Tuto2_corrections

Question 1

You will find the correct syntax (for the first part of your code) below. Note:

- An equivalence check (such as verifying whether column is equal to 1) is performed by using the operator `==` and not `=`. The `=` operator is an allocation (You allocate a value to a variable name). `==` is a that takes two arguments and checks whether they are equal, either in memory, or the stack, or both (dependent on the programming language)
- `ui` can be declared as a function to be iterated over the empty matrix you created (either with for loops, like you did) or with the use of R functions like `apply()`.
- Opening a block, like a `if` or `for` block requires a condition set in parentheses `(...)` and an action set in brackets `{...}`. Your nested for-loops didn't work because of missing brackets and ill-declared conditions (e.g. using `=` instead of `==`)

Note: I reduced n to 10 and p to 4 for visualization purposes.

```
n = 10
p = 4

X = matrix(0, nrow = n, ncol = 2*p + 1) #this is an empty matrix
#number of rows is n because every row represents a time point for the same cell

ui <- function(i) {2*pi*i/n}

for (k in 1:p) {
  for (row in 1:nrow(X)) {
    for (column in 1:ncol(X)) {
      if (column == 1) {
        #column here is the same as j
        X[row, column] = 1
      } else if (column == 2*k) {
        #k is the same as p
        X[row, column] = cos(k*ui(row-1))
      } else if (column == 2*k + 1) {
        X[row, column] = sin(k*ui(row-1))
      }
    }
  }
}

X
```

```
##       [,1]      [,2]          [,3]      [,4]          [,5]      [,6]
##  [1,]    1  1.000000  0.000000e+00  1.000000  0.000000e+00  1.000000
##  [2,]    1  0.809017  5.877853e-01  0.309017  9.510565e-01 -0.309017
##  [3,]    1  0.309017  9.510565e-01 -0.809017  5.877853e-01 -0.809017
##  [4,]    1 -0.309017  9.510565e-01 -0.809017 -5.877853e-01  0.809017
##  [5,]    1 -0.809017  5.877853e-01  0.309017 -9.510565e-01  0.309017
##  [6,]    1 -1.000000  1.224647e-16  1.000000 -2.449294e-16 -1.000000
##  [7,]    1 -0.809017 -5.877853e-01  0.309017  9.510565e-01  0.309017
##  [8,]    1 -0.309017 -9.510565e-01 -0.809017  5.877853e-01  0.809017
##  [9,]    1  0.309017 -9.510565e-01 -0.809017 -5.877853e-01 -0.809017
## [10,]    1  0.809017 -5.877853e-01  0.309017 -9.510565e-01 -0.309017
##               [,7]      [,8]          [,9]
##  [1,]  0.000000e+00  1.000000  0.000000e+00
##  [2,]  9.510565e-01 -0.809017  5.877853e-01
##  [3,] -5.877853e-01  0.309017 -9.510565e-01
##  [4,] -5.877853e-01  0.309017  9.510565e-01
##  [5,]  9.510565e-01 -0.809017 -5.877853e-01
##  [6,]  3.673940e-16  1.000000 -4.898587e-16
##  [7,] -9.510565e-01 -0.809017  5.877853e-01
##  [8,]  5.877853e-01  0.309017 -9.510565e-01
##  [9,]  5.877853e-01  0.309017  9.510565e-01
## [10,] -9.510565e-01 -0.809017 -5.877853e-01
```

Question 2

/! `t(X) %*% X` does not yield the identity matrix but a diagonal one. There is a mistake in your explanation: `orthogonality` and `orthonormality` are not the same. An orthonormal matrix is orthogonal, but an orthogonal matrix is not always orthonormal. As such $Q^T Q = I$ only holds for orthonormal matrices, contrary to what you said in the assignment.

- Two vectors are orthogonal if $\langle u, v \rangle = 0$. An orthonormal basis is a basis where all basis vectors have length 1 and are orthogonal to each other. wiki (https://en.wikipedia.org/wiki/Linear_algebra)

Normalizing the column vectors of an orthogonal matrix yields an orthonormal matrix

```
round(t(X) %*% X) # the matrix is orthogonal
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]   10    0    0    0    0    0    0    0    0
##  [2,]    0    5    0    0    0    0    0    0    0
##  [3,]    0    0    5    0    0    0    0    0    0
##  [4,]    0    0    0    5    0    0    0    0    0
##  [5,]    0    0    0    0    5    0    0    0    0
##  [6,]    0    0    0    0    0    5    0    0    0
##  [7,]    0    0    0    0    0    0    5    0    0
##  [8,]    0    0    0    0    0    0    0    5    0
##  [9,]    0    0    0    0    0    0    0    0    5
```

```
# rounding helps see the matrix due to floating point operation results
```

```
X_prime = matrix(0, nrow = n , ncol = 2*p + 1)
for (j in 1:ncol(X)){
    X_prime[,j] =  X[,j]/(sqrt(sum(X[,j]^2)))
}

round(t(X_prime)%*%X_prime) # post normalization, the matrix is orthonormal
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    1    0    0    0    0    0    0    0    0
##  [2,]    0    1    0    0    0    0    0    0    0
##  [3,]    0    0    1    0    0    0    0    0    0
##  [4,]    0    0    0    1    0    0    0    0    0
##  [5,]    0    0    0    0    1    0    0    0    0
##  [6,]    0    0    0    0    0    1    0    0    0
##  [7,]    0    0    0    0    0    0    1    0    0
##  [8,]    0    0    0    0    0    0    0    1    0
##  [9,]    0    0    0    0    0    0    0    0    1
```

Question 3

Nothing to add per se.

As part of the code `d=seq(1,p-1,1)` is not used. To have a condition check such that $d < p$, this could be implemented within the function itself.

Question 4

Nothing to add to the R code.

The explanation of the problem of overfitting and taking a too low number of dimensions is missing.

Question 5

The implementation of the Mallow's Cp criterion seems correct.

The computation of the estimator that minimizes the criterion for the two previous function Y_1 and Y_2 is missing.

Question 6

Question was not answered.