1 L1 Norm

Derive an upper bound on the $\ell_1$ minimum distance of a full-dimensional lattice (as a function of the determinant) from Minkowski’s convex body theorem, similarly to what we did in class for the $\ell_\infty$ and $\ell_2$ norms. Give special cases of your bound in dimension 1, 2 and 3, as well as a general formula for dimension $n$.

2 Covering Radius

The covering radius of a lattice $\Lambda$ is the smallest radius $\rho$ such that spheres of radius $\rho$ centered around all lattice points cover the entire linear span of the lattice, i.e., $\text{span}(\Lambda) \subset \bigcup \{B(\vec{x}, r) \mid \vec{x} \in \Lambda\}$. Alternatively, the covering radius can be defined as the maximum distance of any point $\vec{t} \in \text{span}(\Lambda)$ to the lattice $\Lambda$, i.e., $\rho = \sup \{\text{dist}(\vec{x}, \Lambda) \mid \vec{x} \in \text{span}(\Lambda)\}$

1. Determine the minimum distance $\lambda$ and the covering radius $\rho$ of the integer lattice $\mathbb{Z}^n$, for arbitrary $n$

2. Prove that if $\Lambda = \mathcal{L}(\mathcal{B})$, then $\rho(\Lambda) \leq \frac{1}{2} \sqrt{\sum_i \|\vec{b}_i^*\|^2}$, where $\vec{b}_i^*$ are the Gram-Schmidt orthogonalized basis vectors.

3. Find a lattice basis $\mathcal{B}$ (for a full rank lattice in arbitrary dimension $n$) such that $\rho(\Lambda) = \frac{1}{2} \sqrt{\sum_i \|\vec{b}_i^*\|^2}$ holds with equality.

3 Cryptanalysis

You have intercepted a secret message. (You will receive the message over email soon!) You know the message as been encrypted using a truncated linear congruential generator with parameters

- $p = 258535798238006737310015446227842039611$
- $a = 55899726347639041718090144141206057179$
- $b = 206062773649155181136659667127305895548$
As a reminder, the linear congruential generator starts from a random secret seed \( x_0 \in \mathbb{Z}_p \) (unknown to you), computes the sequence \( x_{i+1} = ax_i + b \pmod{p} \in \mathbb{Z}_p \), and outputs the 8 least significant bits \( y_i = x_i \pmod{256} \) from each value in the sequence.

You know these values \((y_0, y_1, \ldots)\) have been used as a one-time pad to encrypt the message by breaking the message into a sequence of characters \((m_0, m_1, \ldots)\), interpreting each character as an 8-bit number using the standard ASCII encoding, and computing the ciphertext as \( c_i = m_i + k_i \pmod{256} \). You also know that the message begins with the string “CSE206A Cryptanalysis Challenge:”. (Pay attention to upper and lower case.)

Decrypt the message, and include it in your answer together with a brief description of how you solved the problem.

As part of this problem, you will need to run a combination of LLL lattice basis reduction and the nearest plane algorithm to solve an instance of the bounded distance decoding problem. You are neither required nor expected to implement your own lattice algorithms as part of this problem. Instead, use one of the many available implementations linked on the course webpage. The easiest way is probably to use “fplll”, which provides both a library and a command line interface. The command line interface should be enough to solve this problem. You can install fplll from github following the link on the course webpage. (It may also be available through your operating system package manager.) Then, if you want to solve a BDD or CVP instance, write the basis matrix and vector in a file and then run “fplll -a cvp < inputfile”. For example, if the input file contains “[[3 0] [0 1]] [5 6]”, the command should output the vector “[6 6]”. Here fplll first runs the LLL basis reduction algorithm (you don’t need to reduce the basis before calling fplll) and then finds the lattice vector closest to the target [5 6] using a variant of the nearest plane algorithm. (Specifically, it first runs the nearest plane algorithm, and if a suitable solution is not found, it performs an exhaustive search for the closest lattice vector.) In general, this may take a lot (exponential!) time to produce a solution, because the closest vector problem is NP-complete. But for the BDD instances involved in this problem, the running time will be polynomial, and, in fact, pretty small, say just a fraction of a second (including the time to perform basis reduction.)

If you want to see what an LLL reduced basis looks like, just run “fplll -a lll” on the lattice basis.