2章 誤り訂正符号の例と将来展望

萩原 学†

キーワード：誤り訂正符号，ノイズ，符号化，ビット反転，誤り，消失誤り，削除誤り

1. 誤り訂正符号における誤りと符号

誤り訂正符号とは，情報の記述法の1つである．誤りと呼ばれる好ましくない現象が発生しても，元来の情報を推定できるように工夫する方法である．誤りの例として，映像の分野におけるいわゆるノイズや，音声の分野における雑音などがある．他にも，通信の分野や記憶メディアの分野でも，情報の誤りは発生する．誤りは情報機器を扱う分野における一般的な現象である．

本稿は代表的な誤りのモデルを紹介し，それに対する誤り訂正符号を紹介していく．各誤りモデルに対し，小さな誤りを訂正する符号を，抽象論ではなく具体的に挙げるようにした．例を考察することで，誤り訂正可能なことを読者が実感できるのが狙いである．

2. ビット反転誤りに対する符号

情報機器で扱う情報には映像，音声，文章などのさまざまな種類がある．種類は違えどそれらの情報は，0と1の2種類の記号で構成される系列(01系列)として記述される．

本稿では，情報を01系列で記述することを符号化と呼ぶ．

例2.1 二進表記による符号化：誤り訂正できない符号化として

整数の0から255は，8桁の二進数として表示できる．1は00000001，254は11111110として表記できる．この二進表記による符号化を，以下，本稿では f_B と記述する．この記法の使い方として

$f_B(1) = 00000001$, $f_B(254) = 11111110$
のように書く．

01系列に対する誤りとして，0が1に，もしくは，1が0に変わってしまう誤りが考えられる．そのような誤りはビット反転誤りと表現される．ビット反転誤りは符号理論(誤り訂正符号の理論)では代表的かつ基礎的な誤りのモデルとして知られている．先の f_B によって符号化された $f_B(1)$ にビット反転誤りがちょうど1つ生じれば10000001や00001001や

00000000などに変換される．

ここで次の問題を考えよう．ビット反転誤りが高々1つだけ生じた結果，系列00010000を得たとする．符号化される前の値(0から255のどれか)は幾つだろうか．

ビット反転誤りが生じる前の系列の候補としては，誤りが1つだけ生じた10010000, 01010000, 00110000, 00000000, 00011000, 00010100, 00010010, 00010001, もしくは1つも誤りが生じない00010000の9通りが考えられ，それぞれの元の値として144, 80, 48, 0, 24, 20, 18, 17, そして16という合計9つの整数が候補となる．これらの整数のどれが元の情報であるかは，判断する術がない．なぜなら符号化 f_B には，誤り訂正符号としての機能が備わっていない為である．

ビット反転誤りに対する誤り訂正符号の最も基本的なものとして，繰り返し符号と呼ばれるものがある．0もしくは1を何回か繰り返して表す符号化である．

例2.2 繰り返し符号

ここでは3回繰り返す符号化を f_{3R} と書くことにする：

$$f_{3R}(0) = 000, f_{3R}(1) = 111$$

である．

符号化 f_{3R} を施して得られた系列は高々1つまでのビット反転誤りを容易に訂正できる．それは $f_{3R}(0)$ に高々1つまでのビット反転誤りを生じて得られる系列を観察することで確かめられる．得られる系列を列挙すると，000, 001, 010, 100である．この4つの系列の何れに対しても，0の現れる個数が1の現れる個数よりも多いことを確認頂きたい．同様に $f_{3R}(1)$ に高々1つまでのビット反転誤りを生じて得られる系列も4通りあり，何れに対しても，1の現れる個数が0の現れる個数よりも多いことを確認できる．つまり，個数による多数決で，元来の情報に訂正できる．

演習問題として，繰り返し符号 f_{3R} で符号化した後に高々1つまでのビット反転誤りが生じた系列が011であるとき，元来の情報(f_{3R} を施す前の情報)は1であったと言える理由を考えて頂きたい．

さて，この繰り返し符号 f_{3R} を f_B による符号化結果に対してビット毎に用いれば，整数1は

$$f_{3R} \circ f_B(1) = 00000000000000000000111$$

に符号化される． $\circ$ は符号化の合成を表す記号である．こ

† 千葉大学

"Examples and Future Outlook of Error-Correcting Codes" by Manabu Hagiwara (Chiba University, Chiba)

のように符号化すれば、各3ビット毎に高々1つまでのビット反転誤りが生じてても先と同様に誤りを訂正できる。ここで、ビットとは0もしくは1の記号を意味する。

繰り返し符号よりも少し複雑なものとしてハミング符号と呼ばれる誤り訂正符号がある¹⁾。ハミング符号は4ビットの情報を7ビットに符号化して構成される。先の3ビット繰り返し符号 f_{3R} と同様、高々1つまでのビット反転誤りを訂正できる。利点として符号化率(もしくは伝送効率)の改善が挙げられる。繰り返し符号 f_{3R} を用いると、1ビットが3ビットに変換されるため、元来の情報は符号化された情報のうちの1/3のみとなる。逆に言えば、残りの2/3=66%は冗長な情報である。一方、ハミング符号は4ビットを7ビットに変換するため、元来の情報は符号化された情報のうちの4/7を占める。冗長な情報の割合を3/7=43%程度に抑えられる。

ここからハミング符号がどのように構成されるか述べていく。符号化前の4ビットの情報を (m_1, m_2, m_3, m_4) と表せば、符号化後の7ビットの情報は $(m_1 \oplus m_2 \oplus m_4, m_1 \oplus m_3 \oplus m_4, m_1, m_2 \oplus m_3 \oplus m_4, m_2, m_3, m_4)$ と表される。ただし m_1, m_2, m_3, m_4 は0もしくは1のどちらかを表し、 $\oplus$ は $0 \oplus 0 = 0, 1 \oplus 1 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1$ によって定義される演算を表す。普通の足し算と異なるのは $1 \oplus 1 = 0$ という性質である。これは、足し算の結果を2で割った結果、つまり偶数が奇数かだけを気にしていると考えれば良い。偶数であれば0、奇数であれば1が対応している。では、ハミング符号による符号化の例をみていこう。

例2.3 ハミング符号

符号化前の情報が0000であるとき、符号化後は上で述べた定義より

$$(0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0, 0 \oplus 0 \oplus 0, 0, 0, 0) = (0, 0, 0, 0, 0, 0, 0)$$

と計算で求められる。以下、ハミング符号の符号化写像を f_H と書くことにして、

$$f_H(0000) = 0000000$$

と表す。

符号化前の情報が0001であるとき、符号化後は上で述べた定義より

$$(0 \oplus 0 \oplus 1, 0 \oplus 0 \oplus 1, 0, 0 \oplus 0 \oplus 1, 0, 0, 1) = (1, 1, 0, 1, 0, 0, 1)$$

と計算で求められる。つまり $f_H(0001) = 1101001$ である。

符号化前の情報が1111であるとき、符号化後は上で述べた定義より

$$(1 \oplus 1 \oplus 1, 1 \oplus 1 \oplus 1, 1, 1 \oplus 1 \oplus 1, 1, 1, 1) = (1, 1, 1, 1, 1, 1, 1)$$

と計算で求められる。1 $\oplus$ 1 $\oplus$ 1=1に注意されたい。

符号化前の情報が1110であるとき、符号化後は $f_H(1110) = 0010110$ と計算で求められる。これは読者自身で確認頂きたい。

簡単のため、高々1つのビット反転誤りに耐性をもつ符号を紹介した。一般にはもっと多くの誤りに耐性をもつ符号を構成できる。リード・ソロモン符号²⁾やLDPC符号³⁾

がその例である。本特集の松島氏や笠井氏による解説記事を参照されたい。

3. 消失誤りに対する符号

消失誤りとは、元の0とも1とも判断できないシンボルに変わる誤りを意味する。例えば、誤りの発生する前に1101001と書かれていた情報が、1_01001のように換わる場合は、2番目のシンボルに消失誤りが発生したと考えて良い。同様に、1H01001のように変わった場合も、2番目のシンボルに消失誤りが発生したと考えて良い。つまり、0か1か判別不能なシンボルがどの位置にあるかわかっている状況を消失誤りと呼ぶ。消失誤りが発生する現実的な状況として、一定の間隔で情報を送っていたところ一部の情報が届かなかった場合は消失誤りとみなすことができる。届かなかったタイミングの情報を_と表記すれば、上で例示した1_01001のような消失誤りとしてとらえられる。

消失誤りを訂正する符号として、ビット反転誤りを訂正する符号を利用できる。先の繰り返し符号 f_{3R} やハミング符号 f_H を用いると高々2つまでの消失誤りを訂正できる。繰り返し符号に関して解説すると、 f_{3R} で作られる符号語*1が000もしくは111のどちらかであることを思い出せば良い。これらの符号語のどちらかを選び、高々2つまでの消失誤りを施したとする。もしも選んだ符号語が000であれば、消失誤り発生後の系列に記号0が残る。同様に、選んだ符号語が111であれば、消失誤り発生後の系列に記号1が残る。例えば消失誤りによって_0_に変わったとしても、元来の符号語が000であることは容易に推定できる。つまり、消失誤りが発生しても、元の符号語の痕跡がしっかりと残るのである。ハミング符号でも同様であることが、各符号語に高々2つの消失誤りパターンをすべて観察することで、誤りに耐性を持つことが確認できる。ただし、符号語が16個、誤りパターンが29通り、それらの組合せで464通りの考察が必要となるため、ここでは省略する。

実は、ビット反転誤りと消失誤りには密接な関係がある。高々 t 個までのビット反転誤りを訂正できる符号は、その2倍の高々 $2t$ 個までの消失誤りを訂正できる。より一般に、高々 t 個までのビット反転誤りを訂正できる符号であれば、以下の式を満たす範囲で t_b 個のビット反転誤りと t_e 個の消失誤りを組合せた t_b+t_e 個の誤りを訂正できる：

$$t_b + t_e / 2 \leq t$$

この事実は、文献⁴⁾において最少距離という概念の視点から解説されている。

多数のビット反転誤りに耐性を持つ符号が構成できることは、前節末に述べた。一方、上の不等式があるから、多数のビット反転誤りと消失誤りの両方を同時に訂正する符号が構成できることになる。

*1 符号化の結果、得られる系列全体のこと。

4. 削除誤りに対する符号

削除誤りは元来の符号語を構成するシンボルの一部が欠損する誤りである。欠損することで、系列の長さが真に短くなる。例として、系列010101010に1つの削除誤りが発生することで、短い系列01011010に変わることがある。この場合、真ん中の0が削除されている。

高々1つの削除誤りに耐性のある符号としてLevenshtein符号と呼ばれる符号がある⁵⁾。Levenshtein符号は繰返し符号と同様に符号長(各符号語を構成するビットの個数)の自由度が高い。任意の正整数 n の符号長をもつLevenshtein符号が構成できるという特徴をもつ。ここでは簡単に、符号長5のLevenshtein符号を紹介する。

例4.1 Levenshtein符号

符号語を5つのビット(整数としての0か1)からなる系列 $x_1x_2x_3x_4x_5$ と表す時

$$x_1+2x_2+3x_3+4x_4+5x_5=0 \pmod{6}$$

という条件をみたすものが符号語として扱われる。ここで $=0 \pmod{6}$ とは、左辺が6の倍数であることを意味している。また、+は整数の一般的な足し算を表す。この条件を満たす系列全体を書き下すと

00000, 10001, 01010, 11011, 11100, 00111

であり、全部で6つある。

これらの各符号語に対し、1削除誤りが発生したらどのような系列が得られるか考察する。符号語00000であれば、どのシンボルが削除されても系列0000を得る。次に符号語10001であれば、先頭のシンボルが削除されると0001、2～4番目のシンボルが削除されると1001、5番目のシンボルが削除されると1000である。6つの符号語に対して削除した結果をまとめると以下を得る。

- ・符号語00000：0000
- ・符号語10001：0001, 1001, 1000
- ・符号語01010：1010, 0010, 0110, 0100, 0101
- ・符号語11011：1011, 1111, 1101
- ・符号語11100：1100, 1110
- ・符号語00111：0111, 0011

長さが4となり短くなった系列を眺めて、どこにも重複がないことを確認頂きたい。重複がないことから、元の長さ5の符号語へと一意に戻すことが可能となる。なぜなら、もしも誤り発生後に1101となっていれば、上の結果から、符号語は11011であったことが一意に推定できるからである。

他の長さのLevenshtein符号を作る際には、上の条件を

$$x_1+2x_2+\cdots+(n-1)x_{n-1}+nx_n=0 \pmod{n+1}$$

に変えればよい。このようにしても、1つの削除誤りを訂正できる⁶⁾。

もっと多くの削除誤りを訂正する符号としてHelberg符号が知られている。詳細は文献⁷⁾を参照されると良い。Helberg符号はLevenshtein符号と同様に符号が構成され

る。2つまでの削除誤りを訂正する符号を構成する場合は、上の条件の x_i の係数をフィボナッチ数列^{*2}の第 $i+2$ 項から1引いた値にする。もしも長さが6の符号で2つまでの削除誤りを訂正できる符号であれば

$$x_1+2x_2+4x_3+7x_4+12x_5+20x_6=0 \pmod{33}$$

を満たす系列 $x_1x_2x_3x_4x_5x_6$ 全体とすればよい。

5. 挿入誤りに対する符号

削除誤りの双対的概念として挿入誤りがある。挿入誤りが発生することで、系列の長さが真に長くなる。例として、系列010101010に1つの挿入誤りが発生することで、長い系列0101001010に変わることがある。この場合、真ん中の0が挿入されている。ちなみに、真ん中には2つの0があるが、どちらが挿入されたかは判別できない。

実は、挿入誤り訂正符号と削除誤り訂正符号には密接な関係がある。高々 t 個までの削除誤りを訂正できる符号は、おなじく高々 t 個までの挿入誤りを訂正できる。より一般に、高々 t 個までの削除誤りを訂正できる符号であれば、以下の式を満たす範囲で t_d 個の削除誤りと t_i 個の挿入誤りを組合せた t_d+t_i 個の誤りを訂正できる：

$$t_d+t_i \leq t.$$

この事実は、文献⁵⁾にて解説されている。とくに、現在ではLevenshtein距離⁷⁾とよばれる概念の視点で理解されている。

前節で1削除誤り訂正符号として例示した長さ5のLevenshtein符号が1挿入誤り訂正符号であることを確かめる。そこで、これらの各符号語に対し、1挿入誤りが発生したらどのような系列が得られるか考察する。符号語00000であれば、どの位置にシンボル0が挿入されても系列000000を得る。また、シンボル1が挿入されれば100000, 010000, 001000, 000100, 000010, 000001という系列を得る。6つの符号語に対して挿入した結果をまとめると以下を得る。

- ・符号語00000, 000000, 100000, 010000, 001000, 000100, 000010, 000001
- ・符号語10001：010001, 100001, 100010, 110001, 101001, 100101, 100011
- ・符号語01010：001010, 010010, 010100, 101010, 011010, 010110, 010101
- ・符号語11011：011011, 101011, 110011, 110101, 110110, 111011, 110111
- ・符号語11100：011100, 101100, 110100, 111000, 111100, 111010, 111001
- ・符号語00111：000111, 001011, 001101, 001110, 100111, 010111, 001111

長さが6となり長くなった系列を眺めて、どこにも重複がないことを確認頂きたい。重複がないことから、元の長さ5

*2 フィボナッチ数列 a_i とは $a_1=1, a_2=1, a_3=2, a_4=3, a_5=5, \dots, a_n=a_{n-1}+a_{n-2}$ と表される数列である。

の符号語へと一意に戻すことが可能となる。なぜなら、もしも誤り発生後に110100となっていれば、上の結果から、符号語は11100であったことが一意に推定できるからである。

挿入誤りと削除誤りを組合せると、ビット反転誤りを発生できることを注意しておく。例として、系列000の真ん中を削除し00としたあと、間に1を挿入すれば010を得る。これは、真ん中のビットにビット反転を施したものと区別がつかない。そこで、先ほどと同様に以下の結論を得る。

高々 t 個までの削除誤りを訂正できる符号であれば、以下の式を満たす範囲で t_d 個の削除誤りと t_i 個の挿入誤りと t_b 個のビット反転誤りを組合せた $t_d + t_i + t_b$ 個の誤りを訂正できる：

$$t_d + t_i + 2t_b \leq t.$$

6. 応用分野からの刺激：展望

本稿では、ビット反転誤り、消失誤り、削除誤り、挿入誤りという誤りを解説した。これらの誤りは、通信や記憶媒体で現実的に起こる雑音を数学的にモデル化したものである。また、それらに対して訂正能力をもつ誤り訂正符号をそれぞれ紹介した。誤りのモデルが与えられた時、誤り訂正符号の研究者らは新たな符号の理論を産み出してきた。紙面の都合上ここでは紹介できなかったが、最近ではフラッシュメモリならではの誤りモデル⁸⁾が提案され、置換符号⁹⁾と呼ばれる対象の研究が活発に行われている。今後、情報技術の発展にともない、新たな誤りモデルが提案され続けると予想される。同時に、その誤りを解決する誤り訂正符号も構成され続けていくであろうと著者は考えている。

映像メディアの分野でも、雑音等が実装技術の問題になっていけば、符号理論の研究者に声掛けして頂きたい。

誤り訂正符号の創出に繋がるかも知れない。本稿をきっかけに、分野を超えた研究交流が生まれれば冥利につきる。符号理論のテキストとして文献^{4) 10)}などを挙げておく。

(2016年3月29日受付)

〔文 献〕

- 1) R.W.Hamming: "Error detecting and error correcting codes", Bell Syst. Tech. J., 29, pp.147-160 (1950)
- 2) I.S.Reed and G.Solomon: "Polynomial codes over certain finite fields", SIAM J. Appl. Math., 8, 2, pp.300-304 (1960)
- 3) D.J.C. MacKay: "Good error-correcting codes based on very sparse matrices", IEEE Transactions on Information Theory, 45, 2, pp.399-431 (Mar. 1999)
- 4) 今井秀樹：「符号理論」，信学会（1990）
- 5) V. Levenshtein: "Binary codes capable of correcting deletions, insertions and reversals", Dokl. Akad. Nauk SSSR 163, pp.845-848 (1965) [in Russian] ; English Translation Soviet Phys.-Dokl. 10, pp.707-710 (1966)
- 6) M. Hagiwara: "A short proof for the multideletion error correction property of Helberg codes", IEICE Communication Express, 5, 2, pp.49-51 (2016), DOI: <http://doi.org/10.1587/comex.2015XBL0182>
- 7) A.S.J. Helberg and H.C. Ferreira: "On multiple insertion/deletion correcting codes", IEEE Trans. Inf. Theory, 48, 1, pp.305-308 (Jan. 2002), DOI:10.1109/18.971760
- 8) A. Jiang, M. Schwartz and J. Bruck: "Error-correcting codes for rank modulation", Proc. of IEEE International Symposium on Information Theory (ISIT), IEEE, Toronto, Canada, pp.1736-1740 (2008)
- 9) D. Slepian: "Permutation Modulation", Proc. of IEEE, IEEE, pp.228-236 (1965)
- 10) 萩原学：「符号理論」，日本評論社（2012）

萩原 学 はぎわら まなぶ 千葉大学理学部数学科卒業。東京大学大学院理学研究科修了。東京大学生産技術研究所（ポスドク）、産業技術総合研究所（主任研究員）を経て、現在、千葉大学准教授。符号理論と組合せ論の研究に従事。趣味は怪談話と海外ドラマ鑑賞。博士（数理科学）。