Exploring Interleavers in Turbo Coding

California State Polytechnic University, Pomona

and

Loyola Marymount University

Department of Mathematics Technical Report

Benjamin Moreno[*], Laura Smith[†], Andrea Viteri[‡], Kouadio David Yao[§]

Applied Mathematical Sciences Summer Institute
Department of Mathematics & Statistics
California State Polytechnic University Pomona
3801 W. Temple Ave.
Pomona, CA 91768

Faculty Mentor: Dr. Edward Mosteig [¶]
Research Assistant: Alaina Jones[‖]

August 2005

[*]California State Polytechnic University, Pomona
[†]Western Washington University
[‡]The College of St. Catherine
[§]University of Arkansas at Little Rock
[¶]Loyola Marymount University, Los Angeles
[‖]Texas A&M University

Abstract

The primary aim of coding theory is the successful transmission of information across noisy channels. For half a century, coding theory has been used in a variety of applications such as communications, the design of computer memory systems, and compact discs. Our research focuses on a class of codes called turbo codes, which are currently used in deep-space and satellite communications. In particular, we examine one component of these codes called an interleaver; this component permutes data before transmission. We study properties of interleavers such as spread, dispersion, and cyclic decomposition. The project focuses on the effectiveness of turbo codes, examining how the abovementioned characteristics of interleavers affect the error rates. We use computer simulations to test our theoretical findings.

1 Turbo Codes

Turbo Coding is a topic within Coding Theory. Coding theory consists of successfully transmitting a message through a noisy channel. A message is a block of data that consists of bits. When this block of data is sent through the encoder, some redundancy occurs. Therefore, the length of the original message increases. While this new block of data is sent through the noisy channel, some of the bits might become corrupted. Once the decoder receives the message, it will have a greater chance to retrieve the original message due to the redundancy generated by the encoder. Figure 1 depicts the function of turbo codes, where M is the message, E is the encoder, I is an interleaver, and D is the decoder.

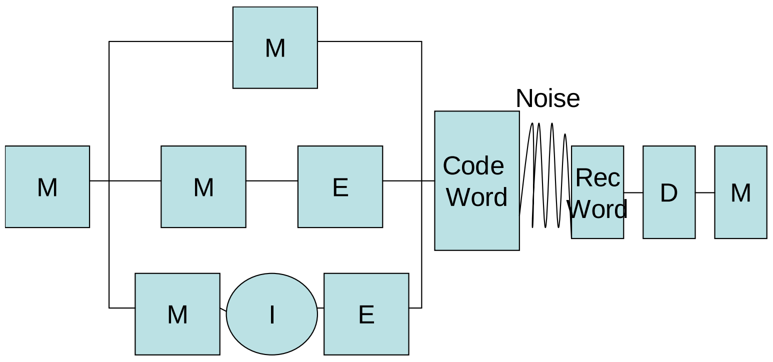


Figure 1: The Function of Turbo Codes

The process begins by making three copies of the message one wishes to communicate. The first copy remains untouched. The second is sent through an encoder that takes a message and produces another message. The third copy is sent through an interleaver, which rearranges the order of the elements, and is then sent through the same encoder as the second copy of the message. The type of encoder used in this project employs the use of shift registers. The best way to understand the function of shift registers is to look at an example.

Example 1. Let 0101011 be the message to be processed, and let the encoder be as shown in Figure 2.

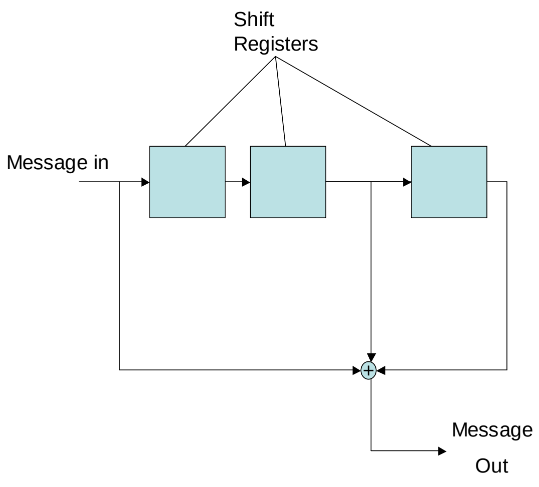


Figure 2: Example Encoder

Each shift register initially contains zero. Then the rightmost entry of the message enters the shift registers, pushing the contents of each register along the path of the arrows. Consequently, the shift registers will appear as in Figure 3.

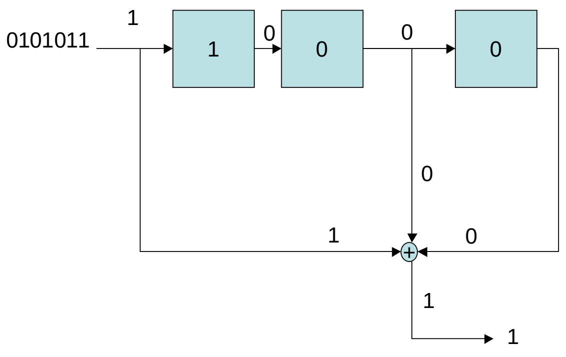


Figure 3: A Message in a Shift Register

Notice that the output is 1. The next entry then enters the registers and the process repeats. Once the entire message has entered, zeros follow until all shift registers have a zero in them.

Table 1 represents the input and output.

Input	Shift Register	Output
	000	
1	100	1
1	110	1
0	011	1
1	101	1
0	010	0
1	101	0
0	010	0
0	001	1
0	000	1

Table 1: Summary of Example 1

The encoded message is 111100011.

Some shift registers have feedback. This is where some of the entries in shift registers may be sent back to be incorporated into other shift register entries. Figure 4 is one example of an encoder whose shift registers have feedback.

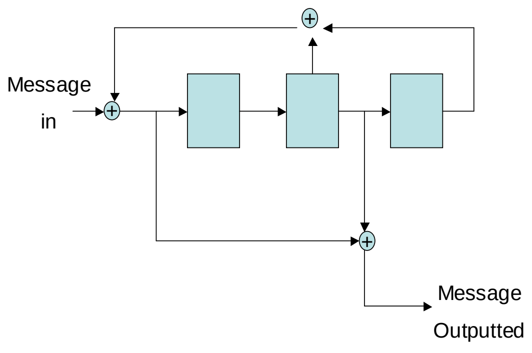


Figure 4: A Shift Register with Feedback

The shift registers can be thought of as polynomials, where the position of the element that is about to enter the first shift register can be considered as 1, the first shift register as D, the second shift register as D^2, and so forth. This is illustrated in Figure 5.

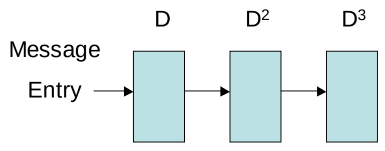


Figure 5: Shift Registers as Polynomials

Consequently, the encoder can be described by a rational function of the form

$$\frac{g_n(D)}{g_d(D)}.$$

Then $g_n(D)$ is the sum of the registers that provide the output. In example 2, $g_n(D) = 1 + D^2$. This is obtained by looking only at the register that provide the output while ignoring any feedback. In addition, $g_d(D)$ is the sum of the registers that produce feedback. In this case, $g_d(D) = 1 + D^2 + D^3$. Consequently, this encoder is $\frac{1+D^2}{1+D^2+D^3}$. This project utilized the following encoder:

$$\frac{1 + D + D^2}{1 + D^2}.$$

Once all three copies of the original message have been processed accordingly, they are combined to produce a codeword. In this project, the codeword takes all entries of the first copy of the message and every other entry of the altered second and third messages. Therefore, the code word that is sent is twice as long as the original message. The codeword is sent over a noisy channel and then processed by a decoder. The goal is to obtain the original message despite any corruption that might occur in transmission.

Example 2. A blocklength of 4 is chosen and the permutation π used is given by

$$\pi(v_0, v_1, v_2, v_3) = (v_0, v_3, v_2, v_1).$$

A block of data 1011 is taken and sent through the turbo code (refer to Figure 1). Three copies of the message are made. The first copy is left intact, and the second one is sent through the encoder in Figure 4. In this case, the message 1011 is sent through the encoder, and 1100 is the output. The third copy of the message is sent through the interleaver, where it is permuted, yielding 1110. This new message is then sent through the same encoder used for the second copy, producing another encoded block of data, 1110. In last step the three processed blocks of data are combined to give 11011010 which is the codeword that is sent through the noisy channel.

This project examines the effectiveness of interleavers. In order to determine which interleaver produces the best performance, various error rates that are run through simulations were examined. A good performance will occur when the error rates are small. Interleavers can be produced randomly or systematically. Random interleavers, however, use up much memory since they must be stored for decoding. For this reason, it is worthwhile to study systematic interleavers. Therefore, this project focuses on systematic interleavers, specifically those which utilize finite fields. Fields will be discussed in the next section.

2 Fields

The function of an interleaver is to permute blocks of data. Since an interleaver is a permutation of data, it is a bijection from one set to another. Consequently, finite fields can be used to construct interleavers.

Definition 3. Let $\mathbb{F}$ be a set with two operations defined on the set, denoted by "+" (addition) and "·" (multiplication). By definition, $\mathbb{F}$ is a field if it satisfies the following axioms for all $a, b, c \in \mathbb{F}$:

(i) $a \cdot b \in \mathbb{F}$, and $a + b \in \mathbb{F}$ (closure)

(ii) $a + b = b + a$, and $a \cdot b = b \cdot a$ (commutativity)

(iii) $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ (associativity)

(iv) $a + 0 = a,\ a \cdot 1 = a$, where $0, 1 \in \mathbb{F}$ (identity)

(v) For all $a \in \mathbb{F}$, there exists $-a \in \mathbb{F}$ such that $a + (-a) = 0$ (additive inverses)

(vi) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ (distributivity)

(vii) For every nonzero element $a \in \mathbb{F}$, there exists $b \in \mathbb{F}$ such that $a \cdot b = 1$, denoted as $b = a^{-1}$ (multiplicative inverses)

These axioms are satisfied for the common fields: the real numbers $\mathbb{R}$, the rational numbers $\mathbb{Q}$, and the complex numbers $\mathbb{C}$. There are other examples of fields that we introduce in the next section.

2.1 Modular Arithmetic on $\mathbb{Z}_q$

Let $\mathbb{Z}_k$ be the set of elements defined by $\mathbb{Z}_k = \{0, 1, ..., k - 1\}$.

An equivalence class on $\mathbb{Z}_k$ is defined as follows: $m \equiv n \mod k$ whenever k divides $m - n$. This is equivalent to saying n is the remainder when m is divided by k.

Example 4. Notice that $11 \equiv 4 \mod 7$ since $11 - 4 = 7$ and 7 divides 7.

Arithmetic over $\mathbb{Z}_k$ is defined in the following way. Given $a, b \in \mathbb{Z}_k$, addition is defined as $a + b = (a + b) \mod k$ and multiplication is defined as $a \cdot b = (a \cdot b) \mod k$.

Example 5. Consider $\mathbb{Z}_4 = \{0, 1, 2, 3\}$. Let $a = 2$ and $b = 3$. Here $2 + 3 = 5$, but $5 \equiv 1$ mod 4. Similarly, $2 \cdot 3 = 6$, but $6 \equiv 2 \mod 4$. Therefore, in $\mathbb{Z}_4$, one defines 2 + 3 =1 and 2 $\cdot$ 3 = 2.

2.2 Finite Fields

A **finite field** is defined to be a field with a finite number of elements.

Theorem 6. *The set $\mathbb{Z}_m$ is a field if and only if m is prime.*

For other cases, the following applies:

Theorem 7. *There exists a finite field with q elements if and only if $q = p^r$, where p is prime.*

Example 8. Notice that $8 = 2^3$, so there exists a finite field with 8 elements. There does not exist a finite field of 12 elements, however, because $12 = 3 \cdot 2^2$, which cannot be expressed in the form p^r.

Define $\mathbb{F}_q$ to be a finite field with q elements.

2.3 Irreducibility

The collection of all polynomials in indeterminate x with coefficients in the field $\mathbb{F}$ is denoted by $\mathbb{F}[x]$.

A polynomial $f(x)$ is said to be **reducible** over $\mathbb{F}$ if there exist nonconstant polynomials $g(x), h(x) \in \mathbb{F}[x]$ such that $f(x) = g(x)h(x)$. A polynomial is **irreducible** if it is not reducible.

Example 9. Note that $f(x) = (x^2 - 1) = (x + 1) \cdot (x - 1)$, which implies $f(x)$ is reducible over $\mathbb{R}$. However, $g(x) = (x^2 + 1) = (x + i) \cdot (x - i)$ is irreducible over $\mathbb{R}$ but reducible over $\mathbb{C}$.

Theorem 10. *If a polynomial of degree at most 3 has no roots, then it is irreducible.*

Proof. Let $P(x)$ be a reducible polynomial over $\mathbb{F}$ of degree at most 3. Then there exists $f(x), g(x)$, where $1 \leq \deg f(x), \deg g(x) \leq 2$ and $f(x) \cdot g(x) = P(x)$. It follows that $\deg f(x) + \deg g(x) = P(x) \leq 3$. Without loss of generality, assume that $\deg f(x) = 1$.

$$\text{Let} f(x) = \beta_1 x + \beta_0, \text{where } \beta_0, \beta_1 \in \mathbb{F}.$$

Note that $-\beta_1^{-1} \cdot \beta_0 \in \mathbb{F}$. Hence,

$$f(-\beta_1^{-1}\beta_0) = \beta_1(-\beta_1^{-1}\beta_0) + \beta_0 = -\beta_0 + \beta_0 = 0,$$

which implies

$$\begin{aligned} P(-\beta_1^{-1}\beta_0) &= f(-\beta_1^{-1}\beta_0) \cdot g(-\beta_1^{-1}\beta_0) \\ &= 0 \cdot g(-\beta_1^{-1}\beta_0) \\ &= 0. \end{aligned}$$

Thus, $-\beta_1^{-1}\beta_0$ is a root of $P(x)$. Therefore by the contrapositive, if a polynomial with a degree of at most 3 or less has no roots, then it is irreducible. □

Example 11. It can be shown that $P(x) = x^3 + x + 1$ is irreducible over $\mathbb{F}_2$. Evaluate $P(x)$ at $0, 1$. By substitution, $P(0) = 0^3 + 0 + 1 \not\equiv 0 \mod 2$. This implies that 0 is not a root of $P(x)$. Now, $P(1) = 1^3 + 1 + 1 = 3 \equiv 1 \not\equiv 0 \mod 2$. Then 1 is not a root of $P(x)$. Therefore, $P(x)$ has no roots in $\mathbb{F}_2$. By the above theorem, $P(x)$ is irreducible over $\mathbb{F}_2$.

Example 12. One can verify whether $f(x) = x^4 + x + 1$ is irreducible over $\mathbb{F}_2$. Begin by finding all the nonconstant polynomials with degree of at most 3. Suppose $f(x)$ an be factored as $f(x) = g(x)h(x)$, where $g(x)$ and $h(x)$ are nonconstant.
Since $\deg f(x) = 4,\ \deg g(x)h(x) = 4$, it follows that $0 < \deg\ g(x), \deg\ h(x) < 4$. Then the possible factors are the following:

$$x,\ x{+}1,\ x^2,\ x^2{+}1,\ x^2{+}x,\ x^2{+}x{+}1,\ x^3,\ x^3{+}1,\ x^3{+}x^2{+}1,\ x^3{+}x{+}1,\ x^3{+}x^2{+}x{+}1,\ x^3{+}x^2{+}x.$$

If $f(x) = g(x)h(x)$, then there is no need to test factors of degree 3. This is because testing for polynomials of degree 1 will account for both factors of degree 1 and degree 3. Furthermore, one needs to only test those factors that are not factorable themselves. For instance, if x^2 divides a polynomial, then x does too. Hence, it is only necessary to check the following factors:

$$x,\ x+1,\ x^2+x+1.$$

If none of these prospective factors divide $f(x)$, then $f(x)$ is irreducible. Dividing x^4+x+1 by x yields $x^3 + 1$ with remainder 1, meaning that x is not a divisor. In addition, dividing $x^4 + x + 1$ by $x + 1$ results in $x^3 + x^2 + x$ with remainder 1, meaning that $x + 1$ is not a divisor. One concludes that $f(x)$ is irreducible over $\mathbb{F}_2$

2.4 Finite Fields of the Form $\mathbb{F}_{p^r}$

Definition 13. Let $h(x)$ be an irreducible polynomial over $\mathbb{F}_p[x]$ and let $r = \deg\ h(x)$. The set $\mathbb{F}_p[x]/\langle h(x)\rangle$ is defined to be the collection of all polynomials over $\mathbb{F}_p$ of degree at most $r - 1$; that is,

$$\mathbb{F}_p[x]/\langle h(x)\rangle = \{\beta_{r-1}\alpha^{r-1} + \ldots + \beta_0 \mid \beta_i \in \mathbb{F}_p\},\ \text{where } \alpha \text{ is an indeterminate.}$$

Note that $h(x)$ does not affect the definition of $\mathbb{F}_p[x]/\langle h(x)\rangle$ other than the fact that it restricts the degree of the polynomials in the set. In the next section, $h(x)$ will be used to define additive and multiplicative structures on $\mathbb{F}_p[x]/\langle h(x)\rangle$.

For example, $\mathbb{F}_{2^3}[x]/\langle x^3 + x^2 + 1\rangle = \{0,\ 1,\ \alpha,\ \alpha + 1,\ \alpha^2,\ \alpha^2 + 1,\ \alpha^2 + \alpha,\ \alpha^2 + \alpha + 1\}$.

Theorem 14. *Given two irreducible polynomials of the same degree, $h_1(x),\ h_2(x) \in \mathbb{F}_p[x]$, $\mathbb{F}_p[x]/\langle h_1(x)\rangle$ and $\mathbb{F}_p[x]/\langle h_2(x)\rangle$ are isomorphic.*

Consequently, since $\mathbb{F}_p[x]/\langle h_1(x)\rangle$ and $\mathbb{F}_p[x]/\langle h_2(x)\rangle$ are isomorphic whenever $r = \deg\ h_1(x) = \deg\ h_2(x)$, the notation $\mathbb{F}_{p^r}$ is used for all isomorphic copies of this field.

2.5 Addition and Multiplication in Finite Fields

The addition of elements in $\mathbb{F}_{p^r}$ is straightforward. Adding in $\mathbb{F}_{p^r}$ requires modular arithmetic to reduce resulting coefficients to elements in the field.

Example 15. Let $\alpha^3 + \alpha^2 + 1$ and $\alpha^2 + \alpha + 1$ be elements of $\mathbb{F}_{2^4} = \mathbb{F}_2[x]/\langle x^4 + x^3 + 1\rangle$. In this case $p = 2$. Adding these two elements together results in $\alpha^3 + 2\alpha^2 + \alpha + 2$. But $2 \equiv 0 \mod 2$. Therefore the final result is $\alpha^3 + \alpha$.

If two given polynomials in c are multiplied together in the usual manner, the resulting degree might be larger than deg $h(x) - 1$, in which case $\mathbb{F}_{p^r} = \mathbb{F}_p[x]/\langle h(x)\rangle$ is not closed under multiplication. Therefore, multiplication must be defined in a different manner.

Given $f(\alpha)$, $g(\alpha) \in \mathbb{F}_p[x]/\langle h(x)\rangle$, divide $f(\alpha) \cdot g(\alpha)$ by $h(\alpha)$ to get a quotient and remainder:

$$f(\alpha) \cdot g(\alpha) = q(\alpha) \cdot h(\alpha) + r(\alpha)$$

where $r(\alpha) = 0$ or deg $r(\alpha) <$ deg $h(\alpha)$. In $\mathbb{F}_p[x]/\langle h(x)\rangle$, $f(\alpha) \cdot g(\alpha)$ is defined to be the remainder $r(\alpha)$.

Example 16. Consider $\mathbb{F}_2[x]/\langle h(x)\rangle$ where $h(x) = x^4 + x^3 + 1$. Then

$$\begin{aligned}(\alpha^3 + \alpha + 1) \cdot (\alpha^2 + 1) &= (\alpha^5 + \alpha^2 + \alpha + 1) \\ &= (\alpha + 1) \cdot (\alpha^4 + \alpha^3 + 1) + (\alpha^3 + \alpha^2).\end{aligned}$$

Thus, the product $(\alpha^3 + \alpha + 1) \cdot (\alpha^2 + 1)$ is defined to be $(\alpha^3 + \alpha^2)$.

Theorem 17. *Over $\mathbb{F}_p$, all irreducible polynomials of degree r divide $x^{p^r-1} - 1$*

To find all the irreducible polynomials of degree r, factor $x^{p^r-1} - 1$.

2.6 Generating a field $\mathbb{F}_{p^r}$ with α

It is possible to generate the nonzero elements of $\mathbb{F}_{p^r}$ using the powers of an element α.

Definition 18. An irreducible polynomial $h(x) \in \mathbb{F}_p[x]$ of degree r is called primitive if $\mathbb{F}_p[x]/\langle h(x)\rangle = \{0,\ 1,\ \alpha,\ \alpha^2,\ \ldots,\ \alpha^{p^r-2}\}$.

Theorem 19. *For every prime p and positive integer r, there exists a primitive polynomial $h(x) \in \mathbb{F}_p[x]$ of degree r.*

Example 20. Consider $\mathbb{F}_8 = \mathbb{F}_{2^3}$ and $h(\alpha) = \alpha^3 + \alpha^2 + 1$. The members of this field are as follows:

$$\begin{aligned}
0 &= 0 \\
\alpha^0 &= 1 \\
\alpha^1 &= \alpha^1 \\
\alpha^2 &= \alpha^2 \\
\alpha^3 &= \alpha^2 + 1 \\
\alpha^4 &= \alpha^2 + \alpha + 1 \\
\alpha^5 &= \alpha + 1 \\
\alpha^6 &= \alpha^2 + \alpha \\
\alpha^7 &= 1.
\end{aligned}$$

Therefore,

$$\mathbb{F}_8 = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\},$$

which can also be expressed as

$$\mathbb{F}_8 = \{0, 1, \alpha, \alpha^2, \alpha^2 + 1, \alpha^2 + \alpha + 1, \alpha + 1, \alpha^2 + \alpha\}.$$

3 Permutations

In order to construct these permutations, finite field are used. In the process of examining finite fields, it was natural to take a look at sets of the form $\mathbb{Z}_q$ because they sometimes are finite fields. It was also of interest to find out any condition between the set $\mathbb{Z}_q$ and the monomial x^i that could be used to create the permutations.

3.1 Theorems of $\mathbb{Z}_q$

Theorem 21 (The Prime Factorization Theorem)**.** *An integer $n \geq 2$ can be written uniquely in the form $n = \prod p_i^{\alpha_i}$ where the p_i are the distinct prime divisors of n, and $\alpha_i \geq 1$.*

For example, 18 has prime factorization $2 \cdot 3^2$.

Definition 22. If n has only one prime divisor, then it is called a **prime power**. On the other hand, if all of the exponents n_i are 1, then n is **square-free**.

Some prime powers are $25 = 5^2, 49 = 7^2, 27 = 3^3$. An example of a square-free number is $210 = 2 \cdot 3 \cdot 5 \cdot 7$.

Theorem 23 (Part 1)**.** *If q is not square-free, then $\pi : \mathbb{Z}_q \to \mathbb{Z}_q$ given by $\pi(x) = x^i$ is a permutation if and only if $i = 1$.*

Proof. Suppose q is not square-free. Then there exists positive $n, m \in \mathbb{Z}$ where $n > 1$, such that $q = n^2m$. Note that $1 < nm < q$. Then $\pi(nm) = (nm)^i, i \in \mathbb{Z}$ where $i > 1$. Suppose $i > 2$. Then

$$\begin{aligned}
(nm)^i &= (nm)^2 \cdot (nm)^{i-2} = n^2 \cdot m^2 \cdot (nm)^{i-2}. \\
&= (n^2 \cdot m) \cdot [m \cdot (nm)^{i-2}] = q[m \cdot (nm)^{i-2}].
\end{aligned}$$

But $q \equiv 0 \mod q$ implies that $(nm)^i \equiv 0 \cdot [m \cdot (nm)^{i-2}] \equiv 0$. Since $\pi(0) \equiv 0$ and $\pi(nm) \equiv 0$, it follows that π is not one-to-one. Thus π is not a permutation if $i \geq 2$. Hence, $\pi(x) = x^i$ is a permuation if and only if $i = 1$. $\square$

Definition 24. The number of invertible elements in $\mathbb{Z}_q$ is denoted by $\mid \mathbb{Z}_q^* \mid$ where $q \geq 2$. This is equivalent to the Euler-phi function, which is defined as

$$\phi(q) = \mid \{x \in \mathbb{Z} | \gcd(q, x) = 1,\ 1 \leq x < q\} \mid.$$

Furthermore, $\phi(q) = (p_1^{n_1} - p_1^{n_1-1})(p_2^{n_2} - p_2^{n_2-1}) \ldots (p_k^{n_k} - p_k^{n_k-1})$, where $m = p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}$. Note that if q is square-free, then $q = p_1 p_2 \ldots p_k$, where p_i are distinct primes. Therefore,

$$\phi(q) = (p_1 - 1)(p_2 - 1) \ldots (p_k - 1) = \prod_{i=1}^{n} (p_i - 1).$$

Example 25. Observe that $\phi(12) =\mid \{1, 5, 7, 11\} \mid = 4$.

Theorem 26. *If q is square-free, then $a^{\phi(q)+1} \equiv a$ mod q*

Lemma 27. *Let $q = p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}$, where p_i are distinct primes and n_i are integers. Then there is an isomorphic copy of $\mathbb{F}_p$ inside $\mathbb{Z}_q$. In particular, if p_i divides q, the elements $\{0, (\frac{q}{p_i}), 2(\frac{q}{p_i}), \ldots, (p_i - 1)(\frac{q}{p_i})\}$ are contained in $\mathbb{Z}_q$. These elements form a finite field that is isomorphic to $\mathbb{F}_{p_i}$.*

Theorem 28 (part 2)**.** *Let q be square-free. Then $\pi : \mathbb{Z}_q \to \mathbb{Z}_q$ given by $\pi(x) = x^i$ is a permutation if and only if gcd$(i, \phi(q)) = 1$.*

Proof. Begin by considering the contrapositive of the implication that $x \mapsto x^i$ is a permutation of $\mathbb{Z}_q$ if $\gcd(i, \phi(q)) = 1$. Suppose $\gcd(i, \phi(q)) \neq 1$. Suppose i is prime. Since $\gcd(i, \phi(q)) \neq 1$, it follows that $i \mid \phi(q)$, which implies $i \mid \prod_{j=1}^{n} (p_{j-1})$. Thus, there exists at least one integer k such that $1 \leq k \leq n$ and $i \mid (p_k - 1)$. By Lemma 27, $\mathbb{F}_{p_k}$ is embedded in $\mathbb{Z}_q$. Since $\mathbb{F}_{p_k}$ is a finite field, it follows that $x \mapsto x^i$ is a permutation of $\mathbb{F}_{p_k}$ if and only if $\gcd(i, (p_k - 1)) = 1$. However, since $i | (p_k - 1)$, $\gcd(i, (p_k - 1)) \neq 1$. Thus, $x \mapsto x^i$ is not a permutation of $\mathbb{F}_{p_k}$ and is therefore not a permutation of $\mathbb{Z}_q$.

This can be generalized for any i not necessarily prime such that $\gcd(i, \phi(q)) \neq 1$. For any positive integer an i there exists an integer $m \geq 1$ such that $i = mp$ where p is a prime and $p \mid \phi(q)$. Therefore, there exists an integer s such that $\mathbb{F}_{p_s}$ is embedded in $\mathbb{Z}_q$, $1 \leq s \leq n$, and $p \mid (p_s - 1)$. This implies $\gcd(p, (p_s - 1)) \neq 1$. Consequently, $x \mapsto x^p$ is not a permutation of $\mathbb{F}_{p_s}$, and is therefore not one-to-one or onto. Hence, at least two elements, w_1 and w_2, of $\mathbb{F}_{p_s}$ map to the same element in $\mathbb{F}_{p_s}$. Thus, $x \mapsto x^p$ is not a bijection of $\mathbb{Z}_q$, and therefore not a permutation. Now consider $x \mapsto x^{mp} = x^i$. Let $(w_1)^p = (w_2)^p = v$. Then

$$\begin{aligned} (w_1)^i &= ((w_1)^p)^m = v^m \\ (w_2)^i &= ((w_2)^p)^m = v^m. \end{aligned}$$

These two elements are mapped to the same element, and therefore $x \mapsto x^i$ is not a bijection, and hence not a permutation of $\mathbb{Z}_q$.

Now, consider the other direction. Let $x \in \mathbb{Z}_q$. It will be shown that x has a pre-image under π, thus showing π is onto. Let $q = p_1 p_2 \ldots p_k$, where the p_j are distinct primes and suppose $\gcd(i, \phi(q)) = 1$. Then there exists $a, b \in \mathbb{Z}, a, b > 0$ such that $ai - b\phi(q) = 1$, which implies $ai = 1 + b\phi(q)$. Then $x^{ai} = x^{1+b\phi(q)}$, which implies that $x^{b-1} \cdot x^{ai} = x^{b-1} \cdot x^{1+b\phi(q)}$. This then implies that $x^{b-1}x^{ai} = (x^b)^{1+\phi(q)}$. However, by Theorem 26 $(x^b)^{1+\phi(q)} \equiv x^b$. Consequently, $x^{b-1}x^{ai} - x^b \equiv 0$. This implies that $x^{b-1}(x^{ai} - x) \equiv 0$.

Without loss of generality, we can rearrange the primes $p_1, \ldots,\ p_k$ so that $p_1, p_2, \ldots, p_\alpha \mid x$ and $p_{\alpha+1}, p_{\alpha+2}, \ldots, p_k \nmid x$ for some index α. Since $x^{b-1}(x^{ai} - x) \equiv 0$, it follows that $q \mid x^{b-1}(x^{ai} - x)$. Since each p_j is prime, $p_1 p_2 \cdots p_\alpha \mid x$, and so $p_{\alpha+1}, p_{\alpha+2}, \cdots p_k \mid (x^{ai} - x)$. However, $x^{ai} - x \equiv x \cdot (x^{ai-1} - 1)$. Thus, $p_1, p_2, \ldots, p_\alpha \mid x$ and $p_{\alpha+1}, p_{\alpha+2}, \cdots p_k \mid (x^{ai-1} - 1)$. Consequently, $p_1, p_2, \ldots, p_\alpha \mid x \cdot (x^{ai-1} - 1)$. This implies $q \mid (x^{ai} - x)$. Hence, $(x^{ai} - x) \equiv 0$, which implies $x^{ai} \equiv x$. Thus $\pi(x^a) = x^{ai} \equiv x$, and so π is onto. Since $\mathbb{Z}_q$ has a finite number of elements, π is also one-to-one, and therefore a bijection. Thus π is a permutation. $\square$

4 Monomial Orderings

Constructing permutations from $\mathbb{Z}_q$ to $\mathbb{Z}_q$ using finite fields requires the use of a construct called monomial orders. Let S be a set. A **total order** on S is a binary relation "$<$" where the following hold for all $\alpha,\ \beta,\ \gamma \in \mathbb{Z}^n$.

(i) For all $\alpha \neq \beta \in \mathbb{Z}^n, \alpha < \beta$ or $\beta < \alpha$.

(ii) If $\alpha < \beta$ and $\beta < \gamma$, then $\alpha < \gamma$.

(iii) $\alpha \not< \alpha$.

A **monomial ordering** on $\mathbb{Z}^n$ is a total order " $<$ " such that

a. For all $\alpha \in \mathbb{N}^n, \alpha \geq \overrightarrow{0}$.

b. $\alpha > \beta \implies \alpha + \gamma > \beta + \gamma$.

Two types of monomial orders are the Lexicographic and Graded Lexicographic Orders.

4.1 Lexicographic Order

Let $\overrightarrow{a} = (a_1, a_2, \ldots, a_n),\ \overrightarrow{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{Z}^n$.

Comparisons under the lexicographic order are given by the following rule: $\overrightarrow{a} < \overrightarrow{b}$ if and only if the first nonzero entry of $\overrightarrow{b} - \overrightarrow{a}$ is positive.

4.2 Graded Lexicographic Order

Let $\overrightarrow{a} = (a_1, a_2, \ldots, a_n),\ \overrightarrow{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{Z}^n$.

Comparisons under the graded lexicographic order are given by the following rule: $\overrightarrow{a} < \overrightarrow{b}$ if and only if $\sum_{i=1}^n\ a_i < \sum_{i=1}^n\ b_i$ or $\sum_{i=1}^n\ a_i = \sum_{i=1}^n\ b_i$ and the first nonzero entry of $\overrightarrow{b} - \overrightarrow{a}$ is positive.

Example 29. Let $\overrightarrow{a} = (1,1,0)$, $\overrightarrow{b} = (0,1,1)$, and $\overrightarrow{c} = (1,0,0)$. For the lexicographic order, $\overrightarrow{c} - \overrightarrow{b} = (1,-1,-1)$ which implies $\overrightarrow{b} < \overrightarrow{c}$. Next, $\overrightarrow{a} - \overrightarrow{c} = (0,1,0)$ means that $\overrightarrow{c} < \overrightarrow{a}$. One concludes that $\overrightarrow{b} < \overrightarrow{c} < \overrightarrow{a}$. For the graded lexicographic order, $\sum_{i=1}^{3} a_i = 2$, $\sum_{i=1}^{3} b_i = 2$, $\sum_{i=1}^{3} c_i = 1$. Then $\overrightarrow{c} < \overrightarrow{a}$ and $\overrightarrow{c} < \overrightarrow{b}$. Also $\overrightarrow{a} - \overrightarrow{b} = (1,0,-1)$, which indicates $\overrightarrow{b} < \overrightarrow{a}$. Hence, $\overrightarrow{c} < \overrightarrow{b} < \overrightarrow{a}$.

Theorem 30. *Let M be an n by n invertible matrix with nonnegative, real entries. For $\alpha, \beta \in \mathbb{N}^n$, define $\alpha < \beta$ if and only if the leftmost nonzero component of $M(\beta - \alpha)$ is positive. Then, this order is a monomial order.*

Proof. First one must show that this is a total order.

(i) Suppose $M(\beta - \alpha) = \overrightarrow{0}$. From this, it follows that

$$\begin{aligned}(\beta - \alpha) &= I_n \cdot (\beta - \alpha) \\ &= M^{-1} \cdot M \cdot (\beta - \alpha) \\ &= M^{-1} \cdot \overrightarrow{0} \\ &= \overrightarrow{0}\end{aligned}$$

Thus, if $\beta \neq \alpha$, then $M(\beta - \alpha) \neq \overrightarrow{0}$

(ii) Let $\alpha < \beta$ and $\beta < \gamma$. Then $M(\gamma - \alpha) = M(\gamma - \beta + \beta - \alpha) = M[(\gamma - \beta) + (\beta - \alpha)] = M(\gamma - \beta) + M(\beta - \alpha)$. Let $i > 0$ be the first nonzero component of $M(\gamma - \beta)$ and let $j > 0$ be the first nonzero component of $M(\beta - \alpha)$. Hence, one of the following occurs:

(a) $M(\gamma - \alpha) = [0, 0, \ldots, 0, i + j, \ldots]$.

(b) $M(\gamma - \alpha) = [0, \ldots, i + 0, \ldots, x + j, \ldots]$ where x is an element of $M(\gamma - \beta)$.

(c) $M(\gamma - \alpha) = [0, \ldots, 0, 0 + j, \ldots, i + y, \ldots]$ where y is an element of $M(\beta - \alpha)$.

In any of the above cases, the first nonzero element of $M(\gamma - \alpha)$ is positive. Therefore, $\alpha < \gamma$.

(iii) Now, $M(\alpha - \alpha) = M(\overrightarrow{0}) = \overrightarrow{0}$. All of the components of $\overrightarrow{0}$ are zero, and so $\alpha \not< \alpha$ Thus, this is a total order.

(a) Let $\alpha \in \mathbb{N}^n$. The i^{th} entry of $M(\alpha - \overrightarrow{0}) = \sum_{j=1}^{n} (M)_{ij}\alpha_j$. But since the entries of M and α are nonnegative, $\sum_{j=1}^{n} (M)_{ij}\alpha_j \geq 0$. If $\alpha \neq \overrightarrow{0}$, then the first nonzero entry is positive. Thus $\alpha \geq \overrightarrow{0}$.

(b) Let $\alpha < \beta$. Then the first nonzero entry of $M(\beta - \alpha)$ is positive. But $M(\beta - \alpha) = M[(\beta + \gamma) - (\gamma + \alpha)]$, and so the first nonzero entry of this must also be positive. Thus, $(\gamma + \alpha) < (\beta + \gamma)$.

Consequently, this order is a monomial order. □

5 Creating Permutations using Finite Fields

Simulations are used to test the different interleavers. See section 7 on simulations for more information. To construct the interleavers that are used in the simulations, a computer program was developed that produces a monomial permutation $\pi : \mathbb{Z}_{p^r} \to \mathbb{Z}_{p^r}$.

$$\mathbb{Z}_{p^r} \xrightarrow{\pi_1} (\mathbb{Z}_p)^r \xrightarrow{\pi_2} \mathbb{F}_{p^r} \xrightarrow{\pi_3} \mathbb{F}_{p^r} \xrightarrow{\pi_2^{-1}} (\mathbb{Z}_p)^r \xrightarrow{\pi_1^{-1}} \mathbb{Z}_{p^r} \tag{1}$$

The function π_1 uses a monomial ordering to order the vectors of $(\mathbb{Z}_p)^r$. Recall that a monomial order is a total order on $\mathbb{N}^r$; since $\mathbb{Z}_p \subset \mathbb{N}$ as sets, $(\mathbb{Z}_P)^r$ inherits the order structure on $\mathbb{N}^r$. Order the vectors of $(\mathbb{Z}_P)^r$ from smallest to largest according to this monomial order. That is, write $(\mathbb{Z}_P)^r = \{v_0, \ldots, v_{p^r-1}\}$ where $v_p \leq v_{i+1}$. Then π_1 is defined by $\pi_1(i) = v_i$.

The function $\pi_2 : (\mathbb{Z}_p)^r \to \mathbb{F}_{p^r}$ is given by $(\beta_{r-1}, \ldots, \beta_0) \mapsto \sum_{i=0}^{r-1} \beta_i \alpha^i$.

The function $\pi_3 : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$ is given by $x \mapsto x^i$.

6 Interleavers

Interleavers are functions that permute data. That is, an **interleaver** is a bijection π that maps $\mathbb{Z}_q$ onto $\mathbb{Z}_q$.

Example 31. The following is a permutation and can thus be used as an interleaver:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 1 & 4 & 6 & 3 & 2 & 5 \end{pmatrix}.$$

Dispersion, spread, and cyclic decomposition are the properties of interleavers that this project examines. This project investigates the relationship between these characteristics and the interleaver's performance in turbo coding.

6.1 Dispersion

Given a permutation π, one wishes to consider the distance between two values in $\mathbb{Z}_q$ and the distance between the images of these values under the given permutation. A high dispersion indicates a variety of the permutation distances of the elements. In order to compute the dispersion, the list of differences must first be calculated.

Definition 32. Given a permutation π of $\mathbb{Z}_q$, the list of differences of π is defined to be the set

$$D(\pi) = \{(j - i, \pi(j) - \pi(i)) \mid 0 \leq i < j < q\}.$$

Example 33. To list out the set of differences one starts by comparing the difference between elements in $\mathbb{Z}_q$ and the difference of their permutations. Looking at the first two entries in $\mathbb{Z}_q$, 0 and 1, one notices they have a difference of one, and that the difference between their permutations is 1. Thus, the first element of the set is $(1, 1)$. Similarly, between 1 and 2, the difference between them is one while the difference between their permutations is three. Then the second element in the set of differences is $(1, 3)$. Continuing in this manner, the permutation in Example 31 has the following list of differences:

$$D(\pi) = \left\{ \begin{array}{llllll} (1, 1) & (1, 3) & (1, 2) & (1, -3) & (1, -1) & (1, 3) \\ (2, 4) & (2, 5) & (2, -1) & (2, -4) & (2, 2) & \\ (3, 6) & (3, 2) & (3, -2) & (3, -1) & & \\ (4, 3) & (4, 1) & (4, 1) & & & \\ (5, 2) & (5, 4) & & & & \\ (6, 5) & & & & & \end{array} \right\}$$

Because $D(\pi)$ is a set, and (1, 3) and (4, 1) are repeated, there are 19 elements total. In other words, $|D(\pi)| = 19$.

Definition 34. Given a permutation π, the **dispersion** of π is given by

$$\frac{|D(\pi)|}{\binom{q}{2}} = \frac{2 \cdot |D(\pi)|}{q \cdot (q-1)}.$$

In example 33, $q = 7$. Therefore, the dispersion is

$$\frac{(2 \cdot 19)}{(7 \cdot 6)} = \frac{19}{21} \approx 0.905$$

.

Theorem 35. *For any permutation π, the dispersion is at most 1. Moreover, there exists a permutation with dispersion exactly 1 (see [HeSt]).*

The maximum dispersion is 1, and so the closer this value is to 1, the more disperse the permutation.

6.2 Spread

Given a permutation π, one wishes to consider the distance between two values in $\mathbb{Z}_q$ and the distance between the images of these values under the given permutation. Loosely speaking, spread measures how these two distances compare. If the images typically have larger distances between them, then the spread is larger. **Spread** is the maximum value of s such that

$$|i - j| < s \text{ implies } |\pi(i) - \pi(j)| \geq s, \text{ for } 1 \leq s \leq q. \qquad [\heartsuit]$$

Suppose $s = 1$. By looking at the list of differences, it is obvious that there are no differences where the first term is less than 1. Therefore, there does not exist an $i \neq j$ such that $|i-j| < 1$ is true. Thus, the antecedent of $\heartsuit$ itself is always false. Consequently, the implication is always true. Thus, the minimum spread is 1. Once a value for s fails to satisfy $\heartsuit$, then there is no need to check for higher values of s. This is because the ordered pair of the list of differences that failed for s will also fail for $s + 1$. The best way to understand spread is to look at an example.

Example 36. For the example 31, let $s = 2$. Considering the element $(1, 1)$ of the list of differences, it is evident that this fails the implication $\heartsuit$. This is because $1 < 2$, yet $1 \not\geq 2$. Therefore the maximum value of s is 1. This is the spread.

Theorem 37. *For any permutation π, the spread is at most $\frac{\sqrt{q}}{2}$*

6.3 Cyclic Decomposition

A permutation can be divided into cycles. A **cycle** is an ordered list of elements $x_0, x_1, \ldots, x_n$ such that

$$\begin{aligned} \pi(x_0) &= x_1 \\ \pi(x_1) &= x_2 \\ &\vdots \\ \pi(x_{n-1}) &= x_n \\ \pi(x_n) &= x_0. \end{aligned}$$

Thus the elements make a cycle: $x_0 \to x_1 \to x_2 \to \ldots \to x_n \to x_0$. This cycle is written as $(x_0, x_1, \ldots, x_n)$. Since it has $(n + 1)$ elements, it is called a $(n + 1)$-cycle.

Example 38. Consider the permutation

$$\pi = \left(\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 1 & 4 & 6 & 3 & 2 & 5 \end{array} \right).$$

Notice $0 \to 0$ which implies π contains the cycle (0). Next, $1 \to 1$ which implies π contains (1). Now, $2 \to 4 \to 3 \to 6 \to 5 \to 2$ which implies π contains the cycle (2 4 3 6 5). Therefore, the cyclic decomposition of π is

$$(0)(1)(2\ 4\ 3\ 6\ 5),$$

with two 1-cycles and one 5-cycle.

7 Simulations

In order to test the performance of different interleavers, the permutations of the data are sent through turbo code simulations on Matlab. This project utilized Yufei Wu's turbo simulation code (see [Wu]). A few settings must be made before the simulator can be run.

A frame is the block length of the interleaver. For this project the number of frames in error was sixty. This means a series of codeword's (frames) were sent over a noisy channel until a total of sixty frames could not be decoded properly.

After a code word has been sent across the noisy channel, the received word is sent through a decoder. Within this particular decoder, there are two separate decoding entities. Each one takes half of the bits of the received word. Based on the information they have, they attempt to reconstruct the original code word. Each swaps its reconstruction with the other decoder and then proceeds to make another attempt at producing the original code word by incorporating the other decoder's suggestion into its decision. The information is then swapped again, and the decoders repeat the process. The number of times this process is repeated is the number of iterations. For this project twenty iterations were used.

The **signal to noise ratio (SNR)** is the ratio of the signal strength to the strength of the noise. Since corruption of bits is a problem, the SNR is varied to test to see how well the decoder can determine the original message over a range of noise levels. The SNR levels used in this project were 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, and 2.0.

The Matlab simulation returns two values, the **Bit Error Rate (BER)** and the **Frame Error Rate (FER)**. A codeword consists of components known as bits. The BER is the ratio of the number of corrupted bits that cannot be corrected to the total number of bits transmitted. The FER is the ratio of the number of corrupted frames that cannot be corrected to the total number of frames sent.

8 Results

By running interleavers of various lengths in simulations, several topics were studied. Noting that monomial permutations that contain the elements 0 and 1 will always have a spread of 1, a goal of the project became to find and study interleavers with higher spreads. Consequently, the performance of interleavers with and without the 0 element were compared. Using the interleavers without the 0 element, several comparisons were made. The project examined fields generated from various irreducible polynomials and ordered with different monomial orderings. In addition, the interleaver properties of spread, dispersion, and cyclic decomposition were studied. Finally, interleavers that were constructed by using finite fields were compared to both randomly constructed interleavers and interleavers that were created using a monomial permutation of $\mathbb{Z}_q$.

8.1 Removing Zero

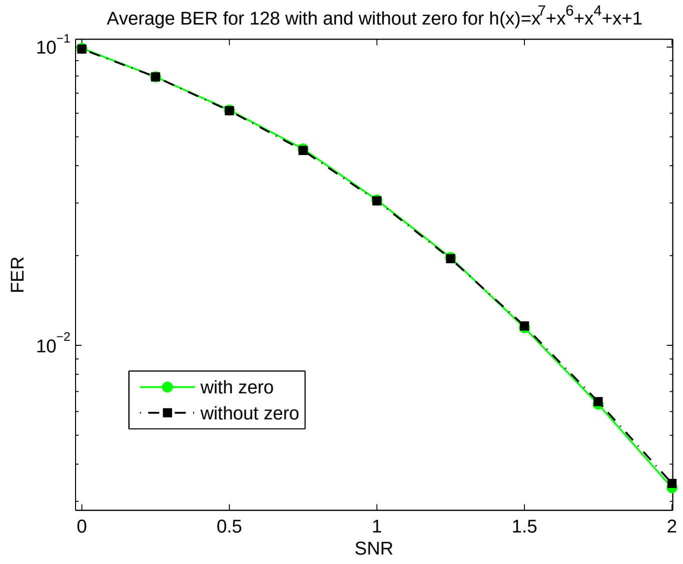
When constructing the permutations from $\mathbb{F}_q$ to $\mathbb{F}_q$ of the form $x \mapsto x^i, \pi(0)$ is always 0 and $\pi(1)$ is always 1. Consequently, $(1, 1)$ is always an element in the list of differences. This forces the spread to be 1 for all permutation monomials of this form. In order to examine the effects of the interleavers' spread, the permutations were restricted to the set of nonzero elements of $\mathbb{F}_q$. The following graphs and table show that removing zero from the permutation did not significantly affect the dispersion or error rates.

In Figure 6, the dispersions and spreads were recorded for $\mathbb{F}_{128}$, with and without the zero element. Notice that the ratios for dispersion are close to 1.

	Dispersion			Spread	
Power	Without Zero	With Zero	Ratio of Without Zero to With Zero	Without Zer	With Zero
1	0.015748032	0.015625	1.007874	1	1
2	0.460692413	0.456447	1.009301	1	1
3	0.8223972	0.822589	0.999767	2	1
4	0.492938383	0.489665	1.006684	1	1
5	0.817647794	0.817913	0.999675	2	1
6	0.817897763	0.817298	1.000734	1	1
7	0.807149106	0.807087	1.000077	1	1
8	0.536307962	0.532726	1.006723	1	1
9	0.825271841	0.825787	0.999376	1	1
10	0.8183977	0.81533	1.003763	2	1
11	0.825021872	0.826156	0.998627	1	1
12	0.820897388	0.819882	1.001239	1	1
13	0.821272341	0.821973	0.999147	1	1
14	0.816272966	0.815453	1.001006	1	1
15	0.820272466	0.820743	0.999427	1	1
16	0.536307962	0.532726	1.006723	1	1
17	0.820272466	0.820743	0.999427	1	1
18	0.82352206	0.821973	1.001884	1	1
19	0.803149606	0.803273	0.999847	1	1
20	0.82752156	0.827141	1.00046	1	1
…	…	…	…	…	…
125	0.815148107	0.814222	1.001137	1	1
126	0.805774278	0.806718	0.998831	1	1
Averages	0.795111921	0.794699	1.007126	1.15873	1

Figure 6: Dispersion and Spread for $\mathbb{Z}_{128}$

Figure 7 compares the average BER and average FER for the monomial permutations of $\mathbb{Z}_{128}$, with and without the zero element.



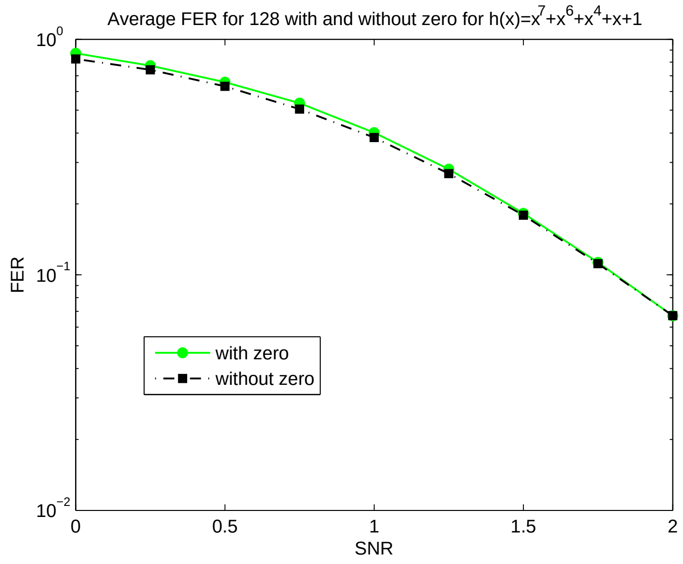


Figure 7: Average BER and FER for $q = 128$

The difference between error rates is insignificant. Consequently, the permutations without the 0 element will behave very similarly to the permutations with 0.

8.2 Dispersion

In order to examine how different dispersions affect the performance of the interleavers, $\mathbb{F}_{64}$ was used. For the irreducible polynomial $h(x) = x^6 + x^5 + x^2 + x + 1$, the signal to noise ratio (SNR) for values of 0, 1.0, and 2.0 were graphed.

For SNR 0, Figures 8 and 9 represent BER versus Dispesion and FER versus Dispersion. Both the bit and frame error rates are about the same for all dispersions.

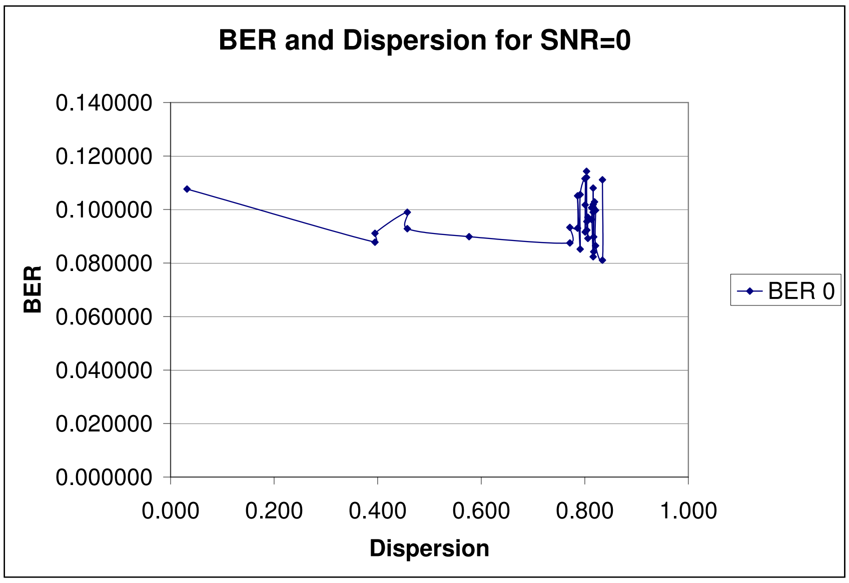


Figure 8: BER with SNR 0

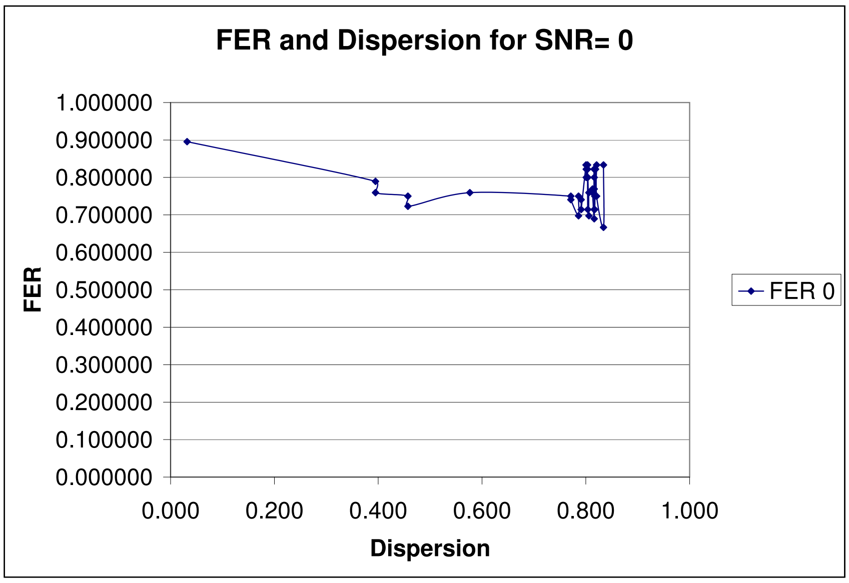


Figure 9: FER wth SNR 0

For SNR 1.0, Figures 10 and 11 below represent BER versus Dispesion and FER versus Dispersion. The error rates for this SNR value are slightly lower as the dispersion increases.

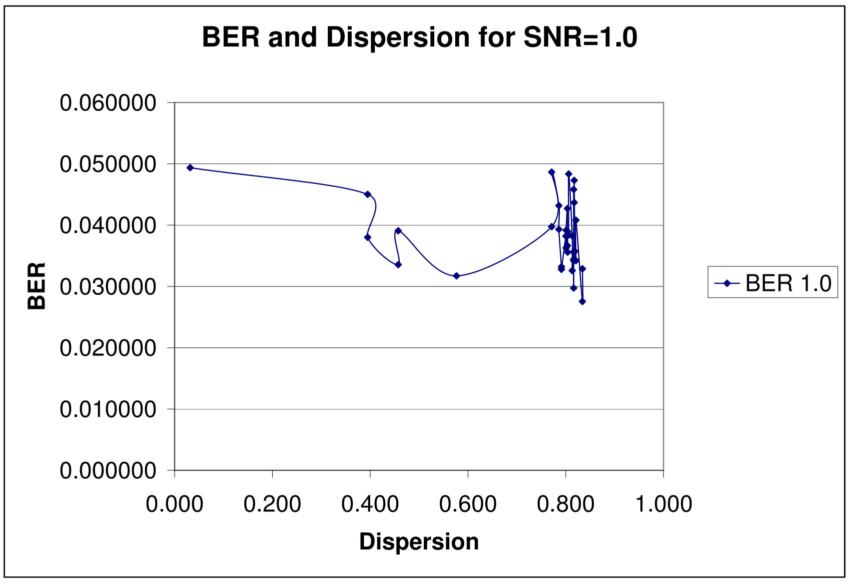


Figure 10: BER wth SNR 1.0

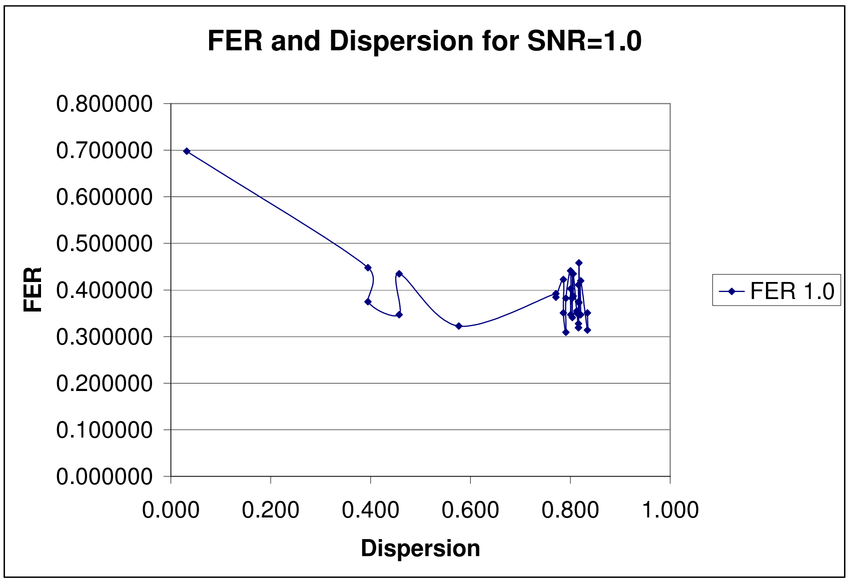


Figure 11: FER wth SNR 1.0

For SNR 2.0 in general, the error rates decrease as the dispersion increases. Figures 12 and 13 represent BER versus Dispesion and FER versus Dispersion for SNR 2.0.

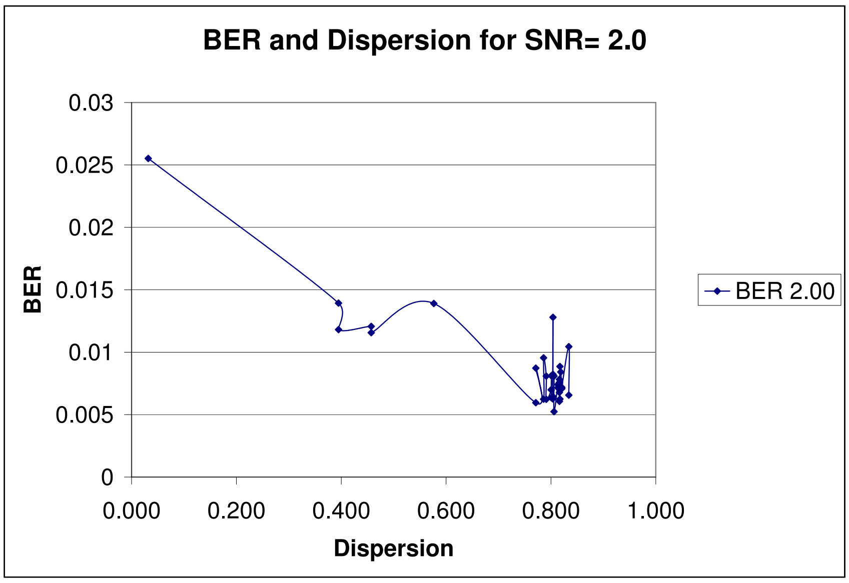


Figure 12: BER wth SNR 2.0

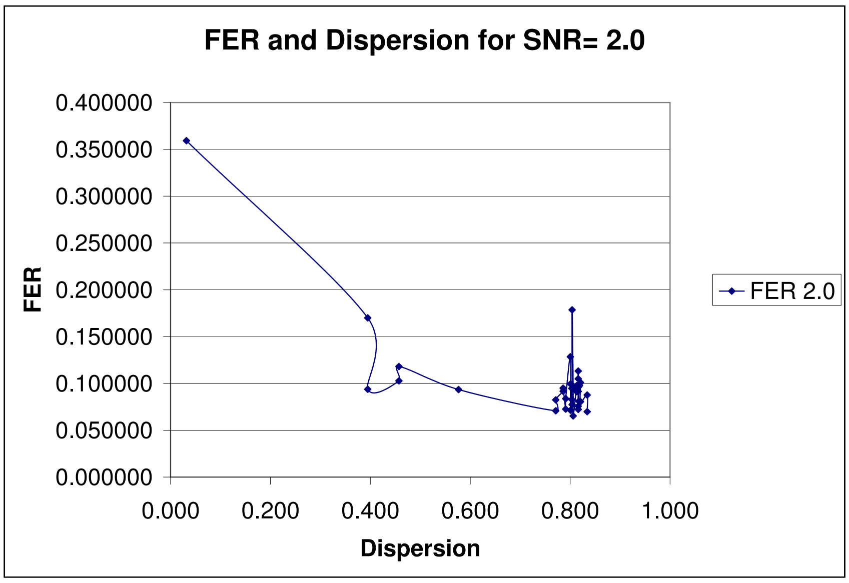


Figure 13: FER wth SNR 2.0

Consequently, it appears that as the SNR increases, higher dispersion rates will yield a better performance.

8.3 Spread

For analyzing how spread affects the performance of interleavers, $\mathbb{F}_{32}$ was used with the polynomial $h(x) = x^5 + x^4 + x^2 + x + 1$. Figures 14 and 15 show the average error rates for spread equal to 1 and for spread equal to 2 across various SNRs.

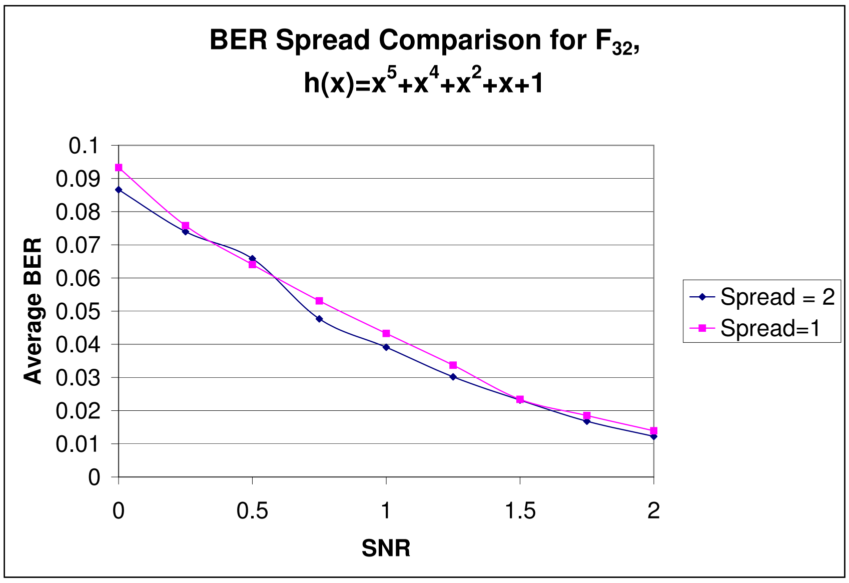


Figure 14: Average BER for Spread Comparison

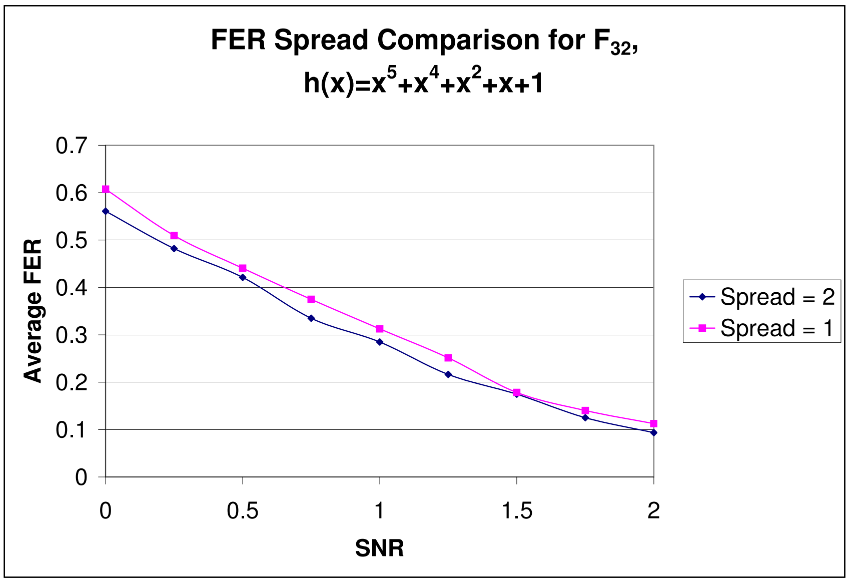


Figure 15: Average FER for Spread Comparison

For both of the bit and frame error rates, when the spread is 2 the error rates are lower. This suggests that interleavers with higher spread will perform better.

8.4 Cyclic Decomposition

No conclusion could be drawn from the data obtained in this project about relating the cyclic decomposition to the performance of the interleaver. In the data obtained from the simulations, interleavers of the same cyclic decomposition did not have the same behavior. However, by examining the cyclic decomposition of fields with different monomial orderings and whose multiplication is defined by different irreducible polynomials, an interesting result emerged.

The chart 16 shows the cyclic decomposition for all permutation monomials for $\mathbb{F}_{32}$. This is the same chart for the various fields of $\mathbb{F}_{32}$, independent of the irreducible polynomial and monomial ordering used. In this chart, 1-c, 2-c,…, 30-c stand for the length of the cycles. For example, in the first line for $x \mapsto x$, there are 31 one-cycles. In the second line for $x \mapsto x^2$, there is 1 one-cycle and 6 five-cycles.

	Cyclic Decomposition							
F_{32}	1-c	2-c	3-c	5-c	6-c	10-c	15-c	30-c
x-> x	31	0	0	0	0	0	0	0
x-> x^2	1	0	0	6	0	0	0	0
x-> x^3	1	0	0	0	0	0	0	1
x-> x^4	1	0	0	6	0	0	0	0
x-> x^5	1	0	10	0	0	0	0	0
x-> x^6	1	0	0	0	5	0	0	0
x-> x^7	1	0	0	0	0	0	2	0
x-> x^8	1	0	0	6	0	0	0	0
x-> x^9	1	0	0	0	0	0	2	0
x-> x^10	1	0	0	0	0	0	2	0
x-> x^11	1	0	0	0	0	0	0	1
x-> x^12	1	0	0	0	0	0	0	1
x-> x^13	1	0	0	0	0	0	0	1
x-> x^14	1	0	0	0	0	0	2	0
x-> x^15	1	0	0	0	0	3	0	0
x-> x^16	1	0	0	6	0	0	0	0
x-> x^17	1	0	0	0	0	0	0	1
x-> x^18	1	0	0	0	0	0	2	0
x-> x^19	1	0	0	0	0	0	2	0
x-> x^20	1	0	0	0	0	0	2	0
x-> x^21	1	0	0	0	0	0	0	1
x-> x^22	1	0	0	0	0	0	0	1
x-> x^23	1	0	0	0	0	3	0	0
x-> x^24	1	0	0	0	0	0	0	1
x-> x^25	1	0	10	0	0	0	0	0
x-> x^26	1	0	0	0	5	0	0	0
x-> x^27	1	0	0	0	0	3	0	0
x-> x^28	1	0	0	0	0	0	2	0
x-> x^29	1	0	0	0	0	3	0	0
x-> x^30	1	15	0	0	0	0	0	0

Figure 16: Cyclic Decomposition for $\mathbb{F}_{32}$

It is interesting to note that each monomial permutation and its inverse had the same spread, dispersion, and cyclic decomposition.

Theorem 39. *If σ_1, σ_2 are permutations of $\mathbb{Z}_q$ then σ_1 and its conjugate $\sigma_2^{-1}\sigma_1\sigma_2$ have the same cyclic decomposition.*

Recall the process of generating the permutation $\pi : \mathbb{Z}_{p^r} \rightarrow \mathbb{Z}_{p^r}$,

$$\mathbb{Z}_{p^r} \xrightarrow{M} (\mathbb{Z}_p)^r \xrightarrow{R} \mathbb{F}_{p^r} \xrightarrow{f_i} \mathbb{F}_{p^r} \xrightarrow{R^{-1}} (\mathbb{Z}_p)^r \xrightarrow{M^{-1}} \mathbb{Z}_{p^r},$$

where $f_i(x) = x^i$, M is a monomial ordering from $\mathbb{Z}_{p^r} \rightarrow (\mathbb{Z}_p)^r$ and M^{-1} is its inverse mapping from $(\mathbb{Z}_p)^r \rightarrow \mathbb{Z}_{p^r}$. Thus, the permutation π is

$$\pi = M^{-1}\ R^{-1}f_iR\ M.$$

Theorem 40. *If π_1, π_2 are permutations of $\mathbb{Z}_{p^r} \rightarrow \mathbb{Z}_{p^r}$ with monomial orders of M_1 and M_2, respectively, then they have the same cyclic decomposition.*

Proof. Let $\pi_1 = M_1^{-1}\ R^{-1}f_iR\ M_1$ and $\pi_2 = M_2^{-1}\ R^{-1}f_iR\ M_2$. Then

$$\begin{aligned} \pi_2 &= (M_2^{-1}M_1)\pi_1(M_1^{-1}M_2) \\ &= (M_1^{-1}M_2)^{-1}\ \pi_1(M_1^{-1}M_2). \end{aligned}$$

Since $M_1^{-1}M_2$ is the mapping from

$$\mathbb{Z}_{p^r} \xrightarrow{M_2} (\mathbb{Z}_p)^r \xrightarrow{M_1^{-1}} \mathbb{Z}_{p^r},$$

$M_1^{-1}M_2$ and $(M_1^{-1}M_2)^{-1}$ are permutations of $\mathbb{Z}_{p^r}$. Therefore, π_2 is the conjugate of π_1, and so by Theorem 39 it follows that they have the same cyclic decomposition. □

Similarly, the cyclic decomposition is the same across irreducible polynomials used to generate a finite field.

Theorem 41. *Different irreducible polynomials that generate $\mathbb{F}_q$ produce the same cyclic decomposition.*

Proof. To get a permutation of $\mathbb{Z}_{p^r}$, the following composition of functions is used:

$$\pi : \mathbb{Z}_{p^r} \xrightarrow{M} (\mathbb{Z}_p)^r \xrightarrow{R} \mathbb{F}_{p^r} \xrightarrow{f_i} \mathbb{F}_{p^r} \xrightarrow{R^{-1}} (\mathbb{Z}_p)^r \xrightarrow{M^{-1}} \mathbb{Z}_{p^r}.$$

This means $\pi = M^{-1}R^{-1}f_iRM$ where $f_i(x) = x^i$. Every field $\mathbb{F}_{p^r}$ has a primitive polynomial. Let $h(\alpha)$ be the primitive polynomial for $\mathbb{F}_{p^r}$, i.e. $\mathbb{F}_{p^r} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^r-2}\}$. Let $g_i(x) = x^i$, $g_i : \mathbb{F}_{p^r} \longrightarrow \mathbb{F}_{p^r}$. Take an irreducible polynomial of degree r, $h_j(x)$. By Theorem 14, there exists an isomorphism $z_j : \mathbb{F}_p[x]/\langle h_j(x)\rangle \longrightarrow \mathbb{F}_p[x]/\langle h(x)\rangle$. Since $\mathbb{F}_p[x]/\langle h_j(x)\rangle = \mathbb{F}_{p^r}$ and $\mathbb{F}_p[x]/\langle h(x)\rangle = \mathbb{F}_{p^r}$, it follows that $\mathbb{F}_{p^r} \xrightarrow{f_i} \mathbb{F}_{p^r}$ can be replaced by

$$\mathbb{F}_{p^r} \xrightarrow{z_j^{-1}} \mathbb{F}_{p^r} \xrightarrow{g_i} \mathbb{F}_{p^r} \xrightarrow{z_j} \mathbb{F}_{p^r},$$

i.e., $f_i = z_j^{-1} g_i z_j$. Hence, $\pi = M^{-1}R^{-1}z_j^{-1}g_i z_j RM$. Consider two irreducible polynomials of degree r, $h_1(x)$ and $h_2(x)$. Then there exists z_1 and z_2 such that

$$\begin{aligned}\pi_1 &= M^{-1}R^{-1}z_1^{-1}g_i z_1 RM \\ \pi_2 &= M^{-1}R^{-1}z_2^{-1}g_i z_2 RM.\end{aligned}$$

But

$$\begin{aligned}\pi_1 &= (M^{-1}R^{-1}z_2^{-1}z_1 RM)\pi_2(M^{-1}R^{-1}z_1^{-1}z_2 RM) \\ &= (M^{-1}R^{-1}z_2^{-1}z_1 RM)\pi_2(M^{-1}R^{-1}z_2^{-1}z_1 RM)^{-1}.\end{aligned}$$

Hence, π_1 and π_2 are conjugates and therefore have the same cyclic decomposition.

$\square$

8.5 Lexicographic Versus Graded Lexicographic Ordering

In this project two types of monomial orderings were used, lexicographic and graded lexicographic. Figures 17 and 18 shows the average BER and FER for

$$\mathbb{F}_{64} = \mathbb{F}_2[x]/\langle h(x) = x^6 + x^5 + x^2 + x + 1\rangle.$$

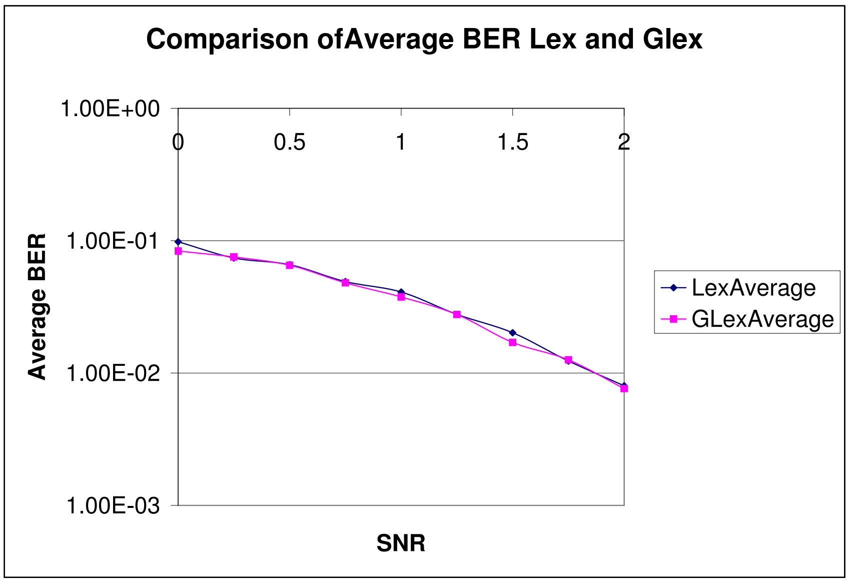


Figure 17: Average BER for Monomial Orderings

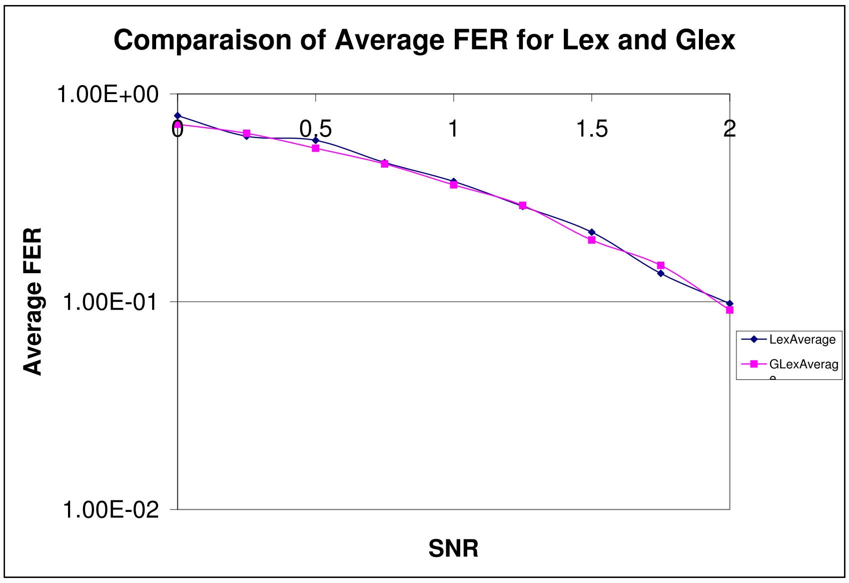


Figure 18: Average FER for Monomial Orderings

According to these graphs, there is not much of a difference in performance when comparing the two monomial orderings. This knowledge is beneficial for programming and simulations since the graded lexicographic ordering is costly and time consuming, whereas the lexicographic ordering is inexpensive and quick.

8.6 Different Irreducible Polynomials

Different irreducible polynomials were used to generate permutations of the form $x \mapsto x^i$, as in equation 1. Figures 19 and 20 show the average BER and FER for six different irreducible polynomials.

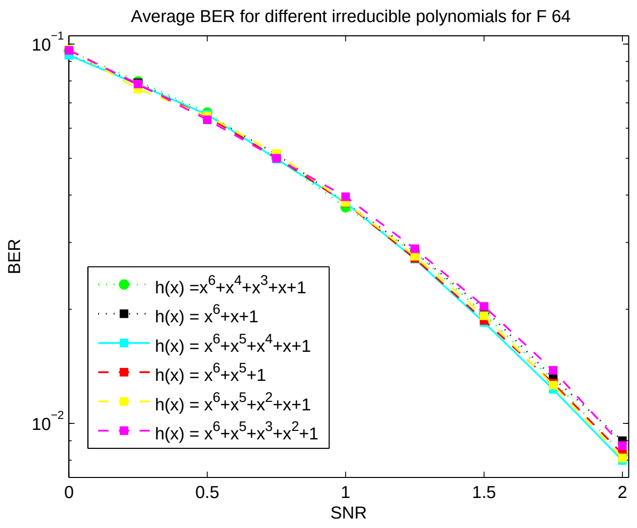


Figure 19: Average BER for Different Irreducible Polynomials

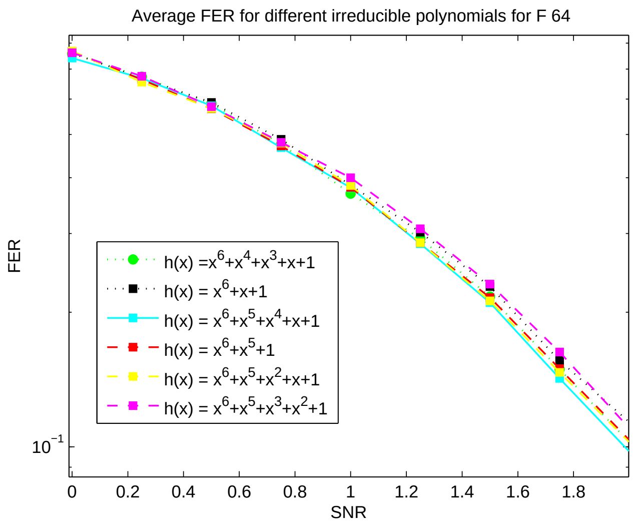


Figure 20: Average FER for Different Irreducible Polynomials

All six polynomials have similar behavior. Although one polynomial, $h(x) = x^6 + x^5 + x^4 + x + 1$, has slightly lower error rates, the difference is insignificant. This suggests that using different irreducible polynomials produces no significan difference in performance.

8.7 Comparing $\mathbb{Z}_q$ versus $\mathbb{F}_q$ Interleavers

All monomial permutations $x \mapsto x^i$ for $\mathbb{Z}_q$ are known due to Theorems 23 and 28. Therefore, permutations of $\mathbb{Z}_q$ that do not use finite fields can be compared to permutations of $\mathbb{Z}_q$ that do use finite fields. Figures 21 and 22 show the average BER and FER for $q = 256$.

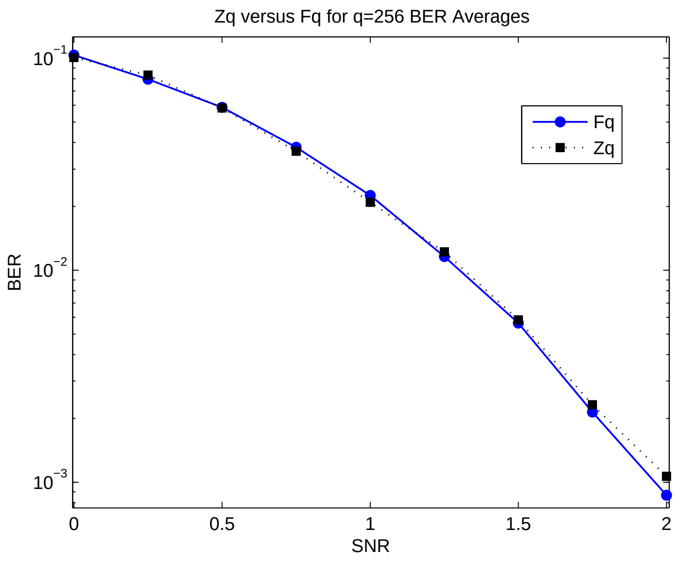


Figure 21: Average BER

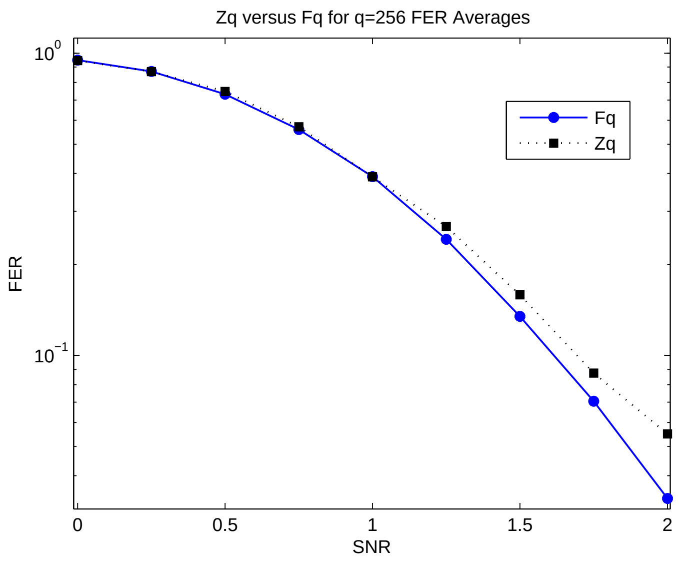


Figure 22: Average FER

In the BER graph, $\mathbb{F}_q$ performs slightly better as the SNR increases. However, in the FER graph, $\mathbb{F}_q$ performs much better as SNR increases.

Figure 23 compares the dispersion of $\mathbb{Z}_{128}$ to $\mathbb{F}_{128}$.

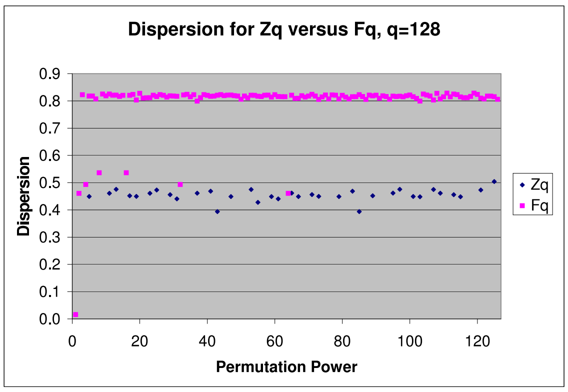


Figure 23: Dispersion Comparison

Notice that the most all of the dispersions for $\mathbb{Z}_{128}$ are about 0.45, whereas almost all of the dispersions for $\mathbb{F}_{128}$ are about 0.8.

8.8 Comparing Randomly Constructed Interleavers versus those made with $\mathbb{F}_q$

Algebraically constructing the permutations for $\mathbb{Z}_q$ and $\mathbb{F}_q$ using monomial permutations is systematic. Therefore, there is a need to compare these permutations to completely random permutations. Below are BER and FER graphs for q =32, 64, 128, and 256, where the $\mathbb{F}_q$ graph is one particular permutation.

For $q = 32$, the random permutation had higher frame error rates. As for the bit error rates, the lower SNR values produced lower error rates for the random permutation. Higher SNR values yielded higher error rates for the random permutation.

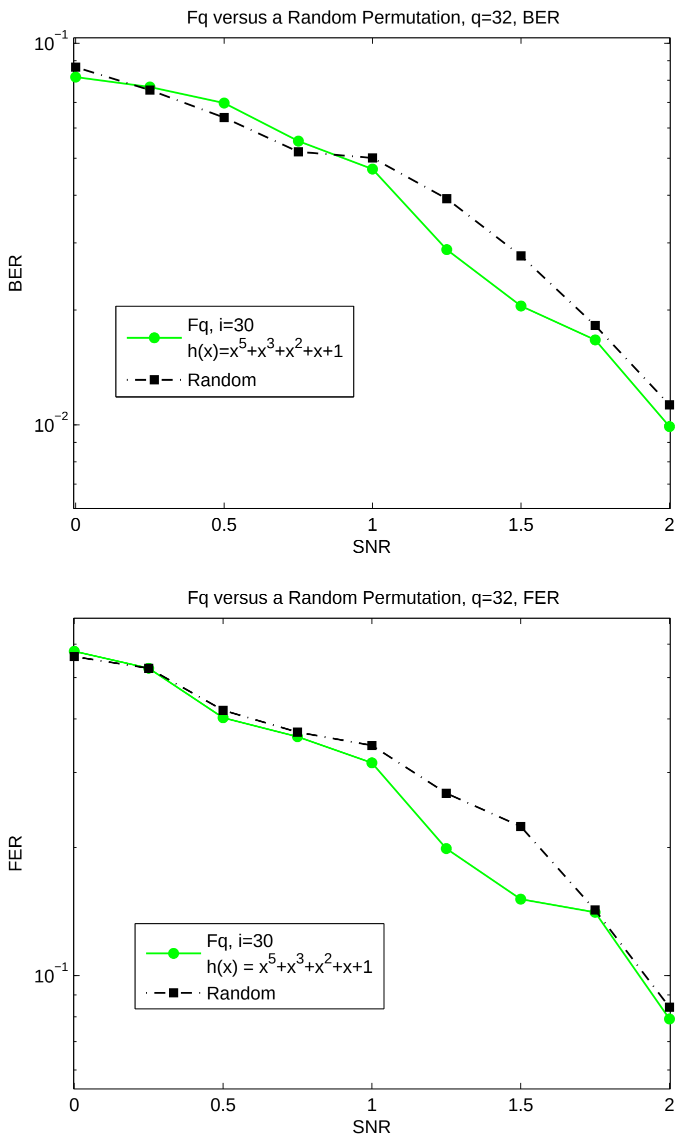


Figure 24: BER and FER for $q = 32$

For $q = 64$ the random permutation had better frame error rates. However, for the bit error rates, neither permutation had an advantage over the other.

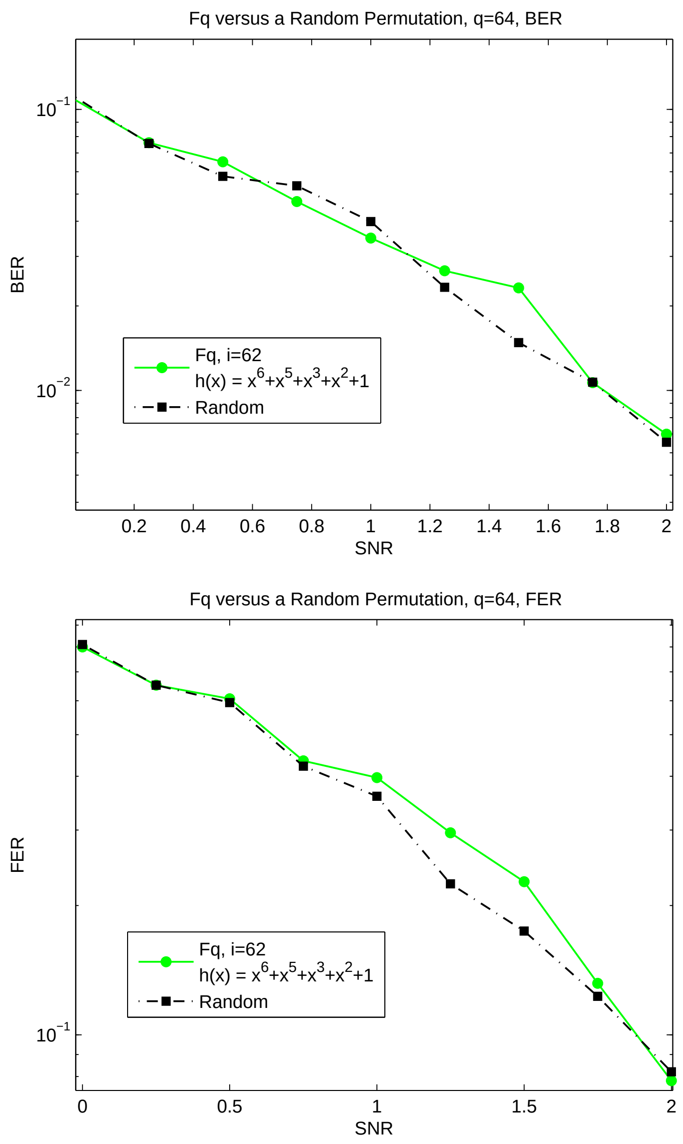
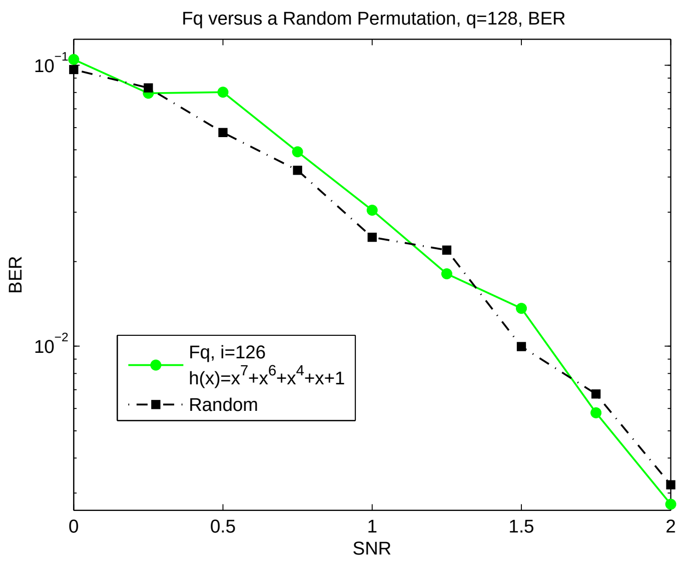


Figure 25: BER and FER for $q = 64$

For $q = 128$, both permutations behaved in a similar manner. Consequently, neither permutation outperformed the other.



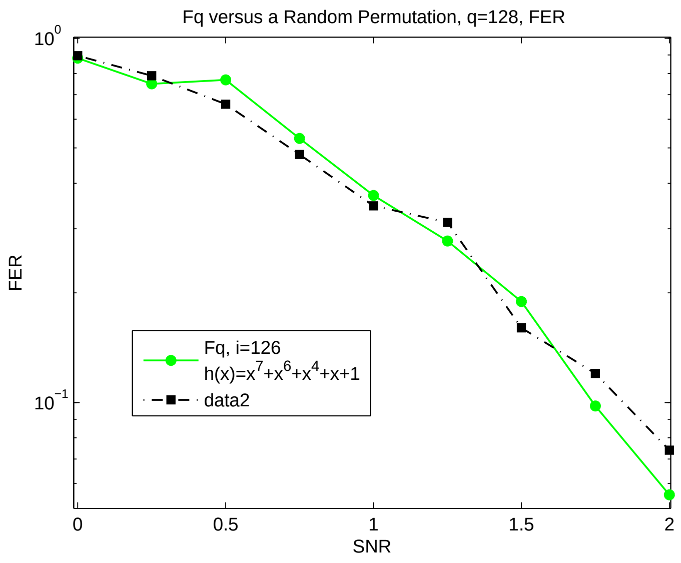
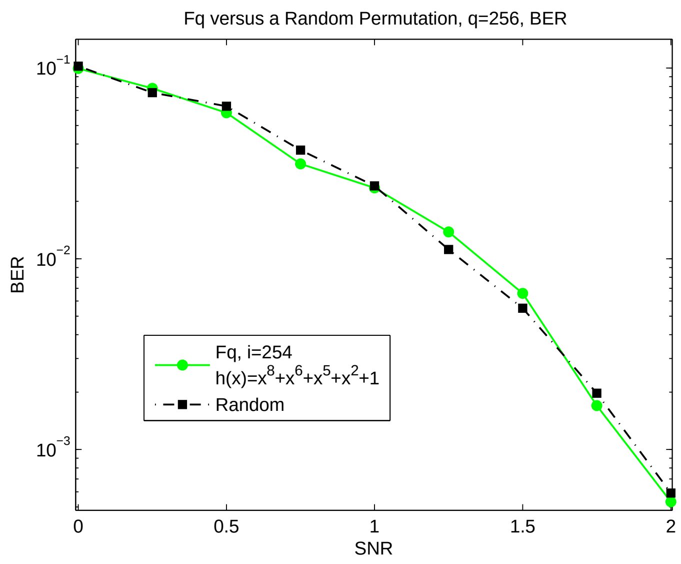


Figure 26: BER and FER for $q = 128$

For $q = 256$, the differences in both the bit error rates and frame error rates are too small to be of consequence.



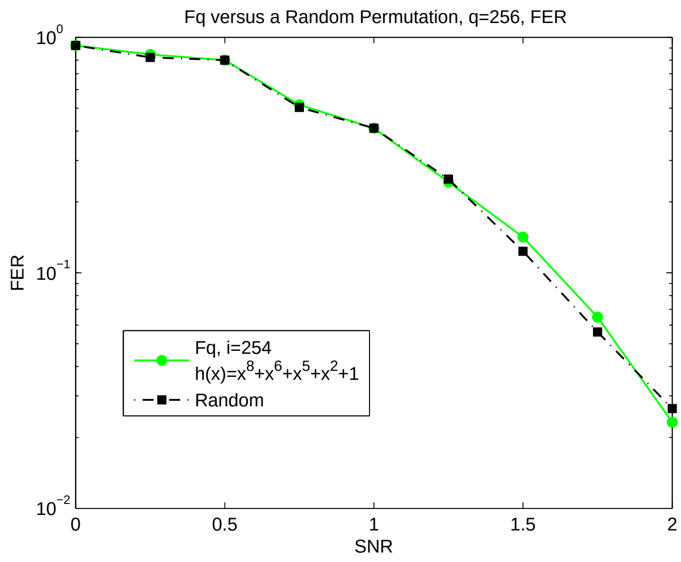


Figure 27: BER and FER for $q = 256$

Overall, it appears that as q increases, the random permutations perform very similarly to $\mathbb{F}_q$ in simulations. This suggests that using interleavers that are constructed using finite fields perform as well as interleavers that are constructed randomly.

9 Conclusion

This project focused on the effects of various interleavers in turbo codes. These interleavers were constructed through the use of finite fields. The main reason for constructing interleavers in this systematic way instead of randomly is to minimize the storage use. Through simulations their performance was measured through Bit Error Rates and Frame Error Rates and then compared to random interleavers and interleavers of $\mathbb{Z}_q$ whose construction did not include the use finite fields. Interleavers using finite fields performed better than those using $\mathbb{Z}_q$ directly, however, as q increased, those using finite fields performed very similarly to a random interleaver. Interleavers with higher dispersion generally performed better as the SNR increases. Also, interleavers with higher spread had lower error rates, whereas no conclusions could be drawn about cyclic decomposition. There was no significant difference in interleavers that used different irreducible polynomials or monomial ordering.

10 Work to be Considered for the Future

- Since there were no conclusions that could be drawn from the data on cyclic decomposition from this project, further studies could be done on this property.
- Blocks of greater length should be analyzed.
- Do fields generated by different irreducible polynomials provide the same permutations besides the identity?
- If there are two given monomial orders, do they have either all permutations the same or all different, excluding the identity?
- Can two permutations generated by different irreducible polynomials and monomial orders be the same, excluding the identity.
- How do fields generated by primitive polynomials versus non-primitive polynomials perform?

References

[Co] Cook, John., Finding Irreducible Polynomials and Finite Field Procedures. `http://www-rohan.sdsu.edu/~mosolliv/courses/coding02/codsch.html`

[HeSt] heegard Heegard, C., Stephen, B. (1999). Turbo Coding. *Kluwer Academic Publishing.*

[LuPe] Luis, Y., B., Perez, L., O. (2005). Permutations of Z_q Constructed Using Several Monomial Orderings. *University of Puerto Rico*, 1–8.

[Wu] Wu, Yufei., Turbo Code Simulator., Nov 1998., MPRG lab, Virginia Tech. `http://www.ee.vt.edu/~yufei`

A Appendix

A.1 A Note on Maple Programs for this Project

The Maple procedures written for this project require the following line, which loads four different packages:

```
with(ListTools,LinearAlgebra,PolynomialTools,linalg):
```

A.2 Program for Construction of Permutations

Before getting to the main program that constructs the permutations, there are a couple of preliminary programs that must be run.
The first program, “lc”, takes in an $n \times n$ matrix and two vectors of length n. The matrix is a monomial ordering matrix. The output tells which vector is larger according to this ordering.

```
lc:=proc(u,v,w) local t,f,nn,ii,z;
   t:= v-w;
   nn:= op(1,u.t);
      f:=(u.t);
          for ii from 1 to nn do
            if (f[ii]>0) then
      z:=1;
      break;
              elif
                (f[ii]=0)then
                z:=0
                 elif (f[ii]< 0) then
          z:=-1;
      break;
            end if;
         end do;
z; end proc:
```

The“ben” program takes in a list of elements, a vector that contains the coefficients of a polynomial, and an integer. This program maps each element according to the received polynomial vector modulo the inputed integer.

```
ben:= proc(a,coeffvector,g) local outputvector,sizea,sizec,j,d,poly;
  sizec:=op(1,coeffvector):
  poly:=0:
  for d from 1 to sizec do
      poly:=poly+coeffvector[d]*x^(sizec-d);
  end do;
  sizea:=op(1,a);
  outputvector:=Vector[row](sizea);
  for j from 1 to sizea do
      outputvector[j]:=expand((eval(poly, x=a[j]))) mod g:
  end do;
  outputvector;
end proc:
```

This is the main program that constructs the permutations. The input is p, n where $\mathbb{F}_q = \mathbb{F}_{p^n}$. It takes in a polynomial coefficient vector, a monomial ordering matrix, and an irreducible polynomial of degree n. It outputs a permutation of $\mathbb{Z}_q$ according to equation 1.

```
permutations:=proc(p,n,poly,matrix,irreduciblepoly)
    local q,ordervector,ordervector2,j,flag,m,qq,storevector,aa,y,
        xvector,yy,powers,temporaryvector,yvector,yvector2,kk,b,
        vectormatrix,jj,k,r,searchvector,rr,flagg,storevector2,u,w;
  q := p^n:
  storevector:=Vector(p^n):
  storevector[1]:=Vector[row](n):
  for aa from 1 to q-1 do
      storevector[aa+1]:=convert(aa,base,p):
      if nops(storevector[aa+1])<n then
          storevector[aa+1]:=convert([storevector[aa+1],Vector[row](n-nops
      end if;
    storevector[aa+1]:=convert(ListTools[Reverse]
                (convert(storevector[aa+1],list)),Vector[row]):
  end do:
  ordervector:=Vector(q,q);
  ordervector[1]:=0;
  ordervector2:=[1];
  for j from 2 to q do
  flag:=0;
  if ordervector[j]=q then
      m:=1:
      while (m<(op(1,ordervector2)+1) and flag=0) do
          qq:=lc(matrix,LinearAlgebra[Transpose](storevector[j]),LinearAlgebra
          if qq=-1 then
              if m=1 then
                  ordervector2:=convert([j,ordervector2],Vector);
                  flag:=1;
              else
                    ordervector2:=convert([ordervector2[1..m-1],j,
                  flag:=1;
              end if;
          elif qq=1 then
              if m=op(1,ordervector2) then
```

```
                ordervector2:=convert([ordervector2,j],Vector);
                flag:=1;
            else
                m:=m+1;
            end if;
        elif qq=0 then
            print("error");
        end if;
    end do;
end if;
end do:
for y from 1 to q do
ordervector[ordervector2[y]]:=y-1:
end do:
storevector2:=Vector(q);
for u from 1 to q do
storevector2[ordervector[u]+1]:=storevector[u];
end do;
for w from 0 to q-1 do
    ordervector[w+1]:=w;
end do;
xvector:=Vector(q);
for yy from 1 to q do
temporaryvector:=0;
        for powers from 1 to n do
                 temporaryvector:=temporaryvector
                      + storevector2[yy][powers]*x^(n-powers);
        end do;
xvector[yy]:=temporaryvector;
end do;
xvector:=LinearAlgebra[Transpose](xvector);
yvector:=ben(xvector,poly,p);
alias(alpha=RootOf(irreduciblepoly));
yvector2:=subs(x=RootOf(irreduciblepoly),yvector);
for kk from 1 to q do
   yvector2[kk]:=Normal(yvector2[kk]) mod p;
end do;
yvector2;
vectormatrix:=Vector(q);
for jj from 1 by 1 to q do
b:=ListTools[Reverse](CoefficientList(RootOf(irreduciblepoly)^n
  +yvector2[jj],RootOf(irreduciblepoly))):
vectormatrix[jj]:=convert(b[2..n+1],Vector[row]):
end do:
vectormatrix:=convert(vectormatrix,list);
```

```
  storevector2:=convert(storevector2,list);
  for k to q do
     storevector2[k] := convert(storevector2[k], list);
     vectormatrix[k] := convert(vectormatrix[k], list);
  end do:
  searchvector:=Vector[row](q);
  for r from 1 to q do
  flagg:=0;
  for rr from 1 to q while flagg=0 do
      if (vectormatrix[r]=storevector2[rr]) then
          searchvector[r]:=ordervector[rr];
          flagg:=1;
      end if;
  end do;
  end do:
  searchvector;
end proc:
```

In the program "findpolynomial," the input is p, n where $\mathbb{F}_q = \mathbb{F}_{p^n}$. The output is a list of irreducible polynomials of degree n.

```
findpolynomial:=proc(a,b) local c,SF,PrimFactors,FactorIndx;
  c:=a^b;
  SF := Factors(x^(c-1) - 1) mod a:
  PrimFactors:=[];
  for FactorIndx from 1 to nops(SF[2]) do
      if (degree(SF[2,FactorIndx,1])= b) then
              if (Primitive(SF[2,FactorIndx,1]) mod a) then
                      PrimFactors := [op(PrimFactors),SF[2,FactorIndx,1]];
          end if;
  end do;
PrimFactors; end proc:
```

B Program for Dispersion

This program takes as input a permutation as a list and computes its dispersion.

```
dispersion:=proc(v)::Vector; local
x,w,sizev,factorv,y,z,i,j,k,a,b,c;
   sizev:=nops(v);
   w:=[v[2..sizev],v[1]];
   x:=convert(w,Vector);
   y:=sum(q,q=1..sizev-1);
   z:=Vector(y);
   k:=0;
   for i from 1 by 1 to sizev-1 do
       for j from i+1 by 1 to sizev do
           k:=k+1;
           z[k]:=[i,x[j]-x[j-i]];
       end do;
   end do;
   a:=MakeUnique(convert(z,listlist));
   b:=nops(a);
   c:=2*b/(sizev*(sizev-1));
end proc:
```

C Program for Spread

This program takes as input a permutation in the form of a list and computes its spread.

```
spreading:=proc(v) local x,w,sizev,factorv,y,z,i,j,k,m,n,
        differences,marker,s,spread;
   sizev:=nops(v);
   w:=[v[2..sizev],v[1]];
   x:=convert(w,Vector);
   y:=sum(q,q=1..sizev-1);
   z:=Vector(y);
   k:=0;
   for i from 1 by 1 to sizev-1 do
       for j from i+1 by 1 to sizev do
           k:=k+1;
           z[k]:=[i,abs(x[j]-x[j-i])];
       end do;
   end do;
   differences:=(convert(z,listlist));
   n:=nops(differences);
   marker:=0;
   s:=2;
   while marker=0 do
       for m from 1 to n do
           if differences[m,1]<s then
               if differences[m,2]<s then
                   marker:=1;
                   spread:=s-1;
               end if;
           end if;
       end do;
       if marker=0 then
```

```
           if s<(n-1) then
               s:=s+1;
           else
               spread:=n-1;
               marker:=1;
           end if;
       end if;
   end do;
   spread;
end proc:
```

D Program for Cyclic Decomposition

This program takes as input a permutation in the form of a list. It initially computes the cyclic decomposition, and then it proceeds to determine how many cycles there are of each length.

```
decompose:=proc(perm) local
len,reflist,seen,begin,idx,j,k,cycle,final,b,c,f,
        u,w,uu,ww,sizestore,flag;
  len:=nops(perm);
  reflist:=convert([seq(i,i=1..len)],'array');
  seen:=convert([seq(0,i=1..len)],'array');
  seen[1]:=1;
  idx:=1;
  begin:=1;
  cycle:=[1];
  final:=[];
  for j from 1 by 1 to len do
      idx:=perm[idx];
      if (begin=idx) then
          final:=[FlattenOnce(final),cycle];
          for k from 1 by 1 to len do
             if(seen[k]=0) then
                  idx:=k;
              end if;
          end do;
          begin:=idx;
          cycle:=[idx];
          seen[idx]:=1;
      else
          cycle:=[Flatten([cycle,idx])];
          seen[idx]:=1;
      end if;
  end do;
  final:=FlattenOnce(final);
  b:=nops(final);
  c:=[];
  for f from 1 to b do
      c:=[Flatten([c,nops(final[f])])];
  end do;
u:=nops(c[1]); sizestore:=[[c[1,1],1]]; for w from 2 to u do
  uu:=nops(sizestore);
  flag:=0;
  for ww from 1 to uu while flag=0 do
      if c[1,w]=sizestore[ww,1] then
          sizestore[ww,2]:=sizestore[ww,2]+1;
          flag:=1;
      end if;
  end do;
  if flag=0 then
      sizestore:=[op(sizestore),[c[1,w],1]];
  end if;
end do; sizestore; end proc:
```