Lab Report

Coding Theory for Storage and Networks

David de Andrés Hernández
Technical University of Munich
deandres.hernandez@tum.de

July 17, 2022

Lab 1: Regenerating Codes

In this lab, we consider a distributed storage system with n nodes, each nodes stores α symbols, such that a regenerating node can reconstruct the whole storage by contacting any k nodes. The repair degree of the system is d and the repair bandwidth of each node if β.

Problem 1: MSR Codes by ZigZag Codes

In this task, we use an $(n, k = n-2, d = n-1)$ ZigZag code ([Tamo, Wang and Bruck, '11]) to construct an $(n, k, d) = (5, 3, 4), \alpha = 4, \beta = 2$ Regenerating code. The $(5, 3, 4)$ ZigZag code for the message $u = (u_1, ..., u_{12})$ over $\mathbb{F}_3$ is given by:

Table 1: Symbols contained at each node

Node 1	u_1	u_2	u_3	u_4
Node 2	u_5	u_6	u_7	u_8
Node 3	u_9	u_{10}	u_{11}	u_{12}
Node 4	$u_1 + u_5 + u_9$	$u_2 + u_6 + u_{10}$	$u_3 + u_7 + u_{11}$	$u_4 + u_8 + u_{12}$
Node 5	$u_1 + 2u_7 + 2u_{10}$	$u_2 + 2u_8 + u_9$	$u_3 + u_5 + u_{12}$	$u_4 + u_6 + u_{11}$

We denote the content of storage at each node $i \in \{1, \dots, 5\}$ by $\mathbf{s_i} = (s_i^1, s_i^2, s_i^3, s_i^4)$. With this notation, the storage matrix looks as follows

$$\mathbf{S} = \begin{bmatrix} s_1^1 & s_2^1 & s_3^1 & s_4^1 & s_5^1 \\ s_1^2 & s_2^2 & s_3^2 & s_4^2 & s_5^2 \\ s_1^3 & s_2^3 & s_3^3 & s_4^3 & s_5^3 \\ s_1^4 & s_2^4 & s_3^4 & s_4^4 & s_5^4 \end{bmatrix}.$$

Moreover, we can now determine the encoding matrix $\mathbf{A}_i \in M_{m\times n}(\mathbb{F}_3)$, $i = 1, \dots, 5$ of each user such that $\mathbf{s_i} = \mathbf{A}_i\mathbf{u}$:

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{A}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{A}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

Now, before computing every $\mathbf{s_i} = \mathbf{A}_i\mathbf{u}$, we retrieve the system's time and after the calculation, we retrieve it again. The difference is the time required for the encoding. We measured 0.014721870422363281s.

To continue, we simulate that node 1 has failed and we want to repair it with minimum bandwidth. First, we determine which $t = 2$ symbols must be downloaded from each node. According to table , we observe that u_1 is still available at the parity (node 4) as $u_1 + u_5 + u_9$ at position 1 (s_4^1). So we download this parity symbol. This in turn dictates that we must download both u_5 and u_9 to subtract them. They are available at position 1 from nodes 2 and 3; s_2^1 and s_3^1 respectively. In the same way, to recover u_4 we must download both the parity symbol at position 4 (s_4^4), as well as symbol 4 from nodes 2 and 3; s_2^4 and s_3^4 respectively. Furthermore, a nice property of ZigZag codes, is that the remaining required symbols can be recovered from $t = 2$ additional ZigZag symbols from node 5. For this reason, we download symbols 2 and 3 from Node 5; ; s_5^2 and s_5^3 respectively. Summarizing, the collection of downloaded symbols are

$$\begin{aligned} \mathbf{d_2} &= (s_2^1, s_2^4) \\ \mathbf{d_3} &= (s_3^1, s_3^4) \\ \mathbf{d_4} &= (s_4^1, s_4^4) \\ \mathbf{d_5} &= (s_5^2, s_5^3). \end{aligned}$$

It is now mathematically possible for the regenerating node to recover all symbols which where stored at node 1, as follows

$$\begin{aligned} \hat{u}_1^1 &= d_4^1 - d_2^1 - d_3^1 \\ \hat{u}_1^2 &= d_5^1 - 2d_2^2 - d_3^1 \\ \hat{u}_1^3 &= d_5^2 - d_2^1 - d_3^2 \\ \hat{u}_1^4 &= d_4^2 - d_2^2 - d_3^2. \end{aligned}$$

And the repaired node is $\mathbf{s_1} = (\hat{s}_1^1, \hat{s}_1^2, \hat{s}_1^3, \hat{s}_1^4) = (u_1, u_2, u_3, u_4)$.
Under the assumption that downloading one symbol consumes one second. We can calculate that the repair time is:

$$1[s] \times \#\text{downloaded symbols} + \text{decoding time}[s] = 8.006059646606445[s].$$

In contrast, if we wish to recover the message, we shall download all information symbols $u_1, \cdots, u_{12}$. This means, we must download 4 symbols from nodes 1,2, and 3. The recover time in this case is:

$$1[s] \times \#\text{information symbols} = 12[s].$$

It is clear that it is more efficient to restore one node, using a regenerating code than using just a plain MDS code which would require to retrieve all remaining available symbols for the restoration.

Problem 2: MSR Codes by Product Matrix Codes

In this task, we use an $(n, k, d = 2k - 2)$ Product Matrix Code ([Rashmi, Shah and Kumar, '11]) to construct an $(n, k, d) = (5, 3, 4)$ Regenerating Code over $\mathbb{F}_{13}$. Let the information stored be $u = (2, 2, 3, 5, 6, 10)$. We begin by computing the MSR point

$$\alpha_{MSR} = d - k + 1 = k - 1 = 2$$

$$\beta_{MSR} = \frac{B}{k(d - k + 1)} = \frac{6}{3(4 - 3 + 1)} = 1.$$

To continue, we define the $(d \times \alpha)$ message matrix M as:

$$\mathbf{M} = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$$

where S_1 and S_2 are $(\alpha \times \alpha)$ symmetric matrices constructed such that the $\binom{\alpha+1}{2}$ entries in the upper-triangular part of each of the two matrices are filled up by $\binom{\alpha+1}{2}$ distinct message symbols. Thus, all the $B = \alpha(\alpha+1)$ message symbols are contained in the two matrices S_1 and S_2. The entries in the strictly lower-triangular portion of the two matrices S_1 and S_2 are chosen so that the matrices are symmetric. Plugging in the values we obtain

$$\mathbf{M} = \begin{bmatrix} 2 & 2 \\ 2 & 3 \\ 5 & 6 \\ 6 & 10 \end{bmatrix}.$$

Next, we define the encoding matrix ψ to be a $(n \times d)$ matrix given by $\Psi = [\Phi\ \Lambda\Phi]$ where Φ is an $(n \times \alpha)$ matrix and Λ is an $(n \times n)$ **diagonal matrix**.

The elements of Ψ are chosen such that the following conditions are satisfied:

1. Any d rows of Ψ are linearly independent;
2. Any α rows of Φ are linearly independent;
3. The n diagonal elements of Λ are distinct.

The above requirements can be met, for example, by choosing Ψ to be a Vandermonde matrix with elements chosen carefully to satisfy the third condition. In this case, let the ith row of Ψ be $\psi_i = [1\ x_i\ ...\ {x_i}^{d-1}]$, which gives $\Lambda = diag\{{x_1}^{\alpha}, {x_2}^{\alpha}, ..., {x_n}^{\alpha}\}$. In order to satisfy the third property, one may choose F_q to be any field of size $n(d-k+1)$ or higher, with $x_i = g^{i-1}$, where g is the generator of the multiplicative group of the finite field F_q. We observe that the size of $\mathbb{F}_{13}$ fulfils the requirement. Moreover, we begin the code with a search for a primitive element g which generates $\mathbb{F}_{13} \setminus \{0\}$. We find that $g = 2$ is a a possible solution and proceed

$$\mathbf{\Psi} = [\Phi\ \Lambda\Phi] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 4 & 3 & 12 \\ 1 & 8 & 12 & 5 \\ 1 & 3 & 9 & 1 \end{bmatrix}.$$

Hence the (5×2) matrix Φ and the (5×5) diagonal matrix Λ are

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \\ 1 & 8 \\ 1 & 3 \end{bmatrix}, \mathbf{\Lambda} = \begin{bmatrix} 1 & & & & \\ & 4 & & & \\ & & 3 & & \\ & & & 12 & \\ & & & & 9 \end{bmatrix}.$$

We encode the information by $S = \Psi M$ and obtain

$$\mathbf{S} = \begin{bmatrix} 2 & 8 \\ 9 & 8 \\ 6 & 9 \\ 4 & 5 \\ 7 & 10 \end{bmatrix}$$

0.0.1 Exact Regeneration

We now assume that node 4 has failed and we would like to repair the node. This means we must contact $d = 4$ other nodes and download $\beta = 1$ symbols to repair node 4. The process begins with each node sending the inner product $\psi_l^t M[1,8]^t$ for $l = 1, 2, 3, 5$ to the regenerating node. Moreover, the regenerating node builds the Ψ_{repair} matrix, which originates from deleting from Ψ the row corresponding to the node to be repaired, 4 in this case. Ψ_{repair} is

$$\Psi_{\text{repair}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 4 & 3 & 12 \\ 1 & 3 & 9 & 1 \end{bmatrix}.$$

The regenerating node now multiplies the downloaded symbols with $\Psi_{\text{repair}}^{-1}$ and obtains:

$$\begin{bmatrix} \mathbf{S_1}\phi_\mathbf{4} \\ \mathbf{S_2}\phi_\mathbf{4} \end{bmatrix} = \begin{bmatrix} u_1 + 8u_2 \\ u_2 + 8u_3 \\ u_4 + 8u_5 \\ u_5 + 8u_6 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 1 \\ 8 \end{bmatrix}.$$

Finally, using the locator $\lambda_4 = 12$, the DC reconstructs node 4 as:

$$\mathbf{s_4} = [(u_1 + u_2) + \lambda_4(u_4 + u_5), (u_2 + u_3) + \lambda_4(u_5 + u_6)] = [5 + 12 \times 1, 0 + 12 \times 8] = [4, 5].$$

Under the assumption that downloading one symbol consumes one second. We can calculate that the repair time is:

$$1[s] \times \#\text{downloaded symbols} + \text{decoding time}[s] = 4.002313613891602[s].$$

0.0.2 Data Reconstruction

To reconstruct the file, the data collector contacts $k = 3$ node; i.e. Nodes 1,2,3. It builds

$$\mathbf{\Psi}_{\text{DC}} = \begin{bmatrix} \Phi_{\text{DC}} & \Lambda_{\text{DC}}\Phi_{\text{DC}} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \end{bmatrix}$$

Next, it receives the symbols matrix

$$\mathbf{S}_{\text{DC}} = \begin{bmatrix} 2 & 8 \\ 9 & 8 \\ 6 & 9 \end{bmatrix}.$$

which corresponds to

$$\mathbf{S}_{\text{DC}} = \mathbf{\Psi}_{\text{DC}}\mathbf{M} = \begin{bmatrix} \Phi_{\text{DC}} & \Lambda_{\text{DC}}\Phi_{\text{DC}} \end{bmatrix} \begin{bmatrix} \mathbf{S_1} \\ \mathbf{S_2} \end{bmatrix} = \begin{bmatrix} \Phi_{\text{DC}}\mathbf{S_1} & \Lambda_{\text{DC}}\Phi_{\text{DC}}\mathbf{S_2} \end{bmatrix}$$

The data collector can post-multiply this term with Φ_{DC}^T, obtaining

$$\mathbf{S}_{\text{DC}}\Phi_{\text{DC}}^T = \begin{bmatrix} 10 & 5 & 8 \\ 4 & 12 & 2 \\ 2 & 11 & 3 \end{bmatrix} \doteq \mathbf{R}$$

from which we are interested in the non-diagonal elements to construct a system of equations. To solve it, we define the matrices P and Q as

$$\mathbf{P} = \Phi_{\text{DC}}\mathbf{S_1}\Phi_{\text{DC}}^T$$
$$\mathbf{Q} = \Phi_{\text{DC}}\mathbf{S_2}\Phi_{\text{DC}}^T.$$

Now, we can rewrite the (i, j)th, $1 \leq i, j \leq k$, element of $\mathbf{R}$ as

$$P_{ij} + \lambda_i Q_{ij},$$

while the (j,i)th element is given by

$$P_{ji} + \lambda_j Q_{ji} = P_{ij} + \lambda_i Q_{ij}$$

thanks to the symmetry of P and Q. The data collector can now solve for the values of P_{ij} and Q_{ij} for all $i \neq j$. We obtain

$$\begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} P_{12} \\ Q_{12} \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix} \Rightarrow P_{12} = 1, Q_{12} = 4$$
$$\begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} P_{13} \\ Q_{13} \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \end{bmatrix} \Rightarrow P_{13} = 11, Q_{13} = 10$$
$$\begin{bmatrix} 1 & 4 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} P_{23} \\ Q_{23} \end{bmatrix} = \begin{bmatrix} 2 \\ 11 \end{bmatrix} \Rightarrow P_{23} = 12, Q_{23} = 4.$$

We can reconstruct the P and Q matrices as

$$\mathbf{P} = \begin{bmatrix} * & 1 & 11 \\ 1 & * & 12 \\ 11 & 12 & * \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} * & 4 & 10 \\ 4 & * & 4 \\ 10 & 4 & * \end{bmatrix}$$

To finish, we need to recover $\mathbf{S_1}$ and $\mathbf{S_2}$ from $\mathbf{P}$ and $\mathbf{Q}$; we can exploit the fact that the matrix is symmetric. Furthermore, we construct a system of 3 equations for each of the matrices as Φ_{DC} and Φ_{DC}^T are known to the data-collector. After solving, we obtain $\mathbf{u} = (2, 2, 3, 5, 6, 10)$.
Under the assumption that downloading one symbol consumes one second. We can calculate that the retrieval time is:

$$1[s] \times \#\text{downloaded symbols} + \text{decoding time}[s] = 6.009028673171997$$

We observe again that it is more efficient to repair a node than to recover a file. In conclusion, Regenerating codes are ideal for maintaining a high availability of the data so that the data is resilient to possible errors. However, the price to pay is the increased complexity to retrieve the original data. Whether to use these codes depends on how often the data must be accessed.

Lab 2: Interleaved Reed-Solomon Codes

Problem 1: Homogeneus IRS Code

Let $\mathcal{C}_1$ denote an IRS code consisting of $l = 2$ Reed–Solomon codes $\mathcal{RS}(15, 8)$ over $\mathbb{F}_{2^4}$. Assume that the codewords of $\mathcal{C}_1$ were transmitted over a bursty channel and we therefore choose a common error-locator polynomial for both Reed–Solomon (RS) codes in the decoding process.
We first wish to calculate the decoding radius $t_{\max,1}$ of $\mathcal{C}_1$ for collaborative IRS decoding. We know the result

$$t_{\max} \doteq \lfloor \frac{l}{l+1}(n-k) \rfloor = \lfloor \frac{l}{l+1}(d-1) \rfloor.$$

Plugging in the values we obtain $t_{\max} = 4$.
Furthermore, we would like to encode 4 random vectors over $\mathcal{GF}(2^4)$ of length $k = 8$, which turn to be

$$\mathbf{u_1} = (a^3 + a + 1, a^3 + a, a, a^3 + a^2, 1, a^2 + 1, a^2, a + 1)$$
$$\mathbf{u_2} = (a^2 + a, a^3 + a + 1, a, a^2, a^3 + a^2 + 1, a^3 + a^2 + 1, a^3 + a^2 + a, a + 1)$$
$$\mathbf{u_3} = (a^2, 0, a, a^2 + a, a^2 + a, 0, a^3 + 1, a^2 + a + 1)$$
$$\mathbf{u_4} = (0, 1, a^2 + a, a^3 + a + 1, a^3 + 1, 1, a^2 + a + 1, a^2 + a)$$

We use Sage to obtain the generator matrix

```
k = 8
p = 2
m = 4
q = p^m
F.<a> = GF(q, modulus = 'primitive')
# IRS code
n = q - 1 # code length
# Given evaluation points and a RS construction
L = [a^i for i in range(n)]
RS1 = codes.GeneralizedReedSolomonCode(L, k_1)
G1 = RS1.generator_matrix()
```

and encode the vectors

$$\mathbf{c_1} = \mathbf{u_1}\mathbf{G_1}$$
$$\mathbf{c_2} = \mathbf{u_2}\mathbf{G_1}$$
$$\mathbf{c_3} = \mathbf{u_3}\mathbf{G_1}$$
$$\mathbf{c_4} = \mathbf{u_4}\mathbf{G_1}.$$

Moreover, we define two error vectors e_1 and e_2 whose non-zero positions are

$$e_1^1 = a^2, e_1^5 = a^4, e_1^9 = a^{10}, e_1^{10} = a^3$$
$$e_2^1 = a^6, e_2^5 = a^7, e_2^9 = a^2, e_2^{10} = a^3.$$

And the received vectors become

$$\mathbf{r_1} = \mathbf{e_1} + \mathbf{c_1}$$
$$\mathbf{r_2} = \mathbf{e_2} + \mathbf{c_2}$$
$$\mathbf{r_3} = \mathbf{e_1} + \mathbf{c_3}$$
$$\mathbf{r_4} = \mathbf{e_2} + \mathbf{c_4}$$

whose syndrome is computed as

$$\mathbf{s_1} = \mathbf{r_1}\mathbf{H_1^T}$$
$$\mathbf{s_2} = \mathbf{r_2}\mathbf{H_1^T}$$
$$\mathbf{s_3} = \mathbf{r_3}\mathbf{H_1^T}$$
$$\mathbf{s_4} = \mathbf{r_4}\mathbf{H_1^T}.$$

While the error locations are equal, we observe that the syndromes $s_1 = s_3$ and $s_2 = s_4$ are pairwise equal. Which means that by using two different syndromes, we shall be able to decode further than the MDS decoding radius.
The next step is to implement a function *irs_decoding()* that returns the error positions given the syndrome. For this we construct the key-equation for joint error location

$$\mathbf{S_t} \cdot \mathbf{\Lambda_t} = \mathbf{T_t}$$
$$\begin{bmatrix} \mathbf{S_t^{(1)}} \\ \mathbf{S_t^{(2)}} \end{bmatrix} \cdot \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \end{bmatrix} = \begin{bmatrix} \mathbf{T_t^{(1)}} \\ \mathbf{T_t^{(2)}} \end{bmatrix}$$

where the matrices with syndrome coefficients are

$$\mathbf{S_t^{(j)}} = \begin{bmatrix} s_{(j)}^{t-1} & \cdots & s_{(j)}^{1} & s_{(j)}^{0} \\ s_{(j)}^{t} & \cdots & s_{(j)}^{2} & s_{(j)}^{1} \\ \vdots & \ddots & & \\ s_{(j)}^{2t-2} & \cdots & s_{(j)}^{t} & s_{(j)}^{t-1} \end{bmatrix}, \forall j = 1, \cdots, l.$$

Moreover, we use an iterative algorithm to find t. The algorithm begins by setting $t = \lfloor \frac{d-1}{2} \rfloor$ and $\mathbf{S_t}$ as above. Then, while $\mathbf{S_t}$ remains Singular, we decrease t and set $\mathbf{S_t}$ again. We iterate this process until $\mathbf{S_t}$ is non-singular. Plugging in the problem values we obtain t=4. We have now a linear system of equations which we can solve for $\mathbf{\Lambda}$. Finally, we set the error locator polynomial

$$\Lambda(x) = 1 + \sum_{i=1}^{4} \Lambda_i x^i = (a^2+a+1)x^4 + (a^3+a^2+a+1)x^3 + (a^3+a^2+a)x^2 + (a^3+1)x + 1$$

and iterate over $\Lambda(x)$ evaluated at the inverses of the RS's evaluation points as

$$\Lambda(\alpha_i^{-1}) = 0 \Leftrightarrow i \in \mathcal{E}.$$

This yields that the error locations are $\mathcal{E} = \{1, 5, 9, 10\}$.

Problem 2: Heterogeneous IRS Code

We extend $\mathcal{C}_1$ to a heterogeneous IRS code $\mathcal{C}_2$ of interleaving order $l = 3$ by adding an $\mathcal{RS}$ code over $\mathbb{F}_{2^4}$ of length $n = 15$ and dimension $k_2 = 6$. We assume one common error-locator polynomial for all three RS codes.
First, we compute the new decoding radius $t_{\max,1}$ of $\mathcal{C}_1$ for collaborative IRS decoding. This time, we need the average dimension $\overline{k}$. Plugging in the values we obtain $t_{\max} = 5$. We are able to correct an additional error. Because of the additional RS code, we also obtain an additional syndrome, which we denote $\mathbf{s_5}$ The decoding procedure follows the same process as in Problem 1, although this time each syndrome will not yield the same number of equations for the joint decoding. Furthermore, we observe that we don't need all the syndromes. Because $t_{\max} = 5$, we only need 5 equations. With $\mathbf{s_5}$ and $\mathbf{s_1}$ we already have 6 equations for decoding. Finally, the location of the errors are this time $\mathcal{E} = \{1, 5, 9, 10, 12\}$.

Problem 3: Virtual Interleaving (Power Decoding)

In order to decode the $\mathcal{RS}(15, 2)$ code over $\mathbb{F}_{2^4}$ beyond half the minimum distance, we extend it virtually to a heterogeneous IRS code. For this problem $k = 2$, the random information vector $u = (a^3+a^2+a, a^3+a^2+a)$, and the error

$$e = (a^5, 0, a^2, a^6, a^1, 0, a, 0, 0, a^5, 0, a^6, 0, a^7, a^9).$$

The received codeword is therefore,

$$y = (a^2{+}a, 1, a^2{+}a{+}1, a^3{+}a{+}1, a^3{+}a^2{+}1, a^3{+}a^2, a^3, a^2{+}a, a^3{+}a^2{+}1, a^3{+}a^2{+}a, a, a^3{+}1, a^3{+}a{+}1, a^3{+}a^2{+}a{+}1, a{+}1).$$

We begin by calculating the maximal decoding radii $t_{\max} = \max_{i=2,\cdots,6} t_{\max}^{(i)}$ and determine $i_{\max}$

$$\begin{aligned} i = 2, t_{max}(2) &= 8 \\ i = 3, t_{max}(3) &= 9 \\ i = 4, t_{max}(4) &= 9 \\ i = 5, t_{max}(5) &= 9 \\ i = 6, t_{max}(6) &= 9. \end{aligned}$$

We select $i = 3$, which is the smallest i producing $t_{\max}$. Further we implement the function *Power_dec()*, which makes use of *irs decoding()* to power decode the $\mathcal{RS}(15, 2)$. The resulting program listing is

```
def Power_dec(F, k, y, t_max, i_max):
    """
    Perform power decoding for a received RS codeword. i.e. construct a virtual IRS by
    powering the RS codeword, then do IRS decoding.
    Inputs:
        F: the finite field
```

```
        k: the dimension of the RS code
        y: the received RS codewords
        t_max: the maximum decoding radius
        i_max: the sufficient virtual interleaving order to obtain tmax
    """
    L = [a^i for i in range(n)] # evaluation points for RS code construction
    s_vec = [] # list of syndromes of virtual IRS code
    for i in range(1, i_max + 1):
    # The power decoding algorithm has to do the following steps:
        # Calculate element-wise power of the received word
        y_i = y.apply_map(lambda x: x^i)

        # Calculate the dimensions of virtual codes
        k_i = i * (k - 1) + 1
        RS_i = codes.GeneralizedReedSolomonCode(L, k_i)

        # Calculate original syndrome and virtual syndromes
        H_i = RS_i.parity_check_matrix()
        s_vec.append(y_i * H_i.transpose())
    # Calculate the error locations using irs_decoding()
    [errloc, t] = irs_decoding(F, t_max, s_vec)

    return errloc, t
```

where we have followed these steps

1. Calculate the element-wise powers of the received word up to interleaving order $i_{\max}$ to obtain y_i, for $i = 2, \cdots, i_{\max}$
2. Calculate the dimensions k_i, for $i = 2, \cdots, i_{\max}$ of the virtual codes;
3. Calculate the corresponding syndromes s_i, for $i = 2, \cdots, i_{\max}$ where s_1 denotes the syndrome of the original received word
4. Calculate the error positions by solving a system of equations

The *Power_dec()* returns the correct error positions, $\mathcal{E} = \{0, 2, 3, 4, 6, 9, 11, 13, 14\}$.
To conclude, we have observed that power decoding allows us to extend the decoding radius from 5, when using an heterogeneous IRS, to 9, when using power decoding. In the next problem we study the failure probability of this algorithm.

Decoding Failure of Power Decoding

To analyse the chances of decoding failure, we run power decoding over 1000 random error vectors of length 15 and weight $t_{\max}$ and calculate the failure probability.
We begin by generating the random error vectors. This task has to be done carefully since I haven't found a way to shuffle a vector in sage after sampling t_max non-zero values and padding until the required length n. For this reason, we sample n values and then correct the weight of the vector. If by some chances, the vector is still not of the desired weight, we continue generating until they satisfy the requirements. The results yield percentages in the range $(7, 10)\%$ which is not bad, but deviates from the expected value 6.667%. This can be because of some imperfections in the power decoding algorithm which may raise some exceptions which are treated as decoding failure erroneously.