Lund University, Information Theory

Third hand in problem

Sergio Gutierrez,

May 2019

Exercise

In the course material, error correcting encoding and decoding is described through a Hamming code. There are many other codes defined over the years. The one treated here is called Reed-Muller code.

For a positive integer m and a non-negative integer r, the generator matrix for the Reed- Muller code RM(r,m) is defined as:

$$G(r,m) = \begin{pmatrix} G(r,m-1) & G(r,m-1) \\ 0 & G(r-1,m-1) \end{pmatrix} \tag{1}$$

To start the induction define G(0,m) as a row vector of 2^4 ones, i.e. the generator matrix for a length 2m repetition code, and G(m,m) the unit matrix of size 2^m. In this problem the RM(1, 3) code will be treated.

- **Find the generator matrix.**
- **list all code-word and determine d_{min}.How many errors can be corrected? How many errors can be detected?**
- **Find the parity matrix.**
- **Write the syndrome table for all single errors that can occur on the channel.**
- **Send a message with an error and correct it.**

Solution

I present a straight forward solution of the Reed-Muller coding scheme done in python running off jupyters platform. As can bee seen in the appended code we first start be defining module two addition. With this we can construct the matrix and array multiplication taking into account the internal products of this space.

From these basic definitions we can go ahead an implement the Reed-Muller code by using Eq.1 and the following the initial conditions:

$$G(m,m) = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}_{2^m \times 2^m} \qquad G(0,m) = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}_{2^m} \tag{2}$$

this was all done in the beginning of the code. Form this one can continue and find the generation matrix for the case RM(1,3) which gives the matrix:

$$RM(1,3) = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{3}$$

With this we can find all the code words by making all the linear combination of the rows in the generator matrix, we do this by multiplying all the elements of the space of 1×4 with binary entries. The set of Code Words that I found are shown next:

```
Code Words
--------------------------
00000000
00001111
00110011
00111100
01010101
01011010
01100110
01101001
10101010
10100101
10011001
10010110
11111111
11110000
11001100
11000011
```

With this code words we can find the d_min that for our case is the minimum of the Hamming weight since we have a linear code. Which gives us $d_min = 4$, letting us detect three errors and we could correct 1 error this is computed with whats known from the class.

The parity matrix of our generator matrix is the transpose of it self since we can compute the product and determine that

$$G \cdot G^T = 0$$

taking into account that this must be computed using the base two inner product. The parity matrix allows us to find the syndrome coding of this system and implement an error correcting scheme. The errors that we can correct and its syndrome were computed and shown in the following segment.

```
Error      Syndrom
---------------------------
00000000       0000
10000000       1000
01000000       0100
00100000       1010
00010000       0110
00001000       1001
00000100       0101
00000010       1011
00000001       0111
```

Correcting an error is now straight forward, one receives some code word CW, next we multiply it with the parity matrix giving of a syndrome. With the syndrome we can detect if there were any errors and correct the received code word:

$$CW = 01110110 \rightarrow CW \cdot G^T = 0110 \rightarrow E = 00010000 \tag{4}$$

which means our real error corrected code word is 01100110 which is what was originaly sent.

Reed-muller code and Error correction

May 24, 2019

1 Reed-Muller Code

```
In [1]: %pylab inline
        %matplotlib inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.1 Internal Product Definitions

```
In [2]: def mod2(x,y):
            return (x+y)%2
        def vecmatri(x,y):
            a=zeros(len(y[0]))
            for k in range(len(y[0])):
                for i in range(len(x)):
                    a[k]=mod2(a[k],x[i]*y[i][k])
            return a
        def matripro(x,y):
            a=zeros((len(x),len(y[0])))
            for i in range(len(x)):
                for j in range(len(y[0])):
                    for k in range(len(x[0])):
                        a[i][j]=mod2(a[i][j],x[i][k]*y[k][j])
            return a
        def numzeros(x):
            n=0
            for i in range(len(x)):
                if(x[i]==1):
                    n+=1
            return n
```

1.2 Reed-Muller Functions

```
In [3]: def G0m(m):
            return ones(2**m)
        def Gmm(m):
```

```
            l=2**m
            return identity(l)
```

```
In [15]: def RM(r,m):
             if(r==0):
                 return G0m(m)
             if(r==m):
                 return Gmm(m)
             else:
                 up=c_[RM(r,m-1),RM(r,m-1)]
                 down=concatenate((zeros(2**(m-1)),RM(r-1,m-1)),axis=None)
                 return concatenate((up,[down]),axis=0)
```

1.3 RM(1,3) Solution

1.3.1 Generator matrix

```
In [16]: g=RM(1,3)
         print(g)
```

```
[[ 1.  0.  1.  0.  1.  0.  1.  0.]
 [ 0.  1.  0.  1.  0.  1.  0.  1.]
 [ 0.  0.  1.  1.  0.  0.  1.  1.]
 [ 0.  0.  0.  0.  1.  1.  1.  1.]]
```

```
In [6]: vec=array([[0,0,0,0],[0,0,0,1],[0,0,1,0],[0,0,1,1],
                   [0,1,0,0],[0,1,0,1],[0,1,1,0],[0,1,1,1],
                   [1,0,0,0],[1,0,0,1],[1,0,1,0],[1,0,1,1],
                   [1,1,0,0],[1,1,0,1],[1,1,1,0],[1,1,1,1]])
```

1.3.2 CodeWords

```
In [7]: List=[]
        wH=[]
        for i in vec:
            List.append(vecmatri(i,g))
            wH.append(numzeros(vecmatri(i,g)))
```

```
In [8]: dmin=min(wH[1:])
        detec=dmin-1
        correct=(dmin-1)/2
        print('Posible errors detected: %.1f, Posible errors corrected: %.1f'%(dete
```

```
Posible errors detected: 3.0, Posible errors corrected: 1.5
```

1.3.3 Parity check

```
In [9]: print(matripro(g,g.T))
```

```
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

1.3.4 Syndrome decoder

```
In [10]: def conv(X,Y):
             for i in range(len(X)):
                 Y.append("".join(str(int(x)) for x in X[i]))
         def pr(X,Y,name1,name2,namef):
             f=open("%s"%namef,"+w")
             f.write("%s      %s\n"%(name1,name2))
             f.write("---------------------------\n")
             for i in range(len(X)):
                 f.write("%s      %s\n"%(X[i],Y[i]))
             f.close()
         def wrt(arr,name,namef):
             f= open("%s"%namef,"+w")
             f.write("%s      \n"%name)
             f.write("---------------------------\n")
             for  i in range(len(arr)):
                 f.write("%s\n" %arr[i])
             f.close()
```

```
In [11]: e=[[0,0,0,0,0,0,0,0],[1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],
            [0,0,1,0,0,0,0,0],[0,0,0,1,0,0,0,0],[0,0,0,0,1,0,0,0],
            [0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,1]]
```

```
In [12]: s=[]
         for i in e:
             s.append(vecmatri(i,g.T))
```

```
In [13]: CodeWords=[]
         Syn=[]
         Er=[]
         conv(List,CodeWords)
         conv(s,Syn)
         conv(e,Er)
         pr(Er,Syn,'Error','Syndrom','Error.txt')
         wrt(CodeWords,'Code Words','Code.txt')
```

```
In [14]: y=List[6]+e[4]
         sol=vecmatri(y,g.T)
         print('Input                                    Syndrome                    Co
         print(y ,sol,y-e[4])
```

```
Input                                 Syndrome                    Corrected
[ 0.  1.  1.  1.  0.  1.  1.  0.] [ 0.  1.  1.  0.] [ 0.  1.  1.  0.  0.  1.  1.  0
```