# Assignment: Error-Control Codes, Due Wednesday, 11:59pm

*Chris Betsill*

*November 16, 2016*

## 1. New ISBN numbers

In 2007, a new ISBN-13 system was introduced. The check digit $a_{13}$ is chosen so that

$$a_1 + 3a_2 + a_3 + 3a_4 + a_5 + 3a_6 + a_7 + 3a_8 + a_9 + 3a_{10} + a_{11} + 3a_{12} + a_{13} = 0$$

in $\mathbb{Z}_{10}$.

a. Show (by finding an example), that it is possible to transpose two adjacent digits without changing the check digit.

```
getISBNCheck <- function(ISBN){
  digits <- strsplit(ISBN, "")[[1]]
  if(length(digits) != 12)
    return("Length must be twelve to compute check digit")
  x <- 0
  for(i in 1:length(digits)){
    if(i %% 2 == 1)
      c <- 1
    else
      c <- 3
    x <- x + c*as.numeric(digits[i])
  for(i in 0:10){
    if(((x+i)%% 10) == 0)
      return(i)
  }
  }
}
getISBNCheck("982133810906")
```

```
[1] 1
```

```
getISBNCheck("982138310906")
```

```
[1] 1
```

b. Show (by example), that it is possible to change a single digit without changing the check digit.

```
num <- "982133810906"
check <- getISBNCheck(num)
findMatch <- function(){
   digits <- strsplit(num, "")[[1]]
   for(i in 1:length(digits)){
     for(j in 0:9){
       digits[i] <- j
       newISBN <- paste(digits, collapse = "")
       if(getISBNCheck(newISBN) == check && newISBN != num)
         return(newISBN)
     }
     digits <- strsplit(num, "")[[1]]
   }
}
findMatch()
```

```
[1] "902133810906"
```

```
getISBNCheck("982133810906")
```

```
[1] 1
```

```
getISBNCheck("902133810906")
```

```
[1] 1
```

## 2. Code rate

Compute the code rate for the Two-Dimensional Parity Check code discussed in the notes. Explain your computation.

The code rate is the number of bits of information communicated divided by the number of bits in a code word. The two-dimensional parity check from the notes, there are 16 data bits sent, along with 8 check bits for a total of 25 bits.

$$\frac{16}{25} = .8$$

## 3. Hamming $[7, 4]$ code words

The following example illustrates how matrices are declared and multiplied in R. You can just use regular numeric variables (not `raw` or `bigz`), and the modulo operator `%%` works on matrices.

```
## Matrix examples
rowVec <- c(1,2,3)
M <- matrix(c(1,1,0,2,3,4, 0,1,2,1,1,4, 3,0,0,1,3,1), nrow=3, byrow=TRUE)
rowVec %*% M
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   10    3    4    7   14   15
```

```
as.vector(rowVec %*% M)
```

```
[1] 10  3  4  7 14 15
```

Compute all of the Hamming $[7, 4]$ code words using matrix multiplication in $R$. Represent the matrix $G$ on page 395 as a matrix, then get the code words by multiplying by the 16 different row vectors.

```
rowVec1 <- c(0,0,0,0)
rowVec2 <- c(1,0,0,0)
rowVec3 <- c(0,1,0,0)
rowVec4 <- c(0,0,1,0)
rowVec5 <- c(0,0,0,1)
rowVec6 <- c(1,1,0,0)
rowVec7 <- c(1,0,1,0)
rowVec8 <- c(1,0,0,1)
rowVec9 <- c(0,0,1,1)
rowVec10 <- c(0,1,1,0)
rowVec11 <- c(1,1,1,0)
rowVec12 <- c(1,1,0,1)
rowVec13 <- c(1,0,1,1)
rowVec14 <- c(0,1,1,1)
rowVec15 <- c(0,1,0,1)
rowVec16 <- c(1,1,1,1)
G <- matrix(c(1,0,0,0,1,1,0, 0,1,0,0,1,0,1, 0,0,1,0,0,1,1, 0,0,0,1,1,1,1), nrow=4, byrow=TRUE)
G
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    0    0    0    1    1    0
[2,]    0    1    0    0    1    0    1
[3,]    0    0    1    0    0    1    1
[4,]    0    0    0    1    1    1    1
```

```
as.vector((rowVec1 %*% G) %% 2 )
```

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

```
as.vector((rowVec2 %*% G) %% 2 )
```

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

```
as.vector((rowVec3 %*% G) %% 2 )
```

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

```
as.vector((rowVec4 %*% G) %% 2 )
```

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

```
as.vector((rowVec5 %*% G) %% 2 )
```

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

```r
as.vector((rowVec6 %*% G) %% 2 )
```

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

```r
as.vector((rowVec7 %*% G) %% 2 )
```

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

```r
as.vector((rowVec8 %*% G) %% 2 )
```

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

```r
as.vector((rowVec9 %*% G) %% 2 )
```

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

```r
as.vector((rowVec10 %*% G) %% 2 )
```

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

```r
as.vector((rowVec11 %*% G) %% 2 )
```

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

```r
as.vector((rowVec12 %*% G) %% 2 )
```

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

```r
as.vector((rowVec13 %*% G) %% 2 )
```

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

```r
as.vector((rowVec14 %*% G) %% 2 )
```

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

```r
as.vector((rowVec15 %*% G) %% 2 )
```

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

```r
as.vector((rowVec16 %*% G) %% 2 )
```

```
[1] 1 1 1 1 1 1 1
```