Hill Ciphers

Olivia Delffs

This paper explores the use of linear algebra, specifically matrices, in encoding and decoding secret messages to enhance security. Traditional substitution ciphers, like those commonly featured in puzzle books, involve replacing each letter of the alphabet with another letter. While they are easy to implement, they can be susceptible to decryption due to recognizable patterns of common phrases. To mitigate the predictability of substitution ciphers, a numerical representation is introduced. Assigning numerical values to letters (A=0, B=1, etc.) transforms messages into matrices. For example, the phrase "abc" becomes [0 1 2], adding a layer of complexity to the encryption process. Utilizing modular arithmetic becomes essential to the encryption process. This operation, denoted by the symbol (%), ensures that all calculations stay within the range of the alphabet's length, 26. For instance, the result of 27 % 26 is 1, and 25 % 26 is 25. This method of encryption was created by mathematician Lester Hill and is referred to as a Hill Cipher.

For message encoding, we will use 3x3 key matrices. The key matrix must be in the set Z_3^{26}, which is spanned by all vectors with 3 elements in the range of 0 through 25. Multiplying a vector in this set yields another vector, also in Z_3^{26}, serving as an encoded version. The effectiveness of this encoding method relies on having a way to decode the encoded messages, which can be done if A is invertible. However, to keep the values of the encoded message within 0 to 25, the inverse of A must be calculated using modular arithmetic.

When choosing a key matrix, besides the range of the values within it, we must make sure that all the calculations are modded by 26. But, if the determinant of the key matrix is a multiple of 26, the determinant would be 0 when modded by 26, and therefore not invertible for our case. Additionally, a matrix can not be a key matrix if the determinant is a multiple of 26's prime factors, 13 and 2.

A way to generate a 3x3 key matrix in R is to create a matrix with 9 slots and fill those slots with randomly chosen numbers valued from 0 to 25 and if the matrix is not invertible modulo 26, change the values in the matrix again until it can be used as a key matrix, which can be done with the function below.

```r
randomKeyMatrix <- function() {
  running <- TRUE
```

```
    while (running) {
      key <- matrix(sample(0:25, 9, replace = TRUE), nrow = 3)
      determinant_mod_26 <- det(key) %% 26

      if (determinant_mod_26 != 0 && gcd(determinant_mod_26, 26) == 1) {
        if(isValidMatrix(key)){
          return(key)
        }

      }
    }
  }
```

To encode a message using the key matrix, we can use this function below. This takes a phrase and converts it to a matrix where the letters become numbers based on the order they fall in the alphabet (A = 0, B = 1, etc.). Then it multiplies the phrase by the key matrix multiplication and mods by 26, and converts the new matrix to letters to reveal the encoded message.

```
  encode <- function(phrase, key){
    phraseMatrix <- convertToNum(phrase)
    numEnc <- (key %*% phraseMatrix) %% 26
    return(convertToLetters(numEnc, convertArray))
  }
```

To find a decoding matrix, we must find the key matrix's inverse modulo 26. This function augments a 3x3 identity matrix to the key matrix in order to find the inverse through transforming the left side of the matrix to an identity.

$$\begin{bmatrix} a & d & g & 1 & 0 & 0 \\ b & e & h & 0 & 1 & 0 \\ c & f & i & 0 & 0 & 1 \end{bmatrix}$$

It performs RREF on the left side starting with the first item, a, scaling the first row by a factor so that:

$$(a \cdot \text{factor}) \mod 26 = 1$$

It then moves on to item b, using a as a pivot to change b to a zero. The first row is multiplied by a factor and added to the second row so that:

$$(a \cdot \text{factor} + b) \mod 26 = 0$$

This step is repeated for item c, and the process is continued onto the second and third columns so that e and i become 1, and d, f, g, and h become zero. Once the left side is successfully transformed to the identity matrix, the right side of the matrix is the inverse of the key and is returned as the decoding matrix.

```r
invMod26 <- function(key){
  identity <- matrix(c(1,0,0,0,1,0,0,0,1), nrow = 3, ncol = 3)
  key <- cbind(key,identity )
  if(key[1,1] != 1){
    key <- scaleRow(key,1,findFactor(key,1))
  }
  if(key[2,1] != 0){
    key[2, ] <- addRows(key, 2, 1, elimFactor(2,1,key))
  }
  if(key[3,1] != 0){
    key[3, ] <- addRows(key, 3, 1, elimFactor(3,1,key))
  }

  if(key[2,2] != 1){
    key <- scaleRow(key,2,findFactor(key,2))
  }
  if(key[1,2] != 0){
    key[1, ] <- addRows(key, 1, 2, elimFactor(1,2,key))
  }
  if(key[3,2] != 0){
    key[3, ] <- addRows(key, 3, 2, elimFactor(3,2,key))
  }

  if(key[3,3] != 1){
    key <- scaleRow(key,3,findFactor(key,3))
  }
  if(key[1,3] != 0){
    key[1, ] <- addRows(key, 1, 3, elimFactor(1,3,key))
  }
  if(key[2,3] != 0){
    key[2, ] <- addRows(key, 2, 3, elimFactor(2,3,key))
  }
  inverseValues <- key[, (ncol(key) - 2):ncol(key)]
  keyInverse <- matrix(nrow = 3, ncol = 3)
  for(i in 0:2){
  keyInverse[i+1, ] <- inverseValues[(i*3 + 1):(i*3 + 3)]
}

  return(t(keyInverse))
}
```

Now that we can get a decoding matrix, we can use this function to decode messages. It converts the encoded message to a numerical matrix that is multiplied by the key inverse modulo 26 and returns the decoded matrix.

```
decode <- function(message, key){
  phraseMatrix <- convertToNum(message)
  key <- invMod26(key)
  numDec <- (key %*% phraseMatrix) %% 26
  return(convertToLetters(numDec, convertArray))
}
```

To test this method of encryption, we will use the phrase "Hill Cipher v Classic Cipher".

```
key <- randomKeyMatrix()
encodedMessage <- encode("hillciphervclassiccipher",key)
encodedMessage
```

```
[1] "lzztjwxkrbmdtrmgsueavnrb"
```

Now, we will input our encoded message above into the decode function to make sure our message was properly encoded.

```
decodedMessage <- decode(encodedMessage, key)
decodedMessage
```

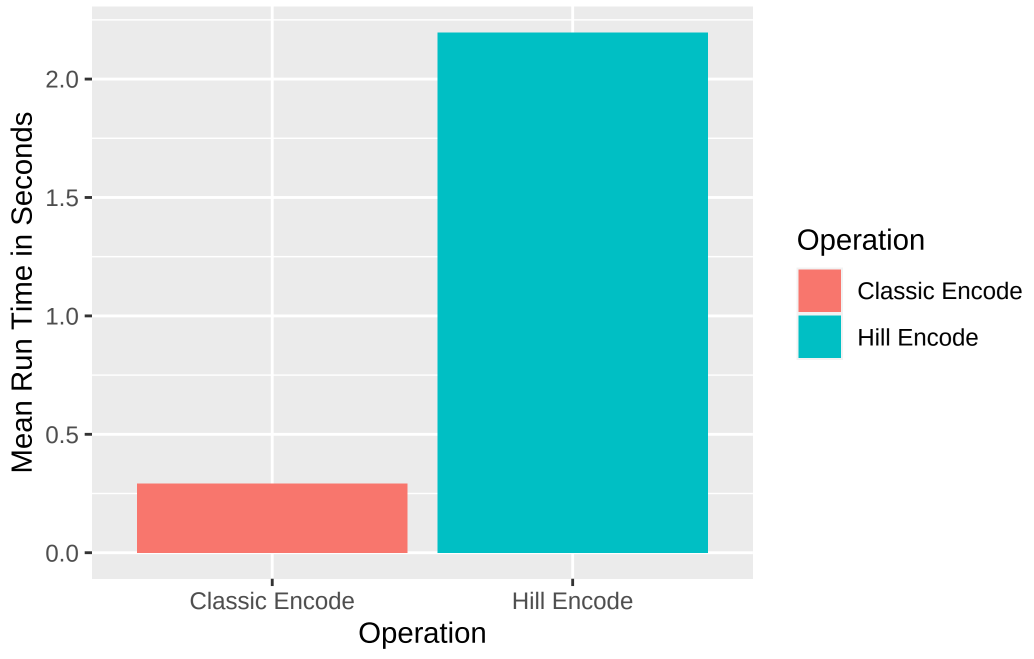
```
[1] "hillciphervclassiccipher"
```

Because the decoded message is the same as the phrase we inputted, it is clear that our functions properly encode and decode. To see the benefits of Hill's encryption method, we will compare it to the classic method of switching letters around.

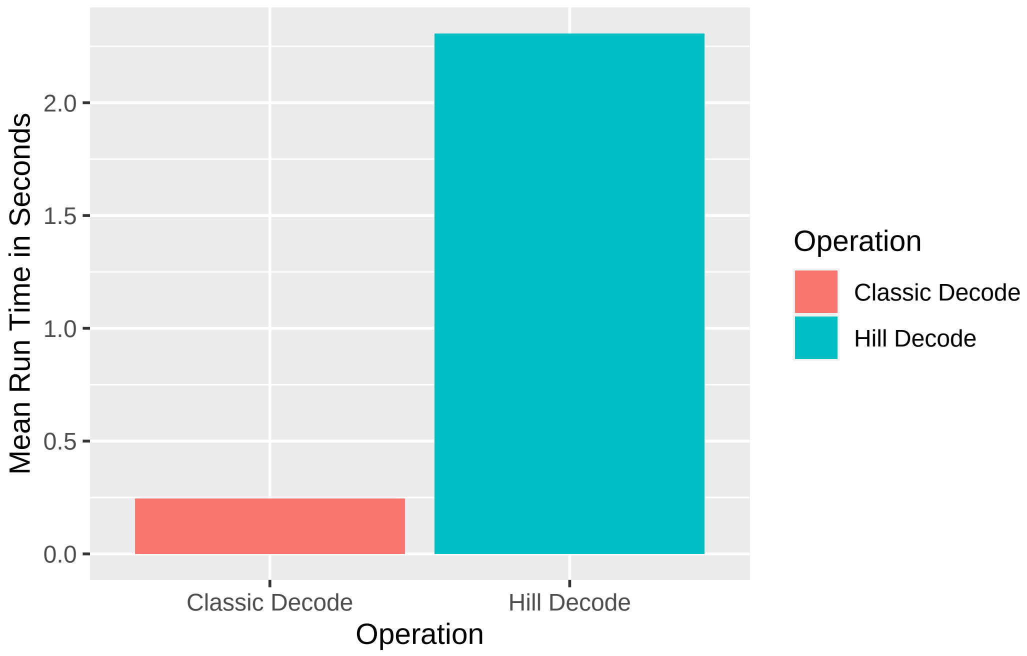
The following code encodes and decodes the phrase "Hill Cipher v Classic Cipher" using each method. It performs each operation 1000 times and records how long it takes, and then repeats 100 times. This allows us to gauge which method takes longer to do, which can be indicative of how difficult it would be to solve these encryptions.

```
phrase <- "hillciphervclassiccipher"
classicEncode <- microbenchmark(
  classic = sapply(1:1000, function(i) encode_phrase(phrase, generate_encoding_cipher())),
  times = 100
)
classicDecode <- microbenchmark(
  classic = sapply(1:1000, function(i) decode_classic(phrase, generate_encoding_cipher()))
  times = 100
)
hillEncode <- microbenchmark(
  hill = sapply(1:1000, function(i) encode(phrase, randomKeyMatrix())),
  times = 100
)
hillDecode <- microbenchmark(
  hill = sapply(1:1000, function(i) decode(phrase, randomKeyMatrix())),
  times = 100
)

compareFrame <- data.frame(classicEncode = classicEncode$time, classicDecode = classicDeco
```



From this graph, it is clear that encoding using a Hill Cipher takes substantially longer than a classic cipher. From this sample, it takes 7.5 times longer to encode with a Hill Cipher than classic.



Again, Hill Ciphers require a lengthy process to decode, specifically 9.4 times as long. While 2 seconds doesn't seem that long, it is important to consider that the key matrix used is only a 3x3 matrix which can greatly lessen the amount spent on the calculations in addition to the numerical matrix formed by the phrase "Hill Cipher v Classic Cipher" only being a 3x8 matrix.

In conclusion, Hill Ciphers are a great method for encryption, as it's extra steps greatly increase the difficulty of decryption in comparison to the traditional letter swap. Further exploration and use of Hill Ciphers can be done with larger matrices or different values used to represent letters to better ensure the protection of secret messages.