Algorithm Design for Finding Generator Matrices of Minimum m-Heights

March 18, 2025

1 Background

Let k and n be two positive integers, where $k < n$. Let

$$G = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}$$

be a ful-rank matrix of k rows and n columns, where each element $g_{i,j}$ is a real number. (Since here $k < n$, the rank of the matrix G is k. And we assume that no column of G is the all-zero vector, namely, for $j = 0, 1, \cdots, n-1$, $(g_{0,j}, g_{1,j}, \cdots, g_{k-1,j}) \neq (0, 0, \cdots, 0)$.) The matrix G is called *systematic* if its first k columns form an identity matrix. That is,

$$G = \begin{bmatrix} 1 & 0 & \cdots & 0 & g_{0,k} & g_{0,k+1} & \cdots & g_{0,n-1} \\ 0 & 1 & \cdots & 0 & g_{1,k} & g_{1,k+1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & g_{k-1,k} & g_{k-1,k+1} & \cdots & g_{k-1,n-1} \end{bmatrix}$$

In this case, we can use $I_{k\times k}$ to denote the $k \times k$ identity matrix, and use the matrix $P_{k,n-k}$ to denote the last $n-k$ columns of G, then we have

$$G = \left[I_{k\times k} \mid P_{k\times(n-k)} \right]$$

Example 1. *For example, when $k = 2$ and $n = 5$, the matrix G might be*

$$G = \begin{bmatrix} 1 & 0 & 0.4759809 & 0.9938236 & 0.819425 \\ 0 & 1 & -0.8960798 & -0.7442706 & 0.3345122 \end{bmatrix}$$

□

Let

$$\mathbf{x} = (x_0, x_1, \cdots, x_{k-1}) \in \mathbb{R}^k$$

be any vector of length k, where each of the k elements is a real number. (The symbol $\mathbb{R}$ denotes the set of all real numbers. And $\mathbb{R}^k$ denotes the k-dimensional real vector space, namely, the set of all real-valued vectors of length k.) When we multiple the vector $\mathbf{x}$ with the matrix G, we get a real-valued vector

$$\mathbf{c} = (c_0, c_1, \cdots, c_{n-1})$$

of length n:

$$\mathbf{c} = \mathbf{x}G = (x_0, x_1, \cdots, x_{k-1}) \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \in \mathbb{R}^n$$

For $i = 0, 1, \cdots, n-1$, we have

$$c_i = (x_0, x_1, \cdots, x_{k-1}) \cdot \begin{pmatrix} g_{0,i} \\ g_{1,i} \\ \vdots \\ g_{k-1,i} \end{pmatrix} = x_0 g_{0,i} + x_1 g_{1,i} + \cdots + x_{k-1} g_{k-1,i} = \sum_{j=0}^{k-1} x_j g_{j,i}$$

Example 2. *Let G be the matrix in the previous example, where $k = 2$ and $n = 5$. Let $\mathbf{x} = (-1.2, 3.8)$. Then the vector $\mathbf{c} = \mathbf{x}G$ is*

$$\mathbf{c} = (c_0, c_1, c_2, c_3, c_4) = (-1.2, 3.8, -3.97628032, -4.0208166, 0.28783636)$$

□

The vector $\mathbf{c} = \mathbf{x}G \in \mathbb{R}^n$ is called an *analog codeword*, or simply *codeword*. (The word *analog* means "real value" or "continuous value", instead of "discrete value".) The set of all such codewords

$$\mathcal{C} = \{\mathbf{x}G \mid \mathbf{x} \in \mathbb{R}^k\}$$

is called an *Analog Code*, or simply *code*. Note that the code $\mathcal{C}$ is a k-dimensional subspace inside an n-dimensional space. Since this code $\mathcal{C}$ is generated using the matrix G, G is called the *Generator Matrix* of the code $\mathcal{C}$.

Let the function

$$\pi \ : \ \{0, 1, \cdots, n-1\} \to \{0, 1, \cdots, n-1\}$$

be a permutation that maps every integer $i \in \{0, 1, \cdots, n-1\}$ to another integer $\pi(i) \in \{0, 1, \cdots, n-1\}$. A permutation is a bijection (a one-to-one mapping), which implies that no two integers are mapped to the same integer; in other words, for any $i \neq j$, we have $\pi(i) \neq \pi(j)$. Given a codeword $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1}) \in \mathcal{C}$ and a permutation π, we get a permutated vector

$$\mathbf{c}_\pi = (c_{\pi(0)}, c_{\pi(1)}, \cdots, c_{\pi(n-1)})$$

Note that $\mathbf{c}_\pi$ is not necessarily a codeword in $\mathcal{C}$.

Example 3. *Let the codeword be*

$$\mathbf{c} = (c_0, c_1, c_2, c_3, c_4) = (-1.2, 3.8, -3.97628032, -4.0208166, 0.28783636),$$

and let the permutation π *be*

$$\pi(0) = 3,\ \pi(1) = 2,\ \pi(2) = 1,\ \pi(3) = 0,\ \pi(4) = 4$$

Then

$$\mathbf{c}_\pi = (-4.0208166, -3.97628032, 3.8, -1.2, 0.28783636)$$

□

Given a codeword $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1}) \in \mathcal{C}$ and a permutation π, if

$$|c_{\pi(0)}| \geq |c_{\pi(1)}| \geq \cdots \geq |c_{\pi(n-1)}|$$

then π is called a *Ranking Permutation* of $\mathbf{c}$.

Example 4. *The permutation* π *in Example 3 is actually a* Ranking Permutation *of* $\mathbf{c}$*, because*

$$|-4.0208166| \geq |-3.97628032| \geq |3.8| \geq |-1.2| \geq |0.28783636|$$

□

Consider any integer

$$m \in \{0, 1, \cdots, n-1\}.$$

Given a codeword $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1}) \in \mathcal{C}$ and its *Ranking Permutation* π, the "m-height of $\mathbf{c}$" is defined as follows:

- If $\mathbf{c}$ is not the all-zero vector $(0, 0, \cdots, 0)$ and $c_{\pi(m)} \neq 0$, the "m-height of $\mathbf{c}$" is defined as
$$h_m(\mathbf{c}) = \left| \frac{c_{\pi(0)}}{c_{\pi(m)}} \right|$$
- If $\mathbf{c}$ is not the all-zero vector $(0, 0, \cdots, 0)$ and $c_{\pi(m)} = 0$, the "m-height of $\mathbf{c}$" is defined as $h_m(\mathbf{c}) = \infty$.
- If $\mathbf{c}$ is the all-zero vector $(0, 0, \cdots, 0)$, the "m-height of $\mathbf{c}$" is defined as $h_m(\mathbf{c}) = 0$.

We can see that given a codeword $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1}) \in \mathcal{C}$, its m-height sequence

$$h_0(\mathbf{c}),\ h_1(\mathbf{c}),\ \cdots,\ h_{n-1}(\mathbf{c})$$

is determined, and we have

$$h_0(\mathbf{c}) \ \leq \ h_1(\mathbf{c}) \ \leq \ \cdots \ \leq \ h_{n-1}(\mathbf{c}).$$

Example 5. *For the codeword* $\mathbf{c}$ *in Example 3, since*

$$|-4.0208166| \geq |-3.97628032| \geq |3.8| \geq |-1.2| \geq |0.28783636|$$

we have

$$\begin{aligned} h_0(\mathbf{c}) &= \left|\tfrac{-4.0208166}{-4.0208166}\right| = 1 \\ h_1(\mathbf{c}) &= \left|\tfrac{-4.0208166}{-3.97628032}\right| = 1.0112004879977878 \\ h_2(\mathbf{c}) &= \left|\tfrac{-4.0208166}{3.8}\right| = 1.0581096315789473 \\ h_3(\mathbf{c}) &= \left|\tfrac{-4.0208166}{-1.2}\right| = 3.3506805 \\ h_4(\mathbf{c}) &= \left|\tfrac{-4.0208166}{0.28783636}\right| = 13.969105918376677 \end{aligned}$$

□

For the analog code $\mathcal{C}$ (which consists of infinitely many codewords), its m-height is defined as

$$h_m(\mathcal{C}) = \max_{\mathbf{c}\in\mathcal{C}} h_m(\mathbf{c})$$

that is, the m-height of the code $\mathcal{C}$ is the maximum m-height of all its codewords.

Example 6. *Let* $\mathcal{C}$ *be the code in the previous examples, and let* $m = 2$. *In Example 5, it has been shown that for the codeword* $\mathbf{c} = (c_0, c_1, c_2, c_3, c_4) = (-1.2, 3.8, -3.97628032, -4.0208166, 0.28783636)$, *its 2-height is*

$$h_2(\mathbf{c}) = 1.0581096315789473.$$

So we know that

$$h_2(\mathcal{C}) \geq 1.0581096315789473$$

namely, $h_2(\mathbf{c})$ *(the 2-height of the codeword* $\mathbf{c}$*) gives us a lower bound to* $h_2(\mathcal{C})$ *(the 2-height of the code* $\mathcal{C}$*).*

There exist other codewords in $\mathcal{C}$ *that have greater 2-heights than* $\mathbf{c}$ *does. For example, for the codeword*

$$\begin{aligned} \mathbf{c}^* &= (-0.435, -1.924) \begin{bmatrix} 1 & 0 & 0.4759809 & 0.9938236 & 0.819425 \\ 0 & 1 & -0.8960798 & -0.7442706 & 0.3345122 \end{bmatrix} \\ &= (-0.435, -1.924, 1.5170058436999998, 0.9996633684, -1.0000513478) \end{aligned}$$

When we order the 5 elements of $\mathbf{c}^*$ *by their absolute values (from high to low), they become*

$$|-1.924| \geq |1.5170058436999998| \geq |-1.0000513478| \geq |0.9996633684| \geq |-0.435|$$

So the 2-height of the codeword $\mathbf{c}^*$ *is*

$$h_2(\mathbf{c}^*) = \left|\frac{-1.924}{-1.0000513478}\right| = 1.9239012119053513$$

It can be shown that the actual 2-height of the code is

$$h_2(\mathcal{C}) = 1.9242387$$

So the 2-height of $\mathbf{c}^*$ *is quite close to the maximum 2-height of all codewords.* □

Let

$$d(\mathcal{C})$$

be the smallest positive integer such that there exists a codeword $\mathbf{c} = (c_0, c_1, \cdots, c_{n-1}) \in \mathcal{C}$ that has exactly $d(\mathcal{C})$ non-zero numbers (out of its n numbers). That is, if π is its Ranking Permutation, then

$$|c_{\pi(0)}| \geq |c_{\pi(1)}| \geq \cdots \geq |c_{\pi(d(\mathcal{C})-1)}| > |c_{\pi(d(\mathcal{C}))}| = |c_{\pi(d(\mathcal{C})+1)}| = \cdots = |c_{\pi(n-1)}| = 0$$

So for that codeword $\mathbf{c}$, we have

$$1 = h_0(\mathbf{c}) \leq h_1(\mathbf{c}) \leq \cdots \leq h_{d(\mathcal{C})-1}(\mathbf{c}) < h_{d(\mathcal{C})}(\mathbf{c}) = h_{d(\mathcal{C})+1}(\mathbf{c}) = \cdots = h_{n-1}(\mathbf{c}) = \infty$$

Therefore for the Analog Code $\mathcal{C}$, we have

$$1 = h_0(\mathcal{C}) \leq h_1(\mathcal{C}) \leq \cdots \leq h_{d(\mathcal{C})-1}(\mathcal{C}) < h_{d(\mathcal{C})}(\mathcal{C}) = h_{d(\mathcal{C})+1}(\mathcal{C}) = \cdots = h_{n-1}(\mathcal{C}) = \infty$$

It is known that $d(\mathcal{C}) \leq n - k + 1$ (by a result known as the Singleton bound). Therefore even if we do not know what $d(\mathcal{C})$ is, we still know that

$$h_{n-k+1}(\mathcal{C}) = h_{n-k+2}(\mathcal{C}) = \cdots = h_{n-1}(\mathcal{C}) = \infty$$

2 Definition of the "m-Height Problem"

Given a $k \times n$ generator matrix G and a positive integer $m \leq n - k$, we would like to compute the m-height of the corresponding Analog Code $\mathcal{C}$. We call it the m-**Height Problem**. It is defined as follows.

m-Height Problem

Input: The problem has the following inputs:

1. Positive integers k, n and m, where $k < n$ and $m \leq n - k$.
2. A $k \times n$ real-valued full-rank generator matrix

$$G = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}$$

   where no column is the all-zero vector.

3. An Analog Code

$$\mathcal{C} = \{\mathbf{x}G \mid \mathbf{x} \in \mathbb{R}^k\}$$

Output: the m-height of $\mathcal{C}$.

3 LP-based Algorithm for m-Height Problem

In this section, we present an algorithm that solves the "m-height problem" based on linear programming (LP). The proof for the algorithm is shown in [Jia24].

Let $[n\rangle$ denote the integer set $\{0, 1, \cdots, n-1\}$, namely

$$[n\rangle = \{0, 1, \cdots, n-1\} \text{ for any positive integer } n.$$

Let $\Psi = \{-1, 1\}^m$ be the set of 2^m binary vectors of length m whose elements are either 1 or -1, namely

$$\Psi = \{(s_0, s_1, \cdots, s_{m-1}) \mid s_i \in \{-1, 1\} \text{ for } i \in [m\rangle\}.$$

Let (a, b, X, ψ) be a tuple where

$$a \in [n\rangle,$$

$$b \in [n\rangle \setminus \{a\},$$

$$X \subseteq [n\rangle \setminus \{a, b\},$$

$$|X| = m - 1,$$

and

$$\psi = (s_0, s_1, \cdots, s_{m-1}) \in \Psi.$$

Let Γ denote the set of all the

$$n(n-1)\binom{n-2}{m-1}2^m$$

such tuples.

Given a tuple $(a, b, X, \psi) \in \Gamma$, let $x_1, x_2, \cdots, x_{m-1}$ denote the $m-1$ integers in X such that

$$x_1 < x_2 < \cdots < x_{m-1}.$$

Define $Y \triangleq [n\rangle \setminus X \setminus \{a, b\}$, and let $x_{m+1}, x_{m+2}, \cdots, x_{n-1}$ denote the $n-m-1$ integers in Y such that

$$x_{m+1} < x_{m+2} < \cdots < x_{n-1}.$$

Let $x_0 = a$ and $x_m = b$. Then $x_0, x_1, \cdots, x_{n-1}$ are the n distinct integers in $[n\rangle$. Let τ denote the permutation on $[n\rangle$ such that $\tau(j) = x_j$ for $j \in [n\rangle$. We call τ the *quasi-sorted permutation given* (a, b, X, ψ).

Theorem 1. *Let*

$$m \in \{1, 2, \cdots, \min\{d(\mathcal{C}), n-1\}\}.$$

Let

$$(a, b, X, \psi) \in \Gamma,$$

where $\psi = (s_0, s_1, \cdots, s_{m-1})$. *Define*

$$Y \triangleq [n\rangle \setminus X \setminus \{a, b\},$$

and let τ be the quasi-sorted permutation given (a, b, X, ψ). Let

$$LP_{a,b,X,\psi}$$

denote the following linear program with k real-valued variables u_0, u_1, $\cdots$, u_{k-1}:

$$\begin{array}{lll} \textit{maximize} & \sum_{i\in[k\rangle}(s_0 g_{i,a}) \cdot u_i & \\ \textit{s.t.} & \sum_{i\in[k\rangle}(s_{\tau^{-1}(j)} g_{i,j} - s_0 g_{i,a}) \cdot u_i \leq 0 & \textit{for } j \in X \\ & \sum_{i\in[k\rangle}(-s_{\tau^{-1}(j)} g_{i,j}) \cdot u_i \leq -1 & \textit{for } j \in X \\ & \sum_{i\in[k\rangle} g_{i,b} \cdot u_i = 1 & \\ & \sum_{i\in[k\rangle} g_{i,j} \cdot u_i \leq 1 & \textit{for } j \in Y \\ & \sum_{i\in[k\rangle} -g_{i,j} \cdot u_i \leq 1 & \textit{for } j \in Y \end{array}$$

Let $z_{a,b,X,\psi}$ be the optimal objective value of $LP_{a,b,X,\psi}$ if it is bounded, let $z_{a,b,X,\psi} = \infty$ if the optimal objective value of $LP_{a,b,X,\psi}$ is unbounded, and let $z_{a,b,X,\psi} = 0$ if $LP_{a,b,X,\psi}$ is infeasible. Then

$$h_m(\mathcal{C}) = \max_{(a,b,X,\psi)\in\Gamma} z_{a,b,X,\psi}$$

□

The above theorem naturally leads to an algorithm that computes all the m-height of a code $\mathcal{C}$, for $m = 0, 1, \cdots, n-1$, and also discovers the minimum distance $d(\mathcal{C})$:

1. $h_0(\mathcal{C}) = 1$.
2. For $m = 1, 2, 3 \cdots$, compute

   $$h_m(\mathcal{C}) = \max_{a,b,X,\psi\in\Gamma} z_{a,b,X,\psi}$$

   by solving the linear programs $LP_{a,b,X,\psi}$ for all $(a, b, X, \psi) \in \Gamma$. Stop as soon as we meet the first value m^* such that $h_{m^*}(\mathcal{C}) = \infty$. We get $d(\mathcal{C}) = m^*$.
3. For $m = m^* + 1, m^* + 2, \cdots, n-1$, $h_m(\mathcal{C}) = \infty$.

The above algorithm solves $n(n-1)\binom{n-2}{m-1}2^m$ linear programs of k variables to compute an $h_m(\mathcal{C})$.

4 Your Task

Your task for the project is to design an algorithm and use it to find systematic generator matrices whose m-heights are as small as possible. The input to your algorithm is:

- Three positive integers n, k and m, where $k < n$ and $m \leq n - k$.

The output of your algorithm includes:

- A $k \times (n-k)$ matrix $P_{k\times(n-k)}$, which forms the last $n-k$ columns of a systematic $k \times n$ generator matrix $G = [I_{k\times k}|P_{k\times(n-k)}]$. Every number in $P_{k\times(n-k)}$ is **an integer between -100 and 100**. No column of $P_{k\times(n-k)}$ is the all-zero vector.
- The m-height of the corresponding Analog Code, whose systematic generator matrix is $G = [I_{k\times k}|P_{k\times(n-k)}]$.

You can design one algorithm or a mixture of different algorithms. The key is to find generator matrices of the smallest possible m-heights in the fastest possible way.

In this project, we will focus on the following values of n, k and m:

$$n \in \{9, 10\}, \quad k \in \{4, 5, 6\}, \quad m \in \{2, 3, \cdots, n-k\}$$

They include the following parameters:

1. $n = 9$, $k = 4$, $m = 2$.
2. $n = 9$, $k = 4$, $m = 3$.
3. $n = 9$, $k = 4$, $m = 4$.
4. $n = 9$, $k = 4$, $m = 5$.
5. $n = 9$, $k = 5$, $m = 2$.
6. $n = 9$, $k = 5$, $m = 3$.
7. $n = 9$, $k = 5$, $m = 4$.
8. $n = 9$, $k = 6$, $m = 2$.
9. $n = 9$, $k = 6$, $m = 3$.
10. $n = 10$, $k = 4$, $m = 2$.
11. $n = 10$, $k = 4$, $m = 3$.
12. $n = 10$, $k = 4$, $m = 4$.
13. $n = 10$, $k = 4$, $m = 5$.

14. $n = 10$, $k = 4$, $m = 6$.
15. $n = 10$, $k = 5$, $m = 2$.
16. $n = 10$, $k = 5$, $m = 3$.
17. $n = 10$, $k = 5$, $m = 4$.
18. $n = 10$, $k = 5$, $m = 5$.
19. $n = 10$, $k = 6$, $m = 2$.
20. $n = 10$, $k = 6$, $m = 3$.
21. $n = 10$, $k = 6$, $m = 4$.

So when you design your algorithm, you can just focus on the above parameters (instead of more general values).

What you should submit:

1. A file named

   "generatorMatrix-NAME-UIN-SESSION-SP25-P#"

   that contains a dictionary in Python. Here "NAME" is your name, "UIN" is your UIN, "SESSION" refers to the session you are in (such as "629-601" or "629-700"), "SP25" refers to the Spring 2025 semester, and "P#" is "P1" or "P2" or "P3" depending on if your submission is for Project 1, Project 2 or Project 3. (Please use pickle.dump() to save the dictionary in the file, so that we can use pickle.load() to read the dictionary from the file.) Each key in the dictionary should be the string version of the list of three integers $[n, k, m]$, such as $[9, 4, 2]$ and $[10, 6, 4]$. (Please use json.dumps() to turn a list of integers $[n, k, m]$ into a string, so that we can use json.loads() to turn the string back to the list.) Corresponding to each key $[n, k, m]$, the value in the dictionary is a $k \times (n - k)$ numpy array that stores the matrix $P_{k\times(n-k)}$.

   Note: every number in $P_{k\times(n-k)}$ should be **an integer between -100 and 100**, and no column of $P_{k\times(n-k)}$ can be the all-zero vector; *otherwise there will be no point for the submission.*

2. A file named

   "mHeight-NAME-UIN-SESSION-SP25-P#"

   that contains a dictionary in Python. (Please use pickle.dump() to save the dictionary in the file, so that we can use pickle.load() to read the dictionary from the file.) Each key in the dictionary should be the string version of the list of three integers $[n, k, m]$, such as $[9, 4, 2]$ and $[10, 6, 4]$. (Please use json.dumps() to turn a list of integers $[n, k, m]$ into a string, so that

we can use json.loads() to turn the string back to the list.) Corresponding to each key $[n, k, m]$, the value in the dictionary is a real number that is the m-height of your submitted generator matrix (corresponding to the matrix $P_{k\times(n-k)}$ in the previous dictionary).

Note: the m-height submitted above should be greater than or equal to 1; *otherwise there will be no point for the submission.*

3. A project report in PDF named

   "report-NAME-UIN-SESSION-SP25-P#.pdf"

   that clearly explains the main idea and the pseudo code of your algorithm, as well as the analysis of its time complexity. If your algorithm can provide any performance guarantee (e.g., it guarantees to minimize the m-height, or guarantees to find a generator matrix whose m-height is within an approximation ratio of the minimum m-height), please include the proof as well.

4. A file named

   "codes-NAME-UIN-SESSION-SP25-P#"

   that contains complete codes for your algorithm, with clear and detailed explanations (namely, documentation of your codes).

The performance of your solution will be measured as follows. Given a tuple (n, k, m) (such as $n = 9$, $k = 4$, $m = 2$), let $P_{k\times(n-k)}$ be your **submitted matrix**, let

$$y \in [1, \infty)$$

be the **true** m**-height** of the Analog Code whose generator matrix is $G = [I_{k\times k}|P_{k\times(n-k)}]$ (whose value we will compute), and let

$$\hat{y} \in [1, \infty)$$

be your **submitted** m**-height** (which hopefully agrees with y). The **score** of the submitted matrix $P_{k\times(n-k)}$ and its computed m-height value $\hat{y}$ depends on two factors:

- How good the generator matrix $G_{n,k,m} = [I_{k\times k}|P_{k\times(n-k)}]$ is: that is, how small the m-height of the Analog Code is.
- How accurate the submitted m-height $\hat{y}$ is: that is, how well $\hat{y}$ matches y.

To measure how good the generator matrix $G_{n,k,m}$ is, let $\theta_{n,k,m}$ and $\alpha_{n,k,m}$ be two positive real parameters. The **score** of your generator matrix $G_{n,k,m}$ is set as

$$\beta_{n,k,m} \triangleq 100 \cdot (e^{\alpha_{n,k,m}((\log_2 y)-\theta_{n,k,m})} + 1)^{-1} + 100 - 100 \cdot (e^{-\alpha_{n,k,m}\cdot\theta_{n,k,m}} + 1)^{-1}$$

The two parameters $\theta_{n,k,m}$ and $\alpha_{n,k,m}$ are chosen in the following way:

- In the first submission of the project, say that there are totally M generator matrices submitted for the parameter set (n, k, m), whose m-heights are sorted as
$$y^{(1)} \le y^{(2)} \le \cdots \le y^{(M)}$$
where $y^{(j)}$ is the actual m-height of the jth student's Analog Code for $j = 1, 2, \cdots, M$. Let $M' \le M$ be the greatest integer such that $y^{(M')}$ is finite (instead of infinity). Then
$$\theta_{n,k,m} = \log_2 y^{(\lceil M'/2 \rceil)}$$
and $\alpha_{n,k,m}$ is chosen to be the value such that
$$e^{\alpha_{n,k,m}(\log_2(y^{(\lceil M'/4 \rceil)}) - \theta_{n,k,m})} = \frac{1}{3}$$
namely,
$$\alpha_{n,k,m} = \frac{\ln 3}{\log_2 y^{(\lceil M'/2 \rceil)} - \log_2 y^{(\lceil M'/4 \rceil)}}$$

- In the followup submissions of the project (namely, the 2nd submission, the 3rd submissions, $\cdots$, if they exist), $\theta_{n,k,m}$ and $\alpha_{n,k,m}$ will be set to different values, based on the performance of previous submissions.

To measure how accurate the submitted m-height $\hat{y}$ is, we define
$$\sigma(y, \hat{y}) = \begin{cases} 1 & \text{if } \frac{|y-\hat{y}|}{\min\{y,\hat{y}\}} \ge 0.01 \\ 0 & \text{otherwise} \end{cases}$$
that is, $\sigma(y, \hat{y})$ is 1 if $\hat{y}$ differs from y by 1% or more.

The **score** of your submitted generator matrix and m-height for the parameter setting (n, k, m) is
$$\gamma_{n,k,m} \triangleq \beta_{n,k,m} - 5 \cdot \sigma(y, \hat{y}).$$

Let S denote the set of all values for the parameter set (n, k, m). (For example, with $n \in \{9, 10\}$, $k \in \{4, 5, 6\}$ and $m \in \{2, 3, \cdots, n-k\}$, the set S has 21 values for the parameter set (n, k, m).) Then the **overall score** of your submitted generator matrices and m-heights is set as
$$\text{SCORE}_{\text{matrices}} \triangleq \frac{1}{|S|} \sum_{(n,k,m) \in S} \gamma_{n,k,m} \in [-5, 100]$$
which is the average score over the different parameter settings in S.

The above score for generator matrices and m-heights accounts for 90% of the final grade of your submission. The other 10% is based on the project report and the codes, including (but not limited to) if the project report is clear and detailed, if the codes are complete and with good explanations, etc. Let
$$\text{SCORE}_{\text{report}} \in [0, 100]$$

denote the score for your project report and codes. Furthermore, if your algorithm is quite novel and efficient AND the project report explains your novel algorithm well (at the level of publishable papers for top academic conferences), there can be up to 30 points of bonus points. Let

$$\text{SCORE}_{\text{bonus}} \in [0, 30]$$

denote the bonus points. Then the **final grade** for your submission is

$$\text{SCORE}_{\text{matrices}} \times 90\% \ + \ \text{SCORE}_{\text{report}} \times 10\% \ + \ \text{SCORE}_{\text{bonus}}$$

References

[Jia24] Anxiao Jiang. Analog error-correcting codes: Designs and analysis. *IEEE Transactions on Information Theory*, 70(11):7740–7756, 2024.