lambda_i \leq 2^{\frac{i-1}{2}} ||b_i||$$

   $$2^{\frac{1-i}{2}} \lambda_i \leq ||b_i||$$

4. Por 2.11, se tiene que $\det(\Lambda) = \prod_{i=1}^{n} ||\widetilde{b}_i||$.

   Usando que $||\widetilde{b}_i|| \leq ||b_i|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i||$, se puede tomar el producto de todos los posibles i:

   $$\prod_{i=1}^{n} ||\widetilde{b}_i|| \leq \prod_{i=1}^{n} ||b_i|| \leq \prod_{i=1}^{n} 2^{\frac{i-1}{2}} ||\widetilde{b}_i||$$

   Se puede notar que la cota inferior se vuelve $\det(\Lambda)$, mientras que la cota superior se convierte en $2^{\sum_{i=1}^{n} \frac{i-1}{2}} \det(\Lambda)$:

$$\det(\Lambda) \leq \prod_{i=1}^{n} ||b_i|| \leq 2^{\frac{n(n-1)}{4}} \det(\Lambda)$$

5. De 2.15 se tiene que:

$$||b_1|| \leq 2^{\frac{i-1}{2}} ||\widetilde{b}_i||$$

Tomando producto para todos los posibles i:

$$||b_1||^n \leq 2^{\frac{n(n-1)}{4}} \det(\Lambda)$$

Finalmente:

$$||b_1|| \leq 2^{\frac{n-1}{4}} \det(\Lambda)$$

□

2.2. Algoritmo Lenstra-Lenstra-Lovász (LLL)

El algoritmo LLL transforma una base de un retículo Λ en una LLL-reducida. Su procedimiento iterativo construye una base reducida en tamaño a partir del prefijo de vectores $b_1, b_2, \ldots, b_i$ y verifica que se cumpla la condición de Lovász: En caso de no cumplirse, los vectores b_i y b_{i-1} son cambiados de posición y deshacer el último vector de la construcción.

Este algoritmo tiene la ventaja de ser simple en términos de implementación y concepto, pero presenta bastantes incógnitas y propiedades interesantes al momento de

analizarlo. El pseudocódigo del algoritmo es el siguiente:

Algoritmo 2: Algoritmo LLL

Entrada: $b_1, b_2, \ldots, b_n \in \mathbb{R}^m$ linealmente independientes, $\frac{1}{4} < \delta < 1$

Salida: Una base LLL-reducida $b_1, b_2, \ldots, b_n \in \mathbb{R}^m$

Ejecutar el algoritmo Gram-Schmidt para obtener $\widetilde{b}_1, \ldots, \widetilde{b}_n$ y los coeficientes $\mu_{i,j}$

Procesar $B_i \leftarrow \left\langle \widetilde{b}_i, \widetilde{b}_i \right\rangle$ para todo $1 \leq i \leq n$

$k \leftarrow 2$

mientras $k \leq n$ **hacer**

  para $j = k-1 \rightarrow 1$ **hacer**

    $q_{k,j} \leftarrow \lfloor \mu_{k,j} \rceil$

    $b_k \leftarrow b_k - q_{k,j} b_j$

    Actualizar los valores de $\widetilde{b}_1, \ldots, \widetilde{b}_n$ y $\mu_{i,j}$ correspondientemente

  si $B_k \geq \left(\delta - \mu_{k,k-1}^2\right) B_{k-1}$ **entonces**

    $k \leftarrow k+1$

  en otro caso

    Intercambiar b_k y b_{k-1}

    Actualizar los valores $\widetilde{b}_i$, $\mu_{i,j}$ y B_i que se vean afectados

    $k \leftarrow \max\{2, k-1\}$

Devolver $\{b_1, b_2, \ldots, b_n\}$

Es de natural interés el análisis de la complejidad del algoritmo y la calidad de su salida.

La primera pregunta que se plantea es si el algoritmo propuesto terminará en todos los posibles casos. Antes de responder a esta incógnita, plantearemos el siguiente lema:

Lema 2.17. *Sea la notación b_i', $\mu_{i,j}'$ y B_i' usada para referirse a los nuevos valores de las bases, coeficientes del algoritmo Gram-Schmidt y longitudes al cuadrado de cada vector, respectivamente. Luego de intercambiar los vectores b_k y b_{k-1} en el algoritmo LLL, entonces se dan los siguientes cambios:*

1. *El nuevo valor de $\widetilde{b}_{k-1}$ será $\widetilde{b}_k + \mu_{k,k-1} b_{k-1}$ y el nuevo valor de B_{k-1} será $B_k +$*

$\mu_{k,k-1}^2 B_{k-1}$.

2. *El nuevo valor de* $\widetilde{b}_k$ *será* $\left(\frac{B_k}{B'_{k-1}}\right)\widetilde{b}_{k-1} - \left(\mu_{k,k-1}\frac{B_{k-1}}{B'_{k-1}}\right)\widetilde{b}_k$ *y el nuevo valor de* B_k *será* $\frac{B_{k-1}B_k}{B'_{k-1}}$.

Demostración. 1. Como se tiene que:

$$\widetilde{b}'_{k-1} = b'_{k-1} - \sum_{j=1}^{k-2} \mu'_{k-1,j}\widetilde{b}'_j$$

$$\widetilde{b}'_{k-1} = b_k - \sum_{j=1}^{k-2} \mu_{k,j}\widetilde{b}_j$$

$$\widetilde{b}'_{k-1} = b_k - \sum_{j=1}^{k-2} \mu_{k,j}\widetilde{b}_j - \mu_{k,k-1}\widetilde{b}_{k-1} + \mu_{k,k-1}\widetilde{b}_{k-1}$$

$$\widetilde{b}'_{k-1} = \widetilde{b}_k + \mu_{k,k-1}\widetilde{b}_{k-1}$$

Por consiguiente, se tendrá que:

$$B'_k = B_k + \mu_{k,k-1}^2 B_{k-1}$$

2. Análogamente al cambio 2:

$$\widetilde{b}'_k = b'_k - \sum_{j=1}^{k-1} \mu'_{k,j}\widetilde{b}'_j$$

$$\widetilde{b}'_k = b_{k-1} - \sum_{j=1}^{k-1} \mu_{k-1,j}\widetilde{b}'_j$$

$$\widetilde{b}'_k = b_{k-1} - \sum_{j=1}^{k-2} \mu_{k-1,j}\widetilde{b}_j - \mu_{k-1,k-1}\widetilde{b}'_{k-1}$$

$$\widetilde{b}'_k = \widetilde{b}_{k-1} - \left(\frac{\left\langle b_{k-1}, \widetilde{b}'_{k-1} \right\rangle}{B'_{k-1}} \right) \left(\widetilde{b}_k + \mu_{k,k-1} \widetilde{b}_{k-1} \right)$$

$$\widetilde{b}'_k = \widetilde{b}_{k-1} - \left(\frac{\left\langle b_{k-1}, \widetilde{b}_k + \mu_{k,k-1} \widetilde{b}_{k-1} \right\rangle}{B'_{k-1}} \right) \left(\widetilde{b}_k + \mu_{k,k-1} \widetilde{b}_{k-1} \right)$$

$$\widetilde{b}'_k = \widetilde{b}_{k-1} - \left(\frac{\mu_{k,k-1} B_{k-1}}{B'_{k-1}} \right) \left(\widetilde{b}_k + \mu_{k,k-1} \widetilde{b}_{k-1} \right)$$

$$\widetilde{b}'_k = \left(1 - \mu^2_{k,k-1} \frac{B_{k-1}}{B'_{k-1}} \right) \widetilde{b}_{k-1} - \left(\frac{\mu_{k,k-1} B_{k-1}}{B'_{k-1}} \right) \widetilde{b}_k$$

$$\widetilde{b}'_k = \left(\frac{B_k}{B'_{k-1}} \right) \widetilde{b}_{k-1} - \left(\frac{\mu_{k,k-1} B_{k-1}}{B'_{k-1}} \right) \widetilde{b}_k$$

Finalmente, no es difícil deducir que:

$$B'_k = \frac{B_{k-1} B_k}{B'_{k-1}}$$

□

Teorema 2.18. *Sea Λ un retículo en $\mathbb{R}^m$ con base $b_1, b_2, \ldots, b_n$ y sea $X \in \mathbb{R}_{\geq 2}$ tal que $\|b_i\|^2 \leq X$ para todo $1 \leq i \leq n$. Sea $\frac{1}{4} < \delta < 1$ el factor del algoritmo LLL; entonces, bajo la condición de que $vol(B)^2 \geq Y$ en todo momento, este terminará en $O(n^2 \log\left(\frac{X}{Y}\right))$ iteraciones.*

El considerar las operaciones aritméticas a lo largo de la ejecución del algoritmo también es importante, ya que si se desea tener datos lo suficientemente precisos, se deberá manipular las variables de manera adecuada.

Teorema 2.19. *Sea Λ un retículo en $\mathbb{R}^m$ con base $b_1, b_2, \ldots, b_n$ y sea $X \in \mathbb{R}_{\geq 2}$ tal que $\|b_i\|^2 \leq X$ para todo $1 \leq i \leq n$. Sea $\frac{1}{4} < \delta < 1$ el factor del algoritmo LLL; entonces, bajo la condición de que $vol(B)^2 \geq Y$ en todo momento, se ejecutarán*

$O(n^5 m \log X^2 \log\left(\frac{X}{Y}\right))$ *operaciones de bits dado que los valores numéricos usados son almacenados en variables de* $O(n \log X)$ *bits.*

2.3. Aplicación del algoritmo LLL

Se explicará la aplicación principal del algoritmo LLL para este estudio, no sin antes presentar

Teorema 2.20 (Teorema de Kronecker)**.** *Dados* m *vectores* $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{R}^n$ *y* $\beta \in \mathbb{R}^n$*, se cumple que:*

$$\forall \varepsilon > 0, \exists q_i, p_j \in \mathbb{Z} : \left| \sum_{i=1}^{m} \alpha_{i,j} q_i - p_j - \beta_j \right| < \varepsilon, 1 \leq j \leq n$$

si y solo si, para todo $r_1, r_2, \ldots, r_n \in \mathbb{Z}$*:*

$$\sum_{j=1}^{n} \alpha_{ij} r_j \in \mathbb{Z}, i = 1, \ldots, m \rightarrow \sum_{j=1}^{m} \beta_j r_j \in \mathbb{Z}$$

Proposición 2.21 (Lenstra et al. 1982)**.** *Existe un algoritmo de complejidad polinomial tal que, dado un entero positivo* n *y racionales* $\alpha_1, \alpha_2, \ldots, \alpha_n, \varepsilon$*, con* $0 < \varepsilon < 1$*, encuentra enteros* $p_1, p_2, \ldots, p_n, q$ *que satisfacen:*

$$|p_i - q\alpha_i| \leq \varepsilon, \forall 1 \leq i \leq n$$

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

La proposición es demostrada aplicando el algoritmo LLL sobre el retículo Λ formado por:

$$b_i = e_i, \forall 1 \leq i \leq n$$

$$b_{n+1} = \left(-\alpha_1, -\alpha_2, \ldots, -\alpha_n, 2^{-\frac{n(n+1)}{4}} \varepsilon^{n+1}\right)^T$$

Notando que al hallar una base LLL-reducida del mismo implica hallar un vector $b_1 \in \Lambda$, por lo cual:

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

Aprovechando la propiedad 5. de 2.16:

$$||b_1|| \leq 2^{\frac{n}{4}} \det(\Lambda)^{\frac{1}{n+1}} = \varepsilon$$

Se tendrá entonces que:

$$|p_i - q\alpha_i| \leq \varepsilon, \forall 1 \leq i \leq n$$

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

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

Ya que $\varepsilon < 1$, y considerando que si $q < 0$, entonces $-b_1$ cumple con las condiciones y permite que $q > 0$:

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

De la proposición anterior obtendremos el siguiente colorario, que será la base para este estudio:

Corolario 2.22. *Se pueden hallar $\mathbb{Q}$-relaciones entre reales $\alpha_1, \alpha_2, \ldots, \alpha_n$ usando reducción de bases sobre el retículo Λ como $\mathbb{Z}^n$ extendido a $\mathbb{R}^{n+1}$ por:*

$$(m_1, m_2, \ldots, m_n) \mapsto (m_1, m_2, \ldots, m_n, c\sum_{i=1}^{n} m_i \alpha_i')$$

Donde c es una constante grande y α_i' es una buena aproximación racional de α_i.

La pregunta principal será ¿Cómo aprovechar esto para hallar ceros especiales?
La respuesta es bastante simple considerando que se desea una respuesta aproximada: Se puede plantear la existencia de un valor T tal que $\zeta_n(s) \approx \zeta_n(s + (0, T))$ y analizar qué características debe cumplir T.

$$\zeta_n(s) = \sum_{i=1}^{n} i^{-s} = \sum_{i=1}^{n} \mathrm{e}^{-s \log i}$$

Se denotará por $\hat{j}$ a la unidad imaginaria, de forma que:

$$\zeta_n(s + \hat{j}T) = \sum_{i=1}^{n} \mathrm{e}^{-(s+\hat{j}T) \log i}$$

La finalidad principal es que la diferencia sea la mínima posible, entonces:

$$\zeta_n(s + \hat{j}T) - \zeta_n(s) = \sum_{i=1}^{n} \mathrm{e}^{-(s+\hat{j}T) \log i} - \sum_{i=1}^{n} \mathrm{e}^{-s \log i}$$

Se puede agrupar miembro a miembro:

$$\zeta_n(s + \hat{j}T) - \zeta_n(s) = \sum_{i=1}^{n} \mathrm{e}^{-(s+\hat{j}T) \log i} - \mathrm{e}^{-s \log i} = \sum_{i=1}^{n} \mathrm{e}^{-s \log i} \left(\mathrm{e}^{-\hat{j}T \log i} - 1 \right)$$

Dado que las condiciones sobre T aún no están definidas, se puede plantear el volver la expresión:

$$\mathrm{e}^{-\hat{j}T \log i} - 1 \approx 0 \rightarrow \mathrm{e}^{-\hat{j}T \log i} \approx 1$$

Sin embargo, debido a las fórmulas de Moivre, se tiene que:

$$\cos(T \log i) - \hat{j} \sin(T \log i) \approx 1$$

Por lo anterior, se desea aproximar:

$$T \log i \approx 2\pi k, k \in \mathbb{Z}$$

En términos más formales, se elegirá un $\varepsilon > 0$ de forma que:

$$|T \log i - 2\pi k| < \varepsilon$$

Para algún $k \in \mathbb{Z}$. Sin embargo, esta condición se debe mantener para **todo** $i = 1, 2, \ldots, n$, por lo tanto se puede definir $k_i \in \mathbb{Z}$ asociado a cada i:

$$|T \log i - 2\pi k_i| < \varepsilon$$

Lo anterior implicaría resolver el problema para un retículo de dimensión $n + 1$, pero es importante notar que esto no es cierto y que basta un retículo de dimensión $\pi(n) + 1$:

$$T \log p \approx 2\pi k, k \in \mathbb{Z}, \forall p \leq n, p \text{ primo} \rightarrow T \log m \approx 2\pi k, k \in \mathbb{Z}, \forall m \leq n$$

Es importante notar la ambigüedad de la aproximación: Debido a que todo número entero se puede descomponer en potencias de primos diferentes, se termina justificando la proposición anterior, pero la aproximación para los valores primos será más precisa que para sus combinaciones lineales (compuestos). La condición sobre T será:

$$|T \log p_i - 2\pi k_i| < \varepsilon, \forall p_i \leq n, p_i \text{ primo}$$

Ahora, se puede aplicar una pequeña transformación a la condición deseada sobre T:

$$|T \log p_i - 2\pi k_i| < \varepsilon \rightarrow \left| T \left(\frac{\log p_i}{2\pi} \right) - k_i \right| < \frac{\varepsilon}{2\pi} = \varepsilon'$$

Finalmente, se puede aplicar 2.21 para resolver este problema con:

$$\alpha_i = \frac{\log p_i}{2\pi}$$

Capítulo 3

Análisis implementativo

En este capítulo se describirá la implementación del algoritmo LLL usando una librería para mantener una alta precisión sobre los números de punto flotante.

3.1. Lenguaje y librerías

Los resultados presentados en este informe han sido generados usando un código escrito en Python 3, mientras que los gráficos de los ceros obtenidos han sido generados usando gnuplot en formato eps.
Para poder mantener una buena precisión en los valores procesados, se decidió usar la librería *Mpmath* [6], la cual permite realizar cálculos con números reales y complejos manteniendo una precisión de hasta 1000 dígitos significativos. Como referencia, la librería de matemática simbólica *SymPy* usa como base a Mpmath para realizar cálculos numéricos.
Para la reducción de las bases de los retículos, se usó una precisión de 100 dígitos significativos en el entorno de Mpmath.

3.2. Implementación del algoritmo LLL

Se decidió por usar la implementación trivial del algoritmo LLL, propuesta en el artículo original de Lenstra et al. en 1982. Esta versión del algoritmo reconstruye por completo la ortogonalización Gram-Schmidt cada vez que se realiza el intercambio entre los vectores b_{k-1} y b_k. El principal motivo por el cual se usó este algoritmo es debido a que se trabajarán con retículos de dimensión pequeña, así que no es muy necesario el dedicar mucho tiempo a realizar optimizaciones sobre este.
Una alternativa sobre el algoritmo en sí sería el ir construyendo la ortogonalización a medida que el k va a aumentando; sin embargo, esta forma de implementación requiere mucho cuidado.
Se elaboró la implementación en Python 3, para la cual solo se necesitaron unas pocas funciones:
Primero, la ortogonalización Gram-Schmidt, como se estableció en el Algoritmo 1.

```
def getGramSchmidt(v):
    n = len(v)
    bp = [mpf('0') for i in range(n)]
    for i in range(n):
        bp[i] = v[i]
        j = i - 1
        while j >= 0:
            mu = dot(v[i], bp[j]) / dot(bp[j], bp[j])
            for pos in range(len(bp[i])):
                bp[i][pos] -= mu * bp[j][pos]
            j -= 1
    return bp
```

Se debe recalcar que la función $dot(u, v)$ obtiene el producto escalar de los vectores u y v.
Luego, se implementó el algoritmo LLL.

```
def LLL(base, delta):
    b = copy.deepcopy(base)
    n = len(b)
    bp = copy.deepcopy(b)
    bp = getGramSchmidt(bp)
    mu = []
    B = []
    for i in range(n):
        B.append(dot(bp[i], bp[i]))
        j = 0
        cur = []
        while j <= i:
            cur.append(dot(b[i], bp[j]) / dot(bp[j], bp[j]))
            j += 1
        mu.append(cur)
    # Go with LLL
    k = 1
    while k < n:
        for j in range(k - 1, -1, -1):
            q = round(mu[k][j])
            if q == 0: continue
            for pos in range(len(b[k])):
                b[k][pos] -= q * b[j][pos]

        for pos in range(k + 1):
            if pos <= k:
                mu[k][pos] = dot(b[k], bp[pos]) / dot(bp[pos], bp[pos])

        if B[k] >= (delta - mu[k][k - 1] ** 2) * B[k - 1]:
```

```
            k += 1
        else:
            b[k], b[k - 1] = b[k - 1], b[k]
            bp = copy.deepcopy(b)
            bp = getGramSchmidt(bp)
            for i in range(n):
                B[i] = dot(bp[i], bp[i])
                j = 0
                while j <= i:
                    mu[i][j] = dot(b[i], bp[j]) / dot(bp[j], bp[j])
                    j += 1
            if k >= 2: k -= 1
    return b
```

3.3. Obtención de T

Se aplicará la reducción de base como está detallado en 2.21 debido a que una aritmética limitada impide que los valores sean tratados como irracionales; por lo tanto, en la práctica se puede asumir los valores $\log p_i$ y π como racionales.
También se debe recordar que la elección del ε es importante al momento de realizar la aproximación para cada logaritmo usado.
En este caso, se tienen algunas problemáticas debido a la limitada precisión en comparación con el crecimiento exponencial del valor de T.
A medida que aumenta el valor de n, la cantidad de dimensiones aumentará también, así que la cota superior del valor T dada por:

$$2^{\frac{n(n+1)}{4}}\varepsilon^{-n}$$

Y dado que $\varepsilon < 1$, esto implica que ε^{-n} crece a medida que el n aumenta también.

Se tenía una relación entre el error deseado en:

$$\frac{\varepsilon}{2\pi} = \varepsilon'$$

Así que se usó $\varepsilon = 10^{-2}$, para que el valor de ε' sea tal que no genere mucho error de precisión.

$$\varepsilon' \approx 0{,}00159154943091895335768883763372514362034459645740456448 74$$

Es importante recordar que si uno varía el valor de ϵ', los ceros especiales hallados serán diferentes.

3.4. Cálculo de ceros especiales

Para el cálculo de los ceros especiales, luego de calcular el valor T para cada n válido, se usó el algoritmo en paralelo para el cálculo de ceros de las sumas parciales. Este algoritmo fue desarrollado en el Seminario I y calculaba cada cero de la suma parcial en un rectángulo dado.

Dada la naturaleza de la función, se considera la aproximación como un punto de referencia, por lo cual será necesario extenderla a un rectángulo lo suficientemente pequeño (para evitar cálculos innecesario) pero al mismo tiempo suficientemente grande (para garantizar el obtener el cero que está en esa región).

Entonces, se usará un bucle partiendo desde un cero s con $\mathrm{Im}(s)$ pequeño y se buscará el cero con el algoritmo previamente elaborado para Seminario I.

Se usó el siguiente código en la función principal:

```
mp.dps = 500
n = int(sys.argv[1])
T = mpf(input())
first_zero = input().split()
s0 = mpc(mpf(first_zero[0]), mpf(first_zero[1]))
```

```
EPSreal = mpf('0.1')
EPSimag = mpf('5')
last_real = s0.real
last_imag = s0.imag
print(last_real, last_imag)
threads = multiprocessing.cpu_count()
while True:
    LD = mpc(max(mpf('1.0000000000000001'), last_real - EPSreal),
    last_imag + T - EPSimag)
    RU = mpc(min(mpf('2.0000000000000000'), last_real + EPSreal),
    last_imag + T + EPSimag)
    start = time.time()
    S = MutexStack(LD, RU, n)
    R = MutexAnswer(threads)
    initialize(S, R, n, threads)
    with concurrent.futures.ThreadPoolExecutor(max_workers=threads) as executor:
        for index in range(threads):
            executor.submit(solve, S, R, index, n)
    end = time.time()
    if R.size() == 0: break
    res = R.pop()
    last_real = res.real
    last_imag = res.imag
    print(res.real, res.imag)
    while R.size() > 0:
        res = R.pop()
        print(res.real, res.imag)
```

En términos sencillos, el algoritmo aumenta $\hat{j}T$ al cero actual (inicializado en un cero

hallado previamente) hasta que ya no se pueda encontrar algún cero en la región:

$$\text{máx}\left\{1 + 10^{-16}, \text{Re}(S) - 0{,}1\right\} \leq \text{Re}(s) \leq \text{mín}\left\{2, \text{Re}(S) + 0{,}1\right\}$$

$$\text{Im}(S) + T - 5 \leq \text{Im}(s) \leq \text{Im}(S) + T + 5$$

Donde S es una referencia al último cero encontrado durante la ejecución del algoritmo.

Capítulo 4

Resultados

4.1. Tabla resumen de los resultados

Primero se presentarán los valores de T hallados para cada $23 \leq n \leq 50$:

n	T
23-28	553875066323779947646626752628 5.0
29-30	587088566951784144513678754157 4994.0
31-36	818801392366123434120811741787549314015.0
37-40	119763205971933347002301043291234090963463 97.0
41-42	6846623298272882561792008376520728178080262598.0
43-46	178966696696026729062133558671975332854893205674 7747.0
47-50	24365956383576486807836922436817854335335699917445757516.0

Cuadro 4.1: Recopilación de los T obtenidos

Ahora, se resumirán los datos obtenidos para $n = 23, 24, 25, 26, 31$ en la siguiente tabla:

n	$\mathrm{Re}(s_0)$	$\mathrm{Im}(s_0)$
23	1.01044334922698341898	8502832.39912065772578142170
24	1.00404186833602031723	32520751.78599510357179634043
25	1.00044920152434295543	32520751.80223907180402332765
26	1.00635284737135701011	36323746.32695248213955443469
31	1.00710368676439502484	52331955.65876127657415336860

Cuadro 4.2: Recopilación de los ceros especiales iniciales s_0

n	Ceros obtenidos	$\min\{\mathrm{Re}(s)\}$	$\max\{\mathrm{Re}(s)\}$
23	195	1.00002145180704386726	1.01044334922698341898
24	353	1.00001966581048224832	1.00514987687339267741
25	394	1.00003430637672226521	1.00358972824030854411
26	184	1.00000072184326632315	1.00635284737135701011
31	446	1.00002177365653510828	1.00710499384343543061

Cuadro 4.3: Recopilación de los ceros especiales obtenidos

Capítulo 5

Conclusiones

5.1. Conclusión inmediata

Se pudo implementar un algoritmo mediante el cual se puede generar una secuencia relativamente grande de ceros especiales partiendo desde un cero especial s_0.

5.2. Discusión

Durante la experimentación se notó la gran necesidad de una aritmética lo suficientemente precisa, debido a que los valores intermedios de tanto el algoritmo LLL como el algoritmo para buscar los ceros especiales en una región llegan a ser exponencialmente pequeños o grandes. De hecho, tener un ε' muy pequeño conlleva a generar valores incorrectos de T en algunos casos.

Una observación interesante del algoritmo LLL es cómo poder optimizar su rendimiento. Desde un punto de vista algorítmico y con un cierto grado de conocimientos teóricos de la complejidad del LLL, uno puede relacionarlo vagamente con un algoritmo de ordenamiento, siendo el más cercano el *Insertion Sort*. Sin embargo, este algoritmo no sería un Insertion Sort natural, sino que cada elemento a ordenar tendría consigo una función potencial para saber cuántas veces puede ser intercambiado, aunque por el momento no se puede deducir bien si dicha función potencial es respecto

al elemento mismo o a la posición del intercambio. Se podrían plantear optimizaciones basadas en aleatoriedad, de manera que la complejidad esperada resulte buena — Debido a que trabajamos con intercambios, se puede decir que necesitaríamos reducir la cantidad de *inversiones* según algún orden parcial $\preceq$. En estos momentos se debe recordar que la complejidad esperada del *Quick Sort* es $O(n \log n)$, por lo tanto, una idea de optimización del LLL sería ordenar la base en una primera instancia mediante el comparador $\preceq$ y esperar a que la cantidad de intercambios termine siendo $O(n \log n)$ con alta probabilidad. La dificultad final radicaría en que el comparador debería ser lo suficientemente insesgado respecto a la norma $\|\cdot\|$, pues durante el mismo algoritmo este valor se irá reduciendo.

Bibliografía

[1] M. Balazard and O. V. Castanón. Sur l'infimum des parties réelles des zéros des sommes partielles de la fonction zêta de riemann. *Comptes Rendus Mathematique*, 347(7-8):343–346, 2009.

[2] E. Bombieri. Problems of the millennium: The riemann hypothesis, 2000.

[3] P. Borwein, G. Fee, R. Ferguson, and A. van der Waall. Zeros of partial sums of the riemann zeta function. *Experimental Mathematics*, 16(1):21–39, 2007.

[4] E. Dubon. Sobre los ceros de polinomios de dirichlet, en general, y los de las sumas parciales de la función zeta de riemann, en particular. *Proyecto de investigación:*, 2015.

[5] S. D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[6] F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*, December 2013. `http://mpmath.org/`.

[7] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[8] H. L. Montgomery. Zeros of approximations to the zeta function. In *Studies in pure mathematics*, pages 497–506. Springer, 1983.

[9] G. Mora and J. M. Sepulcre. Computing the zeros of the partial sums of the riemann zeta function. *Annali di Matematica Pura ed Applicata (1923-)*, 194(5):1499–1504, 2015.

[10] P. Q. Nguyen and B. Vallée. *The LLL algorithm*. Springer, 2010.

[11] D. J. Platt and T. S. Trudgian. Zeroes of partial sums of the zeta-function. *LMS Journal of Computation and Mathematics*, 19(1):37–41, 2016.

[12] B. Riemann. Consecuencias de la hipótesis de riemann. *Revista de educación matemática*, 30(3), 2015.

[13] C. P. Schnorr. Progress on lll and lattice reduction. In *LLL+ 25 Conf.* Citeseer, 2007.

[14] R. Spira. Zeros of sections of the zeta function. i. *Mathematics of Computation*, 20(96):542–550, 1966.

[15] R. Spira. Zeros of sections of the zeta function. ii. *Mathematics of Computation*, 22(101):163–173, 1968.

[16] P. Turán. *On some approximative Dirichlet-polynomials in the theory of the zeta-function of Riemann*, volume 24. I kommission hos Munksgaard, 1948.

[17] X. Ying and I. N. Katz. A reliable argument principle algorithm to find the number of zeros of an analytic function in a bounded domain. *Numerische Mathematik*, 53(1-2):143–163, 1988.