Chapter 4

Codes as Kernels

In $\mathbb{F}_q^n$, just as in $\mathbb{R}^n$, we can calculate the **dot** (or scalar) **product** of two vectors: $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + \cdots x_n y_n$, and if $\mathbf{x} \cdot \mathbf{y} = 0$ we say that $\mathbf{x}$ and $\mathbf{y}$ are **orthogonal**. (But since we multiply and add mod q, a non-zero vector $\mathbf{x}$ can easily have $\mathbf{x} \cdot \mathbf{x} = 0$, and so be orthogonal to itself. [1])

The **kernel** of a linear map $f : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^m$ is the vectors which it sends to $\mathbf{0}$:
$\ker(f) = \{\mathbf{x} \in \mathbb{F}_q^n \mid f(\mathbf{x}) = \mathbf{0}\}$.

By combining these two ideas we get a new way to specify a code, and to find its minimum distance. We also find a much better algorithm for detecting (and sometimes correcting) errors.

4.1 Dual codes

If C is a code in $\mathbb{F}_q^n$, then 'C dual', written $C^\perp$, is the space of all vectors in $\mathbb{F}_q^n$ which are orthogonal to every codeword in C.

Definition 4.1. Let C be a code in $\mathbb{F}_q^n$. Then its **dual** $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v} \cdot \mathbf{u} = 0 \text{ for all } \mathbf{u} \in C\}$.

But we do not have to check $\mathbf{v}$ against *every* $\mathbf{u}$ in C, one by one.

Proposition 4.2. *If C has generator matrix G, then $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$.*

Proof. The rows of G are a basis for C, say $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$. Then certainly we require $\mathbf{v} \cdot \mathbf{b}_i = 0$ for every $1 \le i \le k$. But also, since the dot product is linear in the second input (in fact, in both), then if $\mathbf{u} = u_1\mathbf{b}_1 + \cdots + u_k\mathbf{b}_k$, we have $\mathbf{v} \cdot \mathbf{u} = u_1 \mathbf{v} \cdot \mathbf{b}_1 + \cdots + u_k \mathbf{v} \cdot \mathbf{b}_k$. Thus it is enough to check that $\mathbf{v} \cdot \mathbf{b}_i = 0$ for all the $\mathbf{b}_i$. We can do this by checking that

$$\mathbf{v} \cdot G^t = (v_1, \ldots, v_n) \begin{pmatrix} | & & | \\ \mathbf{b}_1 & \cdots & \mathbf{b}_k \\ | & & | \end{pmatrix} = (0, \ldots, 0) = \mathbf{0}.$$

[1] Thus the dot product is not (generally) an inner product on $\mathbb{F}_q$, so we cannot use $\mathbf{x} \cdot \mathbf{x}$ as a norm, and we do not have any idea of the length of a vector in $\mathbb{F}_q^n$.

$\square$

Multiplying by G^t is of course a linear map $f_{G^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^k$, and the $\mathbf{v}$s we want are exactly $\ker(f_{G^t})$, or the nullspace of G^t. Draw a picture of these spaces and maps: C and $C^\perp$ are both in $\mathbb{F}_q^n$. They will intersect, at least in $\mathbf{0}$. C is the image of the map f_G coming from $\mathbb{F}_q^k$; $C^\perp$ is the kernel of the map f_{G^t} going to $\mathbb{F}_q^k$.

Proposition 4.3. *Let C be a code in $\mathbb{F}_q^n$. Then $C^\perp$ is a code, and if* $\dim(C) = k$*, then* $\dim(C^\perp) = n - k$*.*

Proof. Since f_{G^t} is a linear map, its kernel is a (linear) subspace, and so a (linear) code. The dimension of the kernel is the 'nullity' of the map, and we know[2] that for the linear map $f_{G^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^k$, we have rank + nullity = $\dim(\mathbb{F}_q^n) = n$. The rank of the map is the row-rank of G^t; in fact row- or column-rank of G or G^t are all four equal to k. So the nullity is $n - k$. $\square$

The 'dual' idea appears in many different areas of mathematics, but it is usually, as in this case, a 'self-inverse' operation:

Proposition 4.4. *For $C \subseteq \mathbb{F}_q^n$, $(C^\perp)^\perp = C$.*

Proof. If C has basis $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ and $C^\perp$ has basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_{n-k}\}$, then we know that for any $\mathbf{u}_i$ and $\mathbf{v}_j$ we have $\mathbf{v}_j \cdot \mathbf{u}_i = 0$. But this also shows that every $\mathbf{u}_i \in (C^\perp)^\perp$, so $C \subseteq (C^\perp)^\perp$. By Proposition 4.3 we know that $\dim(C^\perp)^\perp = n - (n-k) = k = \dim(C)$, so they must be equal. $\square$

Suppose C has generator matrix G with rows $\mathbf{u}_1 \ldots \mathbf{u}_k$, how can we find out more about $C^\perp$? We would like to find a basis, and thus a generator matrix for it. The vectors in $C^\perp$ are those $\mathbf{v}$ such that $\mathbf{v}G^t = (v_1, \ldots, v_n)\begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ | & & | \end{pmatrix} = \mathbf{0}$. As in Section 3.3, G^t is not invertible, but we can solve the k equations $\mathbf{v} \cdot \mathbf{u}_i = 0$. Again, one way to do this is to take transposes, $(G^t)^t \mathbf{v}^t = G\mathbf{v}^t = \mathbf{0}$, and then row-reduce the augmented matrix $(G \mid \mathbf{0})$. Once we have G in RREF, we can find a basis for $C^\perp$ from the new, simpler equations.

The following algorithm "automates" this process, working straight from G in RREF to the basis for $C^\perp$.

Algorithm: Finding a basis for a dual code
Suppose that C has a generator matrix $G = (g_{ij}) \in M_{k,n}(\mathbb{F}_q)$, and G is in RREF.

- Let $L = \{1 \le j \le n \mid G \text{ has a leading 1 in column } j\}$.
- For each $1 \le j \le n$, $j \notin L$, make a vector $\mathbf{v}_j$ as follows:
  * for $m \notin L$: the m^{th} entry of $\mathbf{v}_j$ is 1 if $m = j$, 0 otherwise.
  ** Fill in the other entries of $\mathbf{v}_j$ (left to right) as $-g_{1j}, \ldots, -g_{kj}$.
- These $n - k$ vectors $\mathbf{v}_j$ are a basis for $C^\perp$.

[2] Strictly, in Linear Algebra you only proved this for vector spaces over $\mathbb{R}$, but it is true in general.

Example 30. Let C be the code in $\mathbb{F}_5^7$ with generator matrix

$$G = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix}$$

To find a basis for $C^\perp$ we first note that G is already in RREF, and the leading 1s are in columns 1, 3, and 6. Thus $L = \{1, 3, 6\}$, and we make vectors for a basis $\{\mathbf{v}_2, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_7\}$. Step * fills in the "non-L" entries, so that the incomplete vectors look a bit like a standard basis:

$$\mathbf{v}_2 = (\ \ , 1, \ \ , 0, 0, \ \ , 0) \quad \mathbf{v}_4 = (\ \ , 0, \ \ , 1, 0, \ \ , 0)$$
$$\mathbf{v}_5 = (\ \ , 0, \ \ , 0, 1, \ \ , 0) \quad \mathbf{v}_7 = (\ \ , 0, \ \ , 0, 0, \ \ , 1)$$

Then step ** uses the corresponding columns to complete the vectors. For example, since column 7 is (0,3,4), we complete $\mathbf{v}_7$ with the additive inverses of these: 0, 2, and 1. So we have

$$\mathbf{v}_2 = (3, 1, 0, 0, 0, 0, 0) \quad \mathbf{v}_4 = (2, 0, 4, 1, 0, 0, 0)$$
$$\mathbf{v}_5 = (1, 0, 3, 0, 1, 0, 0) \quad \mathbf{v}_7 = (0, 0, 2, 0, 0, 1, 1)$$

△

Notice that, since G is in RREF, in column j all the entries after the j^{th} will be 0. This is why, in step **, we find that $\mathbf{v}_j$ is all zeros after the j^{th} entry (which is the the 1 from step *).

We will not write out a formal proof that this algorithm works: it is a straightforward calculation but involves a lot of notation. But, having found your $\mathbf{v}_j$, it is easy to check they are indeed a basis: Firstly, step * ensures that each $\mathbf{v}_j$ has a 1 in column j, where all the others have 0, so the vectors are linearly independent. Secondly, to see they are in $\ker(f_{G^t})$, check that each $\mathbf{v}_j G^t = \mathbf{0}$. This shows why we do step **: everything cancels out just right. Since we know that $\dim(\ker(f_{G^t})) = n - k$, this proves we have a basis.

We can now make a generator-matrix H for $C^\perp$, by taking the $\mathbf{v}_j$, in order, as rows. In general, H is not in RREF, but we can row-reduce it if necessary. As in Section 3.4, if G is in standard form, the process is even easier:

Proposition 4.5. *If $C \subseteq \mathbb{F}_q^n$ has generator-matrix $G = (I_k \mid A)$, then a generator-matrix for $C^\perp$ is $H = (-A^t \mid I_{n-k})$.*

Again this is fiddly to prove in general, but becomes obvious with examples; this H is exactly the generator-matrix for $C^\perp$ produced by the algorithm above. Again, H can be row-reduced to RREF, but not necessarily to standard form.

4.2 Check-matrices

In the last section we showed that $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$, where G is a generator-matrix for C. But if we then find H, a generator-matrix for $C^\perp$, it is also true that

$C = (C^\perp)^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}H^t = \mathbf{0}\}$. This is a very useful new way to specify any linear code.

Definition 4.6. Let $H \in M_{n-k,n}(\mathbb{F}_q)$ have linearly independent rows, and let $C = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}H^t = \mathbf{0}\}$. Then H is a **check-matrix** for C.

The name makes sense: we use H (or, in practice, its transpose) to 'check' whether $\mathbf{v}$ is in C or not. Notice that the rank of the map f_{H^t} is the rank of the matrix H^t, which is $n-k$. So the dimension of the code C defined in this way, which is the nullity of f_{H^t}, is $n-(n-k) = k$.

Proposition 4.7. *If the code C has generator-matrix G and check-matrix H, then $C^\perp$ has check-matrix G and generator-matrix H.*

Proof. Suppose $\dim(C) = k$. Then G has k rows, and H has $n-k$ rows. Also, by Proposition 4.3, $\dim(C^\perp) = n-k$.

The rows of G are linearly independent, and by Prop. 4.1 we know that $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$, so G is a check-matrix for $C^\perp$.

The rows of H are orthogonal to every codeword in C, so they are in $C^\perp$. They are also linearly independent, and there are $n-k$ of them, so they form a basis for $C^\perp$. □

The relationships among a code, its dual, and their respective generator- and check-matrices can be clarified by drawing pictures of the spaces and maps involved. They can also be very usefully summarised in the following table:

	C	$C^\perp$
Generator-matrix	G	H
Check-matrix	H	G

In the last section we discussed an algorithm which finds the basis of a dual space. So it finds H from G. But this means it also finds a check-matrix for C from its generator-matrix. Or, if we are given the check-matrix H for C, we can regard H as a generator-matrix for $C^\perp$, and then use the same algorithm to find a generator-matrix for $C = (C^\perp)^\perp$. So we can use the algorithm to move either horizontally or vertically on the table; for this reason we can call it "the $G \leftrightarrow H$ algorithm".

If the matrix you have (either G or H) is in standard form $(I_k \mid A)$, the simpler algorithm of Proposition 4.5 can also be used to find the other one. Moreover, if we have H or G in form $(A \mid I_k)$, we can regard it as a check-matrix corresponding, by Proposition 4.5, to a generator matrix of form $(I_{n-k} \mid -A^t)$. (See Q47) For this reason, $(A \mid I_k)$ can be regarded as standard form for check-matrices. But since every check-matrix for a code C is also a generator-matrix for $C^\perp$ this could be confusing; it seems best to specify each time whether we mean standard form $(I_k \mid A)$ or standard form $(A \mid I_k)$.

Example 31. Let $C = \{\mathbf{v} \in \mathbb{F}_2^5 \mid \mathbf{v}H^t = \mathbf{0}\}$, with the single-row check-matrix $H = (1\,1\,1\,1\,1)$. Then the codewords of C are $\mathbf{c} = (c_1, \ldots, c_5)$ such that $c_1 + \cdots + c_5 = 0$, so those with even weight. Thus H performs a simple "parity check"; to make a codeword we can choose 0 or 1 freely for any four of the entries, but the final entry must make the weight even. To find a basis for this code, since H is in standard form $(I_1 \mid A)$, we can use Proposition 4.5 and write down a generator-matrix $G_1 = (A^t \mid I_4)$. (For a binary code, $A = -A$.) But H is also in form $(A \mid I_1)$, so $G_2 = (I_4 \mid A^t)$ is another generator matrix. In fact, $G_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$ is the RREF form of $G_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$. △

What if $C = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}A^t = \mathbf{0}\}$, but $A \in M_{m,n}(F_q)$ does not have linearly independent rows? Or perhaps we do not know whether its row are independent or not? It is still true that $C = \ker(f_{A^t})$, and we might call A an "acting check-matrix" for C - it is doing the checking job, but it may not be fully qualified. Then, also, the rows of A are a spanning set for $C^\perp = \{\mathbf{v}A \mid \mathbf{v} \in \mathbb{F}_q^m\} = \mathrm{im}(f_A)$, but may not be a basis. We could similarly call A an "acting generator-matrix" for $C^\perp$.

Of course, using a check-matrix (or an acting check-matrix) to define a code is only a convenient new notation for a very familiar idea. You are familiar with defining a subspace using equations in the coordinates.

Example 32. If $H = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 1 \end{pmatrix} \in M_{2,3}(\mathbb{F}_5)$, and $C = \{\mathbf{v} \in \mathbb{F}_5^3 \mid \mathbf{v}H^t = \mathbf{0}\}$, then $C = \{(v_1, v_2, v_3) \in \mathbb{F}_5^3 \mid v_1 + 2v_2 + 3v_3 = 0 \text{ and } 4v_2 + v_3 = 0\}$. △

To solve such sets of equations, you would manipulate them in ways which correspond to elementary row operations on the check-matrix. This confirms that (as with generator-matrices) row-reducing a check-matrix for a code C gives another check-matrix for C.

4.3 Syndrome Decoding

In medicine, a "syndrome" is a collection of symptoms or characteristics which occur together. They are often apparently unrelated, but are assumed to have a single cause; over the last few decades, a genetic cause has been identified for many syndromes.

Similarly, the "syndrome" of a received word is useful evidence as to what error it may have suffered. We find the syndrome using the check-matrix. Thus, just as a generator-matrix makes it easy for a sender to encode a message, a check-matrix can help a receiver to decode a received word.

Definition 4.8. Suppose a code C has check-matrix H, so $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}H^t = 0\}$. For any received word $\mathbf{y}$, its **syndrome** is $S(\mathbf{y}) = \mathbf{y}H^t$.

Thus $S(\mathbf{y}) = \mathbf{0}$ if and only if $\mathbf{y}$ is a codeword. In this case we assume that it is in fact the one which was sent and no error-vector was added. In this way, the syndrome detects errors.

But a non-zero syndrome can also help to correct errors, by helping us to guess an error which is likely to have occurred. We know that $f_{H^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^{n-k}$ is a linear map. So if $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where $\mathbf{c} \in C$, then $S(\mathbf{y}) = S(\mathbf{c}) + S(\mathbf{e}) = \mathbf{0} + S(\mathbf{e}) = S(\mathbf{e})$. So the syndrome of the received word is the same as that of the error-vector $\mathbf{e}$. The syndrome is able to ignore the codeword and just "pick out" the error.

Unfortunately knowing $S(\mathbf{e})$ does not tell us $\mathbf{e}$, because the syndrome map f_{H^t} is not injective: two different errors can have the same syndrome. The following algorithm associates each possible syndrome with a single, likely, error-vector.

Algorithm: Syndrome decoding

Let C be a q-ary $[n, k]$ code, with check matrix $H \in M_{n-k,n}(\mathbb{F}_q)$,
so $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}H^t = \mathbf{0}\}$.

Construction of a syndrome look-up table

1. List the elements of $\mathbb{F}_q^n$ in non-decreasing order of weight.
2. Set up a table with two columns: **syndrome** $S(\mathbf{x})$ | **error-vector** $\mathbf{x}$.
3. Let $\mathbf{x}$ be the next element in the list and calculate $S(\mathbf{x})$.
4. If $S(\mathbf{x})$ is in the syndrome column already, do nothing.
   If it is not, write a new row: $S(\mathbf{x}) \mid \mathbf{x}$.
5. Repeat (3) and (4) until you have q^{n-k} rows.

Decoding (error 'correction') Having received a word $\mathbf{y}$,

1. Compute $S(\mathbf{y}) = \mathbf{y}H^t$.
2. Find $S(\mathbf{y})$ in the syndrome column.
3. Find the error-vector $\mathbf{x}$ that is in the same row.
4. Decode $\mathbf{y}$ to $\mathbf{y} - \mathbf{x}$.

Example 33. Let $C_1 = \{\mathbf{x} \in \mathbb{F}_2^4 \mid \mathbf{x}H^t = \mathbf{0}\}$, where $H = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ is a check-matrix for C_1. We calculate syndromes, starting with words of weight 0, then 1, then 2: $S(0,0,0,0) = (0,0)$, $S(1,0,0,0) = (1,0)$, $S(0,1,0,0) = (1,0)$, $S(0,0,1,0) = (0,1)$, $S(0,0,0,1) = (0,1)$, $S(1,1,0,0) = (0,0)$, $S(1,0,1,0) = (1,1)$, $S(1,0,0,1) = (1,1)\ldots$

Omitting the repeated syndromes, we make the following look-up table:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(0,0)	(0,0,0,0)
(1,0)	(1,0,0,0)
(0,1)	(0,0,1,0)
(1,1)	(1,0,1,0)

We can stop here, as we have 2^{4-2} rows; equivalently, we have every possible syndrome.

Now suppose we receive $\mathbf{y}_1 = (1,1,0,1)$. Then $S(\mathbf{y}_1) = (0,1)$, so the table says that the error-vector was (0,0,1,0), and we decode to $(1,1,0,1) - (0,0,1,0) = (1,1,1,1) = \mathbf{c}_1$. Similarly, $\mathbf{y}_2 = (0,1,0,0)$ decodes to $\mathbf{c}_2 = (1,1,0,0)$. △

By the theory, both these $\mathbf{c}_i$ should be in C_1; we can check this by finding $S(\mathbf{c}_i)$. We could also use the "$G \leftrightarrow H$ algorithm" to find a generator matrix G for C. Surprisingly, we find that $G = H$, so $C_1 = C_1^{\perp}$; C_1 is 'self-dual' [3]. So this is actually the code for which, in Section 2.3, we made this decoding array:

$(0,0,0,0)$	$(1,1,0,0)$	$(0,0,1,1)$	$(1,1,1,1)$
$(1,0,0,0)$	$(0,1,0,0)$	$(1,0,1,1)$	$(0,1,1,1)$
$(0,0,1,0)$	$(1,1,1,0)$	$(0,0,0,1)$	$(1,1,0,1)$
$(1,0,1,0)$	$(0,1,1,0)$	$(1,0,0,1$	$(0,1,0,1)$

We see that the $\mathbf{c}_i$ are in the top row, which lists the code. Also, the left-hand column of the array matches the error-vector column of the look-up table; these are the (guessed) errors we will subtract. And certainly this array gives the same decoding as the look-up table for (1,1,0,1) and (0,0,1,0). We can also see a examples of the following:

Proposition 4.9. *Two words are in the same row of a decoding array if and only if they have the same syndrome.*

Proof. In general, finding the two words in the array (see below) expresses them as $\mathbf{y}_1 = \mathbf{c}_1 + \mathbf{x}_1$ and $\mathbf{y}_2 = \mathbf{c}_2 + \mathbf{x}_2$, with $\mathbf{c}_1 \in C$, and we know already that $S(\mathbf{y}_1) = S(\mathbf{x}_1)$ and $S(\mathbf{y}_2) = S(\mathbf{x}_2)$.

$\mathbf{0}$	$\mathbf{c}_2$	$\mathbf{c}_1$
$\mathbf{x}_1$		$\mathbf{y}_1$
$\mathbf{x}_2$	$\mathbf{y}_2$	

If $\mathbf{y}_1$ and $\mathbf{y}_2$ are in the same row, then $\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{x}$, so $S(\mathbf{y}_1) = S(\mathbf{y}_2) = S(\mathbf{x})$.

Conversely, if $S(\mathbf{y}_1) = S(\mathbf{y}_2)$ then $S(\mathbf{y}_1 - \mathbf{y}_2) = S(\mathbf{y}_1) - S(\mathbf{y}_2) = \mathbf{0}$, so $\mathbf{y}_1 - \mathbf{y}_2 = \mathbf{c} \in C$. Then $\mathbf{y}_1 = \mathbf{y}_2 + \mathbf{c} = \mathbf{x}_2 + \mathbf{c}_2 + \mathbf{c}$. Since $\mathbf{c}_2 + \mathbf{c} \in C$, it must be in the top row, so $\mathbf{y}_1$ is in $\mathbf{x}_2$'s row. □

In effect, syndrome decoding is just a more efficient way to do array decoding; without either making or searching through the array, finding $S(\mathbf{y})$ tells us which row of the array $\mathbf{y}$ would be on. So it follows from Proposition 2.10 that syndrome decoding, also, is nearest-neighbour decoding. (We can also prove this directly: Q53)

As with the array, there is some choice in the construction of the syndrome look-up table; it comes in the initial ordering of the words of $\mathbb{F}_q^n$. If this is different, we may get a different column of error-vectors to subtract, which will certainly result in different decoding of some words.

[3] This could not happen over $\mathbb{R}$.

Example 34. Let $C_2 = \{\mathbf{x} \in \mathbb{F}_3^3 \mid \mathbf{x}H^t = 0\}$, where $H = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{pmatrix}$ is a check-matrix for C_2. Then this is one possible syndrome look-up table:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(0,0)	(0,0,0)
(1,0)	(1,0,0)
(2,0)	(2,0,0)
(0,1)	(0,1,0)
(0,2)	(0,2,0)
(2,2)	(0,0,1)
(1,1)	(0,0,2)
(1,2)	(1,2,0)
(2,1)	(1,0,2)

Here we have used every possible $\mathbf{x}$ of weight 1, so the order in which we considered them did not matter. But the last two lines could instead be:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(2,1)	(0,2,1)
(1,2)	(2,0,1)

We can conclude that any error-vector of weight ≤ 1, but only some errors of weight 2, will be correctly identified and subtracted. Which errors of weight 2 are correctly subtracted, and which are not, depends on which table we use. For this reason we might decide to practice incomplete decoding: cut the table short, and if we receive a word with syndrome (1,2) or (2,1) ask for retransmission. △

Looking back to C_1, we see that the table lists only some $\mathbf{x}$'s of weight 1, so we cannot be sure of reliably correcting even error-vectors of weight 1. But we knew this: $d(C_1) = 2$, so by Proposition 1.7 we will detect a single symbol-error, but nearest-neighbour decoding may not correct it.

On the other hand, using Proposition 4.5 (or by guessing and checking) we find that $C_2 = \{(0,0,0), (1,1,1), (2,2,2)\}$, so $d(C_2) = 3$ and we can indeed reliably correct one symbol-error, but not two. Equivalently we know that for this code, spheres $S(\mathbf{c}, 1)$ around the codewords are disjoint, but the $S(\mathbf{c}, 2)$ intersect. (Q24 and 25 consider alternative arrays for this code.)

The examples we've discussed so far have all been for binary or ternary codes. For codes over a larger alphabet, the number of rows in a syndrome table can get quite large. However, since the syndrome is a linear map on $\mathbb{F}_q^n$, we have $S(\lambda\mathbf{y}) = \lambda S(\mathbf{y})$ for any non-zero $\lambda \in \mathbb{F}_q^n$ – we can see this explicitly in Example 34 above.

For codes with $q > 2$, we can therefore define a *reduced* syndrome table, where we only add new syndromes to our table if they aren't of the form $\lambda S(\mathbf{x})$, for any non-zero $\lambda \in \mathbb{F}_q$, and any $S(\mathbf{x})$ already in our table. To decode a received word $\mathbf{y}$, we then calculate $S(\mathbf{y})$ as normal, but now we need to find the row such that $\lambda S(\mathbf{y})$ is in the first column, for some non-zero λ which we need to calculate. We then decode $\mathbf{y}$ to $\mathbf{y} - \lambda\mathbf{x}$, where $\mathbf{x}$ is the error vector in the corresponding row of our table. See Q52 for an example of this idea.

4.4 Minimum distance from a check-matrix

In the last section, $d(C)$ turned out to be relevant to the reliability of our syndrome look-up table. But to find it, we had first to find the words of the code. We will now establish a way to get $d(C)$ directly from a check-matrix, which links up many of the ideas so far.

In fact, it only needs to be an "acting check-matrix". We start with the following:

Lemma 4.10. *For some $A \in M_{m,n}(\mathbb{F}_q)$, let $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}A^t = \mathbf{0}\}$. Then:*
There are d columns of A which are linearly dependent
$\iff$ *there is some codeword $\mathbf{c} \in C$ with $0 < w(\mathbf{c}) \leq d$.*

Proof. Let the columns of A be $\mathbf{a}_1, \ldots, \mathbf{a}_n$.

$\implies$ Suppose we have d linearly dependent columns, $\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_d}$. This means there exist $\lambda_1, \lambda_2, \ldots, \lambda_d$ in $\mathbb{F}_q$, not all 0, such that $\lambda_1 \mathbf{a}_{i_1} + \cdots + \lambda_d \mathbf{a}_{i_d} = \mathbf{0}$. Now let $\mathbf{c}$ be a word with λ_j in position i_j, 0 elsewhere. Then $0 < w(\mathbf{c}) \leq d$. But also, when multiplying $\mathbf{c}A^t$, each λ_j picks out row i_j of A^t, so

$$\mathbf{c}A^t = (0, \ldots 0, \lambda_1, 0, \ldots, 0, \lambda_d, 0, \ldots, 0) \begin{pmatrix} & \vdots & \\ - & \mathbf{a}_{i_1} & - \\ & \vdots & \\ - & \mathbf{a}_{i_d} & - \\ & \vdots & \end{pmatrix} = \lambda_1 \mathbf{a}_{i_1} + \cdots + \lambda_d \mathbf{a}_{i_d} = \mathbf{0}.$$

So $\mathbf{c} \in C$.

$\impliedby$ If $\mathbf{c} = (c_1, c_2, \ldots, c_n) \in C$, and $0 < w(\mathbf{c}) \leq d$, we know that $c_1 \mathbf{a}_1 + \cdots + c_n \mathbf{a}_n = \mathbf{c}A^t = 0$, and that between 1 and d of the $\mathbf{c}_i$ are non-zero. If we choose $c_{i_1}, \ldots, c_{i_d}$ to include all the non-zero c_i, then we still have $c_{i_1}\mathbf{a}_{i_1} + \cdots + c_{i_d}\mathbf{a}_{i_d} = 0$, with not all $c_i = 0$. Thus $\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_d}$ are linearly dependent. $\square$

Example 35. Let $C = \{\mathbf{x} \in \mathbb{F}_7^5 \mid \mathbf{x}A^t = 0\}$, where $A = \begin{pmatrix} 3 & 1 & 1 & 4 & 1 \\ 2 & 2 & 5 & 1 & 4 \\ 6 & 3 & 5 & 0 & 2 \end{pmatrix} \in M_{3,5}(\mathbb{F}_7)$.

Because $(0, 1, 2, 0, 4) \begin{pmatrix} 3 & 2 & 6 \\ 1 & 2 & 3 \\ 1 & 5 & 5 \\ 4 & 1 & 0 \\ 1 & 4 & 2 \end{pmatrix} = (0, 0, 0)$, we know two things:

- $(0, 1, 2, 0, 4) \in C$, so C contains a codeword of weight 3.
- $1(1, 2, 3) + 2(1, 5, 5) + 4(1, 2, 4) = (0, 0, 0)$, so A has 3 columns which are linearly dependent.

$\triangle$

Theorem 4.11. *For some $A \in M_{m,n}(\mathbb{F}_q)$, let $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}A^t = \mathbf{0}\}$. Then there is some set of $d(C)$ columns of A which are linearly dependent, but any $d(C) - 1$ columns of A are linearly independent.*

Proof. For a linear code, by Proposition 2.7 $d(C) = min\{w(\mathbf{c}) \mid \mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}\}$. So we know:

- There is some $\mathbf{c} \in C$ with $w(\mathbf{c}) = d(C)$. So by Lemma 4.10 there are $d(C)$ columns which are linearly dependent.
- There is no $\mathbf{c} \in C$ with $w(\mathbf{c}) \leq d(C) - 1$. So by Lemma 4.10 there is no set of $d(C) - 1$ columns which are linearly dependent.

□

This theorem is mostly used in reverse: We find the number d such that A has a set of d dependent columns, but no smaller such sets. Then we conclude that d is the minimum distance of the code. One can remember the theorem as something like "$d(C)$ is the size of a smallest set of linearly dependent columns in the check-matrix".

Example 36. For the code C in the example above, we have found that columns 2, 3 and 5 are linearly dependent. But this only tells us that $d(C) \leq 3$. To be sure that $d(C) = 3$, we need also to check that there are no linearly dependent pairs of columns, that is, no column is a multiple of another. For many of the $\binom{5}{2}$ pairs this is easy: its zero means that column 4 is not a multiple of any other, and (since they are not identical) the top entry 1 in columns 2, 3, and 5 means they cannot be multiples of each other. It remains to check that column 1 is not a multiple of column 2, 3 or 5. It is not, so $d(C) = 3$. △