Risa/Asirと誤り訂正符号理論

鈴木祥介
Shousuke Suzuki
東北文化学園大学
Tohoku Bunka Gakuen University*

1 概要

1.1 動機

誤り訂正符号理論（以下符号理論と略す）の数学的に基本的な部分を記述、理解するために具体的な例題、演習問題があればとの希望からプログラムの作成を以前から試みてきた．よりどころとなるテキストとしては [1] を、言語としては Risa/Asir を使用した．さし当たりの目標としては数式処理言語 GAP(Groups, Algorithms and Programming) で記述された、GUAVA[8] という coding theory package の機能の実現であったが現在まで一部分しかできていないがこれまでの結果を報告することにした．GAP は整数論、群論を中心とする数式処理システムであり、GUAVA にもその成果が取り入れられている．他の数式処理システムで記述された符号理論のライブラリも存在すると思われるが手元にはない．代数的符号理論は、代数幾何符号とそれ以前の代数符号とに分けられる．代数幾何符号もグレブナー基底との関連で早く実装したい．GUAVA には代数幾何符号の機能はない．Risa/Asir は文法が C 言語に非常に近いので他の数式処理システムで記述するよりも覚えることが少なくてよいと感じている．それにグレブナー基底演算が高性能なので、代数幾何符号についても Risa/Asir で記述できればと思っている．

1.2 現状

符号理論の数学的な基礎は有限体上の線形代数の理論であり、汎用のC言語では自分で有限体のライブラリを作成しなければならず負担が大きい．[1] を取り上げたのは、$GF(2), GF(2^r)$ 上の符号、すなわち 2 元符号に限定しているので Risa/Asir でプログラムが作成しやすいことと大学の教科書用なので最先端のことを扱っているわけではないが、演習問題が豊富でプログラムの検証に役に立つ．今回は、以前に発表してきたものを見直して、Reed-Muller 符号, BCH 符号, Reed-Solomon 符号 (RS 符号) などの機能を加えた．[1] の後半の 30% は暗号理論であるが、暗号については触れない．残念ながら、時間の都合で [1] の符号に関する部分の およそ 70%についてのみに終わっている．

2 符号理論

符号理論に関しては、日本語、英語を含めて良書が多数出版されているが、圧巻なのは [6] で、I,II 巻を合わせると 2200 ページになる．符号理論の実際の応用はハードウェアにも依存するのでこの方面の勉強も

*ssuzuki@ait.tbgu.ac.jp

欠かせないのであるが、筆者には荷が重いので触れない．さて、符号理論の大きな研究成果は、1950 年の Hamming 符号から始まり、およそ 10 年ごとに必ず現れてきている．1959 年から 1961 年にかけては BCH 符号と RS 符号、1968 年には復号法として Berlekamp-Massey 法 (BM 法) が現れた．1981 年には代数幾何符号が、1989 年には代数幾何符号の復号法が現れた．果たして、2000 年代初頭には代数幾何符号のもっと良い復号法が現れるのであろうか．

3 プログラムの内容

3.1 有限体および線形代数

単機能なものを多数作成し、それらを組み合わせることにより目的の関数が得られるという方針にした．集合に関する演算として、和集合、共通部分、補集合、直積など．集合はベクトルとして表現した．線形代数に関しては行の基本変形、Gauss-Jordan 法による標準形など．多項式に関しては、原始多項式、最小多項式を求めるなど．文法を完全に理解しているわけではないので思い違いなどがしばしば生じる．例えば、行列演算を行う場合は、コピーを作ってから操作しないともとの行列が変わってしまうなど．

3.2 符号理論

巡回符号、双対符号、Hamming 符号、Hamming 重み、重み母関数、MacWilliams 恒等式、最小距離、Reed-Muller 符号および 1st クラスの場合の復号、BCH 符号および 2 個の誤りの場合の復号、Reed-Solomon 符号およびその復号などであり、まだ未完成である．変数名、関数名の命名法に苦労した．見ただけで内容が判断できるようにしたかったが、関数の引数や戻り値は List と Vector のどちらがいいのかなどあいまいなままになっている．現時点で同機能の関数で別名のものも入れるとおよそ 250 個の関数になった．

3.3 関数名、変数名、引数名

vector = V or Vect
list = L or List
matrix = M or Mat
polynomial = P or Poly
to = 2 (例えば、coef2Poly は係数のリストから多項式を作成)
code = C or Code
primitive = Prim
大文字、小文字を適宜使用．

4 例

以下簡単な実例をあげていくことにする．Risa/Asir に標準で含まれていない関数は、すべて作成されたものである．

4.1 拡大 Golay 符号の復号 (Exercises 3.6.6(a), [1] page 82)

Algorithm 3.6.1 (decoding the extended Golay code , [1] p.80) を用いて拡大 Golay 符号の復号化を行う．長さが 24 なので、最初の 12 個と後半の 12 個から得られるシンドロームをそれぞれ以下の通りとする．

```
Sda1=newvect(12,[0,1,0,0,1,0,0,0,0,0,0,0])$
Sda2=newvect(12,[0,1,1,1,1,1,0,1,0,0,0,0])$
```

としたとき、これから復号するには

```
sdegc24(Sda1,Sda2);
[ 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
```

また、拡大 Golay 符号の重さ分布から、各符号語の重さはすべて 4 の倍数になることが分かる．

```
[960] ExtGol24=extendGolay()$ <== 拡大 Golay 符号の生成行列
[961] size(ExtGol24);
[12,24] <== このくらいのサイズだと重さ分布は計算できる
[962] weightDistribution(ExtGol24);
[ 1 0 0 0 0 0 0 0 759 0 0 0 2576 0 0 0 759 0 0 0 0 0 0 0 1 ]<== 2576 は重さ 12 の符号語の個数
```

4.2 Reed–Muller 符号、Algorithm 3.9.4 (decoding the $RM(1,m)$ code, [1] page 89)

r 次の長さ 2^m の Reed–Muller 符号を $RM(r,m)$ で表す．1 次の Reed–Muller 符号 $RM(1,5)$ は、マリーナー 9 号で 1972 年 2 月 19 日に用いられた．生成行列は

```
GRM15=genRM(1,5);
[ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ]
[ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ]
[ 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 ]
[ 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 ]
[ 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ]
```

最小距離は、`minDistanceG(GRM15);` より `[16]` となる．
`T=[(最小距離 - 1)/2] =7` 個以下の誤りは訂正できる．重み分布は

```
 weightDistribution(GRM15);
[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 62 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ]
```

簡単な場合の復号化の例を挙げる．$2^6 = 64$ 個の符号語の中から、符号語 CWV として生成行列の 2 行目をとる．

```
GRM15[1] = [ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ]
```

CWV=GRM15[1] この符号語の最初の 7 桁が全て 0 になったものが受信されたとすると、誤りベクトル ErrV は

```
ErrV=newvect(32,vtol(unitVect(7)));
[ 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
```
となる．受信語は、`RWV=CWV+ErrV;` より
```
[ 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ]
```

この受信語 RWV を復号する．$RM(1,m)$ に対して非常に効率のよい、Fast Hadamard Transformation を用いる．このときに行列の Kronecker product が必要になる．まず送信された Message (MsV) を復号する．

```
MsV=algor394(5,RWV);
[ 0 1 0 0 0 0 ]
```
これを送信語に変換すると
```
SWV=Ms1*GRM15;
[ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ]
CWV-SWV;
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
```
となり復号が正常に行われたことが分かる．

4.3 BCH 符号 (Example 5.5.7,[1] page 125)

巡回符号として 2-誤り訂正 BCH 符号を考える．Algorithm 5.5.4 (decoding 2 error-correcting BCH codes, [1] page 124) を実装した．原始多項式を、 PrimPoly $= 1 + x + x^4$ とし、生成される体 $GF(2^4)$ で考える．

`RWL=[1,1,0,1,1,1,1,0,1,0,1,1,0,0,0]$` を受信語とし、
これを `RWV=newvect(15,RWL)$` としてベクトルに変換し、
`dec2ErrBCHR(RWV,PrimPoly);` より送信ベクトルが以下のように推定される．
```
[ (1) (1) 0 (1) (1) (1) (1) (1) 0 0 (1) (1) 0 0 0 ]
```

受信語を与える代わりにシンドロームを与えても同様の結果が得られることはいうまでもない (dec2ErrBCHS を用いる)．1 つだけプログラム (dec2ErrBCHR = Algorithm 5.5.4 ,[1] page 124) を記述することにする．ただし、コメントは省略してある．

```
/*****************************
   dec2ErrBCHR(RecW,PrimPoly)  [2002.12.08]
   RecW=受信語、 PrimPoly=原始多項式
   return:corrected word
==============================*/
def dec2ErrBCHR(RecW,PrimPoly)
{
  CLen=2^deg(PrimPoly,x)-1;
  setmod_ff(PrimPoly);  B=@;
  if(type(RecW)==4){
    RecW=newvect(length(RecW),RecW);
  }
  ChM=getBinCheckMatBCH13(PrimPoly);
  SyndV0=RecW*ChM;
  SyndV=map(simp_ff,SyndV0);
```

```
  S1=vectProj(SyndV,0); S3=vectProj(SyndV,1);
  Sz=size(S1)[0];
  ZV=newvect(Sz);
  if( (S1==ZV) && (S3==ZV) ){
    print("No errors ");
    return RecW;
  }  else {
    if( (S1==ZV) && (S3!=ZV)){
      print("Retransmission ");
      return;
    }  else {
     S1x=coef2Poly(S1);  S3x=coef2Poly(S3);
     S1B=subst(S1x,x,B); S3B=subst(S3x,x,B);
     if( S1B^3==S3B){
       ErrV1=newvect(2*Sz,vtol(S1));
       RecCW=ErrV1+RecW;
       return map(simp_ff,RecCW);
     }  else {
        S1x=coef2Poly(S1); S3x=coef2Poly(S3);
        S1B=subst(S1x,x,B); S3B=subst(S3x,x,B);
        Alpha=S1B;
        Beta=S3B/S1B +S1B^2;
        for(I=0;I<CLen;I++){
          for(J=0;J<CLen;J++){
           if((B^I+B^J == Alpha) && (B^I*B^J ==Beta) ){
              ErrL=[I,J];
              ErrP=exp2Poly(ErrL);
              ErrV=poly2Code(ErrP,CLen);
              RecCW=ErrV+RecW;
              return map(simp_ff,RecCW);
           }
          }
        }
       }
      }
    }
  }
```

4.4 Reed-Solomon 符号 (Example 6.3.4, [1] page 139)

ここでは、$RS(2^4, 7)$ 符号とその復号について．Algorithm 6.3.2(decoding $RS(2^r, \delta)$, [1] page 137) を用いる．生成多項式は、$g(x) = (1 + x)(\beta + x)(\beta^2 + x)(\beta^3 + x)(\beta^4 + x)(\beta^5 + x), \beta$ は原始元

```
[1142] GF24P=1+x+x^4$ <=== GF(2^4) を定義する原始多項式
[1143] setmod_ff(GF24P)$ <== 体を GF(2^4) に設定
[1144] B=@$ <== B を原始元とする
[1145] RcWCoefL=[1,B^4,0,B,0,B^9,1]$ <== 受信語;
[1,(@+1),0,(@),0,(@^3+@),1]
[1146] WP634=coef2Poly(RcWCoefL); <== 受信語の各成分を係数とする多項式
x^6+(@^3+@)*x^5+(@)*x^3+(@+1)*x+1
[1147] D=7$ <=== 設計距離
[1148] decRSCode(WP634,D,-1,GF24P); <== 訂正された符号語,-1 は g(x) の定義に必要
[ 1 b^4 b^2 b b^12 b^9 1 0 0 0 0 0 0 0 0 ] <== b は B=@ の代わり
```

5 今後の課題

Risa/Asir で符号理論の数学的な基礎的な部分をプログラムしてみた．符号の生成行列 $GenMat$ からすべての符号語を作成する場合、行列のサイズが少し大きくなると不可能になる．$GenMat$ の行のサイズが R のとき、2^R 個の一次結合を作る必要があるからである。2 元符号でも over flow してしまいがちになるので、p 元符号の場合はサイズが小さいものに制限されてしまう．重み分布や最小距離を求める場合には別の方法 (理論的な方法) を考えなければならないであろう．符号の最小距離というのは非常に重要な値なので残念である．実際に利用されている符号を取りあげることも必要であろう．Risa/Asir のグラフィクスの機能を利用して視覚に訴えるプログラムを作成することもこれからの課題である．また、将来の問題として代数幾何符号を実装することができれば有用になるであろう．

参 考 文 献

[1] D. R.Hankerson, D.G.Hoffman,et al., Coding Theory and Cryptography:
The Essentials,Second Edition,Revised and Expanded
Marcel Dekker,Inc. (2000)

[2] 内田興二, 有限体と符号理論
サイエンス社 SGC ライブラリ 5 (2000)

[3] 江藤良純、金子敏信監修 , 誤り訂正符号とその応用
オーム社 (1996)

[4] 今井秀樹 , 符号理論
電子情報通信学会 (1990)

[5] F.J.MacWilliams, N.J.A.Sloane, The Theory of Error-Correcting Codes
North-Holland (1977)

[6] Vera S.Pless, W.C.Huffman (eds.) , Handbook of Coding Theory Volume I,II
Elsevier (1998)

[7] Vera Pless, Introduction to the Theory of Error-Correcting Codes (Third Edition)
John Willey & Sons, Ins.(1998)

[8] R.Baart, J.Cramwinckel,and E.Roijackers, GUAVA,a coding theory package
Delft Univ. Technology (1994)