# 符号理論

- coding theory -

Presented by
Shinji Akematsu & Keisuke Sasaki

# 目次

# はじめに

本学習資料は, 釧路高専情報工学科第 5 学年の必修科目である情報論（符号理論）の試験にスムーズに対応するために作成された学習資料です. なるべく分かりやすさを重視し, あまり拘泥しすぎると理解を損なってしまう恐れがある数学的に厳密な証明等は, あえて省いている部分が多くあります. 数学的に厳密な符号理論を学びたいという方は, 別の参考書[*1]をご参照ください.

この学習資料は, 明松 真司, 佐々木 佳祐の共著によって作成しました. 章ごとの執筆担当は, 以下の通りです.

**第 1 章　誤り訂正理論の概要**　- 佐々木 佳祐 著
**第 2 章　線形符号**　- 佐々木 佳祐 著
**第 3 章　DFT 符号と DFT 行列**　- 明松 真司 著
**第 4 章　有限体**　- 明松 真司 著
**第 5 章　巡回符号**　- 佐々木 佳祐 著
**第 6 章　リード・ソロモン符号**　- 佐々木佳祐 著
**第 7 章　BCH 符号**　- 佐々木佳祐 著
**第 8 章　たたみ込み符号**　- 明松 真司 著

本学習資料において, 何か誤りを検出した場合, 以下のメールアドレスまでご連絡下さい.

j1701@cc.kushiro-ct.ac.jp　or　j1714@cc.kushiro-ct.ac.jp

この 1 冊の冊子が, 少しでも勉強のお役に立てることを祈っています.

[*1] 「誤り訂正符号入門」 J. ユステセン, T. ホーホルト著 / 阪田 省二郎, 栗原 正純 訳 (森北出版) を推薦します.

# 第1章

# 誤り訂正理論の概要

通信分野において, 誤りというものは切ろうとしても完全に切り離すことは難しい. 誤りを無くすのが不可能であるなら, 訂正すれば良いのである. 誤り訂正理論では, そんな誤りの検出, 訂正がどのように行われるかを勉強する.

## 1.1 ブロック符号の定義

ここでは, まず符号とはなんたるか, の定義を行う.

**Def : 符号語とブロック符号**

ある正の整数 $M$ に対して, $M$ 個の符号語の集合を**ブロック符号**と呼び, $C$ とかく.

$$C = \{c_1, c_2, \cdots, c_M\}$$

**符号語** $c_i$ は, $n$ 次元ベクトル

$$c_i = (c_{i0}, c_{i1}, \cdots, c_{i,n-1})$$

で表される.

$c_{ij}$ は $q$ 個の記号からなるアルファベットの元だが, 今は $\{0, 1\}$ のどちらかの値をとるものとする.

この記号アルファベットには**体**の構造が付加され, その結果として, 符号語に対する演算が可能になる.

この符号 $c$ 同士の演算は 2 を法として行われ, この加算のことは “排他的論理和” や “XOR” と呼ばれる.

## 1.2 単純な誤り検出, 訂正

まずは簡単な誤りを検出, 訂正を通して, 符号理論の大まかな概要を掴んでいただきたい.

### 1.2.1 パリティ検査

$k$ 個のビット列が与えられたときに, そのビット列の後ろにある条件を満たすようにビットを付加する. これを**パリティ検査**といい, このときの $k$ 個のビット列を**情報ビット (系列)**, 付加されたビットを含んだ全てのビット列を**符号ビット (系列)** という..

**例** $c_0 c_1 \cdots c_{k-1}$ というビット列 (情報ビット) に $c_k$ というビットを付加する. $c_k$ は,

$$c_0 + c_1 + \cdots + c_{k-1} + c_k = 0$$

を満たすように設定する (この式は, $c_0$ から $c_k$ までのビットの中に, 1 が偶数個ある, ということを意味している). これを偶数パリティ検査という. このビット列の誤り検査法は単純で, 先ほどの加算の結果が 1 になれば, どこかで誤りが生じていることがわかる.

**例** (1011) というビット列に偶数パリティ検査ビットを付加すると (10111) となる. 第 3 ビットが誤って受信されると (10011) となり, 1 の数が奇数になっているので誤っているとわかる. しかし, これに加えて第 4 ビットが誤って受信されると (10001) となるので, 1 の数が偶数になり, 誤っているのに正しいものとして受理されてしまう (誤りが検出されない).

この例から, パリティ検査で検出できる誤りのビット数は 1 ということがわかる. ここで疑問なのが,「訂正はどうなのか」という話であるが, この方法ではどこが誤っているかを明示できないので, 誤り訂正はできない. 誤り訂正を可能とするために, 次の方法が考え出された.

### 1.2.2　垂直水平パリティ検査

**垂直水平パリティ検査**は, 符号を行列の形で表し, それぞれの列と行に対してパリティ検査ビットを付加するものである.

**例** 以下のように, $k \times l$ 個の情報ビットを並べ, それぞれの列, 行にパリティ検査ビットを付加する.

| | | | | |
|---|---|---|---|---|
| $c_{00}$ | $c_{01}$ | $\cdots$ | $c_{0,l-1}$ | $c_{0,l}$ |
| $c_{10}$ | $c_{11}$ | $\cdots$ | $c_{1,l-1}$ | $c_{1,l}$ |
| $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $c_{k-1,0}$ | $c_{k-1,1}$ | $\cdots$ | $c_{k-1,l-1}$ | $c_{k-1,l}$ |
| $c_{k,0}$ | $c_{k,1}$ | $\cdots$ | $c_{k,l-1}$ | $c_{k,l}$ |

この方法では何ビットの誤りを検出, 訂正できるのか. 以下の例を元に考える.

**例** 8 ビットの情報ビットを 2 次元に配置する.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

これにパリティ検査ビットを付加すると

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

ここで, 誤って受信されたビットに ˜ を付加する.

1 ビット誤りの場合

$$\begin{pmatrix} 1 & 0 & \tilde{0} & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

1 行目と 3 列目に異常が見られ, 誤りが検出でき, 更に誤りの箇所も判明する.

2 ビット誤りの場合

$$\begin{pmatrix} 1 & 0 & \tilde{0} & 0 & 0 \\ \tilde{1} & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

これも先ほど同様, 検出が可能であり, 誤り箇所も判明している.

3 ビット誤りの場合

$$\begin{pmatrix} 1 & 0 & \tilde{0} & 0 & 0 \\ \tilde{1} & 0 & 1 & 0 & 1 \\ \tilde{0} & 0 & 0 & 0 & 1 \end{pmatrix}$$

全ての行と 3 列目で誤りが検出できるが, 誤りの箇所がどこかは判別できない.

4 ビット誤りの場合

これまで同様, 誤りビットの位置が列, 行ともに被らなければ検出は可能となるが

$$\begin{pmatrix} 1 & 0 & \tilde{0} & 0 & \tilde{1} \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & \tilde{1} & 0 & \tilde{0} \end{pmatrix}$$

のような誤り方は検出ができない. ここまでの例より, 3 ビットの誤りまでは確実に検出でき, 1 ビットの誤りは確実に修正が可能であることがわかる.

## 1.3 符号化率

**符号化率**とは,「情報ビットの数」に対する「誤り訂正符号化後のビット数」の比のことをいい, $R = \frac{k}{n}$(k:情報ビット数, n:符号ビット数) で表す. R の値が高いということは,「より少ないビットで符号化を行うことができる」ことであり, それは即ち処理の少なさを意味する.(=負担が少ない.) だが, 符号化率を高めると誤り率が大きくなるというネックがあることにも留意されたい.

## 1.4 ハミング重みとハミング距離

ここでは, 符号の誤り訂正能力を決定するための概念を導入する.

**Def : ハミング重み**

**ハミング重み**とはベクトルの零でない成分の数で,$w_H(x)$ とかく. 単に重みということもある.

**Def : ハミング距離**

**ハミング距離**とは 2 つのベクトル $x, y$ の互いに異なる成分の数で,$d_H(x, y)$ とかく. 単に距離ということもある. ハミング距離は以下のような 3 つの性質をもつ.

- $d_H(x, y) = 0 \iff x = y$
- $d_H(x, y) = d_H(y, x)$
- $d_H(x, y) \leq d_H(x, z) + d_H(z, y)$ (三角不等式)

これらの性質により, $d_H$ は線形空間における距離の性質をもつことがわかる.

**例**　$C = \{c_1, c_2, c_3\}, c_1 = (0010), c_2 = (1100), c_3 = (0110)$
とすると, それぞれの重みと距離は

$$\begin{aligned} w_H(c_1) &= 1 \\ w_H(c_2) &= 2 \\ w_H(c_3) &= 2 \\ d_H(c_1, c_2) &= 3 \\ d_H(c_1, c_3) &= 1 \\ d_H(c_2, c_3) &= 2 \end{aligned}$$

である.

また, これらの重み, 距離の最小値はそれぞれ最小ハミング重み, 最小ハミング距離と呼ばれ ($w_{min}, d_{min}$ とかく), 以降で扱う**線形符号**においては, 最小重みと最小距離は等しい関係にあることが知られている. 証明は線形符号の章で行われる.

**例**　先ほどの例の符号 C のそれぞれの最小重みと最小距離は

$$\begin{aligned} w_{min} &= 1 \\ d_{min} &= 1 \end{aligned}$$

である.

最小ハミング距離には, 以下のような性質がある.

**Thm : 最小ハミング距離と符号の誤り検出, 訂正**

ある符号語において, 誤り検出が可能なシンボル (ビット) 数を s, 訂正可能なシンボル数を t とすると,

$$d_{min} \geq s + 1 \tag{1.1}$$
$$d_{min} \geq 2t + 1 \tag{1.2}$$

(1.1) の証明を図で行う.

正しい符号語 A があり, その A とのハミング距離が s 以下であれば「A とは異なっている」という誤り検出が可能な円領域を考える (図 1.1). 同じ条件の符号語 B を用意し, AB 間の距離が $d_{min}$ であるとする. $d_{min} < s + 1$ であるような場合,「A と異なっている」符号語が B である可能性が発生し, 誤り検出が正常に行われない (図 1.2). よって, $d_{min} \geq s + 1$ となる.

次に, (1.2) の証明を図で行う.

正しい符号語 A があり, その A とのハミング距離が t 以下であれば「A に訂正が可能」という円領域を考える (図 1.3). 同じ条件の符号語 B を用意し, AB 間の距離が $d_{min}$ であるとする. $d_{min} < 2t + 1$ であるような場合, 積集合 X 内の誤り符号語を A, B どちらに訂正すればよいかわからなくなる (図 1.4). よって, $d_{min} \geq 2t + 1$ となる.

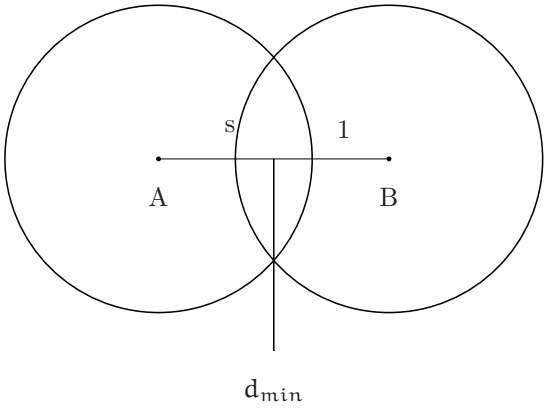



図 1.1　$d_{min} \geq s+1$ の場合

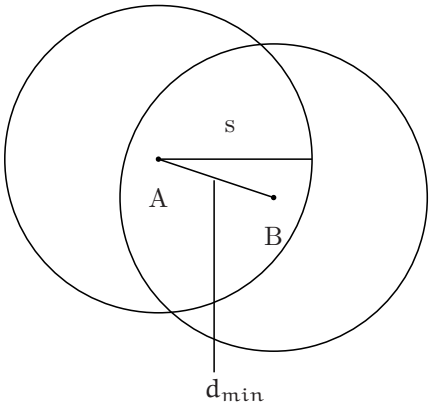



図 1.2　$d_{min} < s+1$ の場合

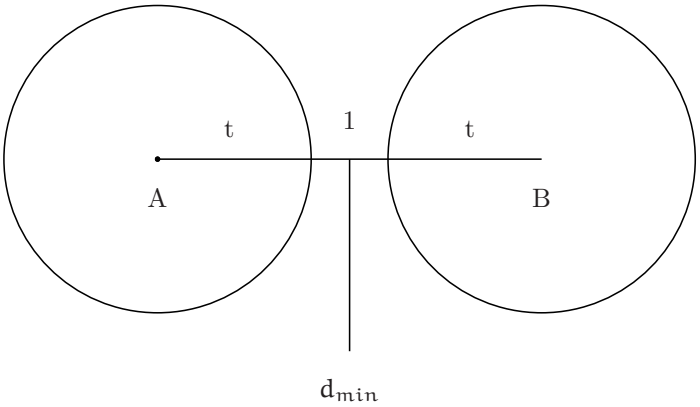



図 1.3　$d_{min} \geq 2t+1$

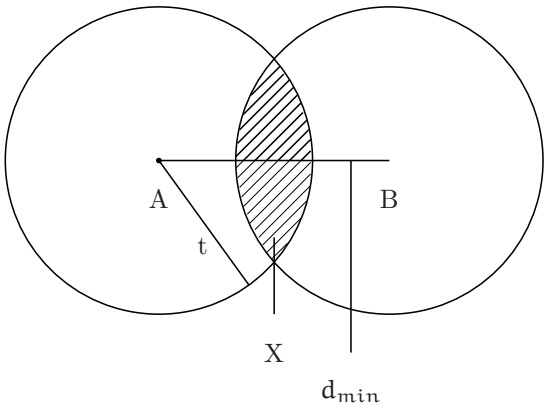



図 1.4　$d_{min} < 2t+1$

## 1.5 練習問題

1. 偶数パリティ検査において, 情報系列の長さを 3 とすると, 全ての情報系列とそれに対応した符号語はどうなるか列挙せよ. また, 符号化率を求めよ.
2. 垂直水平パリティ検査で 5 ビットの誤りがあった場合に, ある 1 行とある 1 列だけに誤りが検出される場合がある. それはどんな場合か.
3. 符号を (0000), (0011), (1100), (0110), (1001), (1010) とすると, 最小ハミング距離はいくらか. また, 何ビットまで誤り検出/訂正が可能か.
4. 4 ビットの情報語 $(c_0c_1c_2c_3)$ に対して垂直水平パリティ検査ビットを付加すると符号語は $(c_0c_1c_2c_3c_4c_5c_6c_7c_8)$ となる. これについて, 以下に答えよ. また, この符号語を行列で書くと

$$\begin{pmatrix} c_0 & c_1 & c_4 \\ c_2 & c_3 & c_5 \\ c_6 & c_7 & c_8 \end{pmatrix}$$

   となる.
   (a) 全ての符号語を $(c_0c_1c_2c_3c_4c_5c_6c_7c_8)$ の形で列挙せよ.
   (b) 符号化率はいくらか.
   (c) 符号語の最小ハミング距離を求めよ.
   (d) 符号語の t と s をそれぞれ求め, 何ビットまで誤り訂正/検出が可能か答えよ.
   (e) (110110101) は符号語か. ハミング距離により誤り検出せよ (ハミング距離が 0 なら符号語である).
   (f) (e) の系列が符号語でないとしたら, 訂正が可能であれば訂正し, 対応する情報語 $(c_0c_1c_2c_3)$ を求めよ.
   (g) (101001101) は符号語か. ハミング距離により誤り検出せよ.
   (h) (5) の系列が符号語でないとしたら, 訂正が可能であれば訂正し, 対応する情報語 $(c_0c_1c_2c_3)$ を求めよ.
   (i) (e)~(h) について, 垂直水平パリティ検査により求めよ.

# 第2章

# 線形符号

現在用いられている符号は, 大抵の場合**線形符号**という構造を持っている. 本章では, 符号が**線形符号である**とはどういうことか. さらに, 線形符号がどのような便利で興味深い性質を持っているのかについて学ぶ.

## 2.1 線形符号の定義

**Def : 線形符号**

ある符号 C を考えたとき, C が線形符号なら,

$$c_i \in C \text{ かつ } c_j \in C(c_i \neq c_j) \Rightarrow c_i + c_j \in C$$
$$c_i \in C \text{ かつ } f \in F(F \text{ はある体}) \Rightarrow fc_i \in C$$

という式が成り立つ.
また, 零ベクトルは常に符号語.
符号語の総数 M は $M = q^k$ に等しい (q は体 F の元の個数).

**例** 次の 16 個のベクトルからなる符号を考える.

| | |
|---|---|
| 0000000 | 1111111 |
| 1000110 | 0111001 |
| 0100011 | 1011100 |
| 0010101 | 1101010 |
| 0001111 | 1110000 |
| 1100101 | 0011010 |
| 1010011 | 0101100 |
| 1001001 | 0110110 |

どの 2 つの符号語の和も, 符号語になっているのがわかる. よって, この符号語からなる符号は線形符号である. また, 線形符号の符号語長が n, 情報語長を k とした時, (n, k) 線形符号とかくこともある. 先の例の符号は (7,4) 線形符号である.

## 2.2 生成行列

**生成行列 (G とかく)** とは, 行が線形符号 C の基底をなす $k \times n$ 行列のことをいう. 生成行列が分かれば, 全ての符号語を知ることができる. また, 符号のパリティ検査行列の構築にも用いられる (逆もある).

**例**　先の (7,4) 符号[*1]の生成行列

符号語のリストから適当に, 4 個の線形独立のベクトルを抜き出すと, 一例として次の生成行列 G が得られる.

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

G の作り方は沢山ある. ベストなのは, **最初の** $k$ **列が** $k \times k$ **の単位行列**[*2]**になるように抜き出すこと**である. 先の例の G はベストな形の生成行列であるといえよう.

また,G は I を使って

$$G = (I, A) \tag{2.1}$$

と表すこともできる. この記法は, 次に登場するパリティ検査行列を構築するときにも役立つので頭の片隅に入れておいて欲しい.

## 2.3　パリティ検査行列

長さ $n$ のベクトル $h$ を用意する. この $h$ が

$$G^t h = 0 \quad (t \text{ は転置を表す})$$

を満たすとき, $h$ を**パリティ検査ベクトル** とよぶ. このパリティ検査ベクトルを行とした $(n-k) \times n$ 行列を**パリティ検査行列**[*3]という. G が生成行列,H が検査行列であるとき

$$G^t H = 0$$

が成り立つ. 0 は $k \times (n-k)$ 零行列である.

G が (2.1) の形であれば, H を次の形でかくことができる.

$$H = (-{}^t A, I) \tag{2.2}$$

ここでの I は $(n-k) \times (n-k)$ 単位行列である.

**例**　先の (7, 4) 符号の検査行列は

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

である.

検査行列は, ある符号語が線形符号の符号語かどうかや, 受信した符号に誤りがないか検査するなど, さまざまな用途がある重要な行列である. ある符号語が線形符号の符号語かどうかは, 符号語 C が

$$H^t C = 0 \tag{2.3}$$

を満たすかどうかをチェックする. この方程式を**パリティ検査方程式**という.

---

[*1] "線形"は面倒なので省略します

[*2] I とおく

[*3] 以降は単に検査行列とかきます

## 2.4 線形符号の性質

線形符号にはさまざまな性質があるが, ここではその中でも重要な 3 つを紹介する.

**Thm : 線形符号の性質 (1)**

$c_1, c_2$ を線形符号とすると, これらの線形結合 $c_3$ も線形符号である.

$$m_1 c_1 + m_2 c_2 = c_3 \quad (\text{但し}, {}^{\forall} m_1, {}^{\forall} m_2 \in GF(2)) \tag{2.4}$$

この証明は, 検査行列を使えば容易となる.

$c_1, c_2$ は符号語なので,

$$Hc_1 = 0$$
$$Hc_2 = 0$$

ここで, (2.4) 式の両辺に左から H を掛けると

$$\begin{aligned} Hc_3 &= H(m_1 c_1 + m_2 c_2) \\ &= m_1 Hc_1 + m_2 Hc_2 \\ &= 0 \end{aligned}$$

よって, $c_3$ も符号語.

**Thm : 線形符号の性質 (2)**

線形符号においては, 最小ハミング距離を最小ハミング重みは以下のような関係を持つ.

$$d_{min} = w_{min} \tag{2.5}$$

証明が少しややこしいが, 頑張ってついてきて欲しい.

重みと距離の定義から, $w(x) = d(0, x)$ であり, $d(x, y) = w(x - y)$ である.
また, c を最小重みの符号語であるとする.
このとき, $w(c) = d(0, c)$ であり, 0 が符号語なので $d_{min} \leq w_{min}$ となる.
次に, 最小距離にある $c_1$ と $c_2$ を取り出すと
$d(c_1, c_2) = w(c_1 - c_2)$ であり, $c_1 - c_2$ も符号語なので $d_{min} \geq w_{min}$ となる.
この二つが矛盾しないようにするには, $d_{min} = w_{min}$ とするしかない.

**Thm : 線形符号の性質 (3)**

線形符号の検査行列 H の任意の r 個の列ベクトルが線形独立であり, そこから更に一つ列ベクトルを取り出した $r + 1$ 個の列ベクトルは線形従属であるとすると,

$$w_{min} = r + 1 \tag{2.6}$$

この証明は省略するが, 次章 **DFT 符号と DFT 行列**においてこの関係が重要な役割を果たすので, しっかりと覚えておこう.

## 2.5 練習問題

1. 以下の検査行列で与えられる符号長 7 のハミング符号の符号語は (0000000), (1101000), (0110100), (1010010), (1110001), (1011100), (0100011), (0111010), (0011001), (1100110), (1000101), (0010111), (0001110), (0101101), (1001011), (1111111) である. この符号語と検査行列について答えよ.

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

   (a) この検査行列の任意の 2 個の列ベクトルは互いに独立である. このことを確認するため, どれか 2 個の列ベクトルを取り出し, それが互いに独立であることを確認せよ.

   (b) この検査行列には互いに従属する 3 個の列ベクトルが存在する. このことを確認するため, 該当する 3 つの列ベクトルを抽出せよ. また, このような従属ベクトルに対応する符号語を求めよ.

   (c) (a), (b) によって, $w_{\min} - 1 = r$ が成立することを示せ.

   (d) (a), (b) によって, 検査行列に対応する符号語の最小距離と最小重みを求め, 誤り検出と訂正可能なビット数を求めよ.

2. 符号語の長さ $n = 5$ の偶数パリティ検査符号に対応する検査行列 H を求めよ.

3. 前章の問題 4 により与えられる水平垂直パリティ検査符号に対応する検査行列 H を求めよ.

# 第3章

# DFT行列とDFT符号

線形符号の誤り訂正, 検出能力は, パリティ検査行列 H の**列ベクトルの線形独立性**と深く関わっている.
このことは, 逆に, 任意の r 個の列ベクトルが線形独立であるようなパリティ検査行列を構成できれば, **思うがままの性能を持つ符号を構築できる**ことを表している.
この章では, 任意の r 個の列ベクトルが線形独立であるようなパリティ検査行列を構築する方法を考えよう.

## 3.1 原始 n 乗根

まず, $x^3 = 1$ という方程式を考えよう[*1].

**Let's thinking**

$x^3 = 1$ の解を求めてみよう.

この方程式の 3 つの解は, $x = 1, \dfrac{1 \pm \sqrt{3}i}{2}$ である. ただし, i は虚数単位なので注意しよう[*2].

これらの x は $x^3 = 1$ という方程式の解なので, 全て, 3 乗すると 1 になるという性質を満たしている. すなわち, これらの 3 つの x は 1 の 3 乗根である.

$$1^3 = 1 \cdot 1 \cdot 1 = \underset{\sim}{1} \quad //$$

$$\left(\frac{1+\sqrt{3}i}{2}\right)^3 = \frac{1+\sqrt{3}i}{2} \cdot \frac{1+\sqrt{3}i}{2} \cdot \frac{1+\sqrt{3}i}{2} = \frac{1-\sqrt{3}i}{2}\frac{1+\sqrt{3}i}{2} = \underset{\sim}{1} \quad //$$

$$\left(\frac{1-\sqrt{3}i}{2}\right)^3 = \frac{1-\sqrt{3}i}{2} \cdot \frac{1-\sqrt{3}i}{2} \cdot \frac{1-\sqrt{3}i}{2} = \frac{1+\sqrt{3}i}{2}\frac{1-\sqrt{3}i}{2} = \underset{\sim}{1} \quad //$$

この 3 つの x に注目すると, 次のことが分かる.

$x = \dfrac{1 \pm \sqrt{3}i}{2}$ については, 3 乗して初めて 1 になる.

[*1] **代数学の基本定理**により, この方程式は 3 次方程式なので, 3 つの複素数解を持つ

[*2] 聞く話では工学では虚数単位を j とか書くらしいが, 普通の人は虚数単位を i と書く. (i は imaginary number の i なんだから, j にしたら意味わかんなくなるだろうが. i は電流で使うとかつまらんことにこだわってないで電流を j にしろボケ)

このように, 3 つの解のうち, 2 つの解が **3 乗して初めて 1** になるという性質を持つことが分かった. このような 1 の 3 乗根を, **1 の原始 3 乗根**という.

**Def : $a$ の原始 $n$ 乗根**

$n$ 乗して初めて $a$ となる数 $\omega$ を, $a$ **の原始 $n$ 乗根**という.

$a$ の原始 $n$ 乗根 $\omega$ は, 以下のような重要な性質を持っている.

$$\omega^1 \neq \omega^2 \neq \cdots \neq \omega^{n-1} \neq \omega^n = a$$

すなわち, $\omega$ は $n$ 乗して初めて $a$ となる数であり, さらに, $\omega^1, \omega^2, \cdots, \omega^n$ は**全て異なる**という性質を持つ. この考え方は, 後に学ぶ**有限体**において大変重要になるので, いまのうちに納得しておくべし.

## 3.2　DFT 行列, DFT 符号

さて, これから **任意の $r$ 列が線形独立**であるような行列を構成することを考えよう. このような行列は, **1 の原始 $n$ 乗根** $\omega$ を使うと, 上手く作ることができる.

天下り式ではあるが, 行列 $D$ を次のように定義することにしよう.

$$D = \begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix}$$

このように定義した行列を, $n$ 次の DFT **行列** (DFT matrix)[*3]という.

**Def : $n$ 次の DFT 行列**

$\omega$ を 1 の原始 $n$ 乗根としたとき, 以下のように定義される行列 $D$ を $n$ 次 DFT 行列という.

$$D = \begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} = \left(\omega^{ij}\right)$$

このように定義した DFT 行列について, 色々なことを調べてみよう. まず, $D$ から連続した $r$ 行を抜き出して作った行列を $D_r$ としよう. ここでは, $\underbrace{r_1 \sim (r_1 + r - 1)}_{\text{全部で } r \text{ 行}}$ 行目までの $r$ 行を取り出すことにする.

$$D_r = \begin{pmatrix} 1 & \omega^{r_1} & \omega^{r_1 \cdot 2} & \cdots & \omega^{r_1 \cdot (n-1)} \\ 1 & \omega^{r_1+1} & \omega^{(r_1+1)\cdot 2} & \cdots & \omega^{(r_1+1)\cdot(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{r_1+r-1} & \omega^{(r_1+r-1)\cdot 2} & \cdots & \omega^{(r_1+r-1)\cdot(n-1)} \end{pmatrix}$$

[*3] 【Advance】信号処理等で活躍する**離散**Fourier**変換** (Discrete Fourier Transform) は線形写像であるので, 表現行列が存在する. その表現行列こそが, この DFT 行列なのである.

いま, この $D_r$ の**任意の r 列が線形独立かどうか？**を調べてみよう. もしこの $D_r$ の任意の r 列が線形独立であるなら, $D_r$ をパリティ検査行列として採用することで, 思うがままの性能を持つ符号を構築できる.

ということで, $D_r$ の任意の r 列が線形独立かどうかを調べるために, $D_r$ の任意の r 列（任意だから, 連続していても, 飛び飛びでも OK）を抜き出して作った行列 $D_{rr}$ を考える. $D_{rr}$ の全ての列ベクトルが互いに線形独立ならば, $D_r$ の任意の r 列が線形独立となることが分かるだろう. ここでは, $s_1, s_2, \cdots, s_r$ 列目 (全部で r 列) をそれぞれ抜きだして $D_{rr}$ を構成したとしよう.

$$D_{rr} = \begin{pmatrix} \omega^{r_1 s_1} & \omega^{r_2 s_2} & \cdots & \omega^{r_1 s_r} \\ \omega^{(r_1+1)s_1} & \omega^{(r_1+1)s_2} & \cdots & \omega^{(r_1+1)s_r} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(r_1+r-1)s_1} & \omega^{(r_1+r-1)s_2} & \cdots & \omega^{(r_1+r-1)s_r} \end{pmatrix}$$

$D_r$ は r 行 n 列の行列だったので, そこから r 列を抜き出して作った $D_{rr}$ は $r \times r$ の**正方行列**となる.
正方行列においては, 実は, 以下に示すような便利な性質が成り立つ.

**Thm : 行列式と線形独立性**

A を正方行列とすると, 以下の事実が成り立つ.

$$\det A \neq 0 \quad \Longleftrightarrow \quad A \text{ の全ての列ベクトル, 行ベクトルが互いに線形独立!!}$$

よって, $\det D_{rr} \neq 0$ であれば, $D_{rr}$ の全ての列ベクトルが線形独立, すなわち, $D_r$ の任意の r 列が線形独立であると示せたことになる. さあ, ゴールは見えてきた. 頑張って $\det D_{rr}$ を調べてみよう.

$$\det D_{rr} = \det \begin{pmatrix} \omega^{r_1 s_1} & \omega^{r_2 s_2} & \cdots & \omega^{r_1 s_r} \\ \omega^{(r_1+1)s_1} & \omega^{(r_1+1)s_2} & \cdots & \omega^{(r_1+1)s_r} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(r_1+r-1)s_1} & \omega^{(r_1+r-1)s_2} & \cdots & \omega^{(r_1+r-1)s_r} \end{pmatrix}$$

$$= \det \begin{pmatrix} \omega^{r_1 s_1} & \omega^{r_2 s_2} & \cdots & \omega^{r_1 s_r} \\ \omega^{r_1 s_1}\omega^{s_1} & \omega^{r_1 s_2}\omega^{s_2} & \cdots & \omega^{r_1 s_r}\omega^{s_r} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{r_1 s_1}\omega^{(r-1)s_1} & \omega^{r_1 s_2}\omega^{(r-1)s_2} & \cdots & \omega^{r_1 s_r}\omega^{(r-1)s_r} \end{pmatrix}$$

まず, $\det D_{rr}$ の各列に着目すると, **全ての列の** $\omega^{r_1 s_1}, \omega^{r_1 s_2}, \cdots, \omega^{r_1 s_r}$ **がくくり出せる**ということに気づく. これらを外にくくりだしてあげよう.

$$\det D_{rr} = \omega^{\left(r_1 \sum_{k=1}^{r} s_k\right)} \underbrace{\det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \omega^{s_1} & \omega^{s_2} & \cdots & \omega^{s_r} \\ (\omega^{s_2})^2 & (\omega^{s_2})^2 & \cdots & (\omega^{s_r})^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\omega^{s_1})^{r-1} & (\omega^{s_2})^{r-1} & \cdots & (\omega^{s_r})^{r-1} \end{pmatrix}}_{(*)} = \omega^{\left(r_1 \sum_{k=1}^{r} s_k\right)} \det D_v$$

次に, $(*)$ の部分 $(\det D_v)$ に着目しよう. 実は, このような形をした行列式は, その面白い性質ゆえに, **Vandermonde**[*4]**の行列式**という名前が付いている.

[*4] ヴァンデルモンドと読む. 教科書によってはバンデルモンド, ファンデアモンデなど, けっこう表記の揺れがある.

**Def : Vandermonde の行列式**

次のような形の行列式を, Vandermonde **の行列式**という.

$$\det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ (x_1)^2 & (x_2)^2 & \cdots & (x_n)^2 \\ \vdots & \vdots & \ddots & \vdots \\ (x_1)^n & (x_2)^n & \cdots & (x_n)^n \end{pmatrix}$$

Vandermonde の行列式は, 実はすごく綺麗な形に**因数分解**することができる. その因数分解の結果を以下に示そう. ちなみに, $\prod$ は**総積 (product)** を表す ($\sum$ の掛け算バージョン).

**Thm : Vandermonde の行列式の因数分解**

$$\det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ (x_1)^2 & (x_2)^2 & \cdots & (x_n)^2 \\ \vdots & \vdots & \ddots & \vdots \\ (x_1)^n & (x_2)^n & \cdots & (x_n)^n \end{pmatrix} = \prod_{i>j} (x_i - x_j).$$

▷ 4 年のときに数 2 を選択していた人は証明せよ (･ω･)b

この因数分解の結果を使って, $\det D_{rr}$ を書き換えてみよう .

$$\det D_{rr} = \omega^{\left(r_1 \sum_{k=1}^{r} s_k\right)} \det D_v = \omega^{\left(r_1 \sum_{k=1}^{r} s_k\right)} \prod_{i>j} (\omega^{s_i} - \omega^{s_j}).$$

さて, 右辺に着目すると, $\det D_{rr}$ が 0 となる条件が分かる.

$$\det D_{rr} = 0 \iff \omega^{s_i} = \omega^{s_j} \quad (i > j).$$

ところで, $s_i, s_j$ というのはどこからどうやって持ってきた数だっただろうか. $s_i, s_j$ は, $D_r$ の任意の $r$ 列を抜き出したときの, **抜き出した列のそれぞれの番号**である. 抜き出す列はそれぞれ異なっていた**（同じ列を 2 回抜き出すことはしなかった）**ので, 任意の $i, j$ について, $s_i \neq s_j$が成り立つ.

更に, $D_r$ の列数は $n$ だったので, $s_i, s_j$ が $n$ より大きくなることはない. よって, $0 < s_i, s_j \leq n, s_i \neq s_j$である. これと, $\omega$ (1 の原始 $n$ 乗根) の性質,

$$\omega^1 \neq \omega^2 \neq \cdots \neq \omega^{n-1} \neq \omega^n = 1.$$

を合わせて考えれば, 次のことが分かるだろう.

$$\omega^{s_i} \neq \omega^{s_j}.$$

このことから, $\det D_{rr}$ について, 以下のことが言える.

$$\det D_{rr} = \omega^{\left(r_1 \sum_{k=1}^{r} s_k\right)} \underbrace{\prod_{i>j} (\omega^{s_i} - \omega^{s_j})}_{\omega^{s_i} \neq \omega^{s_j} \text{ より, } \neq 0}.$$

よって, $\det D_{rr}$ **の全ての列ベクトルは互いに線形独立**なので, 結局, $D_r$ **の任意の** $r$ **個の列ベクトルは線形独立である**と示せたことになる. これで, 示したかった結論を示すことができた！

**Thm : 任意の** $r$ **列が線形独立な行列**

$n$ 次 DFT 行列の連続した $r$ 行により構成された行列

$$D_r = \begin{pmatrix} 1 & \omega^{r_1} & \omega^{r_1 \cdot 2} & \dots & \omega^{r_1 \cdot (n-1)} \\ 1 & \omega^{r_1+1} & \omega^{(r_1+1)\cdot 2} & \dots & \omega^{(r_1+1)\cdot(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{r_1+r-1} & \omega^{(r_1+r-1)\cdot 2} & \dots & \omega^{(r_1+r-1)\cdot(n-1)} \end{pmatrix}$$

の任意の $r$ 列は, **互いに線形独立**である.

$D_r$ を符号のパリティ検査行列として採用すれば, r/2 以下の整数値で表される数の誤りまで訂正できる符号を構成することができる. 更にはこの $r$ は自由に変更することができる（大きい DFT 行列から, 連続する行をごっそり持ってくれば, $r$ はむちゃくちゃ大きくもできる）ので, **望む性能を持つ符号を, 思うがままに構成することができる**のだ. このように, DFT 行列を基に構成する符号を, DFT **符号** (DFT code) と呼ぶ.

# 第4章

# 有限体

符号語（たとえば 0100110）を構成する**シンボル** (symbol) は, **アルファベット** (alphabet) の元により構成される (さっきの例でのアルファベットは $\{0,1\}$ という, 記号の**集合**). そして, **符号語の原子**であるアルファベットは, 一般に, **体** (field) という構造を持つものに限られる.

アルファベットの構造を限定する必要なんてあるのか？ もっと自由にしたほうが, 符号理論の世界は広がりを持つのでは？ いやいや, そんなことはない. この**「意味のある制限」**こそが, 符号理論の壮大な広がりや, 純粋数学との巨大な繋がり, そして, 情報工学の未来を切り開く膨大な可能性をもたらす**「鍵」**なのである.

本章では, まず, アルファベットの構造の基礎となる Galois **体**について学び, 更に, Galois **拡大体**を上手く構築することにより, 更なる符号理論の発展の”きっかけ”をせっせと作って行こう.

## 4.1 体の公理

そんなわけで, アルファベットには**体**という構造が付加されるということを言ったわけだが, その「体」がどういうものなのかを知らなければ元も子もない. 早速 , **体とはこういうものだぞ**という約束を確認しておこう. このような約束のことを**公理** (axiom) という.

例えば, **実数** (real number) の集合 $\mathbb{R}$ を考えてみる.
有理数も無理数も, 整数も自然数も, 全て実数に含まれる. ただし, **虚部** (imaginary part) を持つ数は , 実数ではない. とりあえず, 虚部が無い数は全て実数だと覚えておこう.

さて, ここで質問.

$\mathbb{R}$ の中で**常に自由に行える演算**には何があるだろう？

例えば, 整数全体の集合 $\mathbb{Z}$ を考えよう. $\mathbb{Z}$ では, **除算**を自由に行ってはいけない. なぜなら, $1 \div 2 = 1/2 \notin \mathbb{Z}$ のように[*1], 除算による結果が $\mathbb{Z}$ から**飛び出てしまう**からである. 自由に演算を行えるとは, **そのような事態が絶対に起こらないと保証される**という意味だ. いま, $\mathbb{R}$ における演算は, 以下のものが考えられる.

- **加算** (addition)
- **減算** (subtraction)
- **乗算** (production)
- **除算** (division)[*2]

[*1] $1/2 \notin \mathbb{Z}$ は, $1/2$ が, $\mathbb{Z}$ に含まれないという意味. すなわち, $1/2$ は整数ではないことを表している.
[*2] ただし, **ゼロ割**は除く.

では早速 , $\mathbb{R}$ で常に可能な演算は何かを順番に考えてみよう.

1. **加算**
   $^\forall a, ^\forall b \in \mathbb{R}$ とする[*3]. このとき, $a+b$ がやはり実数であるのは, 明らかだろう.

   $$^\forall a, ^\forall b \in \mathbb{R} \ \Rightarrow \ a+b \in \mathbb{R}.$$ [*4]

   このような $\mathbb{R}$ の性質を, 数学では $\mathbb{R}$ は**和について閉じている**と呼ぶ.

2. **減算**
   例えば, $^\forall a, ^\forall b \in \mathbb{R}$ について, 減算 $a-b$ を考えてみよう. この減算は, $a+(-b)$ と解釈することができる. そして, $-b$ もやはり実数であることが簡単にわかるので, 次のことが言える.

   $$^\forall a, ^\forall b \in \mathbb{R} \ \Rightarrow \ a-b \in \mathbb{R}.$$

   すなわち, $\mathbb{R}$ は**差について閉じている**.

3. **乗算**
   実数の積は, **明らかに実数**である. よって, $^\forall a, ^\forall b \in \mathbb{R}$ について, 次のことが言える.

   $$^\forall a, ^\forall b \in \mathbb{R} \ \Rightarrow \ a \cdot b \in \mathbb{R}.$$

   すなわち, $\mathbb{R}$ は**積について閉じている**.

4. **除算**
   $^\forall a \in \mathbb{R}, ^\forall b \in \mathbb{R}/\{0\}$ を考えよう[*5]. $a, b$ の商 $a/b$ が実数となることは, 容易に想像できるだろう.

   $$^\forall a \in \mathbb{R}, ^\forall b \in \mathbb{R}/\{0\} \ \Rightarrow \ a/b \in \mathbb{R}.$$

   すなわち, $\mathbb{R}$ は**商について閉じている**.

結局のところ, $\mathbb{R}$ は, **四則演算が上手く定義されていて, それらについて閉じている**ことが分かる. このような, **四則演算について閉じている数の体系**のことを**体** (field) と呼ぶことにしよう.

**Def : 体**

集合 K に四則演算が定義されていて, 四則演算について K が閉じているとき, K を**体** (field) と呼ぶ.

**例**　有理数体 $\mathbb{Q}$, 実数体 $\mathbb{R}$, 複素数体 $\mathbb{C}$ は, 全て**体**を成している.

符号語のアルファベットの構造として大変重要な**「体」**は, このようにして定義される[*6].

---

[*3] この式は, **「任意の $\mathbb{R}$ の元（すなわち実数）$a, b$ をとる」**と読む. 数式により記述された文を読むのは慣れるまでは大変だが, 日本語での記述よりも厳密で, 洗練されていることが, 慣れるにつれて分かってくる. 英作文ならぬ数作文だ.
というわけで, この学習資料では, あえて**分かりやすさを重視するため**に, このような記法を遠慮無く使わせてもらう.

[*4] 「$a, b$ が任意の実数ならば, $a+b$ もやはり実数である」と読む. $\in$ は集合の記号で, $a \in A$ を「$a$ は集合 $A$ の元である」と読む.

[*5] $b$ については, ゼロ割を防止するために, $\mathbb{R}/\{0\}$ を考えた. $\mathbb{R}/\{0\}$ は, $\mathbb{R}$ から $\{0\}$ という集合を除いた集合を表す. 差集合.

[*6] 本当は, 体はもっと厳密に定義しなければならない. ”四則演算について閉じている”だけでは, 本来ならば不十分である. が, 符号理論においてはとりあえずこれだけ知っておけば大丈夫. あと, なんか脚注多くてごめんなさい. ごめんねごめんねー！

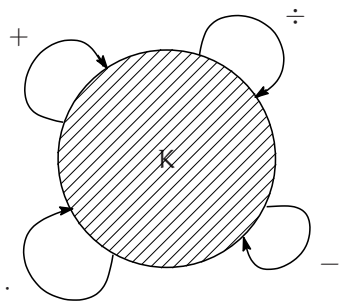



図 4.1 体 K のイメージ

## 4.2 有限体の定義

さて, これから, 符号理論において最も重要な役割を果たす**有限体**を定義し , 使いこなすための準備を始めよう. まずは, 重要な言葉の定義からスタートだ.

### 4.2.1 集合の濃度

体は, 「四則演算について閉じている」という性質を持つ**集合**である. 集合は元（要素）の集まりなので, 当然, **集合に対して元の個数**が対応する. 集合 S の元の個数を, 集合 S の**濃度** (cardinal number) という.

**Def : 集合の濃度**

ある集合 S の**元の個数**のことを集合 S の**濃度** (cardinal number) といい, |S| と表す.

どんな集合に対してでも, 濃度は定義することが可能だ. 例えそれが, 無限集合であっても, である[*7].

### 4.2.2 有限体の定義

さて, 集合の濃度が定義できたところで, **有限体**を定義しよう. 有限体は, 今後の符号理論の展開において最重要の役割を果たす. そして, その定義は極めて簡潔なものである.

**Def : 有限体**

体 K の濃度 |K| が有限であるとき, K を**有限体** (finite field) という.
すなわち有限体は, 有限個の元しか持たない体のことである.

今までの例で示してきた体は, 全て**無限集合**であった. では, 有限個の元からなる体, 有限体にはどのようなものがあるのだろうか？ 実はその体こそが, 最も重要な Galois **体**と呼ばれる体である.

さあ, ここから先, 符号理論はどんどん面白くなる. 有限体がもたらす芳醇な世界を, 存分に味わおう.

[*7] 【Advance】 無限集合は当然, 元を無限に含むので, 濃度を単純な数では表すことはできない. いくつかの無限集合の濃度の例を示すと, $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ の濃度は $\aleph_0$, $\mathbb{R}, \mathbb{C}$ の濃度は $\aleph$ という. こんな風に, 無限集合の濃度を表すための専用の数が用意されているのである. ちなみに, $\aleph_0, \aleph$ 以外の濃度を持つ無限集合を見つけたら, 誰にも言わずに, まずはすぐに私のところに連絡すること（大丈夫, 論文で学会に発表してフィールズ賞を貰ったりなんてしませんから安心してください）.

## 4.3　有限体の構築

それではこれから, 具体的な有限体を構築する作業を始めよう. これから構築する有限体が符号理論において果たす役割は果てしない. **有限体の構築 ＝ 土台の構築**といっても, 全く大げさではないほどに重要だ.

### 4.3.1　p を法とする演算

有限体を構築するときに必要不可欠となるのが, p **を法とする演算**である.

**Def : p を法とする演算**

**通常の四則演算**を行った結果を p で割った**余り**を結果とする演算を, p **を法とする演算**と呼ぶ.

例えば, 第 1 章で登場した**排他的論理和（**exclusive or**）**は, 2 **を法とする和**と捉えることができる.
まずは, 情報数学で習った排他的論理和 $\oplus$ の**真理値表**を以下に示す.

| $\oplus$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

次に, 2 **を法とした和** ($+$ と表す) の真理値表を作ってみると,

| $+$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

となり, 2 **つの真理値表が完全に一致する**. よって, **排他的論理和は** 2 **を法とする和**と言えるのだ.
（ちょっと発想を変えて, p を法とする演算は, **排他的論理和の拡張**という風に考えても良い.）

### 4.3.2　Galois **体**

いよいよ, 具体的な有限体を構築しよう. それには, 先程定義した p を法とする演算が, 大きな威力を発揮する. なぜなら, **これから構築する有限体での四則演算は, 素数** p **を法とするものだから**だ.

まず, 集合 $K_p$ を以下のように作ってみよう. ただし, p **は素数**である.

$$K_p = \{0, 1, \cdots, p-1\}.$$

実はこのように定義した $K_p$ は, p **を法とする四則演算について体を成す**のである. こんなにも単純な集合が, これまた単純な演算（p を法とする四則演算）について体を成してしまう. この驚くほど単純な体は Galois[*8]**体** (Galois field) と呼ばれ, GF(p) と表記することが決まっている[*9].

[*8] ガロアと読む. 不幸にも若くして亡くなった（不本意ながら決闘をするハメになり, うっかり殺されてしまった）数学者だが, この人が 80 まで生きていたら数学は 200 年分進歩していただろうとまで言われるほどの, いわば大天才だった.

[*9] 教科書によっては, Galois 体ではなく, p **元体**と呼んでいるものもある. p 元体と呼ぶ場合は, GF(p) ではなくて, $\mathbb{F}_p$ と表記するのが普通. 例えば 2 元体は $\mathbb{F}_2$, 3 元体は $\mathbb{F}_3$ みたいな感じ. $GF(p) = \mathbb{F}_p$. ちなみに, 1 元体 $\mathbb{F}_1$ というのも存在して, $\mathbb{F}_1$ をうまく使えば Riemann **予想**が解けるんじゃないだろうかと言われている.

**Def : Galois 体 GF(p)**

p を素数とし, GF(p) を次のように定義する.

$$GF(p) = \{0, 1, \cdots, p-1\}.$$

このとき, GF(p) は, p を法とする四則演算について**体を成す**.
この GF(p) を Galois **体** (Galois field) と呼ぶ. GF(p) は明らかに**有限体**である.

GF(p) においては, 四則演算として p **を法とした四則演算**を採用する.
そして, GF(p) において, 実は次の定理**（和と積の逆元の存在）**が**必ず成立**する.

- $^{\forall}a \in GF(p)$, $^{\exists}(-a) \in GF(p)$ s.t. $a + (-a) = 0$.
- $^{\forall}a \in GF(p)$, $^{\exists}a^{-1} \in GF(p)$ s.t. $a \cdot a^{-1} = 1$. *10

すなわち, GF(p) のどんな元 a を取っても, 必ず, $a + (-a) = 0$ を満たす元**（和の逆元）**が GF(p) の中に存在し, さらに, $a \cdot a^{-1} = 1$ を満たす元 **(積の逆元)** も必ず GF(p) に存在する.

そして, GF(p) における**差, 商**という演算は, 和, 積を用いて次のように定義される.

- $a - b = a + (-b)$ （差は**和の逆元との和**）
- $a/b = a \cdot b^{-1}$ （商は**積の逆元との積**）

つまり, 差と商は, 結局**逆元** (GF(p) の元) との**和, 積**により定義されているので, 和, 積について閉じていることを示せば, GF(p) が四則演算について閉じていることを示せたことになる.

ここの話は少し理解しにくい部分だが, とっても重要な部分なので頑張って読み込んで理解しておこう.

**例** GF(2)

具体的な例として, GF(2) を見てみよう. GF(2) は, 次のように構成される集合である.

$$GF(2) = \{0, 1\}.$$

GF(2) は, 2 **を法とした演算**について体を成している. そのことをこれから確認しよう.
まず, 2 を法とした和について, **加算表**を作る.

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

このように, GF(2) の元の演算結果がやはり GF(2) の元となっているので, GF(2) **は和について閉じている**.
さらに, 2 を法とした積について, **乗算表**を作る.

| · | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

*10 ∃ は, **存在する**という意味. $^{\exists}a \in A$ は, **集合 A に元 a が存在する**と読む.
s.t. は such that の略. 意味は辞書でも引いて調べてくだーさい.

このように, やはり GF(2) の元の演算結果が GF(2) の元となっているので, GF(2) **は積について閉じている**. よって, GF(2) は, 2 を法とした四則演算について**体を成す**[*11]. //

**例** GF(3)

GF(3) は次のように構成される集合である.

$$GF(3) = \{0, 1, 2\}.$$

GF(2) のときと同様に, 3 を法とした加算表と乗算表を作ってみよう.

| $+$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 0 |
| 2 | 2 | 0 | 1 |

| $\cdot$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 1 |

このように, GF(3) の元の加算, 乗算を行った結果が必ず GF(3) の元となっている. よって, GF(3) は 3 を法とした四則演算について**体を成している**. //

## 4.4 有限体の性質

有限体の具体的な例として Galois 体を構築することができた. これから, Galois 体が満たす様々な興味深い性質, そして, 符号理論において重要な役割を果たす様々な特徴について学んで行くことにしよう.

### 4.4.1 原始元の定義

Galois 体には, **原始元**と呼ばれる元を考えることができる.

**Def : 原始元**

GF(p) は, 濃度 p の有限体である. このとき,

$$a^{p-1} = 1.$$

かつ, $a \neq a^2 \neq \cdots \neq a^{p-1}$ を満たす GF(p) の元 a を, GF(p) の**原始元** (primitive element) という.

すなわち, GF(p) の原始元とは, $p-1$ **乗して初めて** 1 **になる数**のことをいう. そして, GF(p) の原始元は, $a \neq a^2 \neq \cdots \neq a^{p-1}$ を満たすとする[*12].

この定義に見覚えは無いだろうか？ そう, 実は, DFT 行列において登場した 1 **の原始** n **乗根** $\omega$ と全く同じ性質を持っているのである. 実は, GF(p) の原始元は, GF(p) **における** 1 **の原始** $p-1$ **乗根**に相当する.

### 4.4.2 原始元の性質

原始元は, 興味深く, 大変便利な性質を多数持ち合わせている. それらの重要な諸性質について, これから順番に挙げて行くことにしよう.

[*11] 前述したとおり, 和と積について閉じていることを示せば, 四則演算について閉じていることを示したことになるお (^ω^)
[*12] 原始元は, 本当は体の元の**位数** (order) ord(a) を使って定義される. 厳密な定義に興味のある読者は, 他の教科書を参照すべし.

**Thm : 原始元の性質**

GF(p) の 0 以外の全ての元は, GF(p) の原始元 $\alpha$ のべき乗, すなわち,

$$\alpha^n$$

の形で表される. つまり, **原始元が決まれば, 有限体が一意に決まる**.

この定理の証明は簡単である. 以下にその証明を示そう.

GF(p) の原始元を $\alpha$ とすると, 原始元の定義より, $\alpha$ は以下の性質を満たす.

$$\alpha \neq \alpha^2 \neq \cdots \neq \alpha^{p-1}.$$

原始元は GF(p) の元なので, 当然, $\alpha^n$ も GF(p) の元である. (GF(p) は体なので, 積について閉じている. よって, $\alpha \cdots \alpha \in$ GF(p) なので, $\alpha^n$ が GF(p) に含まれる)　すなわち, $\alpha, \alpha^2, \cdots, \alpha^{p-1}$ は全て GF(p) の元である. ところで, $|\text{GF}(p)| = p$ が成り立つので, GF(p) の元は全部で p 個だ.
いっぽう, $\underbrace{\alpha, \alpha^2, \cdots, \alpha^{p-1}}_{\text{全部で } p-1 \text{ 個}}$ は全て異なり, さらに, 全て GF(p) の元である. よって,
GF(p) の元は $\alpha, \alpha^2, \cdots, \alpha^{p-1}$ と残る 1 つのみとなる. そして, GF(p) は体なので 0 を必ず含むことにより, 残る 1 つが 0 であることが分かる. よって, GF(p) の 0 以外の全ての元が原始元のべきで表せる.

このことから, 原始元さえ決まれば, 対応する有限体が一意に定まるということも理解できるだろう. 原始元は, いわば**体を生成するベース**と捉えることも可能なのである.

**Thm : 原始元の存在**

任意の有限体には, 必ず**原始元が存在**する (1 つとは限らない).

更に, この便利な原始元は, どのような有限体についても必ず存在することが証明されている. よって, どんな有限体 K を考えても, K **の** 0 **以外の元はある原始元のべきで必ず書ける**のである.

が, そんな優秀な原始元だが, 実は, 原始元を上手く求めるアルゴリズムは知られていない. つまり, 原始元知りたければ, **有限体の** 1 **つ** 1 **つの元をしらみつぶしに調べるしか方法はない**のである.

**例**　GF(5) の原始元

まず, GF(5) $= \{0, 1, 2, 3, 4\}$ である. GF(5) の原始元を探してみよう.

| $\alpha$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $\alpha^2$ | 0 | 1 | 4 | 4 | 1 |
| $\alpha^3$ | 0 | 1 | 3 | 2 | 4 |
| $\alpha^4$ | 0 | 1 | 1 | 1 | 1 |

この表により, $\alpha^{5-1} = \alpha^4 = 1$ かつ, $\alpha \neq \alpha^2 \neq \alpha^3 \neq \alpha^4$ を満たしている元 $\alpha$ は, 2, 3 であることが分かる. よって, GF(5) の原始元は 2, 3 である. 2, 3 の列に注目すると, 確かに 2, 3 のべき乗によって, GF(5) の元が全て表示されていることが分かる.　//

## 4.5　有限体の拡大

$x^2+1=0$ という方程式の解は, 実数体 $\mathbb{R}$ 上には存在しない. この**「謎の方程式」**に真っ向から立ち向かう（解を記述する）ために, 数学者達は虚数 $i$ を考えた. そして, 虚数 $i$ の登場により, $\mathbb{R}$ よりも大きな数の体系, 複素数体 $\mathbb{C}$ が現れた. 方程式の解を記述するというごく単純な動機により考えられた虚数だが, 現在では, もの凄く巨大で, 重要で, 複雑な理論が, あろうことか虚数を礎として成り立っている.

このように, 時として我々は今まで考えていたものを上手に拡張し, 様々な実りある結果を得なければならないことがある. $\mathbb{R}$ から $\mathbb{C}$ への飛躍は, まさにその最たる一例である. そして私達がこれから行うのは, **有限体 $GF(p)$ の拡張**だ. 拡張の理由は, $\mathbb{R}\to\mathbb{C}$ のときと全く同じである. 有限体 $GF(p)$ を綺麗に**「広げる」**ことで, 私達の目の前の世界は, 今までとの比にならないほどに広く, 壮大に, そして芳醇に姿を変えるのだ.

### 4.5.1　体 K 上の多項式環

$GF(p)$ に上手い拡張を与えるために, まず, 体 $K$ 上の**多項式環**を定義しよう.

**Def : 体 K 上の多項式環**

$K$ を体としたとき, **体 $K$ の元を係数として持つ多項式全体が成す集合** $K[x]$ を,
体 $K$ 上の**多項式環** (polynomial ring) という.

体上の多項式環[*13]を考えることにより, $GF(p)$ は上手い具合に拡張を行うことができる.
体 $K$ 上の多項式環 $K[x]$ の元 $f(x)$ は, 必ず次のような形で表すことが可能である.

$$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0\ ,\ (a_k\in K, x\in K)$$

$K[x]$ の元のうち, 特に最高次数の項の係数が 1 ($a_n=1$) であるものを, **モニックな多項式**と呼ぶ.

### 4.5.2　既約多項式

さて, 体 $K$ 上の多項式環の定義が終わったところで, 今度は**既約多項式**を定義しよう.

**Def : 既約多項式**

体 $K$ 上の多項式環 $K[x]$ の元のうち, 因数分解を行うことができない多項式 を,
$K[x]$ の**既約多項式** (irreducible polynomial) と呼ぶ.

すなわち, $f(x)\in K[x]$ を考えたときに, $f(\alpha)=0,\ \alpha\in K$ を満たす $\alpha$[*14]が存在しない（$f(x)\in K[x],\ \nexists\alpha\in K\ \text{s.t.}\ f(\alpha)=0$）ならば, $f(x)$ を**既約多項式**と呼ぶ. 既約多項式は, 次の重要な性質を持つ.

- $K[x]$ の任意の元は, $K[x]$ の既約多項式の積に, **一意に因数分解**できる[*15].

---

[*13] 【Advance】 体 $K$ 上の多項式環 $K[x]$ は, その名が表す通り**環** (ring) を成す. また, 体上の多項式環は Euclid **環** (Euclid ring) である.

[*14] このような $\alpha$ を, $f(x)$ の**根** (root), または**零点** (zero) と呼ぶ.

[*15] お気づきだろうか. 実は, $K[x]$ の既約多項式は, $K[x]$ において**素数**と全く同じ役割を演じる. 多項式を既約多項式の積で表示することは, まさに中学校で習った**素因数分解**そのものに対応しているのである.
【Advance】 $K[x]$ の既約多項式全体は環 $K[x]$ の**イデアル** (ideal) を成し, 更には, **素イデアル** (prime ideal) となる.

**例** 体 GF(2) 上の既約多項式

今, GF(2) 上の多項式環 GF(2)[x] を考える.

$$^{\forall}f(x) \in GF(2)[x]\ ,\ f(x) = a_n x_n + \cdots + a_1 x + a_0\ (a_k \in GF(2), x \in GF(2))$$

GF(2) 上の多項式環 GF(2)[x] の既約多項式の例を挙げよう. 以下に示す多項式は, 2 **次の既約多項式**である.

$$g(x) = x^2 + x + 1.$$

なぜなら, $g(0) = 1 \neq 0, g(1) = 1 \neq 0$ となり, 体 GF(2) 上に g(x) の零点が存在しない. すなわち,
GF(2) 上で g(x) は因数分解を行うことができないからだ. また, もっと単純な 1 次以下の既約多項式は,

$$h(x) = x + 1\ ,\ \ell(x) = 1.$$

がある. この h(x), ℓ(x) は, GF(2) に零点を持たないことが容易に確かめられる.

### 4.5.3 Galois 拡大体

さて, それでは, いよいよ Galois 体 GF(p) を**拡大**し, Galois **拡大体** $GF(p^m)$ を構成することを考えよう.Galois 拡大体は, **符号理論の最も大きな基盤であり礎である**. $GF(p^m)$ をしっかり理解しておかなければ, 符号理論を制覇するのは難しいだろう. 逆にいえば, **これさえ理解しておけば, 符号理論はそれほど怖くない**.

では早速, 具体的に $GF(p^m)$ を定義するために, まずは簡単な$GF(2^m)$をどのようにして構成するのかについて示すことにする. そして, $GF(2^m)$ をもとにして, 一般的な $GF(p^m)$ を構成することにしよう. というわけで, さっそく $GF(2^m)$ を定義する.

**Def :** Galois **拡大体** $GF(2^m)$

$GF(2^m)$ を次のように定義する.

$$GF(2^m) = \{(a_{m-1}, a_{m-2}, \cdots, a_1, a_0) \mid a_k \in GF(2), k = 0, 1, \cdots, m-1\}$$

すなわち, $GF(2^m)$ は, **要素として** GF(2) **の元を持つ** m **次元ベクトル（行ベクトル）全体が成す集合**である. $GF(2^m)$ について, $|GF(2^m)| = 2^m$ が成り立つ.

このように, GF(2) 上の m 次元ベクトルを全て集めて, その集合に $GF(2^m)$ という名前を付けるのだ.（ベクトルの各要素は $GF(2) = \{0, 1\}$ の元なので, 例えば要素は $(0, 0, 1, 0, \cdots, 1)$ みたいな感じ）

実はこのように定義した $GF(2^m)$ は, **上手く加算と乗算を定義することによって体を成す**.
というわけでこれから, $GF(2^m)$ が体を成すように, 上手に加算と乗算を定義してみよう. まずは加算から.

**Def :** $GF(2^m)$ **における加算**

$GF(2^m)$ 上での加算を, **成分毎の,** 2 **を法とする和**によって定義する.

このように, $GF(2^m)$ での加算は, 通常のベクトルの加算において, 2 を法とするものを採用する.
こんな風に $GF(2^m)$ に和を定義すると, $GF(2^m)$ は, **和について閉じている体系**となるのだ.

次に, $GF(2^m)$ において積を定義する. が, 積の定義は**少し複雑**である. 積の定義をきちんと理解するための準備として, まずは, $GF(2^m)$ **の元と,** $GF(2)[x]$**（**$GF(2)$ **上の多項式環）の元を対応付ける**という考え方を導入しよう. この考え方は, 今後非常に重要だ.

**Def :** $GF(2^m)$ **と** $GF(2)[x]$ **の対応**

$^\forall a \in GF(2^m)$ を考える.

$$a = (a_{m-1}, a_{m-2}, \cdots, a_0).$$

このとき, $a$ ($m$ 次元ベクトル) と, 以下に示す $GF(2)[x]$ の元（多項式）$a(x)$ を**対応**させることにする.

$$a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0.$$

このように捉えると, $GF(2)$ の $m$ 次元ベクトル $a$ と, $GF(2)$ 上の $m$ 次多項式 $a(x)$ は, **単に表現の仕方が違うだけで, 本質的には同じもの**であるということが分かる. これから, この対応付けはもの凄く重要な役割を演じることになるので, よく読んで理解しておこう.

では, いよいよ, $GF(2^m)$ における乗算を定義しよう.

まず最初に, $f(x)$ を, $GF(2)[x]$ の**ある** $m$ **次の既約多項式**としよう. $GF(2^m)$ において乗算を定義するためには, $m$ 次の既約多項式を必ず 1 つ使わなければならない. この既約多項式は, 特にどう選んでも問題はない.

**Def :** $GF(2^m)$ **における乗算**

$GF(2^m)$ における, $a(x), b(x)$ の**乗算**を, 次のように定義する.

$$a(x)b(x) = \underbrace{a(x)b(x)}_{\text{普通の積}} \mod \underbrace{f(x)}_{\text{既約多項式}}$$

ただし, $f(x)$ ($\in GF(2)[x]$) は, $m$ 次の既約多項式である.

このように, $GF(2^m)$ における積は, mod を使って定義される.
もう少しかみ砕いて説明をすると, まず, $a(x)b(x)$ を普通の多項式の掛け算（$x$+1 と $x$ ならば, $x(x+1) = x^2+x$ のように）によって求め, それを $f(x)$（既約多項式）によって割る.　その余りを, $GF(2^m)$ における積 $a(x)b(x)$ として採用するのだ. ちょっとややこしいが, 理解できるだろうか？

**例**　$GF(2^2) = GF(4)$ における積.

$GF(2^2)$ の 2 つの元 $a = (1,1), b = (1,0)$ を取ろう. この 2 つの元は, $GF(2)[x]$ の元 $a(x) = x + 1, b(x) = x$ と対応している. 更に, $GF(2)[x]$ 上の 2 次の既約多項式 $f(x) = x^2 + x + 1$ を考える. この $f(x)$ を用いて, $a(x)$ と $b(x)$ の $GF(2^2)$ における積を求めてみよう.

まず, $a(x) = x + 1, b(x) = x$ を素直にかけると, $a(x)b(x) = x^2 + x$ となる.
次に, この結果を, 既約多項式 $f(x) = x^2 + x + 1$ によって割った余りを求めよう.

$$
\begin{array}{r}
1 \\
x^2+x+1 \overline{)\, x^2+x} \\
\underline{x^2+x+1} \\
-1
\end{array}
$$

よって, 余り $r(x) = -1$ となり, GF(2) では $-1 = 1$ が成り立つので, 余り $r(x) = 1$ となる. よって, $GF(2^2)$ においては, $a(x)b(x) = 1$ となる. すなわち, 多項式とベクトルの対応により, $a \cdot b = (0,1)$ である. //

このように $GF(2^m)$ における積を定義すれば, $GF(2^m)$ は, **積について閉じている体系**となる. よって, $GF(2^m)$ は和と積について閉じており, 更にそのことは, 差と商（**ゼロ割は除く**）についても閉じていることを意味し, $GF(2^m)$ は**体を成す**. また, $|GF(2^m)| = 2^m$ により, $GF(2^m)$ は**有限体**である.

**Thm : $GF(2^m)$ は体を成す**

$GF(2^m)$ は, 前述のように定義した和と積について**体を成す**.
(差, 商は, それぞれ**逆元との和, 積**によって定義される)

**例** 有限体 $GF(2^2)$ における加算と乗算.

$GF(2^2)$ において, 加算と乗算の結果をまとめた**加算表, 乗算表**を作ってみよう.
ただし, ベクトル $(i,j)$ を, 簡単に $ij$ と表すことにする（例えば $(0,1) = 01$ のように略記する）.

| $+$ | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 00 | 00 | 01 | 10 | 11 |
| 01 | 01 | 00 | 11 | 10 |
| 10 | 10 | 11 | 00 | 01 |
| 11 | 11 | 10 | 01 | 00 |

| $\cdot$ | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 00 | 00 | 00 | 00 | 00 |
| 01 | 00 | 01 | 10 | 11 |
| 10 | 00 | 10 | 11 | 01 |
| 11 | 00 | 11 | 01 | 10 |

乗算表により, $GF(2^2)$ における**積の単位元**[*16]は $(0,1) = 01$ であることが分かる.
更に, 今度は, 乗算表をもとにして, 次のような表を完成させてみよう.

| $a$ | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| $a^2$ | 00 | 01 | 11 | 10 |
| $a^3$ | 00 | 01 | 01 | 01 |

表より, $a^{4-1} = a^3 = \underbrace{(0,1) = 01}_{\text{積の単位元}}, a \neq a^2 \neq a^3$ を満たす元, すなわち**原始元**が $10, 11$ であることが示される.

Galois 体の節で述べたように, $GF(2^2)$ において, 00 以外の全ての元は, 原始元 $10, 11$ のべき乗で表示できている. このように, 原始元を用いると, $GF(2^m)$ における乗算は容易に行うことができる.
今, $\alpha = (1,0) = 10$ として, 次のような**対応表**を作ってみよう. このような対応表を作っておくと, 体の元の対応関係を明確にすることができるので, とても便利である.

[*16] $a \cdot e = e \cdot a = a$ を満たす元 $e$ のこと.

| ベクトル | $\alpha$ のべき | 多項式 |
|---|---|---|
| 00 | - | 0 |
| 01 | $\alpha^3$ | 1 |
| 10 | $\alpha^1$ | x |
| 11 | $\alpha^2$ | $x+1$ |

//

このように, GF($2^m$) を無事に定義することが出来た. 和と積を上手に定義することにより, GF(2) 上の m 次元ベクトル全体が有限体を成す. そして, そのように構成した有限体にもやはり原始元が存在し, 全ての元が原始元によって表示できる. この美しい対応関係をまずはじっくりと味わっておこう.

さて, 今までは GF($2^m$) について述べてきたが, 実はもっと一般的な GF($p^m$) を考えることも出来る.

**Def :** GF($p^m$)

p を素数とし, GF($p^m$) を次のように定義する.

$$GF(p^m) = \{(a_{m-1}, a_{m-2}, \cdots, a_1, a_0) \mid a_k \in GF(p), k = 0, 1, \cdots, m-1\}$$

GF($p^m$) は, p **元** m **次元ベクトル**の集合となる.

このように定義した GF($p^m$) について, 次のことが証明されている.

**Thm :** GF($p^m$) **は体を成す**

GF($p^m$) は, GF($2^m$) と同様の手順で和と積を定義すると**体を成す**.
更に $|GF(p^m)| = p^m$ となり, GF($p^m$) は**有限体**となる.

これで, 任意の素数に対する GF(p) を上手く拡張して, 有限体 GF($p^m$) を構成することができた.
GF($p^m$) の構成は確かに少し複雑な手順を含んでいたが, その複雑さ以上に, GF($p^m$) は非常に美しく , 面白い数学的性質を持っている. これらの有限体と上手くお付き合いするには, たくさんたくさん, とにかく GF($p^m$) と遊んでみることが重要である. 有限体と遊べば遊ぶほど, 有限体の魅力的な部分が沢山見えてくるはずだ. 是非是非, 勉強だという先入観すら一度捨てて, 有限体と自由気ままに戯れてみよう.

というわけで, これで有限体の定義と, 諸性質についての議論はオシマイです. 有限体を上手く構成する巧みな手法, そして有限体が見せる興味深く美しい振る舞い. いかがだっただろうか. これから, この有限体を土台にして, **符号理論**という壮大な世界が見事に組み立てられて行く. 是非楽しんで, 一歩一歩前へ進んで行こう.

## 4.6 練習問題

1. 5 次の DFT 行列について, 3,4,5 行を抜き出し, 列ベクトルの線形独立性を確認せよ.
2. GF(7) における $4^{-1}$*17を求めよ. また, $4^{-1}$ は 1 つしかないことを確認せよ.
   (HINT : $a \times 4 (a = 1,2,3,4,5,6)$ 計算し, その中に $a \times 4 = 1$ を満たす $a$ が 1 つしかないことを確認する. そのような $a$ が, $4^{-1}$ である.)
3. $p = 5$ として次の問に答えよ.
   (1) GF(p) の原始元を全て求めよ.
   (2) $a = 0$ を除いたすべての $a \in GF(p)$ に対して, $a^{p-1} = 1$ であることを確認せよ.
4. GF(7) において, 次の値を求めよ.
   $5 + 3, 6 \times 8, 6 - 19, 5 \times 3^{-1}, 5^{-1} - 6^{-1}$
5. GF(2) 上の原始多項式 $f(x) = x^3 + x^2 + 1$ の根を $\alpha$ とするとき, 以下の問に答えよ.
   (1) GF($2^3$) のすべての要素に関する正規表示, ベクトル表示, 多項式表示と指数表示の対応表を作れ.
   (2) 次を計算し, それぞれの答えを正規表示, ベクトル表示, 多項式表示と指数表示で示せ.
   $(\alpha + 1) \times (\alpha^2 + 1), (\alpha + 1) + (\alpha^2 + 1), \alpha^{11}, \alpha^3 - 1, -(\alpha^4 + 5) \times (\alpha^2 + \alpha), \alpha^{13}, (\alpha^4 - \alpha)^{-3}\alpha^2,$
   $(\alpha - 1)^2/(\alpha^2 + 1)^4$.

*17 $4 \times 4^{-1} = 1$ を満たすような元のこと.

# 第5章

# 巡回符号

符号はベクトルの形で記述されてきていたが, 実際のコンピュータ上に実装されている符号化技術は多項式の形を取ることが多い. この章では, 符号を多項式で表現し, また, 多項式で表現されるいろいろな符号とその性質について学ぶ.

## 5.1 符号多項式

今まで, 符号はベクトルの形で書いてきたが, もっと扱いやすくするために, 符号を多項式の形で書くことを考えていく.

**復習** ある線形符号 C の誤り訂正能力が t であるとすると, 以下の式が成り立つ.

$$\begin{aligned} d_{min} &\geq 2t+1 \\ d_{min} &= w_{min} \\ w_{min} - 1 &= r \end{aligned}$$

この r は, 線形符号により作られる検査行列 H の任意な r 列が互いに独立であることを意味している.
ここで, DFT 行列から連続した r 行を抜き出して作られる検査行列 H を考えると, パリティ検査方程式は

$$H^t C = \begin{pmatrix} (\omega^{r_1})^0 & (\omega^{r_1})^1 & \cdots & (\omega^{r_1})^{n-1} \\ (\omega^{r_1+1})^0 & (\omega^{r_1+1})^1 & \cdots & (\omega^{r_1+1})^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega^{r_1+r-1})^0 & (\omega^{r_1+r-1})^1 & \cdots & (\omega^{r_1+r-1})^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{0}$$

H は $r \times n$ 行列, $\mathbf{0}$ は r 行の零ベクトル, $\omega$ は 1 の原始 n 乗根.
また, 長さ n の符号語 $[c_{n-1}c_{n-2}\cdots c_0]$ を次の**符号多項式** $c(x)$ として表現することにする.

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1 x^1 + c_0$$

先ほどの行列の積を計算すると,

$$\begin{cases} c(\omega^{r_1}) = c_0(\omega^{r_1})^0 + c_1(\omega^{r_1})^1 + \cdots + c_{n-1}(\omega^{r_1})^{n-1} = 0 \\ c(\omega^{r_1+1}) = c_0(\omega^{r_1+1})^0 + c_1(\omega^{r_1+1})^1 + \cdots + c_{n-1}(\omega^{r_1+1})^{n-1} = 0 \\ \vdots \\ c(\omega^{r_1+r-1}) = c_0(\omega^{r_1+r-1})^0 + c_1(\omega^{r_1+r-1})^1 + \cdots + c_{n-1}(\omega^{r_1+r-1})^{n-1} = 0 \end{cases}$$

よって,　$\omega^{r_1}, \omega^{r_1+1}, \cdots, \omega^{r_1+r-1}$ は $c(x)$ の根 (解) であることがわかるから, 因数定理より $c(x)$ を

$$c(x) = (x-\omega^{r_1})(x-\omega^{r_1+1})\cdots(x-\omega^{r_1+r-1})m(x)$$

と表現できる.
ここで,

$$g(x) = (x-\omega^{r_1})(x-\omega^{r_1+1})\cdots(x-\omega^{r_1+r-1})$$

とおくと, $c(x)$ は

$$c(x) = g(x)m(x)$$

となる.
$c(x)$ は $x$ の $n-1$ 次の多項式, $g(x)$ は $x$ の $r$ 次の多項式, $m(x)$ は $x$ の $n-r-1$ 次の多項式であり, $c(x)$ で構築される符号は $t$ 個の誤りを訂正できる[*1] .
ここまでを一度整理しよう.

**Def: 符号多項式と, 符号多項式をなす生成多項式及び情報多項式**

長さ $n$ の符号語を多項式 $c(x)$ で表現すると

$$\begin{aligned} c(x) &= c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x^1 + c_0 \\ &= (x-\omega^{r_1})(x-\omega^{r_1+1})\cdots(x-\omega^{r_1+r-1})m(x) \\ &= g(x)m(x) \\ \text{ただし, } g(x) &= \prod_{i=r_1}^{r_1+r-1}(x-\omega^i) \\ &= g_rx^r + \cdots g_0 (g(x) \text{ を生成多項式という}) \\ m(x) &= m_{k-1}x^{k-1} + \cdots + m_0 (k = n-r) (m(x) \text{ を情報多項式という}) \end{aligned}$$

また, 符号多項式と情報多項式の係数 $[c_{n-1}\cdots c_0], [m_{k-1}\cdots m_0]$ がそれぞれ符号語, 情報語に相当する.
更に, 符号語と符号多項式にはこのような対応関係もある.

$$\begin{array}{c} H^tc = 0 \Rightarrow c \text{ は符号語} \\ \Updownarrow \\ c(x) \text{ が } g(x) \text{ で割り切れる } \Rightarrow c(x) \text{ は符号多項式} \end{array}$$

この形で表される符号多項式によってできる符号語は**非組織符号**と呼ばれる. 非組織符号では, 情報語が符号多項式の全ての項にバラバラに分散していて, 管理が面倒である. そこでこれを改良し, 情報語がいつも同じ項に付属するように多項式を定義したものを組織符号多項式という.

[*1] $2t+1 \geq d_{min} = w_{min} = r+1$ より

**Def: 組織符号多項式**

符号多項式 $c(x)$ において, 最初の $k$ 個の項を情報語に対応付ける. そうして作られる符号多項式 $c'(x)$ は

$$c'(x) = m_{k-1}x^{n-1} + \cdots + m_0x^r + c'_{r-1}x^{r-1} + \cdots + c'_0x^0$$

となる. この方程式が, 組織符号多項式となり, 以下の式と同義である.(以下の式から求める.)

$$c'(x) = m(x)x^r - [m(x)x^r \mod g(x)]$$

この $[m(x)x^r \mod g(x)]$ によって得られる多項式は $r-1$ 次であるから,

$$\begin{cases} m(x)x^r = m_{k-1}x^{n-1} + \cdots + m_0x^r \\ -[m(x)x^r \mod g(x)] = c'_{r-1}x^{r-1} + \cdots + c'_0x^0 \end{cases}$$

という関係が導ける.

この $c'(x)$ も,$g(x)$ で割り切れる. この証明は簡単である.

$[c'(x) \mod g(x)] = 0$ であることを証明する.
$c(x)$ の定義から

$$\begin{aligned} [c'(x) \mod g(x)] &= \left[(m(x)x^r - [m(x)x^r \mod g(x)]) \mod g(x)\right] \\ &= [m(x)x^r \mod g(x)] - \left[[m(x)x^r \mod g(x)] \mod g(x)\right] \\ &= [m(x)x^r \mod g(x)] - [m(x)x^r \mod g(x)] \\ &= 0 \end{aligned}$$

よって,$c'(x)$ は符号多項式.

組織符号と非組織符号によって構築される符号語は, ビットパターンは必ずしも一致しない. しかし, 集合としてみると等しいことが証明されている. この証明は省略する.

**例** 生成多項式 $g(x)$ を

$$g(x) = x^3 + x + 1$$

とし, 情報系列を $(m_2, m_1, m_0)$ とすると情報多項式は,

$$m(x) = m_2x^2 + m_1x + m_0$$

であるから, 非組織符号の符号多項式 $c(x)$ は

$$\begin{aligned} c(x) &= g(x)m(x) \\ &= m_2x^5 + m_1x^4 + (m_2 + m_0)x^3 + (m_2 + m_1)x^2 + (m_1 + m_0)x + m_0 \end{aligned}$$

となり, 情報系列の各シンボルは符号語の要素に複雑に分散している. 一方, 組織符号の符号多項式 $c'(x)$ は

$$\begin{aligned} c'(x) &= m(x)x^3 - [m(x)x^3 \mod g(x)] \\ &= m_2x^5 + m_1x^4 + m_0x^3 + (m_2 + m_1)x^2 + (m_2 + m_1 + m_0)x + m_2 + m_0 \end{aligned}$$

となり, 情報シンボルの系列に検査シンボルが連なった形をしている.

さて, 符号多項式を定義したことによって, 符号理論の世界はまた一つ広がりを見せることとなる. 以降の章では, 符号多項式により構築される様々な符号を紹介していくことにしよう.

## 5.2 巡回符号

**巡回符号 (Cyclic code)** とは, 符号語を巡回シフトした系列がまた符号語になるような符号のことをいう. といきなり解説されたところで何が何やらといったところであると思う. それではまず “巡回とは何か” を説明した上で, 再び巡回符号を定義していこう.

**Def: 符号の巡回シフト**

ある長さ $n$ の符号語が与えられたとき, 最上位のシンボルを最下位に移動させる処理を, 符号語の**巡回シフト**という.

$$[c_{n-1}c_{n-2}\cdots c_1c_0]$$
$$\Downarrow \text{巡回シフト}$$
$$[c_{n-2}c_{n-3}\cdots c_1c_0\mathbf{c_{n-1}}]$$

この巡回シフトを多項式で表現すると, 以下のようになる.

$$xc(x) + c_{n-1} - c_{n-1}x^n = xg(x)m(x) - c_{n-1}(x^n - 1)$$

この巡回シフト後の符号多項式が $g(x)$ で割り切れるならば, シフト後の系列は符号語となる. 実は, **ある生成多項式 $g(x)$ が与えられたとき, $g(x)$ で割り切れる $x^n - 1$ という形の $n$ 次の多項式が必ず存在する**ことが知られているので, 巡回シフト後の符号多項式は**必ず $g(x)$ で割り切れる**. これを踏まえて, 巡回符号の定義を行う.

**Def: 巡回符号**

符号語を巡回シフトさせ, その系列もまた符号語になるような符号を**巡回符号**という.

**例** GF(2) において

$$\begin{cases} c(x) = m(x)g(x) \\ g(x) = x^3 + x + 1 \\ m(x) = x^3 + 1 \end{cases}$$

とすると, 非組織符号多項式 $c(x)$ は

$$c(x) = x^6 + x^4 + x + 1$$

ここで, 情報ビットと符号ビットはそれぞれ

$$[m_3m_2m_1m_0] = [1001]$$
$$[c_6c_5\cdots c_1c_0] = [1010011]$$

となる.$c(x)$ を 1 ビット巡回させた多項式を $c'(x)$ とすると,

$$\begin{aligned} c'(x) &= c_5x^6 + c_4x^5 + \cdots + c_0x + c_6 \\ &= x^5 + x^2 + x + 1 \end{aligned}$$

巡回後の情報多項式は

$$c'(x) \div g(x) = m(x) = x^2 - 1$$

よって, この $c'(x)$ は $g(x)$ で割り切れるので, 符号多項式である. また, 情報ビットと符号ビットはそれぞれ

$$[m_3 m_2 m_1 m_0] = [0101]$$
$$[c_6 c_5 \cdots c_1 c_0] = [0100111]$$

である.

## 5.3 巡回冗長検査

**巡回冗長検査 (Cyclic Redundancy Check, CRC)** とは, 誤り検査法の一種で, 組織符号のように, 情報系列に検査シンボルを付加して送信されたものに対して, 受信側で, 受信した多項式が $g(x)$ で割り切れるかどうかで誤り検出を行う方法のことをいう. 巡回冗長検査 (以下, CRC) は論理回路で簡単に構築でき (ハード的にも簡単), かつ高い誤り検出能力を有することから, 実際のコンピュータに広く実用されている. CRC は, 送信側の符号化回路, 受信側の検出回路によって構築される. まずは符号化の概念から始めよう. また, この資料では簡単のために, **CRC の生成多項式は必ず $(x+1)$ を因数としてもつものとする.**

### 5.3.1 符号化, 検査回路

まず, 送信したい情報語 (長さ $k$) から組織符号 (長さ $n$) を構築する. 生成多項式 $g(x)$ は, 送信側と受信側で共通のものを使う. また, $g(x)$ は $r$ 次の多項式とする.($r = n - k$) 情報多項式と生成多項式から作られる組織符号多項式 $c_s(x)$ は

$$c_s(x) = m(x)x^r - [m(x)x^r \mod g(x)]$$

となる. この多項式を, 図 5.1 に示す **CRC 符号化回路**で符号化する[*1].

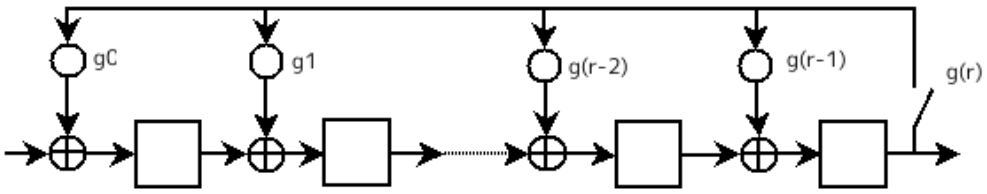



図 5.1 符号化回路

**例** $r = 3$ のときの CRC 符号化

$$\begin{cases} g(x) = x^3 + x + 1 \\ m(x) = x^5 + x^4 + x + 1 \end{cases}$$

とする. $c_s(x)$ は,

$$c_s(x) = m(x)x^3 - [m(x)x^3 \mod g(x)] = \underbrace{x^8 + x^7 + x^4 + x^3}_{\text{情報系列}} + \underbrace{c_2x^2 + c_1x + c_0}_{\text{未決定なシンボル}}$$

となる.CRC では符号化回路によって $c_s(x)$ 中の $[m(x)x^3 \mod g(x)]$ を求め, その多項式を情報系列に付加することで, CRC 符号化を行う[*2].

[*1] これを CRC 符号という.

[*2] つまり, 剰余を計算するのではなく, 多項式を回路に突っ込むことで, 組織符号化を行うのが CRC 符号化である.

では, 符号化を手順を踏んで行っていこう.

Step1. **回路を書く**　まずは回路を書く. 箱[*3] を r 個横に並べ, 箱の前に $\oplus$, $\oplus$ の上に $\otimes$ をそれぞれ書く. 一番右の箱からはスイッチを繋ぎ, スイッチからそれぞれの $\otimes$ に向けて矢印を引こう. 出来上がる回路は図 5.2 のようになる.

Step2. **符号化表を作る**　まず表 5.3.1 を見てほしい. このように, 次入力, 生成多項式の係数, スイッチの状態, 出力を表にする. 表が完成したら, 以下のような手順で表を埋めていく.

1. 次入力を箱 1 に入れる. 箱 1 に入力がある場合, 箱 2 へ移動する. 繰り返し, 箱を埋めていく
2. 箱が全て埋まったら, 最右の箱の値によってスイッチの ON/OFF を記入する (1 なら ON, 0 なら OFF).
3. スイッチが OFF なら, その行の g の値を全て 0 にする.
4. 次入力を箱 1 に格納する際に, 入力の右隣にある g との和をとる. また, 最右の箱の値は出力欄に書き込む.
5. 2 から 4 を繰り返し,$\phi$ の行まで埋まったら終了. 全てを埋めた表を表 5.3.1 に示す.

Step3. **符号多項式を作る**　表の最後の行の箱に入った値を $m(x)x^3$ に加えれば, CRC 符号化は完了である.

こうして完成した CRC 符号多項式は

$$c_s(x) = x^8 + x^7 + x^4 + x^3 + x^2 + x$$

である. これが本当に

$$c_s(x) = m(x)x^3 - [m(x)x^3 \mod g(x)]$$

であるかどうかは, 実際に割り算をしてみればわかる. 時間があれば確かめてみてほしい.

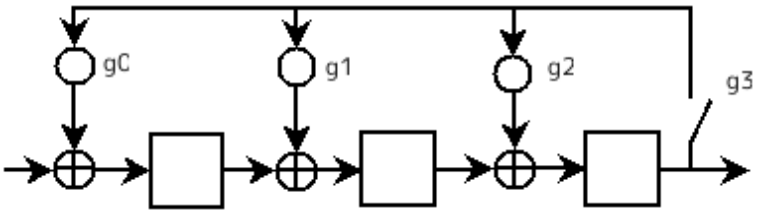



図 5.2　r=3 の時の符号化回路

次に, 受信した系列が誤っていないかの検出だが, これも符号化の時と同様に行うことができる. 検査回路は図 5.3 のようになる. 符号化の時と同様, 受信した系列に対して符号化表を作成し, 最後の行の箱の値が全て 0 となっていなければ, それは $g(x)$ で割り切れていないことと同義であるため, 誤りとなる.

### 5.3.2　CRC の特性

■(1) CRC **符号の符号長**　CRC 符号の長さを, 生成多項式 $g(x)$ が $x^n - 1$ を割り切ることのできる最小の $n$ とする. $g(x)$ を因数分解して, 既約多項式の積にすると, 既約多項式の中で解に原始元をもつもの (これを**原始多項式**という) が現れる. $x^n - 1$ は原始多項式でも割り切ることができる. ここで, 原始多項式の解を $\alpha$ とおくと, $\alpha^n = 1$ を満たす $n$ は原始多項式の係数のパターン数 (=濃度) となるから,$n = 2^p - 1$ となる.(GF(2) 上,p は原始多項式の次数)

[*3] 実はこの箱には**シフトレジスタ**という名前があるが, 面倒なので箱と呼ばせていただく

表 5.1 符号化表 (r=3)

| 次入力 | $g_0$ | 箱 1 | $g_1$ | 箱 2 | $g_2$ | 箱 3 | スイッチ | 出力 |
|---|---|---|---|---|---|---|---|---|
| $x^8 = 1$ | 1 | | 1 | | 0 | | | |
| $x^7 = 1$ | 1 | | 1 | | 0 | | | |
| $x^6 = 0$ | 1 | | 1 | | 0 | | | |
| $x^5 = 0$ | 1 | | 1 | | 0 | | | |
| $x^4 = 1$ | 1 | | 1 | | 0 | | | |
| $x^3 = 1$ | 1 | | 1 | | 0 | | | |
| $x^2 = 0$ | 1 | | 1 | | 0 | | | |
| $x^1 = 0$ | 1 | | 1 | | 0 | | | |
| $x^0 = 0$ | 1 | | 1 | | 0 | | | |
| $\phi$ | 1 | | 1 | | 0 | | | |

表 5.2 完成した符号化表 (r=3)

| 次入力 | $g_0$ | 箱 1 | $g_1$ | 箱 2 | $g_2$ | 箱 3 | スイッチ | 出力 |
|---|---|---|---|---|---|---|---|---|
| $x^8 = 1$ | 1 | | 1 | | 0 | | | |
| $x^7 = 1$ | 1 | $x^8 = 1$ | 1 | | 0 | | | |
| $x^6 = 0$ | 1 | $x^7 = 1$ | 1 | $x^8 = 1$ | 0 | | | |
| $x^5 = 0$ | 1 | $x^6 = 0$ | 1 | $x^7 = 1$ | 0 | $x^8 = 1$ | ON | |
| $x^4 = 1$ | 1 | $x^5 = 1$ | 1 | $x^6 = 1$ | 0 | $x^7 = 1$ | ON | $x^8 = 1$ |
| $x^3 = 1$ | 1 | $x^4 = 0$ | 1 | $x^5 = 0$ | 0 | $x^6 = 1$ | ON | $x^7 = 1$ |
| $x^2 = 0$ | 0 | $x^3 = 0$ | 0 | $x^4 = 1$ | 0 | $x^5 = 0$ | OFF | $x^6 = 1$ |
| $x^1 = 0$ | 1 | $x^2 = 0$ | 1 | $x^3 = 0$ | 0 | $x^4 = 1$ | ON | $x^5 = 0$ |
| $x^0 = 0$ | 0 | $x^1 = 1$ | 0 | $x^2 = 1$ | 0 | $x^3 = 0$ | OFF | $x^4 = 1$ |
| $\phi$ | 1 | $x^0 = 0$ | 1 | $x^1 = 1$ | 0 | $x^2 = 1$ | ON | $x^3 = 0$ |

**例** CRC の生成多項式 $g(x)$ を

$$\begin{aligned} g(x) &= x^{16} + x^{12} + x^5 1 \\ &= (x+1)(\underbrace{x^{15} + x^{14} + x^{13} + x^{12} + x^4 + x^3 + x^2 + x + 1}_{\text{原始多項式}}) \end{aligned}$$

とすると, $n = 2^{15} - 1 = 32767$ となり, 情報語長は $k = n - r$ より $32767 - 16 = 32751$ となる.

**■(2) 誤り検出能力 (ランダムエラー)**

(a) **奇数ビットの誤り** 受信した多項式を $R(x)$ とする. $R(x) = c(x) + e(x)$ である. この $e(x)$ を, **誤り多項式** という.

$[e(x) \mod g(x)] = 0$ であれば誤り検出が不可能 (見逃し) であり, 0 でなければ誤り検出ができたことになる.

ここで, $g(x)$ を既約多項式に因数分解すると $g(x) = b(x)a(x)$ が得られる. このとき, $b(x) = (x+1)$ とおく. すると, 誤り検出の条件を $[e(x) \mod g(x)]$ を $[e(x) \mod b(x)]$ と書き換えることができる.

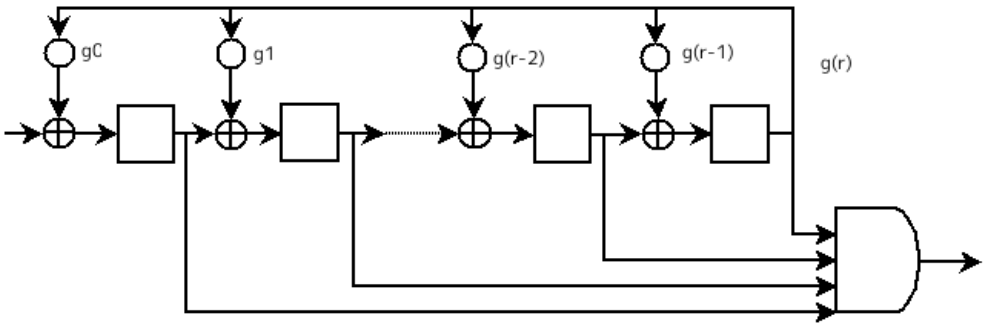



図 5.3　検査回路

$b(x) = 0$ とおくと, $x = 1$ となる. よって誤り多項式は

$$e(1) = e_{x_1} + e_{x_2} + \cdots e_{x_k} = \begin{cases} 0 \ (e_x\text{が偶数個の時}) \\ 1 \ (e_x\text{が奇数個の時}) \end{cases}$$

となり, 誤りビット数が奇数個であれば, 確実に検出が可能となる.

(b) 2 **シンボルの誤り**　$e(x)$ の任意の $e_i, e_j$ $(i \neq j)$ が 1 であるとする (2 ビット誤り). すると誤り多項式 $e(x)$ は

$$e(x) = e_i x^i + e_j x^j = x^i x^j$$

と変形できる. ここで, $i > j$ とすると,

$$e(x) = x^j(1 + x^q) \ (j + q = i)$$

と変形できる. $q$ の取りうる範囲は, $i$ と $j$ の関係から $1 \leq q \leq 2^p - 2$ となる. また, $g(x) = b(x)a(x)$(ただし $a(x)$ は原始多項式) とすれば, $a(x) = 0$ ならば $x^{2^p-1} = 1, x^u \neq 1$ $(0 \leq u \leq 2^p - 2)$ という関係が導かれる. これを踏まえて, $e(x) \mod g(x)$ を考える. $e(x)$ を変形すれば

$$x^j(x^q + 1) \mod g(x)$$

となる. $g(x)$ が割りきれる $x^n - 1$ の $n$ の定義より, $n > q$ なので, $g(x)$ は $(x^q + 1)$ を割り切れない. よって, $e(x) \mod g(x) \neq 0$ となり, 2 ビットエラーは必ず検出が可能となる.

以上のことから, CRC は最小で 3 ビット, または奇数ビットの誤りは必ず検出できることがわかる.

■(3) **バーストエラー**　ある $e(x)$ をビットパターンに直したとき, 最初の零でないシンボルから最後の零でないシンボルまでの長さを**バーストエラー長**という. バーストエラー長 $m$ であれば, $m - 1$ 次の多項式と $x$ のべき乗の積で表すことができる.

**例**　先ほどの生成多項式で作られる CRC 符号を考える.

(a) **バーストエラー長 16 以下のとき**　バーストエラーにより作られる誤り多項式の次数は最大でも 15 となる. よって, $g(x)$ では割り切れず, 必ず誤り検出ができる.

(b) **バーストエラー長 17 のとき**　長さが 17 のバーストエラーは, 両端が 1 で, 中間のビットは任意であるから, $2^{15}$ 通り存在する. このうち, $g(x)$ と等しいものだけ (この場合, $x^i(x^{16} + x^{15} + x^5 + 1)$[*4] が割り切れるので, 誤り見逃し確率は $\frac{1}{2^{15}}$ となる.

[*4] この $x^i$ は, バーストエラーの位置によって決まる

(c) **バーストエラー長** $16+l$ $(l \geq 2)$ **のとき**　バーストエラーのパターン数は $2^{14+l}$ 通りあるが, このうち $e(x) = x^i(x^{l-1} + e_{l-2}x^{l-2} + \cdots + e_1x + 1)g(x)$ というパターンのみ, $g(x)$ で割り切ることができる. このパターンの総数は $2^{l-2}$個 であるから, 見逃し確率は $\frac{2^{l-2}}{2^{14+l}} = \frac{1}{2^{16}}$ となる.

## 5.4 練習問題

1. 生成多項式 $g(x) = x^3 + x^2 + 1$, 情報多項式 $g(x) = x^2 + 1$ とするとき, GF(2) 上の巡回符号に関して以下の問に答えよ. なお, ビット列は次数の高い方を左に書くこと.
   (1) $g(x)$ を原始多項式とすると, 符号長はいくらか. 情報語の長さはいくらか.
   (2) 情報語のビット列を書け.
   (3) 非組織符号多項式 $c(x)$ を求めよ. そのビット列を書け.
   (4) 非組織符号多項式 $c(x)$ の巡回性を確認せよ. ただし, 巡回シフトは 1 回のみで良い.
   (5) 組織符号多項式 $c_s(x)$ を求めよ. また, そのビット列を書け.
   (6) $c_s(x)$ を, CRC 符号化回路で求めよ.
   (7) $c_s(x)$ の巡回性を確認せよ.
   (8) $R_1(x) = x^5 + x^3 + x^2 + x + 1$ は符号多項式か. CRC 誤り検出回路で検査せよ. 符号語である場合, 次の問に答えよ.
      i. $R_1(x)$ を非組織符号多項式として, 対応する情報多項式を求めよ. その時のビット列を書け.
      ii. $R_1(x)$ を組織符号多項式として, 対応する情報多項式を求めよ. その時のビット列を書け.
   (9) $R_2(x) = x^5 + x^4 + x^3 + 1$ について前問と同様に答えよ. 誤り検出はどの方式を使っても良い. (CRC 回路を使っても, $R_2(x)$ を $g(x)$ で割っても良い.)

2. GF(2) 上の巡回符号である, CRC $-$ 12 の生成多項式は $g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1 = (x+1)(x^{11} + x^2 + 1)$ により与えられる. ただし, $x^{11} + x^2 + 1$ は原始多項式である. このとき, 以下の問に答えよ.
   (a) 符号語と情報語の長さはそれぞれいくらか.
   (b) 何ビットまでの誤りを必ず検出できるか.
   (c) 誤りバーストの長さはいくら以下なら必ず検出できるか.
   (d) 誤りバーストの長さが誤り検出可能な長さを超えた場合, 誤り見逃し確率はいくらか.

# 第6章

# リード・ソロモン符号

ここまで扱ってきた巡回符号は, 誤りを検出することしか考えていなかった. 符号理論をリアルタイムに応用するには, 訂正についても考えていかなくてはならない. この章と次の章で扱う BCH 符号は, 現実世界でも応用されている符号である.
また, ここからの演算は簡単のためすべて GF(2) 上のものとする.

## 6.1 リード・ソロモン符号の構築

**リード・ソロモン符号 (Reed-Solomon Coding, 以降は RS 符号と表記)** は, 1960 年にリード氏とソロモン氏が開発した誤り訂正符号で, 符号の生成と復号が複雑だが誤り訂正能力が高く, 地デジや ADSL などに実用化されている.
早速, RS 符号の構築をはじめよう.
$GF(2^s)$ の原始元の一つを $\omega$ とすると,

$$\omega^i \neq 1, (i = 0, 1, \cdots, 2^s - 2) \text{ かつ} \omega^{2s-1} = 1$$

であるから, $\omega$ は 1 の原始 $2^s - 1$ 乗根であり, $n$ 次の DFT 行列から $r$ 行抜き出した行列 $D_r$ において $n = 2^s - 1, r = 2t$ とすれば, $GF(2^s)$ 上の $t$ 重訂正可能な符号の検査行列が得られる. この検査行列から生成多項式 $g(x)$ を作ると,

$$g(x) = (x - \omega^{r_1})(x - \omega^{r_1+1}) \cdots (x - \omega^{r_1+r-1}) = \prod_{i=r_1}^{r_1+r-1} (x - \omega^i)$$

となる. この $r_1$ の値は任意であるが, RS 符号を構築する際はふつう 0 を採用する. つまり, 一般的な RS 符号は以下のように定義できる.

**Def: RS 符号**

$GF(2^s)$ 上の $t$ 重誤り訂正符号 (RS 符号) の生成多項式 $g(x)$ は

$$g(x) = \prod_{i=0}^{2t-1} (x - \omega^i)$$

この生成多項式と, 情報多項式を元に作られる符号が **RS 符号**となる.

この RS 符号の符号と情報のビット長を考える.

符号語の長さ $n$ は $2^s-1$ であるから, この $n$ は即ちシンボル数と等価であり, 1 つのシンボルが $s$ ビットで表現されることから, 符号ビットの長さは $s(2^s-1)$ となる.
また, $k=n-r$ より, 情報語の長さ $k$ は $2^s-1-r=2^s-2t-1$ となり, ビット長は $s$ をかけて $s(2^s-2t-1)$ となる.

**例** GF($2^s$) 上で考える. GF(2) 上の原始多項式 $f(x)$ を $f(x)=x^3+x+1$ とし, $f(\alpha)=0, t=2$ とする[*1].

まず $g(x)$ を構築しよう. $r=2t=4$ であるから, $g(x)$ は

$$g(x)=(x-\alpha^0)(x-\alpha^1)(x-\alpha^2)(x-\alpha^3)$$

の 4 次方程式となる. 符号語の長さは原始多項式の次数 $s=3$ より, $n=2^3-1=7$ よって, ビット長は $ns=7\times3=21$ また, 符号多項式 $c(x)$ が 6 次となるので, 情報多項式は 2 次となり, 情報語の長さ $k=3$, 情報語のビット長 $ks=9$ も導ける.

それではまず, $g(x)$ を展開する. 原始多項式の性質 $(\alpha^3+\alpha+1=0)$ を利用して展開すると

$$g(x)=x^4+\alpha^2x^3+(\alpha^2+\alpha+1)x^2+(\alpha^2+\alpha+1)x+(\alpha^2+1)$$

情報多項式 $m(x)$ を

$$m(x)=m_2x^2+m_1x+m_0$$

とすると, 符号多項式 $c(x)$ は

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= \quad \vdots \\ &= m_2x^6+(m_2\alpha^2+m_1)x^5+\cdots+m_0 \end{aligned} \tag{6.1}$$

このとき, 情報語 $[m_2m_1m_0]$ はベクトル表示で [000 000 000] から [111 111 111] の $2^9$ 個のパターンを取りうる[*2].

## 6.2 リード・ソロモン符号の復号

### 6.2.1 シンドロームと誤り検出

RS 符号の復号の話の前に, まずシンドロームという概念について解説しておく. **シンドローム (syndrome)** は, ある符号を受信したとき, その符号と検査行列を使って生成できる, **符号の誤り位置情報**のことをいう. 具体的には, 以下の式により定義される.

$$\boldsymbol{s}=H\boldsymbol{R}$$

[*1] この $f(\alpha)$ と $t$ の値から,RS 符号を構築する
[*2] 多項式表示で書くこともできる.

Tips: 検査行列 H

$$H = \begin{pmatrix} (\omega^{r_1})^0 & (\omega^{r_1})^1 & \cdots & (\omega^{r_1})^{n-1} \\ (\omega^{r_1+1})^0 & (\omega^{r_1+1})^1 & \cdots & (\omega^{r_1+1})^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega^{r_1+r-1})^0 & (\omega^{r_1+r-1})^1 & \cdots & (\omega^{r_1+r-1})^{n-1} \end{pmatrix}$$

ここで $\boldsymbol{R}$ は, 符号多項式 $c(x)$ と誤り多項式 $e(x)$ の和 $R(x)$ をベクトル表示にしたものである. 受信した符号が正しいかどうかは, $R(x)$ が $g(x)$ で割りきれるかどうかを判定する必要があったが, これは検査行列を使って書き直せば $H\boldsymbol{R}$ が 0 かを判定すればよいことになる. つまり, シンドローム $\boldsymbol{s}$ が 0 ならば $\boldsymbol{R}$ は符号語であり, 0 でなければ符号語ではない. よって, RS 符号の誤り検出は以下のように定義される.

$$\boldsymbol{s} = \begin{pmatrix} s_0 \\ \vdots \\ s_{r-1} \end{pmatrix} \begin{cases} = \mathbf{o} \Rightarrow \text{誤りなし} \\ \neq \mathbf{o} \Rightarrow \text{誤りあり} \end{cases}$$

このシンドロームを使って復号を行うことを**シンドローム復号**という.

### 6.2.2 RS 符号の誤り訂正

RS 符号の $t$ 重誤り訂正は以下のような手順で行われる.

1. **誤り位置検出**　$0 \sim n-1$ のシンボル中, どこに誤りがあるのか調べる.
2. **誤り訂正**　誤りのあったシンボルの正しい値を求める.

これらはそれぞれ $t$ 個の変数を調べる必要があるため, $t$ 重訂正を行うためには合わせて $2t$ 個の変数を調べる必要がある. まずシンドロームを使って, 誤り位置を検出する.

$$\boldsymbol{s} = H\boldsymbol{R} = H(\boldsymbol{c} + \boldsymbol{e}) = H\boldsymbol{e}$$

これを多項式表示すると

$$\begin{cases} (\alpha^0)^0 e_o + \cdots (\alpha^0)^{n-1} e_{n-1} = s_0 \\ \quad \vdots \\ (\alpha^{r-1})^0 e_o + \cdots (\alpha^{r-1})^{n-1} e_{n-1} = s_{n-1} \end{cases} \tag{6.2}$$

これらは $r$ 個の方程式であり, また $r = 2t$ なので, $2t$ 個の方程式であると言える. この式の中で, $e_0, \cdots, e_{n-1}$ は未知であり,$s_0, \cdots, s_{n-1}$ は既知である. また, $e_0, \cdots, e_{n-1}$ は $t$ 個の誤りの場所と誤り内容を孕んでいる. よってこの未知な情報から, $2t$ 個の変数を用意する. 誤り位置を $i$, 誤り内容を $e_i$ とすると, $2t$ 個の変数が取りうる値は

$$\begin{cases} e_{i_1} \in \{0, \alpha^0, \cdots, \alpha^{2^s-2}\} \\ \quad \vdots \\ e_{i_t} \in \{0, \alpha^0, \cdots, \alpha^{2^s-2}\} \\ i_1 \in \{0, \cdots, n-1\} \\ \quad \vdots \\ i_t \in \{0, \cdots, n-1\} \end{cases} \tag{6.3}$$

となる.

$t \geq 3$ では, 式 (6.2) を解くのが非常に困難なため[*3], ここからは t が 1 か 2 の場合について考えていく.
$t = 2$ のとき, $r = 4$ なので, $H\boldsymbol{R}$ は

$$H\boldsymbol{R} = \begin{pmatrix} (\alpha^0)^0 & \cdots & (\alpha^0)^{n-1} \\ \vdots & \ddots & \vdots \\ (\alpha^3)^0 & \cdots & (\alpha^3)^{n-1} \end{pmatrix} \begin{pmatrix} R_0 \\ \vdots \\ R_{r-1} \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

これによって, $s_0 \sim s_3$ が与えられる. また, $H\boldsymbol{R} = H\boldsymbol{e}$ より

$$H\boldsymbol{R} = \begin{pmatrix} (\alpha^0)^0 & \cdots & (\alpha^0)^{n-1} \\ \vdots & \ddots & \vdots \\ (\alpha^3)^0 & \cdots & (\alpha^3)^{n-1} \end{pmatrix} \begin{pmatrix} e_0 \\ \vdots \\ e_{r-1} \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

と変形できるのでこれを展開し,

$$\begin{cases} (\alpha^0)^0 e_o + \cdots (\alpha^0)^{n-1} e_{n-1} = s_0 \\ \quad \vdots \\ (\alpha^3)^0 e_o + \cdots (\alpha^3)^{n-1} e_{n-1} = s_3 \end{cases} \tag{6.4}$$

これから, $e_0, \cdots, e_{n-1}$ を求める[*4].

### 6.2.3　1 重誤りの訂正

誤りシンボルを $e_i (e_i \neq 0, e_j = 0 (j \neq i), 0 \leq i \leq n-1)$ とする. また,$e_i \in \{0, \alpha, \cdots, \alpha^{2^s-2}\}$ である. 訂正に必要な情報は

- i の値はいくつか (誤り位置)
- $e_i$ の値はいくつか (誤りの値)

であるので, 順を追って求めていく. $e_i$ 以外は 0 であるから, 式 (6.4) は

$$\begin{cases} (\alpha^0)^i e_i = s_0 \\ (\alpha^1)^i e_i = s_1 \\ (\alpha^2)^i e_i = s_2 \\ (\alpha^3)^i e_i = s_3 \end{cases} \tag{6.5}$$

となる. ここで,

$$\begin{cases} \frac{s_1}{s_0} = \frac{(\alpha^1)^i e_i}{(\alpha^0)^i e_i} = a^i \\ \frac{s_2}{s_1} = \frac{(\alpha^2)^i e_i}{(\alpha^1)^i e_i} = a^i \\ \frac{s_3}{s_2} = \frac{(\alpha^3)^i e_i}{(\alpha^2)^i e_i} = a^i \end{cases} \tag{6.6}$$

より,

$$\frac{s_1}{s_0} = \frac{s_2}{s_1} = \frac{s_3}{s_2}$$

が成り立つ. これが, $t = 2$ のときの 1 重誤りの条件である.

---

[*3] デジタル TV では, 内部に $t = 8$ の方程式を解く回路が埋め込まれているらしい
[*4] くどいようだが, この式は t=1,2 の時しか使えないので注意

**Def: 1 重誤りの条件**

シンドローム $\boldsymbol{s}$ を求め, 各要素間が以下の関係を満たすとき, その受信系列は 1 重誤りである.

$$\frac{s_1}{s_0} = \frac{s_2}{s_1} = \frac{s_3}{s_2}$$

これを満たさない場合は, 2 重か, それ以上の誤りがあるとみてよい.

シンドロームは既知であるから, これで誤り位置 $i$ が求められる. 更に $i$ がわかれば $e_i$ が計算可能となるので, $s_0$ を使って

$$e_i = \frac{s_0}{(\alpha^0)^i} = s_0$$

この $s_0$ が誤り内容となる. 正しい符号多項式 $c(x)$ は

$$c(x) = R(x) + e(x) = R(x) + e_i$$

によって訂正できる.

### 6.2.4 2 重誤りの訂正

1 重誤りのとき同様, 誤りシンボルを $e_i, e_j (i \neq j, e_i, e_j \neq 0, e_k = 0 (k \neq i, j), 0 \leq i, j \leq n-1)$ とする. $e_i, e_j$ 以外は 0 であるから, 式 (6.4) は

$$\begin{cases} (\alpha^0)^i e_i + (\alpha^0)^j e_j = s_0 \\ (\alpha^1)^i e_i + (\alpha^1)^j e_j = s_1 \\ (\alpha^2)^i e_i + (\alpha^2)^j e_j = s_2 \\ (\alpha^3)^i e_i + (\alpha^3)^j e_j = s_3 \end{cases} \tag{6.7}$$

となる. これらをまとめて, 以下の式で扱う.

$$e_i(\alpha^h)^i + e_j(\alpha^h)^j = s_h \ (h = 0, 1, 2, 3) \quad \cdots\cdots\cdots\cdots\cdots\cdots \ (6.8)$$

方程式 4 つに対し未知数も 4 つ $(i, j, e_i, e_j)$ であるから, 解は必ず求められる. この連立方程式を解くために, 以下のような**誤り位置方程式** $\sigma(x)$ を考える.

$$\sigma(x) = (x - \alpha^i)(x - \alpha^j) = x^2 + (\alpha^i + \alpha^j)x + \alpha^i\alpha^j = 0$$

この解となる $x$ を求めることで, 誤り位置を検出することができる. ここで, $\sigma_1 = (\alpha^i + \alpha^j), \sigma_2 = \alpha^i\alpha^j$ とおくと,

$$\sigma(x) = x^2 + \sigma_1 x + \sigma_2 = 0 \quad \cdots\cdots\cdots\cdots\cdots\cdots \ (6.9)$$

となるので, まずは $\sigma_1\sigma_2$ を求める. $\sigma_2$ について解くと,

$$\sigma_2 = \alpha^i\alpha^j = (\sigma_1 + \alpha^j)\alpha^j = \sigma_1\alpha^j + \alpha^{2j}$$

同様に,

$$\sigma_2 = \sigma_1\alpha^i + \alpha^{2i}$$

よって,

$$\alpha^{2i} + \sigma_1\alpha^i + \sigma_2 = 0$$
$$\alpha^{2j} + \sigma_1\alpha^j + \sigma_2 = 0$$

この 2 つの式と式 (6.8) の積を取ると

$$e_i(\alpha^h)^i(\alpha^{2i}+\sigma_1\alpha^i+\sigma_2)+e_j(\alpha^h)^j(\alpha^{2j}+\sigma_1\alpha^j+\sigma_2)=0$$
$$e_i\alpha^{i(h+2)}+e_j\alpha^{j(h+2)}+\sigma_1(e_i\alpha^{i(h+1)}+e_j\alpha^{j(h+1)})+\sigma_2(e_i\alpha^i+e_j\alpha^j)=0$$

よって,

$$s_{h+2}+\sigma_1 s_{h+1}+\sigma_2 s_h=0\ (h=0,1) \qquad \cdots\cdots\cdots\cdots\cdots\cdots\ (6.10)$$

s の値は既知であるから, この式から σ を求めることができる. ここまでをまとめて整理する.

**Def: RS 符号の 2 重誤り訂正**

まず,

$$s_{h+2}+\sigma_1 s_{h+1}+\sigma_2 s_h=0\ (h=0,1)$$

の 2 本の式によって $\sigma_1,\sigma_2$ を求める.
次に,

$$\sigma(x)=x^2+\sigma_1 x+\sigma_2=0$$

によって根 x を 2 つ求める.

$$x=a^i,a^j$$

であるから, i と j(誤り位置) が求められることになる.
最後に,

$$e_i(\alpha^h)^i+e_j(\alpha^h)^j=s_h\ (h=0,1,2,3)$$

により, $e_i$ と $e_j$ を求める. 正しい符号多項式 c(x) は 1 重誤り同様

$$c(x)=R(x)+e(x)=R(x)+e_i+e_j$$

により求められる.

**例** GF($2^4$) 上で考える. GF(2) 上の原始多項式 f(x) を $f(x)=x^4+x+1$ とし, $f(\alpha)=0, t=2$ とする. いま, 受信系列 R(x) が以下の多項式であったとして, 誤り検出, 訂正を行う.

$$R(x)=x^{14}+\alpha^6x^8+\alpha^3x^7+\alpha^2x^6+\alpha^{12}x^5+\alpha^{13}x^4+\alpha^8x^3+\alpha^4x^2+\alpha^{10}x$$

まず, シンドローム $\boldsymbol{s}$ を求める.

$$H\boldsymbol{R}=\begin{pmatrix}(\alpha^0)^0 & \cdots & (\alpha^0)^{14}\\ (\alpha^1)^0 & \cdots & (\alpha^1)^{14}\\ (\alpha^2)^0 & \cdots & (\alpha^2)^{14}\\ (\alpha^3)^0 & \cdots & (\alpha^3)^{14}\end{pmatrix}\begin{pmatrix}0\\ \alpha^{10}\\ \alpha^4\\ \alpha^8\\ \alpha^{13}\\ \vdots\\ 1\end{pmatrix}=\begin{pmatrix}\alpha\\ \alpha^{11}\\ \alpha^{12}\\ \alpha^2\end{pmatrix}=\begin{pmatrix}s_0\\ s_1\\ s_2\\ s_3\end{pmatrix}=\boldsymbol{s}\neq 0$$

よって, R が符号語でないのが確認できる. ここで注意して欲しいのが, $\boldsymbol{R}$ の順番である. **これを間違えると全て台無しなので注意!!**

1 重誤りのチェックをする.

$$\frac{s_1}{s_0}=\frac{s_2}{s_1}=\frac{s_3}{s_2}$$

を満たさないのは明らかなので, R は 2 重誤りである. 式 (6.10) より

$$\begin{cases} s_3 + s_2\sigma_1 + s_1\sigma_2 = 0 \\ s_2 + s_1\sigma_1 + s_0\sigma_2 = 0 \end{cases}$$

これを解くと,

$$\sigma_1 = \alpha^8, \sigma_2 = \alpha^5$$

式 (6.9) より

$$\sigma(x) = x^2 + \alpha^8 x + \alpha^5 = 0$$

これを因数分解して根を求めなくてはならないが, $x \in 0, 1, \alpha, \cdots, \alpha^14$ であることを利用し, 全てを代入して根を求めても良い.

$$\sigma(0) = \alpha^5 \neq 0$$
$$\sigma(1) = \alpha^8 + \alpha^5 + 1 \neq 0 \tag{6.11}$$
$$\vdots \tag{6.12}$$

結果は,

$$x = \alpha^6, = \alpha^{14}$$

よって, 誤り位置 $i, j$ は 14, 6 となるので, $e_{14}, e_6$ を求める. 式 (6.8) の $h = 0, 1$ を用いて

$$\alpha = e_6 + e_{14}$$
$$\alpha^{11} = e_6\alpha^6 + e_{14}\alpha^{14}$$

解くと,

$$e_6 = \alpha^4, e_{14} = 1$$

よって正しい符号多項式 $c(x)$ は

$$\begin{aligned} c(x) &= e(x) + R(x) \\ &= x^{14} + \alpha^4 x^6 + x^{14} + \alpha^6 x^8 + \cdots + \alpha^{10} x \\ &= \alpha^6 x^8 + \alpha^3 x^7 + \alpha^{10} x^6 + \alpha^{12} x^5 + \alpha^{13} x^4 + \alpha^8 x^3 + \alpha^4 x^2 + \alpha^{10} x \end{aligned}$$

## 6.3 練習問題

1. GF(2) 上の原始多項式 $x^4 + x^3 + 1$ の根 $\alpha$ を用いて, GF($2^4$) 上の 1 重誤り訂正 Reed-Solomon 符号に関して以下の問に答えよ.
   (a) 符号語のビット長はいくらか.
   (b) 生成多項式 $g(x)$ を求めよ (多項式の展開はしなくても良い).
   (c) 情報語のビット長はいくらか.
   (d) 指数表示の検査行列 H を書け.
   (e) 符号の最小 Hamming 距離と最小 Hamming 重みはそれぞれいくらか.

2. GF(2) の原始多項式 $x^3 + x^2 + 1$ の根 $\alpha$ を用いて, GF($2^3$) 上の 2 重誤り訂正 Reed-Solomon 符号に関して以下の問に答えよ.
   (a) 符号語のビット長はいくらか.
   (b) 生成多項式 $g(x)$ を求めよ.
   (c) 情報語のビット長はいくらか.
   (d) 指数表示の検査行列 H を求めよ.
   (e) 符号の最小 Hamming 距離と最小 Hamming 重みはそれぞれいくらか.
   (f) 情報多項式 $m(x) = \alpha x + 1$ とすると, 対応する情報語のビット列, 非組織化符号語の多項式, 非組織化符号語のビット列, 組織化符号語の多項式, 組織化符号語のビット列を求めよ.
   (g) $R_1(x) = \alpha^3 x^3 + \alpha^3 x^2 + \alpha^6 x + \alpha, R_2(x) = 0, R_3(x) = \alpha^6 x^6 + x^5 + \alpha^3 x^4 + x^3 + \alpha^6 x^2 + \alpha^6 x + \alpha^5$ は符号語か.
   (h) 符号語でなく訂正可能な場合, 2 重以下の誤りと仮定し, 正しい符号語に訂正せよ. 正しい符号語に対応する符号多項式, 符号語のビット列を書け. また, 非組織化符号として対応する情報多項式と情報語のビット列を書け. さらに, 組織化符号として対応する情報多項式と情報語のビット列を書け.

# 第7章

# BCH符号

**BCH 符号 (BCH Code)** は, 1959 年 Alexis Hocquenghem が、それとは別に 1960 年には Raj Chandra Bose と D. K. Ray-Chaudhuri が考案した. BCH とは, この 3 人のイニシャルである.
BCH 符号も, シンドローム復号を用いて容易に復号が可能であり, そのための電子回路は非常にシンプルである. 符号としても柔軟性があり, ブロック長や誤り訂正能力を自由に設定でき, 目的に応じてカスタマイズされた符号を設計できる.

## 7.1 BCH符号の構築

さて, 前章の RS 符号は, 生成, 符号, 情報多項式すべてが拡大体 $GF(2^s)$ 上の符号表現だった. これを基礎体 $GF(2)$ に落とし込むことで, 扱う値も 2 つと単純化が可能となる. 具体的には, **生成多項式 $g(x)$ に何かを掛ける**ことで, $g(x)$ を $GF(2)$ の符号表現に変換すること[*1] で符号を構築する. そのために, いくつかの数学的知識が必要となるので, まずはそちらの解説から.

### 7.1.1 最小多項式

**Def: 最小多項式**

$g(\alpha) = 0(\alpha \in GF(p^s))$ を満たす $GF(p)$ 上の多項式中, 次数が最小のものを $GF(p)$ **上の $\alpha$ に対する最小多項式**という.

$GF(p^s)$ 上の $s$ 次原始多項式 $f(x)$ があるとする. この要素は $\{0, 1, \cdots, \omega, 2\omega, \cdots, (p-1)\omega, \cdots, (p-1)\omega^{p^s-1}\}(f(\omega) = 0, \omega$は原始元$)$ である. このとき, 以下のような生成多項式 $g(x)$ を定義すると,

$$g(x) = g_q x^q + \cdots + g_0 \qquad (g_q, \cdots, g_0 \in GF(p))$$

$g(\alpha) = 0$ を満たす多項式は無限に存在するが, その中で $q$ が最小のものが最小多項式となる.

**例** $GF(2^2)$ 上の原始多項式 $f(x) = x^2 + x + 1$ において, $\alpha$ に関する最小多項式を求める. $\alpha$ の候補は $\{0, 1, \omega, \omega^2\}$ であるので, これら全てについて求める必要がある.

[*1] 基礎体の元を係数としてもつ多項式にする, というとわかりやすいだろうか.

$$\begin{cases} \alpha = 0 \text{ のとき} & g(x) = x \\ \alpha = 1 \text{ のとき} & g(x) = x + 1 \\ \alpha = \omega \text{のとき} & g(x) = x^2 + x + 1 \\ \alpha = \omega^2 \text{のとき} & g(x) = x^2 + x + 1 \end{cases}$$

対応する $\alpha$ を各式に代入すれば, 全て 0 となることがわかる. また, $\alpha = \omega^2$ の際に関して, $h(x) = x(x^2 + x + 1)$ とすると, $h(\omega^2) = 0$ となるが, これよりも小さい次数の多項式が存在しているので, $h(x)$ は最小多項式ではないことがわかる.
おそらく定義だけではピンと来ないと思う. 具体的に BCH 符号を構築する際に用いる最小多項式を見れば, なんとなくこういうものなのか, というのはわかるはずだ.

### 7.1.2 共役根

RS 符号の構築に用いる生成多項式 $g(x) = \prod_{i=0}^{2t-1}(x - \omega^i)$ は, 一般には基礎体上の多項式とならない. 基礎体上の多項式とするためには, 生成多項式がすべての共役な根を持つようにしなければならない. 共役な根とは, たとえば $\alpha$ がある多項式 $f(x)$ の根であるとして, $\beta$ も $f(x)$ の根であるようなとき, $\beta$ は $\alpha$ の**共役根**であるという. 共役根を求めるには, 以下の定理を用いる.

**Thm: 共役根についての定理**

$\alpha$ が GF(p) 上の任意な多項式 $f(x)$ の根なら, $\alpha^p$ も $f(x)$ の根である.

$$f(\alpha) = 0 \Rightarrow f(\alpha^p) = 0$$

証明は省略.

**例** 前節の例において, $\omega$ と $\omega^2$ は, 両方とも $g(x) = x^2 + x + 1$ の根となっている. よって, $\omega$ と $\omega^2$ は共役根である.

**例** GF(2) 上の多項式 $f(x) = x^3 + x + 1 = 0$ の根の一つを $\alpha$ とすると,

$$\alpha^3 + \alpha + 1 = 0$$

である. $\alpha^2, \alpha^4$ が $\alpha$ の共役根となることを確認しよう.

$$\begin{aligned} f(\alpha^2) &= \alpha^6 + \alpha^2 + 1 \\ &= (\alpha + 1)^2 + \alpha^2 + 1 = 0 \\ f(\alpha^4) &= \alpha^{12} + \alpha^4 + 1 \\ &= (\alpha + 1)^4 + \alpha(\alpha + 1) + 1 \\ &= \alpha^4 + \alpha^2 + \alpha \\ &= \alpha(\alpha^3 + \alpha + 1) = 0 \end{aligned}$$

**例** GF(2) 上の多項式 $x^3 + x + 1 = 0$ の根の 1 つを $\alpha$ とするときの共役根と, 共役根に含まれない, $GF(2^3)$ の要素を根とする多項式を求めよう. $\alpha$ のべき乗を順次求めていくと, $\alpha, \alpha^2, \alpha^4, \alpha^8, \cdots$. ここで,

$$\alpha^8 = \alpha$$

となる[*2]から, $\alpha$ の共役根は $\alpha, \alpha^2, \alpha^4$ となる. このことから, $\alpha^7 = 1$ ということもわかる.
$\alpha$ の共役根に含まれない GF($2^3$) の要素は $\alpha^3, \alpha^5, \alpha^6$ であるから, まずは $\alpha^3$ の共役根を求める. $\alpha^3$ のべき乗を順次求めていくと, $\alpha^3, \alpha^6, \alpha^12, \alpha^24, \cdots$. ここで, $\alpha^12 = \alpha^5\alpha^7 = \alpha^5$, $\alpha^24 = \alpha^10 = \alpha^3\alpha^7$ であるから, $\alpha^3$ の共役根は $\alpha^3, \alpha^6, \alpha^5$ となる. これを根とする多項式を $f_3(x)$ とすると

$$\begin{aligned} f_3(x) &= (x-\alpha^3)(x-\alpha^6)(x-\alpha^5) \\ &= x^3 - (\alpha^6+\alpha^5+\alpha^3)x^2 + (\alpha^{11}+\alpha^9+\alpha^8)x + \alpha^{14} \\ &= x^3 + x^2 + 1 \end{aligned}$$

となり, この多項式は基礎体の元を係数として持つ多項式となる. 実はこの多項式は $\alpha^3$ に対する最小多項式になっている.

### 7.1.3 BCH 符号の構築

さて, これで BCH 符号を構築する準備は整った. 早速, GF(2) 上の t 重訂正, 原始多項式 $f(x)$ の BCH 符号を構築してみよう. まず, RS 符号の生成多項式 $g(x)$ を思い出そう.

$$g(x) = \prod_{i=r_1}^{r_1+r-1} (x-\omega^i)$$

RS 符号では一般的に $r_1 = 0$ としたが, BCH 符号では $r_1 = 1$ とするのが一般的である. よって, 生成多項式は

$$g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^r)$$

最後に, 各根 $\alpha^i$ に対して全ての共役根を求め, その共役根を根とする最小多項式の積を, BCH 符号の生成多項式とする.

**例** GF(2) 上の原始多項式 $f(x) = x^4 + x + 1$, また誤り訂正能力 $t = 2$ としたときの BCH 符号を構築してみよう. GF($2^4$) の原始元の一つを $\alpha$ とすると, 符号の長さは $n = 15$ である. まず RS 符号における $r_1 = 1$ の生成多項式は,

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)$$

であるから, $\alpha, \alpha^2, \alpha^3, \alpha^4$ の共役根を求める. $\alpha$ の共役根は $\alpha, \alpha^2, \alpha^4, \alpha^8 (\alpha^{16} = \alpha)$ となるから, $\alpha^2, \alpha^4$ の共役根は考えずともよくなる. そして $\alpha^3$ の共役根は $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24} = \alpha^9 (\alpha^{48} = \alpha^3)$ となるから, 生成多項式 $g(x)$ は

$$\begin{aligned} g(x) &= g_1(x)g_2(x) \\ g_1(x) &= (x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8) \\ &= x^4 + x + 1 \\ g_2(x) &= (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^9) \\ &= x^4 + x^3 + x^2 + x + 1 \end{aligned}$$

となる. この多項式は, 係数が 0 か 1, すなわち GF(2) 上の多項式となっているのがわかる. また, 生成多項式が 8 次なので, 情報多項式は 6 次, すなわち情報ビット長は 7 となる.

[*2] この求め方は第 4 章を参照

### 7.1.4　符号化率と検査行列

RS 符号と BCH 符号の符号化率を比べよう. 符号化率 R の式は, $\frac{\text{情報ビット長}}{\text{符号ビット長}}$ だった. BCH 符号の R は先の例でいうと,

$$R = \frac{7}{15}$$

同じものを RS 符号にしたときの符号化率は,

$$R = \frac{11}{15}$$

こうして比べると, BCH 符号は RS 符号に比べて符号化率はさほどよくないように見えるが, BCH 符号の長所はなんといっても単純であることだから, あまり気にする必要はない.

検査行列にはどのような違いがあるだろうか. BCH 符号の検査行列の根は先ほどの例では $\alpha$ および $\alpha^3$ であるから,

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{42} \end{pmatrix}$$

と, RS 符号の検査行列[*3]に比べてかなりコンパクトになっている. ここで, $\alpha$ をベクトル表示 ($1 =^t [0\ 0\ 0\ 1]$ など) にすると, 検査行列は以下のように書き直すこともできる.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**例** GF(2) 上の 5 次の原始多項式の根の 1 つを $\alpha$, また訂正能力 $t = 5$ としたときの BCH 符号の情報ビットはいくつか.

$t = 5$ なので,$\alpha^i (i = 1 \sim 10)$ までの共役根を求める必要がある. $\alpha$ の共役根は

$$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}$$

$\alpha^3$ の共役根は

$$\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48} = \alpha^{17}$$

$\alpha^5$ の共役根は

$$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40} = \alpha^9, \alpha^{18}$$

$\alpha^7$ の共役根は

$$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56} = \alpha^{25}, \alpha^{50} = \alpha^{19}$$

よって, 生成多項式は 5 次多項式 4 つの積, つまり 20 次の多項式となる. また, 符号長 $n = 31$ より, 情報多項式は 10 次であり, 情報ビットは 11 ビットとなる.

[*3] 第 6 章参照

## 7.2 BCH 符号の復号

冒頭で述べたように, BCH 符号の復号もシンドロームを用いて行うという点は RS 符号と相違ない. だが, RS 符号に比べ検査行列が小さくて済むので, 比較的単純な計算で復号が行えるというのも B CH 符号の強みである. 符号の訂正能力 $t = 2$ とすると, 検査行列 H は

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{42} \end{pmatrix}$$

であった. これは $t = 2$ であれば固定であることもわかるだろう. シンドロームを求める.

$$He = \begin{pmatrix} 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{14} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{42} \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_14 \end{pmatrix} = \begin{pmatrix} s_1 \\ s_3 \end{pmatrix}$$

多項式表示すると

$$\begin{aligned} s_1 &= \omega^0 e_0 + \omega^1 e_1 + \cdots + \omega^{14} e_{14} \\ s_3 &= (\omega^3)^0 e_0 + (\omega^3)^1 e_1 + \cdots + (\omega^3)^{14} e_{14} \end{aligned}$$

当然, $s_1, s_3 = 0$ ならば, 誤りはない. 誤り多項式 $e(x) = e_{n-1}x^{n-1} + \cdots + e_0$ の中で, ある $e_i$ のみが 1 であるとすると, シンドロームは

$$\begin{aligned} s_1 &= \omega^i e_i = \omega^i \\ s_3 &= (\omega^3)^i e_i = (\omega^3)^i \end{aligned}$$

であるから,

$$s_3 = (\omega^3)^i = (\omega^i)^3 = s_1^3$$

となる. この条件を満たすとき, 受信系列は 1 重誤りである. 逆に満たさないときは, 受信系列は 2 重誤りである.

**■1 重誤り訂正** 誤り位置は $i$ であるので,

$$e(x) = e_i x^i = x^i$$

よって正しい符号多項式 $c(x)$ は,

$$c(x) = R(x) + e(x) = R(x) + x^i$$

**■2 重誤り訂正** 誤り位置を $i, j (i \neq j)$ とする. シンドロームは

$$\begin{aligned} s_1 &= \omega^i e_i + \omega^j e_j = \omega^i + \omega^j \\ s_3 &= (\omega^3)^i e_i + (\omega^3)^j e_j = (\omega^3)^i + (\omega^3)^j \end{aligned}$$

となるので, $\omega^i$ と $\omega^j$ を求める必要がある. $(x-\omega^i)(x-\omega^j)=0$ であることを利用する.

$$\begin{aligned}(x-\omega^i)(x-\omega^j) &= x^2-(\omega^i+\omega^j)x+\omega^i\omega^j\\ &= x^2-s_1x+\omega^i\omega^j\\ \text{ここで}, s_1^3 &= (\omega^i+\omega^j)^3\\ &= (\omega^{3i}+\omega^{3j})+\omega^i\omega^j(\omega^i+\omega^j)\\ &= s_3+\omega^i\omega^j s_i\\ \omega^i\omega^j &= \frac{s_1^3-s_3}{s_1}\text{より},\\ (x-\omega^i)(x-\omega^j) &= x^2-s_1x+\frac{s_1^3-s_3}{s_1}\end{aligned}$$

これを**誤り位置方程式**とする. これにより位置を求めたら, 残りの手順は 1 重誤りと同様である.

**例** いま, 受信系列のシンドロームが

$$\boldsymbol{s}=H\boldsymbol{R}=\begin{pmatrix}\omega^4\\ \omega^5\end{pmatrix}$$

であったとする. 受信系列が GF($2^4$) 上の原始多項式 $f(x)=x^4+x+1$ で構築された BCH 符号であるとき, 誤りを訂正しよう. まず, GF($2^4$) であるから, $\omega^{15}=1$ であることがわかる. 次にシンドロームを見ると, $\boldsymbol{s}\neq 0$ であるから, R が符号語でないこともわかる.

$$\begin{aligned}s_1 &= \omega^4\\ s_1^3 &= \omega^{12}\\ s_3 &= \omega^5\\ s_1^3 &\neq s_3\end{aligned}$$

より, 2 重誤りと仮定する. 誤り位置方程式は

$$x^2-\omega^4x+\omega^{10}=0$$

となるので, これに $\omega^i(i=1\sim 14)$ を順次代入し, 根を求めると

$$x=\omega^3,\omega^7$$

が求められる. よって誤り多項式 $e(x)$ は

$$e(x)=e_7x^7+e_3x^3=x^7+x^3$$

であるから, 受信系列 R にこれを加算すれば訂正完了である.

## 7.3 練習問題

1. GF(2) 上の原始多項式 $x^4 + x^4 + 1$ を用いて構成される 2 重訂正 BCH 符号に関して, 以下の問に答えよ.
   (a) 符号のビット長はいくらか.
   (b) 生成多項式 $g(x)$ を求めよ.
   (c) 情報語のビット長はいくらか.
   (d) 指数表示とベクトル表示の検査行列 H を答えよ.
   (e) 符号の最小 Hamming 距離と最小 Hamming 重みはいくらか.
   (f) 情報多項式 $m(x) = x + 1$ とすると, 対応する情報語のビット列, 非組織化符号語の多項式, 非組織化符号語のビット列, 組織化符号語の多項式, 組織化符号語のビット列を求めよ.
   (g) $R_1(x) = x^{10} + x^8 + x^2 + x + 1, R_2(x) = x^{14} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^3 + x^2 + x, R_3(x) = x^{13} + x^{12} + x^{11} + x^8 + x^2 + x$ はそれぞれ符号多項式か.
   (h) 符号多項式ではなく, 訂正可能な場合, 2 重以下の誤りと仮定し, 正しい符号多項式に訂正せよ. 対応する符号多項式, 符号語のビット列を書け. また, 非組織化符号として対応する情報多項式と情報語のビット列を書け. さらに, 組織化符号として対応する情報多項式と情報語のビット列を書け.

2. 教科書 例 4-9 に対応するそれぞれの検査行列を答えよ.

# 第8章

# たたみ込み符号

今まで, 我々は様々な符号について学んできた. たとえば Reed-Solomon 符号や, BCH 符号等である. これらの符号は**ブロック符号**という種類の符号であり, これまでに登場した符号はすべてブロック符号であった. そして, これから学ぶのは, **ブロック符号とは全く別物の符号**だ. この符号は, いままでに学んできた符号とは根本的に様々な点が異なっている. が, この符号はブロック符号に隠れる根本的な弱点を見事に補う力を持っており, 現在の情報理論においても, とても重要な役割を果たしている. 本章では, そんな偉大なる符号について学び, さらに, その特性についても考えて行こう

## 8.1 歴史

### 8.1.1 Reed-Solomon 符号の致命的弱点

前々章で学んだ Reed-Solomon 符号は, 極めて高い性能（誤り訂正能力）を持つ符号である. 特に**バーストエラー**に対して極めて高い能力を発揮し, その高い性能ゆえに, 現在も様々な誤り訂正に利用されている. が, Reed-Solomon 符号も完全な符号というわけではなく, 次のような弱点を持っている.

**Reed-Solomon 符号は, まばらなエラーに弱い符号である.**

すなわち, Reed-Solomon 符号は, 符号語の中にまばらに発生するエラーに対して, あまり高い能力を発揮することはできない. この弱点により, Reed-Solomon 符号は**完全な符号とは言い難いもの**となっているのだ.

### 8.1.2 まばらなエラーに強い符号

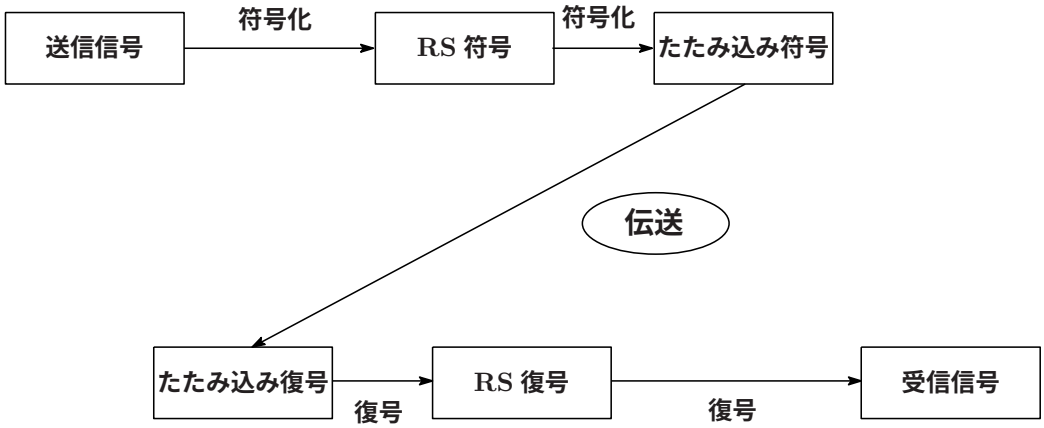
Reed-Solomon 符号の性能の高さは, 様々な符号の中でも突出したものであり, ほかのどのような符号を持ってしても, ほとんどの部分において Reed-Solomon 符号には敵わない. そんな Reed-Solomon 符号の唯一と言っても良い弱点が, **まばらなエラーに弱い**という点なのである.

そして, 世の中には, 他の部分においては Reed-Solomon 符号に敵わずとも, **まばらなエラーに強い符号**が存在する. その符号は**たたみ込み符号 (convolutional code)** と呼ばれ, 今まで考えてきた符号（ブロック符号）とは, 根本的な仕組みから異なっているという特徴を持っている.

### 8.1.3 最強のコンビネーション

Reed-Solomon 符号は**まばらなエラーに弱い**という唯一の短所を持ち, たたみ込み符号は**まばらなエラーに強い**という長所を持っている. ということは, **まばらなエラーに対してはたたみ込み符号の力を利用**し, その

他の部分に Reed-Solomon 符号を利用すれば, Reed-Solomon 符号とたたみ込み符号を組み合わせた, **限りなく完全に近い符号**が構築できるはずである.



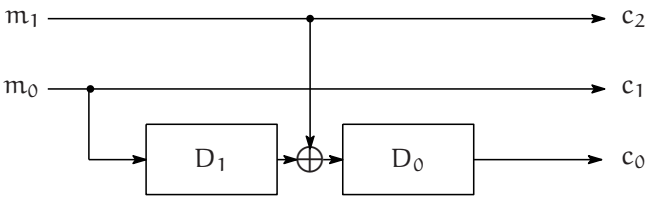

そして事実, このような符号は実際に構築され, NASA による宇宙探査計画である**パイオニア計画**において利用された. その際の誤り訂正能力は, 極めて高いものだったという.

## 8.2　たたみ込み符号

というわけで, たたみ込み符号の歴史について説明したところで, いよいよたたみ込み符号とは何かということを考えることにする. Reed-Solomon 符号の短所を補うたたみ込み符号とは, 一体どのような仕組みによって成り立っているのだろうか.

### 8.2.1　たたみ込み符号器

たたみ込み符号による符号化とは, **たたみ込み符号器 (convolutional encoder)** に情報系列を入力し, その出力 (**符号語**) を得ることである. たたみ込み符号器は, 以下のような回路により表される (あくまで一例であり, 様々なたたみ込み符号器が考えられる).




回路の中の $D_1, D_0$ という箱は, **シフトレジスタ (shift register)**, または**遅延器**と呼ばれる. シフトレジスタは一定時間値を保持するための素子であり, シフトレジスタには, 1 つの値を記憶しておくことができる. シフトレジスタの値を, シフトレジスタの**状態 (state)** という. シフトレジスタの動作を以下に示そう.

1. シフトレジスタの現状態を a とする.　　**状態** a
2. シフトレジスタに b を入力する.　　入力 b → **状態** a
3. シフトレジスタの状態は b に遷移し, a を出力する.　　**状態** b → 出力 a

この動作の様子からも, **遅延器**という名称の由来が理解できるだろう (値を少し遅延させて出力している).

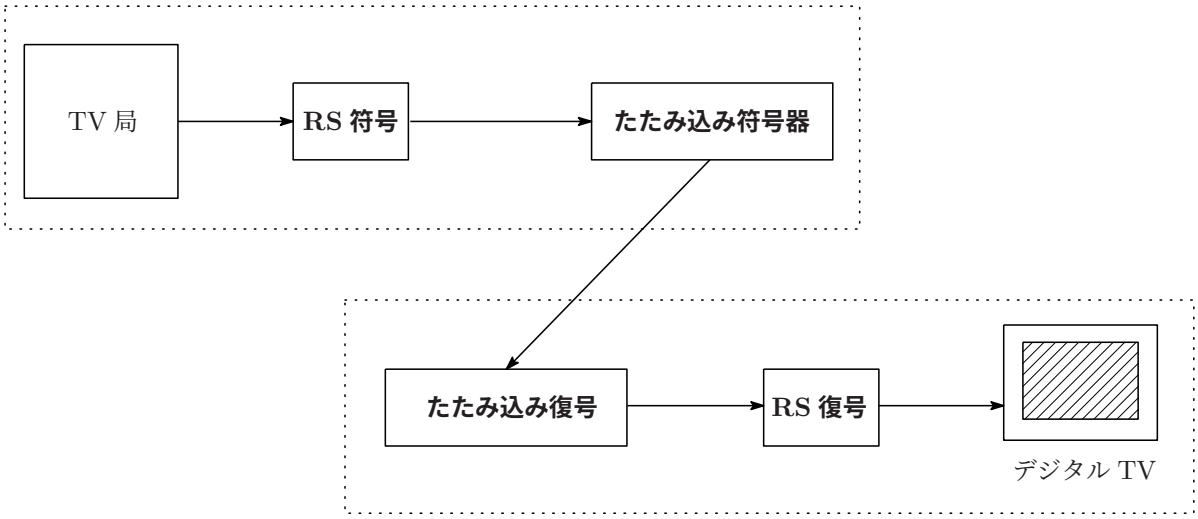
**Def：たたみ込み符号による符号化**

たたみ込み符号における**符号化 (coding)** とは, **たたみ込み符号器に情報系列を入力し, その出力の系列（符号語）を得ること**である. 符号化のことを単に**たたみ込み (convolution)** とも呼ぶ.

また, 符号器の中の $\oplus$ は, $\oplus$ に入力される値の **2 を法とする和 (排他的論理和)** を出力する素子である.

### 8.2.2 入力は, 時間と共に変化する

たたみ込み符号器への入力は, 時間と共に常に変化して行くものである. このことは, 以下のテレビ局とデジタル TV[*1]の関係を考えるとわかりやすい.

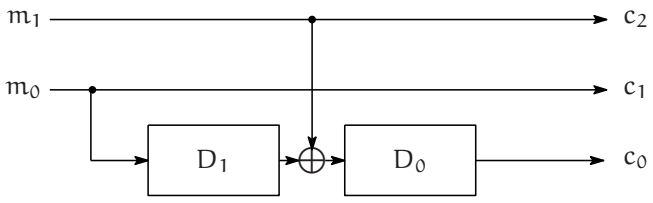



デジタル TV 放送においては, 大体このような仕組みで TV 局から視聴者に映像信号が送信されている. 本章の最初に述べたとおり, 確かに Reed-Solomon 符号とたたみ込み符号が併用されていることが分かるだろう.

さて, デジタル TV の映像は刻一刻と変化し続けることを考えると, そのたびにたたみ込み符号器の入力も, 出力も変化するということが分かる. このことはすなわち, **刻一刻と変化し続ける時間の各時点に, たたみ込み符号器の異なる入出力が対応している**ということを表している. このイメージをしっかりと持っておこう.

### 8.2.3 符号器の動きを追いかけてみよう

では, 実際に, 以下に示す符号器に様々な値を入力したときに符号器がどのように動くのかを, 落ち着いて追いかけてみることにしよう. 前述した通り, 符号器の入出力は, 時間と共に変化して行く.




シフトレジスタ $D_1, D_0$ の状態をそれぞれ $s_1, s_0$ と表すことにしよう. シフトレジスタの状態の組 $(s_1, s_0)$ を, 符号器の**状態 (state)** と呼ぶ. また, 符号器の**初期状態** (最初の状態) は $(s_1, s_0) = (0, 0)$ であるとする.

[*1] ディジタルテレビデスネ

このとき, 各時点 $t = 0, 1, 2, 3$ の入力 $(m_1, m_0)$, 符号器の状態 $(s_1, s_0)$, 出力 $(c_2, c_1, c_0)$ を表にまとめると以下のような結果となる. 各入力に対する符号器の動きをよく観察し, 以下の表が正しいことを確認しよう.

| 時点 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $m_1$ | 0 | 0 | 1 | 0 |
| $m_2$ | 1 | 0 | 1 | 0 |
| $s_1$ | 1 | 0 | 1 | 0 |
| $s_0$ | 0 | 1 | 1 | 1 |
| $c_2$ | 0 | 0 | 1 | 0 |
| $c_1$ | 1 | 0 | 1 | 0 |
| $c_0$ | 0 | 0 | 1 | 1 |

### 8.2.4　符号化率

たたみ込み符号では, 符号器の形に対応して, 符号化率が決まる. すなわち, **符号器の形が変われば, 符号化率も変化する**. これは, 今までに学んできた符号とは決定的に異なる点である.

**Def : 符号化率**

たたみ込み符号の**符号化率** $R$ は, 符号器の**入力シンボル数** $k$ と, **出力シンボル数** $n$ によって, 次のように定義される.

$$R = \frac{k}{n}.$$

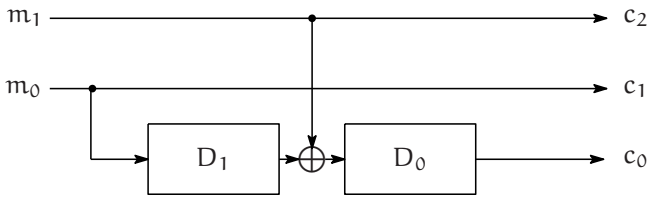
先程の符号器では, 入力シンボル数 2 に対して, 出力シンボル数は 3 である. よって, 符号器の符号化率 $R$ は,

$$R = \frac{k}{n} = \frac{2}{3}.$$

と求めることができる. 簡単ですね !

### 8.2.5　拘束長

再び, 先程の符号器について考える.




この符号器について, 次の表を完成させよう. ただし, シフトレジスタの初期状態は $(s_1, s_0) = (0, 0)$ とする.

| 時点 | 0 | 1 | 2 | 3 | $\cdots$ | k |
|---|---|---|---|---|---|---|
| $m_1$ | $a_{01}$ | $a_{11}$ | $a_{21}$ | $a_{31}$ | $\cdots$ | $a_{k,1}$ |
| $m_2$ | $a_{02}$ | $a_{12}$ | $a_{22}$ | $a_{32}$ | $\cdots$ | $a_{k,2}$ |
| $s_1$ | | | | | | |
| $s_0$ | | | | | | |
| $c_2$ | | | | | | |
| $c_1$ | | | | | | |
| $c_0$ | | | | | | |

入力が見慣れない形なので戸惑ったかもしれないが, 今まで通りの手順で表を完成させてくれれば OK である. この表を完成させると, 以下のような形となる.

| 時点 | 0 | 1 | 2 | 3 | $\cdots$ | k |
|---|---|---|---|---|---|---|
| $m_1$ | $a_{01}$ | $a_{11}$ | $a_{21}$ | $a_{31}$ | $\cdots$ | $a_{k,1}$ |
| $m_2$ | $a_{02}$ | $a_{12}$ | $a_{22}$ | $a_{32}$ | $\cdots$ | $a_{k,2}$ |
| $s_1$ | $a_{10}$ | $a_{20}$ | $a_{30}$ | $a_{40}$ | | $a_{k,0}$ |
| $s_0$ | $a_{11}$ | $a_{10} \oplus a_{21}$ | $a_{20} \oplus a_{31}$ | $a_{30} \oplus a_{41}$ | $\cdots$ | $a_{k-1,0} \oplus a_{k,1}$ |
| $c_2$ | $a_{11}$ | $a_{21}$ | $a_{31}$ | $a_{41}$ | $\cdots$ | $a_{k,1}$ |
| $c_1$ | $a_{10}$ | $a_{20}$ | $a_{30}$ | $a_{40}$ | $\cdots$ | $a_{k,0}$ |
| $c_0$ | 0 | $a_{11}$ | $a_{10} \oplus a_{21}$ | $a_{20} \oplus a_{31}$ | $\cdots$ | $a_{k-2,0} \oplus a_{k-1,14}$ |

さて, ここで, 時点 k における出力ベクトル $\boldsymbol{c}_k = (c_2, c_1, c_0)$ に注目しよう. $\boldsymbol{c}_k = (c_2, c_1, c_0) = (a_{k,1}, a_{k,0}, a_{k-2,0} \oplus a_{k-1,1})$ である. この出力をよく見てみると, 時点 $t = k, k-1, k-2$ における入力が, 出力 $\boldsymbol{c}_k$ に影響を与えていることが分かる. すなわち, 入力ベクトル $\boldsymbol{a}_k, \boldsymbol{a}_{k-1}, \boldsymbol{a}_{k-2}$ が, 時点 k での出力 $\boldsymbol{c}_k$ に影響を与えているのである.

**Def : 拘束長**

出力ベクトルに影響を与える入力ベクトルの本数を, **拘束長 (constraint length)** という.

この例では, 拘束長は 3 である. 拘束長はたたみ込み符号について考える上で非常に重要な役割を果たすので, 是非意味からしっかりと理解しておいてほしい.

拘束長は, わざわざこのように入出力を全て書き出して調べる必要はない. 実は, 拘束長に関して以下の定理が証明されているので, とても簡単な計算で拘束長を求めることができるのだ.

**Thm : 拘束長の計算**

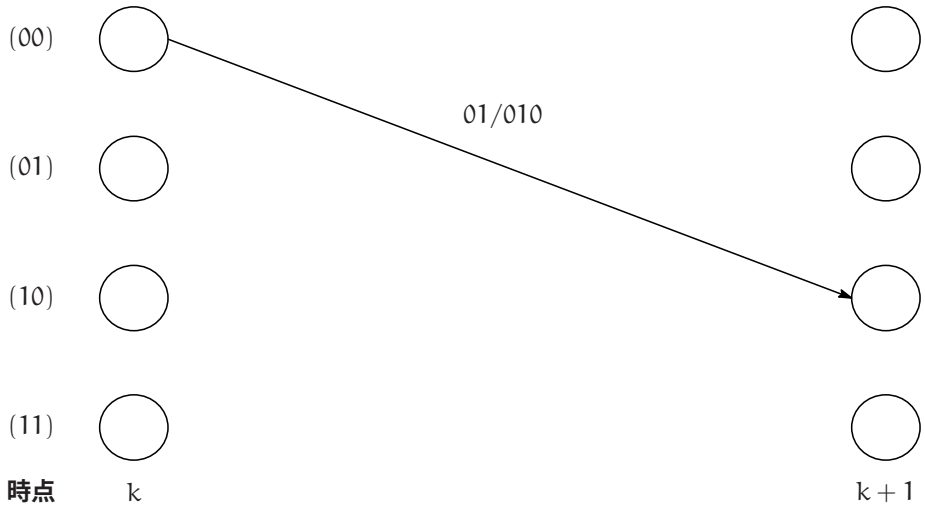
$$(\text{拘束長}) = (\text{シフトレジスタの数}) + 1.$$

この例では, 符号器に含まれるシフトレジスタが 2 つであるから, それに 1 を加えると 3 となり, 確かに拘束長と一致していることが分かる. そして, 先程の定理より, どんな符号器でもこの計算は正しい.

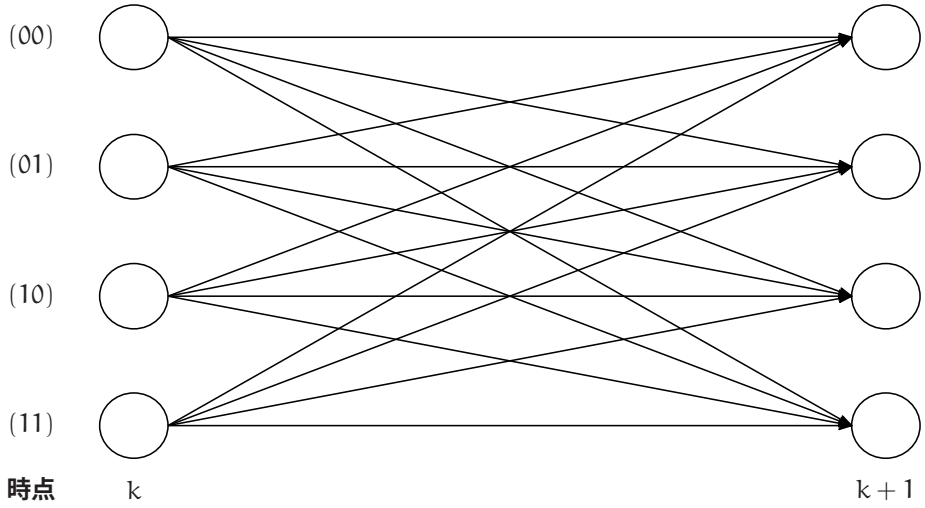
### 8.2.6 トレリス線図

たたみ込み符号器の状態の変化を視覚的にわかりやすい形で表す方法として, **トレリス線図**を用いる方法と, **状態遷移図**を用いる方法がある. まずは, トレリス線図[*2]を用いる方法を紹介しよう.

先程の符号器において, 時点 $t = k$ における状態が $(s_1, s_0) = (00)$ であるとしよう. このとき, 符号器に入力として $(m_1, m_0) = (01)$ を与えると, $(c_2, c_1, c_0) = (010)$ が出力され, 符号器の次状態 (時点 $t = k + 1$ における状態) は $(s_1, s_0) = (10)$ となる. このことは, 図を用いて次のように表現できる.




ただし, 状態の遷移を表す矢印の上の 01/010 において, 分子は**入力ベクトル** $m_1 m_0$ , 分母は**出力ベクトル** $c_2 c_1 c_0$ を表すとする. 同様の手順で, 時点 $k$ の各状態から時点 $k + 1$ の各状態への全ての遷移を図に書き込むと, 以下のような図が出来上がる (ただし, 煩雑になることを避けるため, 入出力の記述は省略した).




この図を, 符号器の**トレリス線図 (Trellis diagram)** という. トレリス線図を使うと, 符号器の状態遷移の様

[*2] トレリスは「格子状の」という意味. 人名ではないので勘違いしないように覚えておこう.

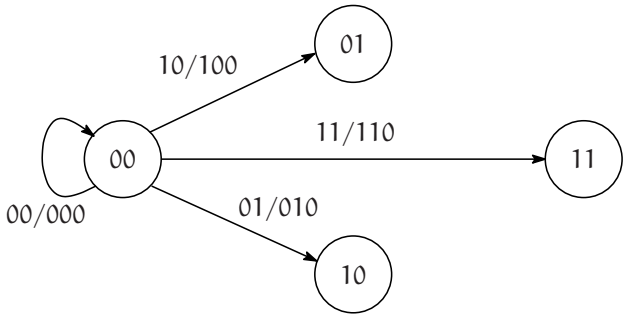
子やそのときの出力を, 視覚的に, 明確に表すことができる. また, トレリス線図は, たたみ込み符号の復号において登場する **Viterbi アルゴリズム**の説明の際, 大活躍することになる.

### 8.2.7 状態遷移図

符号器には, 各地点において状態が決まっており, 入力によって, 状態遷移が発生する (その際同時に出力も発生する). この動作の説明, どこかで聞いたことはないだろうか？

実は, 符号器は**出力付き有限オートマトン**である. 4 年次のオートマトンの授業で学んだ知識を, 頭の引き出しの奥のほうから引っ張り出してみよう. 私たちは, 出力付き有限オートマトンを知っているハズである[*3].

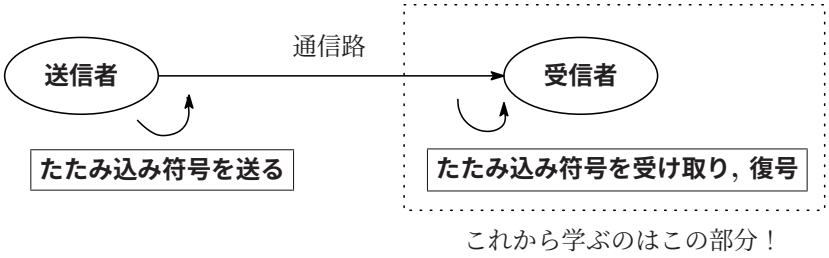
というわけで, 符号器の状態 00, 01, 10, 11 とその間の遷移関係を, オートマトンで習った状態遷移図を使って表現してみよう (煩雑になるのを避けるため, 状態 00 からの遷移のみを記した).




このように状態遷移図を用いると, コンパクトにわかりやすく状態遷移を表現することができる. トレリス線図とは異なり, 状態遷移図では**時点の区別を取り払い, 状態の変化だけを表している**. 各状態からの遷移を考えて, 状態遷移図を完成させてみよう.

## 8.3 たたみ込み符号の復号

ここまで, たたみ込み符号についての大まかな説明を行ってきた. そしてこれからは, たたみ込み符号の**復号 (decode)** について考えることにしよう.




たたみ込み符号の復号を考える上では, 2 つの条件を設定しなければならない. その条件を以下に示そう.

- 情報を伝送する通信路が 2 **元対称通信路**であること.
- 復号に**硬判定復号**を用いること.

[*3] 符号器を出力付き有限オートマトンとみなすと, このオートマトンは, 状態の遷移と同時に出力を行うオートマトンであることが分かる. このことから, 符号器は **Mealy 機械 (Mealy machine)** であると言える. 4 年次の記憶が思い出せない！という方は, 『**オートマトン NOTE（後期中間試験までの範囲分)**』の p.17 を参照すべし.

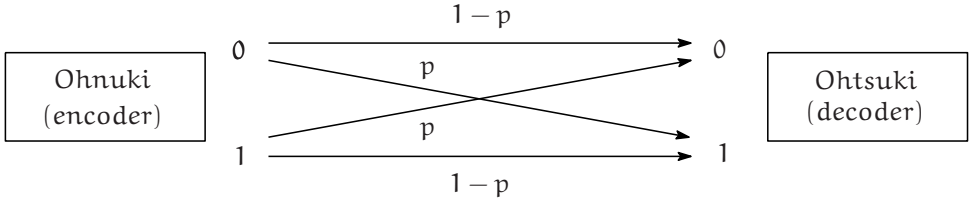
### 8.3.1　2 元対称通信路

符号を復号するということは, **情報が通信路を通って来る過程で生じた誤りを訂正し, もとの情報を復元すること**を表す. すなわち, 情報を伝送する通信路がどのような性質を持つかについても, しっかりと考えなければならない. そこで, たたみ込み符号の復号において, 通信路は **2 元対称通信路**であると仮定する.

**Def : 2 元対称通信路**

2 元対称通信路とは, **シンボル 0,1 のみを伝送し, かつ, 誤りが発生する確率が一定である通信路**を指す.

すなわち, 通信路上を伝送されるシンボルは 0 と 1 のみ. そして, 情報を伝送している過程で誤りが生じ, 0 を誤って 1 として伝送してしまう確率も, 1 を誤って 0 と伝送してしまう確率も, 同じ値 p となるような通信路を, 2 元対称通信路と呼ぶ.




2 元対称通信路は解析が最も容易であり, かつ, 応用の幅が広いため, 情報通信理論において極めてよく使われる通信路モデルである[*4].

### 8.3.2　硬判定復号

送信信号は 0,1 の離散値（デジタル[*5]値）である（たたみ込み符号器の出力は 0,1 のみだから）である. が, 情報を伝送している過程で, 送信信号には**雑音 (noise)** が侵入し, 受信者の元にたどり着く頃には, 元の離散値ではなく, 連続値 (アナログ値) を取る信号に変化してしまう.

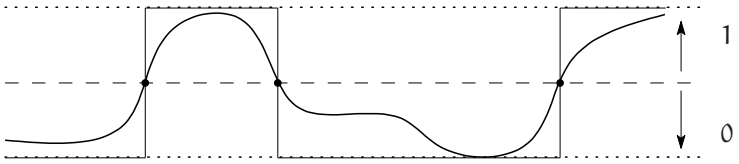

が, 送信された信号はあくまで離散値であるので, 受信者は, 受信した信号を離散値に復元しなければならない. その復元のための方法の 1 つが, **硬判定復号 (Fixed dicision decoding)** である. 硬判定復号では, ある**しきい値 (Threshold)** を決定し, そのしきい値より大きい値を取る部分の値は 1, しきい値より小さな値を取る部分の値は 0 として離散系列を得る, 最も単純な復元方法である[*6].

---

[*4] ちなみにこの 2 元対称通信路の図は Wikipedia の丸パクリです.

[*5] ディジタルテレビデスネ

[*6] 硬判定復号があるということは, 当然, **軟判定復号 (Soft dicision decoding)** も存在する. 硬判定復号よりも複雑な復元手続きが必要だが, 硬判定復号よりも精度の良い復元を行うことができる. が, 一般的には硬判定復号の精度で十分に事は足りるので, この資料で軟判定復号は扱わないこととする.




たたみ込み符号の復号においては, **硬判定復号**を採用することに決めておく.

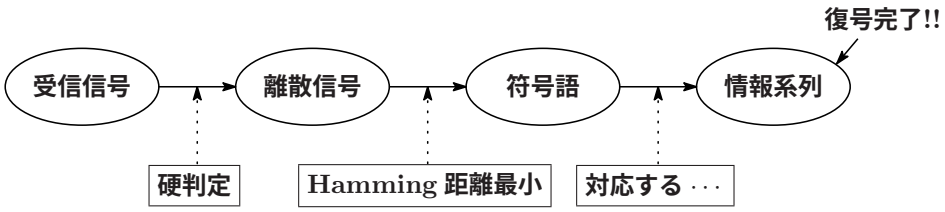
### 8.3.3 たたみ込み符号を復号するとは

さて, 必要な 2 つの条件を設定したところで, いよいよ, たたみ込み符号を**復号する**とはどういうことなのか, を詳しく定義することにする このイメージは, 是非ともしっかりつかんでおこう.

**Def : たたみ込み符号の復号**

受信者が, **雑音が侵入した連続信号**を受け取ったとき,
その信号 (たたみ込み符号) を復号するとは, 以下の手順を実行することである.

1. 受信信号（連続値）を**硬判定復号**し, 0,1 の離散信号を得る.
2. その離散系列と, **Hamming 距離が最小**の符号語を求める.
3. その符号語に対応する情報系列を求める.

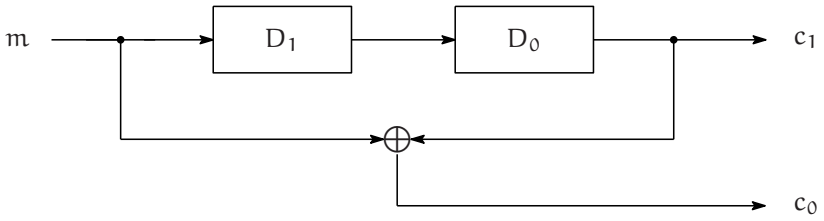



### 8.3.4 Viterbi 復号 (Viterbi アルゴリズム)

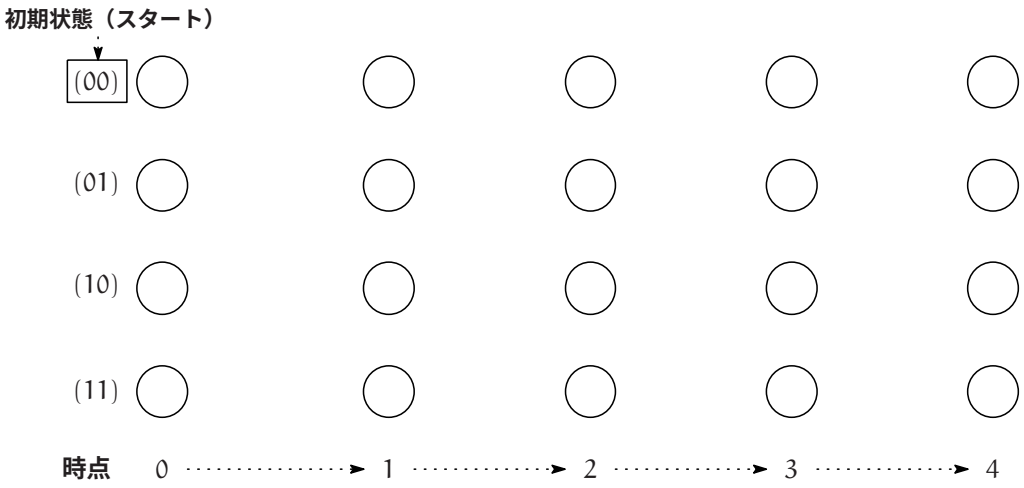
硬判定後の系列と, 全ての符号語の Hamming 距離を計算し, その中で Hamming 距離が最小となる符号語を選べば, 復号は可能である. が, この方法には膨大な計算が必要であり, 現実的な方法とは言い難い.

そこで, たたみ込み符号の復号には, 主に **Viterbi 復号 (Viterbi decoding)** という方法が使われる. Viterbi 復号とは, **Viterbi アルゴリズム**を用いてたたみ込み符号を復号することを表し, この Viterbi アルゴリズムは, **トレリス線図**を用いて説明することができる.

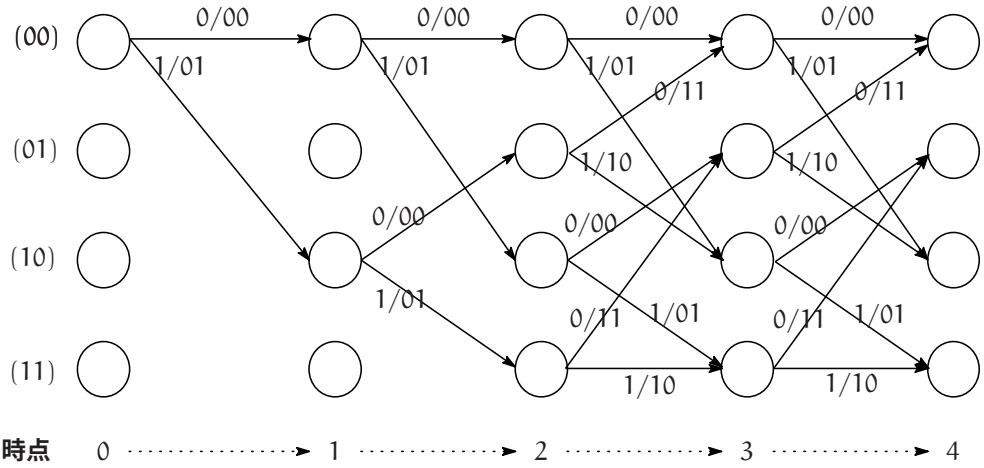
Viterbi アルゴリズムは例を通して見るとうまく理解することができる. ということで, 次のような符号器を考え, そのトレリス線図を用いて, Viterbi アルゴリズムを理解することを目標としよう.



この符号器の初期状態 $(s_1, s_0) = (0, 0)$ とする．このとき, 時点 $t = 0$ から時点 $t = 4$ までのトリレス線図を完成させてみよう．入力が 1 つだけなので, それほど複雑なトレリス線図が出来上がることはない．




初期状態 (00) がスタートとなることに注意すると, 次のようなトレリス線図を得ることができる．

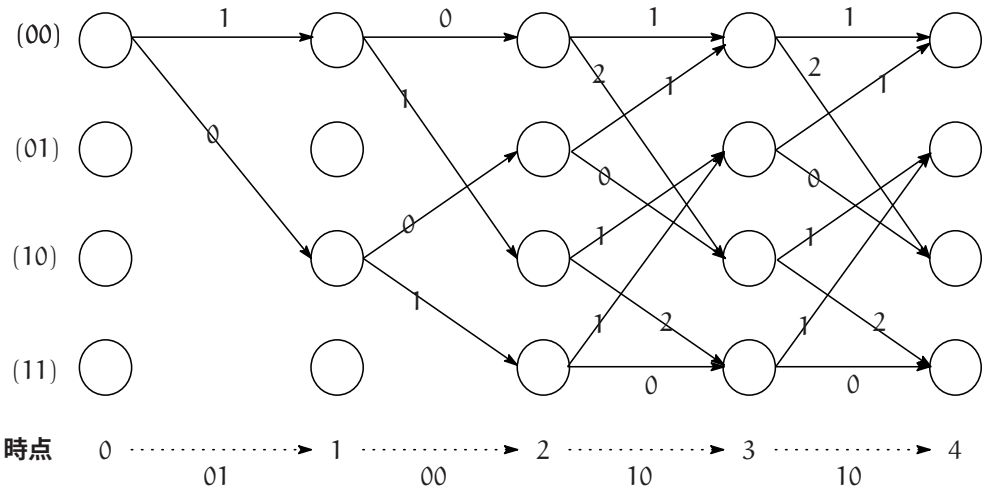



このように, 符号器に対応する, 時点 $t = 0$ から $t = 4$ までのトレリス線図を求めることができた．
さて, ここで, 受信者が受信信号を硬判定復号して得られた離散信号を 01001010 としよう．

$$\underbrace{01}\ \underbrace{00}\ \underbrace{10}\ \underbrace{10}$$

このように, 硬判定により得られた離散信号は, 2 シンボルずつ, トレリス線図の　,　,　,　の部分と対応している．このことを念頭に置いて, 硬判定後の系列を 2 ビットずつ区切った各部分と, 対応するトレリス線図の各枝 [*7]の出力の **Hamming 距離**を計算し, トレリス線図に書き込んでみよう (ただし, 入出力の記述は省略することにする)．

[*7] トレリス線図の矢印を**枝 (brunch)** と呼ぶ



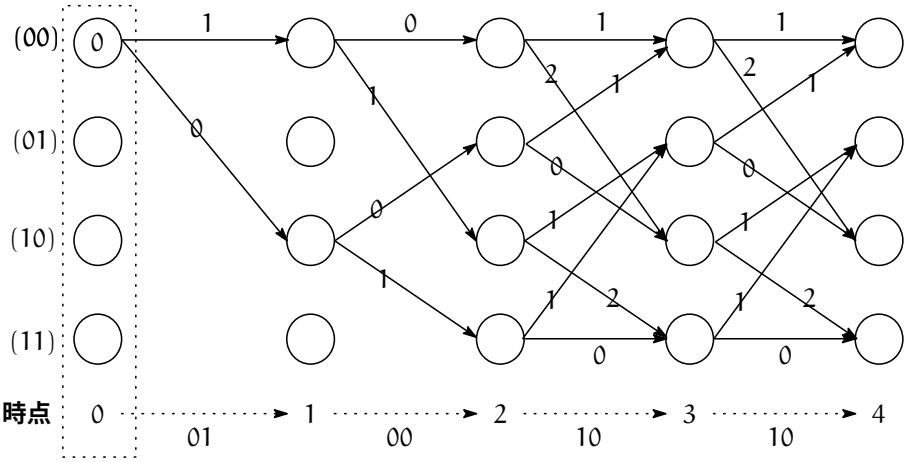

**Def：ブランチメトリック**

トレリス線図の各枝に対応する出力と, 硬判定後の離散系列の対応する各部分の Hamming 距離の値を, **ブランチメトリック (brunch metric)**, または**枝尤度 (brunch likelihood)** という．ブランチメトリックは, トレリス線図の各枝に対して定まる．

さらに今度は, 時点 0 から順番に時点 4 まで, 各状態を表す ◯ の中に, **その ◯ にたどり着くまでにかかるブランチメトリックの総和のうち最小のもの**を記入しよう．この値を, **パスメトリック (pass metric)** という．

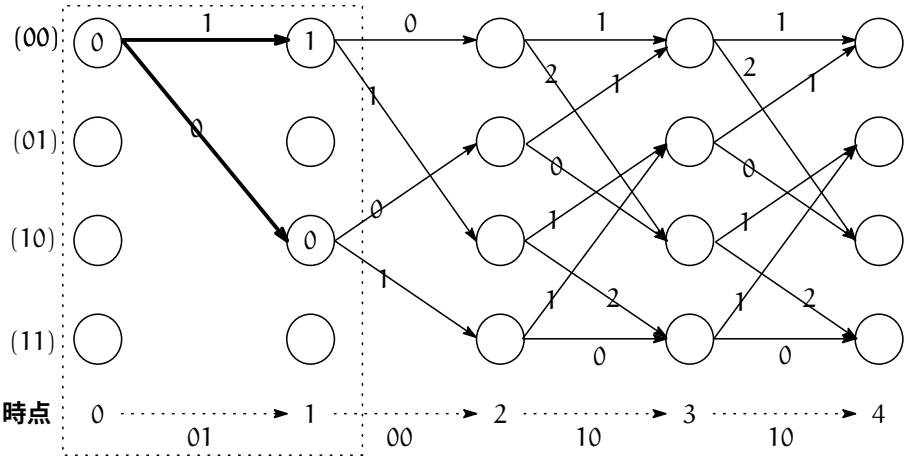
そして, その時点, その状態までのブランチメトリックの総和を最小にするような経路を太線で強調しよう, それ以外の経路は今後使うことはない．この太線で強調することができた経路を**生き残りパス (survivor pass)**[*8]という．全ての ◯ にパスメトリックを記入し, 生き残りパスが全て判明する流れを以下に示そう．

1.

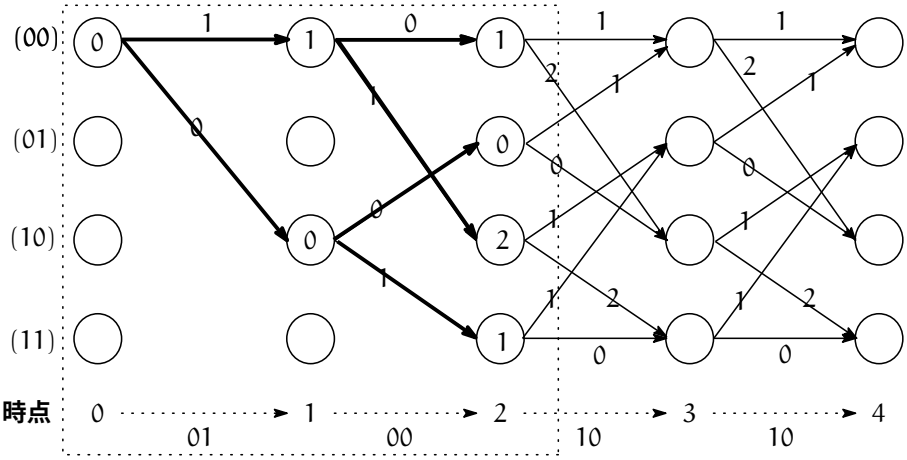



[*8] **残存経路, 生存者経路**とも呼ばれることがある．

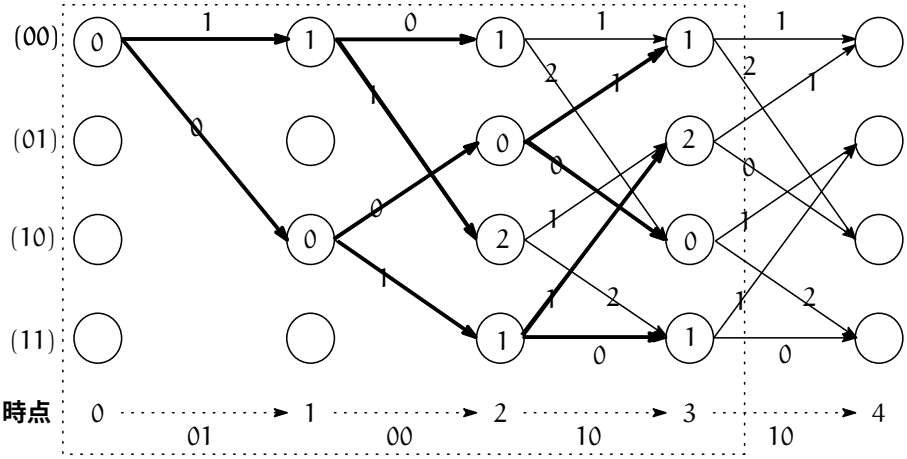
2.


(00)
(01)
(10)
(11)
時点
0
1
2
3
4
01
00
10
10

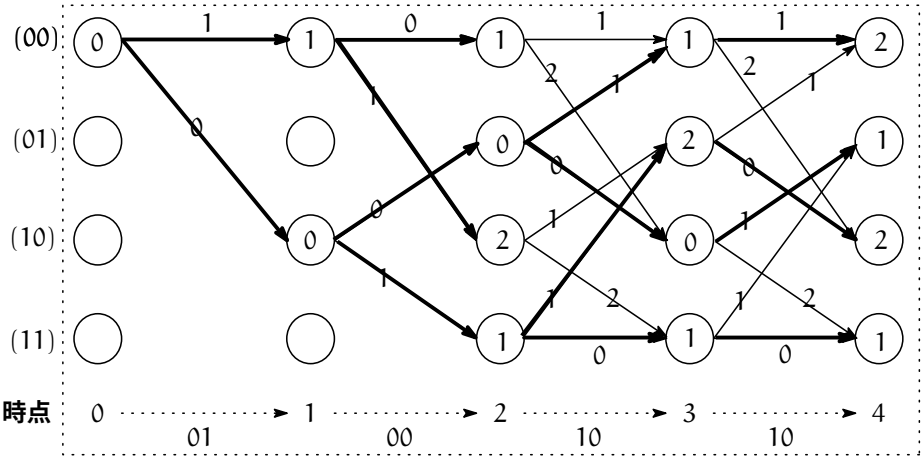
3.


(00)
(01)
(10)
(11)
時点
0
1
2
3
4
01
00
10
10

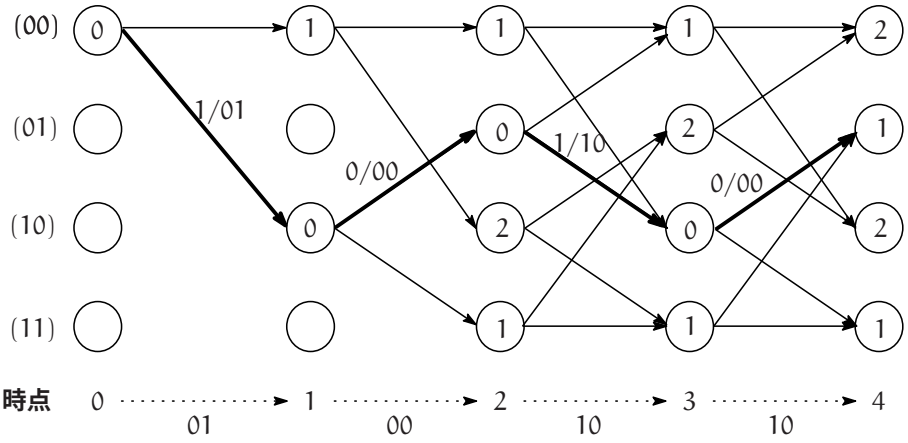
4.


(00)
(01)
(10)
(11)
時点
0
1
2
3
4
01
00
10
10

5.




このように, 全ての時点, 状態について, パスメトリックと生き残りパスが定まった. これで, **たたみ込み符号の復号のための準備は全て終了**である. あとは, 最後の時点 (t = 4) のパスメトリックのうち最小のものを選び, 生き残りパスを逆にたどれば, その経路は, 硬判定後の系列との Hamming 距離が最小の符号語を表す出力となる. 今, 時点 t = 4 を見ると, パスメトリックが最小である状態が 2 つ存在する. このような場合は, **最も上の状態を選ぶことにする**と約束しておこう. よって, t = 4 の状態 (01) から生き残りパスを逆に辿ると,




という唯一の経路が定まる. この経路に対応する出力を順番に並べると,

01001010.

という系列が得られ, この系列が**復号後の系列**である. よって, 復号後の系列を得ることができたので, これで**復号完了**だ. 長い手順, お疲れ様でした. 是非何度も見返して, Viterbi 復号の手順をマスターしておこう.

## 8.4 Viterbi 復号の特性

復号のアルゴリズムについての説明を終えたところで, ここからは復号がどのような特性を持っているのか？ということを考えて行こう. この先, 時点は無限に続いて行く（すなわち, トレリス線図が際限なく続いて行く）と仮定する.

### 8.4.1　何を知りたいのかを決めておこう

Viterbi 復号の特性を調べると言っても, いまいち, 何が知りたいのかがハッキリと分からない（抽象的すぎる）. そこで, 一体これから何を調べたいのかを, ここでハッキリと理解しておこう. この理解をキッチリとしているかどうかで, これからの理解のスムーズさが大幅に変わってくる.

**Def：これから調べること**

送信者が**全零系列**（すなわち, ゼロベクトル）を送信したとき, **Viterbi 復号によって誤りが発生（すなわち, ゼロベクトルを送ったのに 1 を含む系列に復号されてしまう）**する確率について調べる.

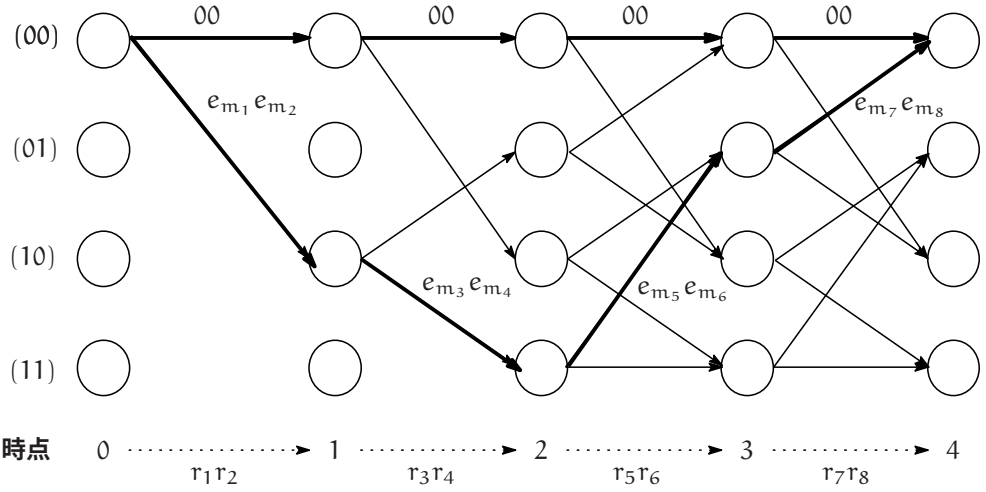
### 8.4.2　イベント誤り

Viterbi 復号の特性を考える上で欠かせない概念として, **イベント誤り (event error)** を定義しよう.

**Def：イベント誤り**

全零系列を復号した系列（復号系列）が, **全零系列から離れ, 再び合流するまで**の誤りを, **イベント誤り (event error)** と呼ぶ.

イベント誤りのイメージは, 以下のトレリス線図を見れば掴みやすいだろう.




$\boldsymbol{e}_{\mathrm{m}} = [e_{m_1}, \cdots, e_{m_8}]$ により作られる 1 つの経路が, 1 つのイベント誤りを表す. **全零系列から離れ, 再び合流している**ことがわかるだろう. イベント誤り $\boldsymbol{e}_{\mathrm{k}}$ を, **第 k のイベント誤り**と呼ぶことにする. また, $\boldsymbol{R} = [r_1, r_2, \cdots, r_8]$ は, **受信系列（硬判定復号後の系列）**である.

復号により誤りが発生するとは, $\boldsymbol{R}$ が全零系列ではない系列となってしまう（もともとの送信系列は全零なのだから, 復号された $\boldsymbol{R}$ も, 本来ならば全零にならなければならない）ことを指す. このことから, **復号の誤りが発生したかどうかを知るためには, 以下の判断を行う必要がある**ことがわかる.

受信系列 $\boldsymbol{R}$ は, $\boldsymbol{0}$ (全零系列) と見なすべきか？ $\boldsymbol{e}_{\mathrm{m}}$（あるイベント誤り）と見なすべきか？

そして, $\boldsymbol{R}$ をどう見なすかを決定するために用いる指標は, **Hamming 距離**である.

**Def : $\boldsymbol{R}$ はどう見なす？**

$\boldsymbol{R}$ を受信系列, $\boldsymbol{e}_{\mathrm{m}}$ をあるイベント誤り (第 $\mathrm{m}$ イベント誤り) であるとしたとき,

- $\mathrm{d_H}(\boldsymbol{R}, \boldsymbol{0}) \leq \mathrm{d_H}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}}) \quad \Rightarrow \quad \boldsymbol{R}$ は $\boldsymbol{0}$ であっただろうと見なす.
- $\mathrm{d_H}(\boldsymbol{R}, \boldsymbol{0}) \geq \mathrm{d_H}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}}) \quad \Rightarrow \quad \boldsymbol{R}$ は $\boldsymbol{e}_{\mathrm{m}}$であっただろうと見なす.

**例** $\boldsymbol{e}_{\mathrm{m}} = [01110100], \boldsymbol{0} = [00000000], \boldsymbol{R} = [00101001]$. このとき, $\boldsymbol{R}$ は, $\boldsymbol{0}$ であった可能性と, $\boldsymbol{e}_{\mathrm{m}}$ であった可能性, どちらが高いだろうか？

$\mathrm{d_H}(\boldsymbol{R}), \boldsymbol{0}) = 3, \mathrm{d_H}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}}) = 5.$
$\therefore \mathrm{d_H}(\boldsymbol{R}.\boldsymbol{0}) \leq \mathrm{d_H}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}})$ より, $\boldsymbol{R}$ は, $\boldsymbol{e}_{\mathrm{m}}$ であった可能性が高い.

つまるところ, Hamming 距離が短いほうが似ているんだから, そっちだった可能性のほうが高いんじゃないの. という至極当然のことを言っているだけなのだが, この条件はこれからずっと有効なので是非ここで理解しておいてほしい.

### 8.4.3 イベント誤り確率

あるイベント誤り $\boldsymbol{e}_{\mathrm{m}}$ を考えたときに, $\boldsymbol{0}$ が誤って $\boldsymbol{e}_{\mathrm{m}}$ に復号されてしまう[*9]確率が一体どの程度になるのかを考えよう. そのためにまず, **イベント誤り確率**を定義する.

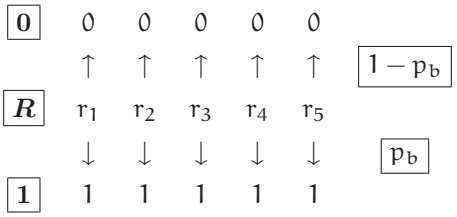
**Def : イベント誤り確率 $\mathrm{P}_{\mathrm{em}}$**

全零系列 $\boldsymbol{0}$ が, あるイベント誤り $\boldsymbol{e}_{\mathrm{m}}$ に復号される確率を $\boldsymbol{e}_{\mathrm{m}}$ の**イベント誤り確率**と呼び, $\mathrm{P}_{\mathrm{em}}$ と表す.

この $\mathrm{P}_{\mathrm{em}}$ は具体的にどうやって計算すれば求められるのだろうか. その計算式を今から導き出そう.

まず, たたみ込み符号の復号においては **2 元対称通信路**を考えていたため, 0 を 1 と復号してしまう確率と, 1 を 0 と復号してしまう確率は一定である. その確率を $\mathrm{p_b}$ と表すことにしよう.

今, 送信系列として全零系列を考えていたので, $\boldsymbol{R}$ のシンボルがもし 1 と見なされてしまったら, それは**誤りが発生した**ということになる. 誤りが発生する確率は $\mathrm{p_b}$ であるので, $\boldsymbol{R}$ の 1 つのシンボルが 1 であると見なされる確率は $\mathrm{p_b}$ である. またそれゆえに, $\boldsymbol{R}$ のひとつのシンボルが 0 と見なされる確率は $1 - \mathrm{p_b}$ である.




[*9] すなわち, 何か適当なイベント誤りを 1 つ作って, 全零系列がそれに復号されるということ.

次に, 以下の例を考えてみよう.

- $w_{\mathrm{H}}(\boldsymbol{e}_{\mathrm{m}})$[*10] $= \mathrm{d}_{\mathrm{m}} = 4$ とする.

このとき, 全零系列が誤って $\boldsymbol{e}_{\mathrm{m}}$ に復号されてしまうのはどのような場合だろうか？ちょっと考えてみよう.

$\mathrm{d}_{\mathrm{m}} = 4$ のイベント誤りの例として, $\boldsymbol{e}_{\mathrm{m}} = [0111001]$ を考える. このとき, 全零系列が誤って $\boldsymbol{e}_{\mathrm{m}}$ と見なされてしまう (すなわち, $\mathrm{d}_{\mathrm{H}}(\boldsymbol{R}, \boldsymbol{0}) \geq \mathrm{d}_{\mathrm{H}}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}})$ となる) のは, 受信系列 $\boldsymbol{R}$ のビットのうち,$\boldsymbol{e}_{\mathrm{m}}$ の 1 の部分に対応する部分のうち, **任意の 2 ビット以上が 1 となったとき**である.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{0}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{R}$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $\boldsymbol{e}_{\mathrm{m}}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{0}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{R}$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| $\boldsymbol{e}_{\mathrm{m}}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{0}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{R}$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $\boldsymbol{e}_{\mathrm{m}}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{0}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{R}$ | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| $\boldsymbol{e}_{\mathrm{m}}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{0}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\boldsymbol{R}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| $\boldsymbol{e}_{\mathrm{m}}$ | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

これらのすべての場合において, $\mathrm{d}_{\mathrm{H}}(\boldsymbol{R}, 0) \geq \mathrm{d}_{\mathrm{H}}(\boldsymbol{R}, \boldsymbol{e}_{\mathrm{m}})$ となる. よって, $\boldsymbol{R}$ は $\boldsymbol{e}_{\mathrm{m}}$ と見なされてしまうということなので, **誤りが発生**していることになる. このように, 一般に, $\boldsymbol{R}$ がイベント誤り $\boldsymbol{e}_{\mathrm{m}}$ と見なされてしまうのは, $\boldsymbol{R}$ のビットのうち $\boldsymbol{e}_{\mathrm{m}}$ と対応する場所 (ビット数 $\mathrm{d}_{\mathrm{m}}$) に $\mathrm{d}_{\mathrm{m}}/2$ 個以上の 1 が発生してしまった場合である. ただし, $\mathrm{d}_{\mathrm{m}}/2$ が自然数とならない場合は, **桁上げ**することとする[*11].

**Thm : $\boldsymbol{R}$ がイベント誤り $\boldsymbol{e}_{\mathrm{m}}$ と見なされる条件**

$\boldsymbol{R}$ のうち, $\boldsymbol{e}_{\mathrm{m}}$ の 1 に対応する部分（その部分のビット数は $\mathrm{d}_{\mathrm{m}}$）で任意の $\lceil \frac{\mathrm{d}_{\mathrm{m}}}{2} \rceil$ 個以上の 1 が発生したならば, $\boldsymbol{R}$ は $\boldsymbol{e}_{\mathrm{m}}$ と見なされる. すなわち, 誤りが発生する.
ただし, $\lceil \mathrm{x} \rceil$ は, x の**桁上げ**を表す. $\mathrm{f}(\mathrm{x}) = \lceil \mathrm{x} \rceil$ のとき, f を**天井関数 (ceiling function)** と呼ぶ.

[*10] イベント誤り $\boldsymbol{e}_{\mathrm{m}}$ の **Hamming 重み (すなわち, 1 の数)** のこと. 覚えてるかな？

[*11] たとえば, $\mathrm{d}_{\mathrm{m}} = 3$ ならば, $\mathrm{d}_{\mathrm{m}}/2 = 1.5$ となり, 整数とならない. この場合は, 桁の切り上げを行い, 2 個以上の 1 が $\boldsymbol{R}$ に発生したときに誤った復号が行われると考える.

先程の $d_m = 4$ のイベント誤りの例についてもう一度考えよう．$\boldsymbol{R}$ のある 1 ビットが 1 となる確率は $p_b$, 0 となる確率は $1-p_b$ であったので, $\boldsymbol{R}$ の中の $\boldsymbol{e}_m$ の 1 に対応する部分のうち, ある 2 $(=\lceil \frac{d_m}{2} \rceil)$ ビットが 1 となる確率は $p_b^2(1-p_b)^{4-2}$ である．そして, 4 ビットのうち 2 ビットが 1 となるパターン数は ${}_{d_m}C_2 = {}_4C_2$ 個存在するため, 4 ビット中の任意の 2 ビットが 1 となる（つまり, すべてのパターンの合計）確率は,

$$ {}_{d_m}C_2 p_b^2 (1-p_b)^{d_m-2} = {}_4C_2 p_b^2 (1-p_b)^{4-2} $$

となる．同様に, 3 ビットの 1 が生じる確率は,

$$ {}_{d_m}C_3 p_b^3 (1-p_b)^{d_m-3} = {}_4C_3 p_b^3 (1-p_b)^{4-3} $$

となり, 4 ビットの 1 が生じる確率は,

$$ {}_{d_m}C_4 p_b^4 (1-p_b)^{d_m-4} = {}_4C_4 p_b^4 (1-p_b)^{4-4} $$

として求められる．これで, $\boldsymbol{R}$ がイベント誤り $\boldsymbol{e}_m$ と見なされてしまうすべてのパターンを網羅したことになるので, このすべての確率の和をとれば $\boldsymbol{e}_m$ のイベント誤り確率 $P_{em}$ が求められるはずである．よって,

$$ P_{em} = \sum_{r=2}^{4} {}_{d_m}C_r p_b^r (1-p_b)^{d_m-r}. $$

という結果が得られる．今までは $d_m = 4$ の場合のみを考えてきたが, 一般の場合にも同様の議論によりイベント誤り確率 $P_{em}$ を求めることができる．以下にその結果を示そう．

**Thm：イベント誤り確率**

受信系列 $\boldsymbol{R}$ が, イベント誤り $\boldsymbol{e}_m$ に見なされてしまう確率,　すなわち $\boldsymbol{e}_m$ の**イベント誤り確率** $P_{em}$ は, 以下のように求められる．ただし, $d_m = w_H(\boldsymbol{e}_m)$ である．

$$ P_{em} = \sum_{r=\lceil \frac{d_m}{2} \rceil}^{d_m} {}_{d_m}C_r p_b^r (1-p_b)^{d_m-r}. $$

### 8.4.4 上界

このようにして, 上手にイベント誤り確率 $P_{em}$ を求めることができた．が, この式はなかなか複雑であり, 計算をするために結構な手間がかかってしまう．そこで, この $P_{em}$ を, **上界**を使って押さえ込んでやることを考えよう．その前に, まずは**上界**とは何かを定義しよう．

**Def：上界 (upper bound)**

ある関数 $f(x)$ が, どんな $x$ に対しても,

$$ f(x) \leq M $$

を満たすとき, $M$ を $f(x)$ の**上界 (upper bound)** と呼ぶ.
$f(x)$ が上界を持つとき, $f(x)$ は**上に有界**であるという．上界のうち最小のものを**上限**と呼ぶ.

上界の考え方はちょっとわかりにくいので, 詳しく考えておこう. 例えば, 以下の関数を考える.

$$f(x) = 2.$$

この関数の上界は, 2 **以上のすべての実数**である. なぜなら, 2 以上の実数 M をとれば, 必ず $f(x) \leq M$ が満たされるからだ. このように, 上界というものはいくつでも存在する. そして, 上界のうち最小のものは 2 であるから, $f(x)$ の上限は 2 である. このように, 上限は**唯一**だ.

上界のイメージは, 以下のような例を使うとわかりやすい. 例えば, 今あなたの目の前にとても高いビルがあるとする. そのビルが一体何 m なのかを一目見てハッキリと知ることはできない（すなわち, ビルの高さの**上限**を求めることはできない）が, 「まあ, 100m 以下ではあるだろう」と, 大雑把な値でビルの高さを評価することができる. この「100m」という値は, ビルの高さの上界である. ビルの高さは「1000m 以下だ」とも, 「10000m 以下だ」とも言えるので, 上界はいくつでも考えられる. が, 上界のうち最小のもの（すなわち, ビルの**本当の高さ**, 上限）はただ 1 つしか存在しない. このことは, 上限が唯一であることと対応する.

### 8.4.5　Bhattacharyya **バウンド**

$\boldsymbol{e}_m$ のイベント誤り確率 $P_{em}$ の上界について考える. このことはすなわち, 「まあ $P_{em}$ はこの数よりは小さいですよ」と言えるその「数」を求めることに相当する.

その数を導くには, 高度な確率統計学の知識, そして, 洞察力を必要とするので, ここでは, $P_{em}$ を上界を使って不等式で評価した結果のみを示すことにしよう. その時に用いる上界を **Bhattacharyya**[*12]**バウンド (Bhattacharyya bound)** と呼ぶ.

**Thm : Bhattacharyya バウンド**

$$P_{em} \leq \left\{2\sqrt{p_b(1-p_b)}\right\}^{d_m}$$

### 8.4.6　**ユニオンバウンド**

そして, Bhattacharyya バウンドを用いて, **Viterbi 復号によって何か (すなわち, 何でも良い) イベント誤りが生じてしまう確率** の上界を導いてみよう. その確率を $P_b$ と置くと, $P_b$ は, イベント誤り $\boldsymbol{e}_1$ または $\boldsymbol{e}_2$ または $\cdots$ または $\boldsymbol{e}_m$ または $\cdots$ が生じる確率であるから,

$$P_b = P(\boldsymbol{e}_1 \sqcup \boldsymbol{e}_2 \sqcup \cdots \sqcup \boldsymbol{e}_m \sqcup \cdots) = P\left(\bigsqcup_{k=1}^{\infty} \boldsymbol{e}_k\right)$$

$$= P(\boldsymbol{e}_1) + P(\boldsymbol{e}_2) + \cdots + P(\boldsymbol{e}_m) + \cdots = \sum_{k=1}^{\infty} P(\boldsymbol{e}_k)$$

ただし, $\sqcup$ は, 直和集合（共通部分が存在しない和集合）を表すこととする. このように, 結局, すべてのイベント誤りが発生する確率の和によって $P_b$ が求められるので, $P_b$ は以下のように変形できる.

$$P_b = \text{【Hamming重みが 1 であるようなイベント誤り発生確率の和】} + \cdots$$

$$+ \text{【Hamming重みが } m \text{ であるようなイベント誤り発生確率の和】} + \cdots$$

[*12] 「ばたちゃりや」と読む.

さらにこの式についても, 次のように変形することができる.

$$\begin{aligned} P_b &= A_1【\text{Hamming重みが } 1 \text{ であるようなイベント誤り発生確率}】+\cdots \\ &\qquad +A_m【\text{Hamming重みが } m \text{ であるようなイベント誤り発生確率}】+\cdots \\ &= A_1P_{e_1}+A_2P_{e_2}+\cdots+A_mP_{e_m}+\cdots=\sum_{k=1}^{\infty}A_kP_{ek} \end{aligned}$$

ただし, $A_k$ は, Hamming 重みが k であるようなイベント誤りの総数である. このようにして, $P_b$ を変形することができた. あとは, それぞれの $P_{ek}$ を Bhattacharyya バウンドを使って評価すると,

$$P_b \le A_1\left\{2\sqrt{p_b(1-p_b)}\right\}+A_2\left\{2\sqrt{p_b(1-p_b)}\right\}^2+\cdots=\sum_{k=1}^{\infty}A_k\left\{2\sqrt{p_b(1-p_b)}\right\}^k$$

このように, $\boldsymbol{R}$ が, 何かのイベント誤りに復号されてしまう確率 $P_b$ の上界を求めることができた. この上界を**ユニオンバウンド (union bound)** と呼ぶ.

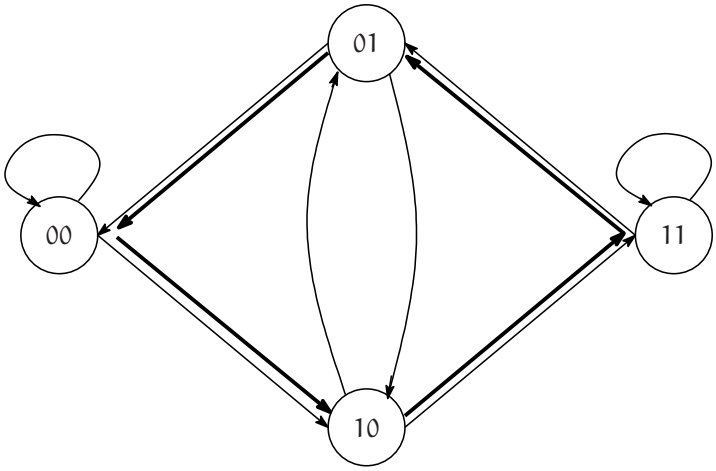
**Thm : ユニオンバウンド**

全零系列 **0** を送信したとき, 受信系列 $\boldsymbol{R}$ が何かのイベント誤りに復号されてしまう確率 $P_b$ について, 以下の評価が成り立つ.

$$P_b \le \sum_{k=1}^{\infty}A_k\left\{2\sqrt{p_b(1-p_b)}\right\}^k.$$

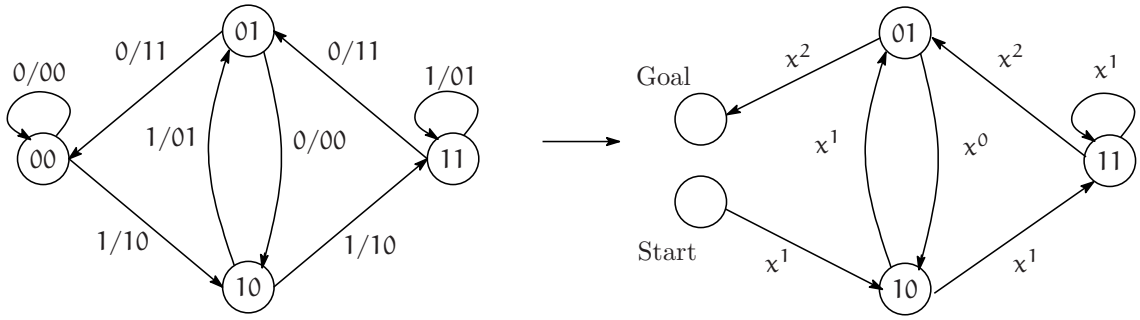
### 8.4.7 ユニオンバウンドを実際に求めるには

さて, このようにユニオンバウンドによる $P_b$ の評価を求めることができた. しかし, Hamming 重み k のイベント誤りの総数 $A_k$ は一般に求めることが難しいので, ユニオンバウンドを求めるためにも工夫が必要となる. そこで, ユニオンバウンドをうまく求める方法を, 符号器の**状態遷移図**を使って考えてみよう. そのために, 以下のような状態遷移図を考える.

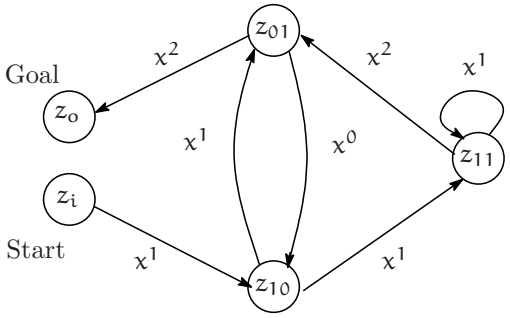



状態遷移図におけるイベント誤りは, 例えば太線の矢印で示している経路のように, 00 からスタートし, 最終的にまた 00 に戻る経路により表される. そして, そのような経路は無数に存在する.

Bhattacharyya バウンドの形をよく見ると, Hamming 重みが k であるようなイベント誤り確率の上界は, $2\sqrt{p_b(1-p_b)}$ を k 乗によって求められる. このことから, 状態遷移図を用いてユニオンバウンドを求めるアイデアを得ることができる. まず, 状態遷移図の各枝に対応する**出力の Hamming 重み**を i とし, 各枝に $x^i$ を対応付けよう. さらに, イベント誤りは状態 00 をスタートかつゴールとするので, 00 を**スタートとゴールに分割**する. 例えば, 以下のような変換となる.




そして, 遷移の際に, 各枝の $x^i$ の積をとると約束しておく. 例えば, スタート (00) から, 10, 11, 01 を経由してゴール (00) に戻るイベント誤りによる遷移を考えると, 出力地点において $x^1x^1x^2x^2 = x^6$ が得られる. $x^6$ の指数は, この定義より明らかに, イベント誤りの Hamming 重みと等しい (各枝の出力の Hamming 重みの和だから). 今, 各状態において, $z_i, z_{01}, z_{10}, z_{11}, z_o$ という値を定義する.




この値は, **スタート地点（00）から, 状態** 01, 10, 11, ゴール地点 **に至るようなすべてのイベント誤りの** $x^k$ **の総和**であると決めておこう. また, $z_i = 1$ と約束しておく. すると, 以下のような連立方程式が得られる.

$$\begin{cases} z_{10} = z_{01}x + z_i x \\ z_{11} = z_{10}x + z_{11}x \\ z_{01} = z_o x + z_{11}x^2 \\ z_o = z_{01}x^2 \\ z_i = 1 \end{cases}$$

この連立方程式をとけば, $z_i, z_{01}, z_{10}, z_{11}, z_o$ の値を求めることができる. ここで, $z_o$ の定義を思い出すと, $z_o$ は, 状態 00 から状態 00 に至るイベント誤り全てについての $x^i$ (i は各イベント誤りの Hamming重み) の総和である. そしてここで $x = 2\sqrt{p_b(1-p_b)}$ を代入すれば, $z_o$ は**全てのイベント誤りについての** $\left\{2\sqrt{p_b(1-p_b)}\right\}^i$ **の総和**, すなわち, **ユニオンバウンド**そのものとなる. このアイデアを用いると, 状態遷移図からユニオンバウンドをうまく求めることができる. この $z_o$ を**伝達関数 (Transfer Function)** といい,

$G_e(x)$ と表すので覚えておこう．この例での $G_e(x) = z_0$ は, 連立方程式より以下のように求められる．

$$G_e(x) = \frac{x^3(1-x+x^3)}{(1-x-x^3)(x-x+x^2)}$$

この関数に $x = 2\sqrt{p_b(1-p_b)}$ を代入すれば, それはユニオンバウンドそのものとなる．

### 8.4.8 情報ビット誤り率

今まで続いてきた情報論も, ついに最後の話題となった．最後の話題は, **情報ビット誤り率**についての考察である．情報ビット誤り率を次のように定義しよう．

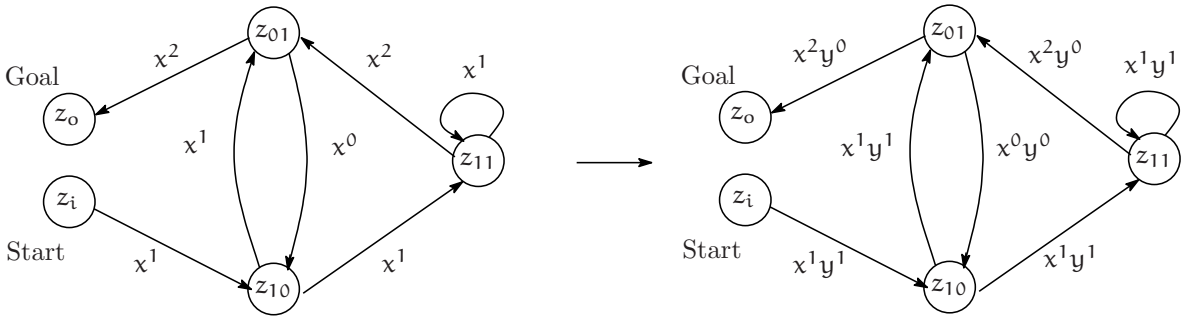
**Def : 情報ビット誤り率**

情報ビット誤り率 $P_b$ を以下のように定義する．

$$P_b = \frac{1}{k}\sum_m L_m P_{em}$$

ただし, $L_m$ は, イベント誤り $\boldsymbol{e}_m$ に含まれる情報ビット列の Hamming 重みである．
また, $k$ は各時点において入力される情報ビットの長さである．

この情報ビット誤り率は, Viterbi 復号における重要な指標だが, 情報ビット誤り率の詳しい意味についてはここでは深く考えないことにする．今から, この情報ビット誤り率を状態遷移図を使って求める方法について考える．まず, 先程の各枝に $x^i$ を割り当てた状態遷移図に, $y^j$ (ただし,$j$ は枝に対応する入力, すなわち情報ビット列の Hamming重み) を割り当てる．




先ほどと同様の規則によって, 遷移ごとに $x^iy^j$ の積を取ることにすると, あるイベント誤り $\boldsymbol{e}_m$ に対応する遷移を終えた時点で,

$$x^{d_m}y^{L_m}$$

が得られる．この結果を用いて, 何とか $L_m p_{em}$ をうまく評価できないかということを考えよう．まず, $L_m p_{em}$ においては, $L_m$ が係数として前にくっついているので, $x^{d_m}y^{L_m}$ についても $L_m$ を前に出すために, $x^{d_m}y^{L_m}$ を $y$ で**偏微分**してみよう．

$$\frac{\partial}{\partial y}x^{d_m}y^{L_m} = L_m x^{d_m}y^{L_m-1}.$$

するとどうだろう. 見事に $L_m$ を前に引きずり出すことができた. さらにここで $y=1$ を代入すると $\cdots$

$$\left.\frac{\partial}{\partial y}x^{d_m}y^{L_m}\right|_{y=1} = L_m x^{d_m} \times 1^{L_m} = L_m x^{d_m}.$$

となる. さらに, $x = 2\sqrt{p_b(1-p_b)}$ を代入すると, $x^{d_m}$ の部分は $P_{em}$ の Bhattacharyya バウンドと等しくなるから, 評価式 $P_{em} \leq \left\{2\sqrt{p_b(1-p_b)}\right\}^{d_m}$ を用いると,

$$\left.\frac{\partial}{\partial y}x^{d_m}y^{L_m}\right|_{x=2\sqrt{p_b(1-p_b)},y=1} = L_m\left\{\sqrt{p_b(1-p_b)}\right\}^{d_m} \geq L_m P_{em}.$$

という評価を行うことができる.

さらにここで, $G_e(x)$ を求めたときと同様に, $z_i, z_{01}, z_{10}, z_{11}, z_o$ に関する連立方程式を作ってみよう.

$$\begin{cases} z_{10} = z_i xy + z_{01} xy \\ z_{11} = z_{10} xy + z_{11} xy \\ z_{01} = z_{10} + z_{11} x^2 \\ z_o = z_{01} x^2 \\ z_i = 1 \end{cases}$$

この連立方程式を解くと, $z_i, z_{01}, z_{10}, z_{11}, z_o$ を求めることができ, さらに, $z_o$ は全てのイベント誤りにわたる $x^{d_m}y^{L_m}$ の総和 $\sum_m x^{d_m}y^{L_m}$ であることがわかる. この $z_o$ を $G_b(x,y)$ と書くことにする (この $G_b(x,y)$ を**伝達関数**と呼ぶ) と, この例では $G_b(x,y)$ が次のように求められる.

$$G_b(x,y) = z_o = \frac{x^3y(1-xy+x^3y)}{(1-xy-x^2y)(1-xy+x^2y)}$$

一方,

$$G_b(x,y) = \sum_m x^{d_m}y^{L_m} = x^{d_1}y^{L_1} + x^{d_2}y^{L_2} + \cdots + x^{d_m}y^{L_m} + \cdots$$

となるから,

$$\left.\frac{\partial}{\partial y}G_b(x,y)\right|_{x=2\sqrt{p_b(1-p_b)},y=1} = L_1\left\{2\sqrt{p_b(1-p_b)}\right\}^{d_1} + \cdots + L_m\left\{2\sqrt{p_b(1-p_b)}\right\}^{d_m} + \cdots \geq \sum_m L_m P_{em}.$$

これより, 情報ビット誤り率 $P_b$ の評価が以下のように得られる.

**Thm : 情報ビット誤り率 $P_b$ の上界による評価**

$$P_b \leq \frac{1}{k}\left.\frac{\partial}{\partial y}G_b(x,y)\right|_{x=2\sqrt{p_b(1-p_b)},y=1}$$

以上により, 情報ビット誤り率を上界を使ってうまく評価することができた.

数学的でテクニカルな考え方が数多く登場し, 混乱してしまったかもしれない. が, 工学的視点と数学的視点が見事に混じり合い, その結果として実りあるものが導かれるこの過程は, 大変美しく, 見事なものである. 是非この部分をしっかりと理解して, テストに万全の状態で望んで欲しい. また, 符号理論に興味が沸くことがあれば, 是非とも発展的な内容にも自ら進んでチャレンジしてみてほしい.

## 8.5 練習問題

1. テキスト p.65 の演習問題 2 を答えよ.
2. 前問において, 情報ビット列 $\boldsymbol{m} = [m(0)m(1)m(2)m(3)] = [1101]$ が符号器に入力されたとき, 対応するトレリス線図を描き, 出力される符号ビット列 $\boldsymbol{c} = [c_1c_0(0)c_1c_0(1)c_1c_0(2)c_1c_0(3)]$ を求めよ. ここで記号 $m(k)$ は符号器の時点 $k$ におおける情報入力ビット, $c_1c_0(k)$ は, 時点 $k$ における符号ビット列 $c_1c_0$ を表し, 符号器 $D_1, D_0$ の時点 $k = 0$ mの状態はそれぞれ $s_1 = 0, s_0 = 1$ とする.
3. テキスト p.65 図 6-14 のたたみ込み符号の復号に関して, パスメモリの長さ $M = 4$, 時点 $k$ の各状態のパスメトリック $PM(00,k) = 1, PM(01,k) = 0, PM(10,k) = 0, PM(11,k) = 1$, 時点 $k+1$ から時点 $k+4$ までの各時点の受信ビット列 $y_1y_0(k+1) = 10, y_1y_0(k+2) = 10, y_1y_0(k+3) = 10, y_1y_0(k+4) = 00$ とするとき, 以下の問に答えよ. ただし, $PM(s_1s_0,k)$ の状態 $s_1s_0$ は遅延器 $D_1D_2$ に, 受信ビット列 $y_1y_0$ は符号ビット列 $C_1C_0$ にそれぞれ対応するとする.
   (a) 時点 $k$ から $k+3$ に対応するブランチメトリック, 時点 $k+1$ から $k+4$ に対応するパスメトリックと生き残りパスを求め, トレリス線図上に書け.
   (b) 時点 $k+1$ に対応する符号ビット列 $C_1C_0(k+1)$ と情報ビット $m(k+1)$ を求めよ.
   (c) さらに時点 $k+5$ で受信ビット列 $y_1y_0(k+5) = 10$ を受信すると新たに得られるブランチメトリック, パスメトリック, 生き残りパスを求め, 新たなトレリス線図を描け.
   (d) 時点 $k+5$ の受信により時点 $k+2$ に対応する符号ビット列 $C_1C_0(k+2)$ と情報ビット $m(k+2)$ を求めよ.
4. テキスト p.65 図 6-14 のたたみ込み符号の復号に関して, 以下の問に答えよ. ただし, 通信路のビット誤り確率 $p_b = 0.001$ とする.
   (a) ユニオンバウンド $G_e(x)|_{x=2\sqrt{p_b(1-p_b)}}$ を求め, その値を計算せよ.
   (b) $G_b(x,y)$ を求め, $G_b(x,y)|_{y=1} = G_e(x)$ を確認せよ.
   (c) 情報ビット誤り率 $P_b$ の上界 $\frac{1}{k}\frac{\partial G_b(x,y)}{\partial y}\Big|_{x=2\sqrt{p_b(1-p_b)},y=1}$ を計算せよ.
   (d) Hamming 距離が $0,1,2,3,4,5,6,7$ であるようなイベント誤りの数はいくらか. たたみ込み符号器の最小 Hamming 距離はいくらか.

# おわりに

符号理論ノートは皆さんの勉強のお役に少しでも立つことができたでしょうか. 説明や内容がまだまだ未熟な部分もあったかとは思いますが, そのような点を著者自身が認識することができたということだけでも, このようなノートを作成した意味があったなと思っています. また, このノートを作成する過程で得た符号理論への興味を大切に, 今後も符号理論について学び続けて行こうと思います. 皆さんの中でも, このノートを通じて符号理論への認識が少しでも変わってくれたのならば, 本当に嬉しく思います.

最後に, 符号理論について深く学ぶきっかけを与えていただいた神谷先生に深く感謝申し上げます.

平成 22 年 2 月 21 日　明松 真司 , 佐々木 佳祐