Biostatistics 615 Mastery Assignment #1 (10 pts)

Due by September 10th 2024 (Tuesday) 11:59pm. Use Gradescope (via Canvas) to submit an R file.

- Your submission should only contain one `R` file named `streamedCov.R` that contains a function named `streamedCov(x, y)`.
- Your code will be evaluated in Gradescope using 10 different test cases using an automated script. Full credit will be given if your code passes all test cases.
- You are allowed to submit multiple times before the deadline, but only the last submission will be graded. Automated feedback will be provided for each submission.
- You need to implement the function to work with arbitrary (valid) input values beyond the 10 cases tested. If you tweak your implementation so that your functions works specifically for the test cases, you will not receive any credit.
- You may test the function in the Google Colab page at https://bit.ly/615hw1xtra to test your code on a subset of test cases.
- Implement your function as efficient as you can. If your program does not finish after running for 2.0 seconds, you will lose the points for those test cases. Note that the official solution finishes within 0.3 seconds for any test case, so this should be a reasonable time limit.
- All the returned values should have at least 8 correct significant digits, so make sure that your implementation retains sufficient numerical precision.

Problem 1 - `streamedCov.R` (10 pts)

Write a function `streamedCov(x, y)`, which returns covariance between two numeric vectors with one pass.

Let n be the sample size ($n > 1$) and $(X_1, \ldots, X_n)$ and $(Y_1, \ldots, Y_n)$ represent the pair of input vectors.

The formulae for the covariance is given as follows:

$$\begin{aligned} \mathrm{cov}(X, Y) &= \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y}) \\ &= \frac{1}{n-1} \left(\sum_{i=1}^{n} X_i Y_i - n\overline{XY} \right) \end{aligned}$$

A most straightforward way to implement this algorithm is a two-pass algorithm as follows:

```
#' twoPassCov(x, y): calculate covariance between vectors without streaming
#' @param x - A numeric vector
#' @param y - Another numeric vector
#' @return covariance between x and y (with n-1 as denominator)
twoPassCov <- function(x, y) {
```

```
    stopifnot(length(x) == length(y))
    mx = mean(x)
    my = mean(y)
    return(sum((x-mx)*(y-my))/(length(x)-1))
}
```

However, in this problem, there are two important requirements:

- The function must be a one-pass algorithm, meaning that each element is read only once sequentiually during the algorithm without being stored in memory after each iteration.
- The estimated covariance should be as accurate as `twoPassCov(x,y)`, at least up to 8 significant digits, regardless of the input sequences.

A simple way to implement a one-pass algorithm for covariance is shown in the `naiveCov()` function below. While this algorithm is one-pass algorithm, it is not always accurate.

```
#' naiveCov(x, y): naively calculate covariance between vectors while
#'                 streaming. The function must read each element of
#'                 x and y without reusing them after an iteration.
#' @param x - A numeric vector
#' @param y - Another numeric vector
#' @return covariance between x and y (with n-1 as denominator)
naiveCov <- function(x, y)
    stopifnot(length(x) == length(y))
    n = sx = sy = sxy = 0
    for(i in 1:length(x)) {
        n <- n + 1
        sx <- sx + x[i]
        sy <- sy + y[i]
        sxy <- sxy + (x[i] * y[i])
    }
    return((sxy - sx*sy/n)/(n-1))
}
```

For example, in the following example, `naiveCov()` appears as accurate as `twoPassCov()`.

```
> set.seed(20220831)
> x = sample(c(0,1),10000,replace=TRUE)
> y = sample(c(0,1),10000,replace=TRUE)
> print(c(naiveCov(x,y), twoPassCov(x,y)), digits=8)
[1] 0.0032811281 0.0032811281
```

However, it there is an offset in the input, `naiveCov()` starts giving slightly inaccurate answers.

```
> set.seed(20220831)
> x = sample(c(0,1),10000,replace=TRUE) + 5e5
> y = sample(c(0,1),10000,replace=TRUE) + 5e5
> print(c(naiveCov(x,y), twoPassCov(x,y)), digits=8)
[1] 0.0032503250 0.0032811281
```

Your task is to implement another one-pass algorithm `streamedCov()` that has much better accuracy regardless of the input. Your algorithm may be inspired by the West algorithm. No error handling for malformed argument is needed.

For clarity, a skeleton code for `streamedCov.R` is provided below. You may only modify the skeleton as instructed. If your modification does not follow the instruction, your score will be zero, regardless of the scores by Gradescope.

```
#' streamedCov(x, y): calculate covariance between vectors accurately while
#'                    streaming. The function must read each element of
#'                    x and y without reusing them after an iteration.
#' @param x - A numeric vector
#' @param y - Another numeric vector
#' @return covariance between x and y (with n-1 as denominator)
streamedCov <- function(x, y) {
    ## ensure that two input vectors are the same size
    stopifnot(length(x) == length(y))
    ######################################################
    ## YOU MAY DEFINE NEW VARIABLES HERE
    ######################################################
    for(i in 1:length(x)) {
        xi = x[i]
        yi = y[i]
        ################################################
        ## YOUR PRIMARY IMPLEMENTATION GOES HERE
        ## YOU MAY ONLY USE xi, yi, and OTHER VARIABLES
        ## YOU DEFINED ABOVE.
        ## YOU MAY NOT USE OTHER ELEMENTS OF x and y
        ## EXCEPT FOR xi=x[i] and yi=y[i]
        ## (e.g. NEVER USE mean(x) or sum(y))
        ################################################
    }
    ######################################################
    ## YOU MAY RETURN CALCULATED COVARIANCE HERE
    ######################################################
}
```

Note that you are NOT allowed to use any functions outside the `base` package in your implementation. Use `help(...)` to check whether a function belongs to the `base` package or not. In addition, you are NOT allowed to use functions that access all elements in the input vectors, such as `mean()` or `sum()`.