MATH6166/MATH6173 – Coursework 1

Instructions for Coursework

1. This coursework assignment is worth 80% of the overall mark for the module MATH6166, and 40% of the overall mark for the module MATH6173.
2. The deadline is 15:00 on Thursday 21st November 2024, and your completed work must be submitted electronically via Blackboard.
3. All of your answers to the coursework are to be written in the R Markdown file "Coursework-template.Rmd", which can be found under the Coursework heading in the Assessment panel on Blackboard. Download this file and write all of your answers to the coursework questions in this file only.
4. The file "Coursework-template.Rmd" is a template answer sheet that is used to write your answers to all of the coursework questions. This is the only file that you need to use to write down your answers. You must **not** delete any of the content from this file; for example, do not change the names or options of code chunks, or change the layout of the file. The layout of the file has been designed specifically so that the correctness of the R functions that you write can be systematically assessed.
5. You must write all of your coding answers to each question in the corresponding empty R Markdown code chunk within the file. Do not create any new code chunks. Each code chunk has a name that matches the corresponding coursework question, and there are comments that signpost where to provide an answer for a specific question. For example, for Question 1 (a), you will see the comment `Write your code for Q1 (a) in the code chunk below:` and below this comment will be a code chunk with the label Q1a. Any changes you make outside of the designated code chunks will likely result in your assessment not being able to be marked. All the coding answers must be carried out through R commands (do not use comments to give your answers).
6. Where relevant, report your answers in the designated section below the R code chunks for each question in the R Markdown file. For example, for Question 1 (a), you will see the comment `Report your answer for Q1 (a) below:`, below which you can write your answer in standard R Markdown formatting.
7. When writing an R function, the

   - function name,
   - argument names and data type, and
   - output names and data type

   must be exactly the same as those specified in the questions; otherwise a heavy penalty will be applied. Please note that uppercase and lowercase of the same alphabet are **not** the same character, so if the question asks to write a function/argument called, `apple`, but a function/argument called `applE` is presented in your answer, then that would be wrong.
8. Your submission must consist of exactly **two** files; the R Markdown file and its associated html file that is created by knitting/rendering the R Markdown file. The R Markdown file is the "Coursework-template.Rmd" file with your answers added in the appropriate places. Using this file, you must generate a html output file using the knit functionality in R Markdown, and submit this along with the R Markdown file.
9. Your R Markdown file must be renamed, and have the file name <your student ID>.Rmd. For example, if your student ID is 12345678, then your script must have the file name 12345678.Rmd.
10. Similarly, the associated html file should be named <your student ID>.html. For example, if your student ID is 12345678, then your html file must have the file name 12345678.html. (This should be correctly named by default provided that you have named the R Markdown file correctly.)

11. Missing R code will be penalised. Please ensure the submitted files have the correct file format; otherwise it cannot be marked. Note that an R Markdown file is a different format to an R script file.
12. Your entire R Markdown file must be reproducible and run without any errors.
13. You must informatively comment your R code where appropriate.
14. Unless specifically stated otherwise, you **do not** have to include error handling in your code.
15. Ensure that all required plots are informative, including the use of appropriate plotting areas, points/lines sizes and colours, as well as including meaningful labels, titles and legends.
16. Except for `ggplot2`, you must **not** load any add-on packages, i.e., using functions `library` or `require` etc.
17. All data files required to complete the Coursework can be downloaded from Blackboard, under the Coursework panel.
18. To help you with the completion of the Coursework, there is an example Coursework question and related files on Blackboard for reference. You can find the following files under the Coursework Example Question panel:

    - An example Coursework question, in the file `Coursework-example.pdf`,
    - A template R Markdown file `Coursework-example-template.Rmd` in which to write your answers to the example Coursework question,
    - A model solution R Markdown file `Coursework-example-solutions.Rmd`, which contains solutions to the Coursework example question and shows you the way you are expected to fill in the `Coursework-example-template.Rmd` file with your answers.
    - The model solution output html file `Coursework-example-solutions.html`, which is simply the rendered output of the `Coursework-example-solutions.Rmd` file.
    - `all-ad-sales.txt` and `youtube-ad-sales.txt`: the data files needed for the example question.

19. When asked to import a data file into your R workspace, you should first set the working directory to the location of the file on your local computer using the command `setwd`, and then read the file in using the appropriate R command (e.g. `read.csv` or `read.table`). You should set the working directory **every time** you have to import a data file. For an example of how to do this, see Question 1 (a) in the example Coursework solutions file.
20. The standard Faculty rules on late submissions apply: for every day, or part of a day, that the coursework is late, the mark will be reduced by a factor of 10%. No credit will be given for work which is more than five days late.
21. All coursework must be carried out **independently**, without collaborating with other students. You are reminded of the University's Academic Responsibility and Conduct Policy, see https://www.southampton.ac.uk/about/governance/regulations-policies/student-regulations/academic-responsibility-conduct.
22. In the interests of fairness, queries which relate directly to the coursework must be made via the Coursework Discussion Forum on Blackboard. This ensures that everybody has access to the same information.

Total marks available: 80

Question 1: Exploratory analysis of Melbourne house price data set

[*21 marks*]

The data set in the file `melbourne_house_price.csv` contains information on properties sold in Melbourne, Australia in 2016 and 2017. The data consists of the following seven columns:

1. `Suburb`: a character variable, giving the name of the Melbourne suburb the property is in.
2. `Address`: a character variable, giving the street address of the property.
3. `Rooms`: a numeric variable, giving the number of bedrooms in the property.
4. `Type`: a character variable, giving the type of the property. This can be either `"h"` (house), `"t"` (townhouse), or `"u"` (unit or duplex).
5. `Price`: a numeric variable, giving the price that the property was sold for (in AU$).
6. `BuildingArea`: a numeric variable, giving the area (in m^2) of the property.
7. `YearBuilt`: a numeric variable, giving the year the property was built.

Not all of the entries in the `melbourne_house_price.csv` data set file have been observed. Some values in the `BuildingArea` and `YearBuilt` columns are missing, and these are represented by the value `NA`. For example, in the `YearBuilt` column, the 3rd entry has been observed, and has value 2008. However, the 1st and 2nd entries are missing, and have value `NA`.

(a) [*2 marks*] Import the data set from the `melbourne_house_prices.csv` file into the R workspace in a variable named `house_data`. How many properties in the data set have more than 4 bedrooms? Store this in a variable named `more4_rooms`.

(b) [*2 marks*] Convert the `Suburb`, `Rooms`, and `Type` columns into factor variables. How many unique categories are there for the `Suburb` variable? Store the number in a variable named `num_suburbs`.

(c) [*2 marks*] Calculate the median building area for properties built in the year 1980 or before. Make sure to remove any missing data (from both variables) from your calculations. Store it in a variable named `median_area`.

(d) [*2 marks*] What is the address of the most expensive townhouse sold in the Brunswick suburb? Store this in a variable named `brunswick_t_max_address`.

(e) [*3 marks*] Create a new column, named `FullyObserved`, that contains the value `TRUE` if all observations in a single row have been observed, and `FALSE` otherwise. What is the name of the column with the most missing values? Store it in a variable named `col_miss`.

(f) [*3 marks*] Create a side-by-side box plot of `Price` for the three `Suburb` values `"Bentleigh East"`, `"Preston"`, and `"Reservoir"`.

(g) [*3 marks*] On a single figure, plot two histograms of the `BuildingArea` variable on the probability scale, with one histogram for properties with 1 bedroom, and the other for properties with 4 bedrooms, using x-axis limits of 0 and 300.

Hint: you can add one histogram plot to another by supplying the argument `add = TRUE` to the `hist()` function.

(h) [*4 marks*] For the columns containing numeric variables, one approach to dealing with `NA` values is to replace the `NA` values with the mean value of the observed variables in the column according to some rule.

Replace `NA` values in the `BuildingArea` variable with the mean value of the `BuildingArea` values that have the same number of `Rooms` variable. If all data for a given number of bedrooms is missing, then use the mean of all of the `BuildingArea` values instead. For example, observation 1 has an `NA` value in the `BuildingArea` column, and 2 in the `Rooms` column, so the `NA` value should be replaced by the mean `BuildingArea` of all 2 bedroom properties.

Question 2: Classifying data using k-nearest neighbours

[*27 marks*]

Scientists use measurements collected from radar signals in order to detect objects in the Earth's ionosphere (a region of the Earth's atmosphere stretching from a height of about 50km to more than 1000km). The scientists record data consisting of the radar measurements from a signal sent to a specific target location in the Earth's ionosphere.

Given new radar readings from a target location in the ionosphere, the scientists are interested in classifying whether or not the radar measurements give evidence of an object being present at that location. Each data observation therefore comes from one of two classes, 1 or 2, with 1 representing an object being present, and 2 representing **no** object being present. The scientists aim to predict, given a new observation of radar measurements, if the observation should be classified as coming from class 1 (object detected) or class 2 (no object detected).

Mathematically, suppose we observe p-dimensional data that comes from one of two classes, 1 or 2. The scientists have access to a *training data set* of n pairs of data points $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)$, where for each $1 \leq i \leq n$, $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{1, 2\}$. The value y_i is the class label of $\boldsymbol{x}_i$, so that in the training data set, the class membership of each observation is known.

The k-nearest neighbours algorithm can be used to predict which class a new observation $\boldsymbol{x}$ belongs to. For a given distance function $f(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p \to [0, \infty)$, we order the data points as $(\boldsymbol{x}_{(1)}, y_{(1)}), (\boldsymbol{x}_{(2)}, y_{(2)}), \ldots, (\boldsymbol{x}_{(n)}, y_{(n)})$ according to the distance of the point to $\boldsymbol{x}$ with respect to f, so that

$$f(\boldsymbol{x}_{(1)}, \boldsymbol{x}) \leq f(\boldsymbol{x}_{(2)}, \boldsymbol{x}) \leq \cdots \leq f(\boldsymbol{x}_{(n)}, \boldsymbol{x}). \tag{1}$$

Then, the predicted class of $\boldsymbol{x}$, $\widehat{y}$, is given by the majority class of the k-nearest data points (neighbours) to $\boldsymbol{x}$:

$$\widehat{y} = \text{argmax}_{y \in \{1,2\}} \left\{ \sum_{i=1}^{k} \mathbb{I}(y_{(i)} = y) \right\}, \tag{2}$$

where $\mathbb{I}$ is the indicator function.

(a) [*5 marks*] Write a function, called `classify_point`, that classifies a new observation $\boldsymbol{x}$ according to the k-nearest neighbours rule in Equation (2) given a set of training observations $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)$ and a user-specified distance function f. The `classify_point` function should have the following features:

Arguments:

1. `x`: a numeric vector of length p, giving the new observation $\boldsymbol{x} \in \mathbb{R}^p$ that is to be classified.
2. `training_data`: a numeric matrix with n columns and $p + 1$ rows, consisting of training data. Each column corresponds to an observation $(\boldsymbol{x}_i, y_i)$. The first p rows contain the p variables from the n vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. The last row contains entries equal to either 1 or 2, that depicts which of the two classes the observation belongs to.
3. `k`: a single numeric value, a positive integer k representing how many nearest neighbours to use.
4. `dist_func`: a function object, giving the distance function f to use as in Equation (1), that will be provided by the user. You do not need to create your own distance function, but you should assume that `dist_func` has two arguments, `a` and `b`, representing the two observations whose distance is being computed. For example, for 1-dimensional data, one possible choice is `dist_func = function(a, b){abs(a - b)}`, the absolute value of the difference between `a` and `b`.

Computation:

The function should compute the ordered distances with respect to a new observation $\boldsymbol{x}$ as given in Equation (1). Then, the function should compute the predicted class of $\boldsymbol{x}$ according to Equation (2).

If there is no majority class, i.e. if k is an even number and $\sum_{i=1}^{k} \mathbb{I}(y_{(i)} = 1) = \sum_{i=1}^{k} \mathbb{I}(y_{(i)} = 2) = k/2$, then $\boldsymbol{x}$ should be predicted as belonging to the class of the $(k+1)$-th nearest neighbour instead.

Return:

- `predicted_class`: a single numeric value, equal to either 1 or 2, giving the predicted class of the new observation $\boldsymbol{x}$.

(b) [*2 marks*] Write a function, called `classify_data`, that classifies a new set of m observations $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \ldots, \boldsymbol{x}^{(m)}$ according to the k-nearest neighbours rule given a set of training observations $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)$. Your function should call the function you wrote in part (a) where appropriate. The `classify_data` function should have the following features:

Arguments:

1. `test_data`: a numeric matrix with m columns and p rows, giving the new observations $\{\boldsymbol{x}^{(i)}\}_{i=1}^{m}$ that are to be classified.
2. `training_data`: a numeric matrix with n columns and $p+1$ rows, consisting of training data. The argument has the same specifications as in part (a).
3. `k`: a single numeric value, a positive integer k representing how many nearest neighbours to use. The argument has the same specifications as in part (a).
4. `dist_func`: a function object, giving the distance function f to use as in Equation (1). The argument has the same specifications as in part (a).

Computation:

The function should classify each of the m observations in `test_data` as belonging to the class 1 or 2, using the `classify_obs` function that you wrote in part (a).

Return:

- `predicted_classes`: a numeric vector of length m, with entries equal to either 1 or 2, whose i-th entry corresponds to the predicted class $\widehat{y}^{(i)}$ of the new observation $\boldsymbol{x}^{(i)}$.

(c) [*4 marks*] Import the data sets `ion_train.txt` and `ion_test.txt` into your R workspace into variables named `ion_train` and `ion_test` respectively. Using the training data set, compute the predicted classes from the test data set for $k \in \{1, 2, \ldots, 13\}$, using the Euclidean distance $f(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{p}(a_i - b_i)^2}$. Store the predicted classes in a data frame object `predicted_classes` of dimensions $m \times 13$, where m is the number of new observations to predict, and where the columns consist of the predictions for the 13 different values of k.

(d) [*3 marks*] By comparing the predicted classes $\widehat{y}^{(1)}, \widehat{y}^{(2)}, \ldots, \widehat{y}^{(m)}$ to the true classes $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$, a 2×2 confusion matrix C can be computed with entries given by:

$$C_{ij} = \sum_{l=1}^{m} \mathbb{I}\left(y^{(l)} = i, \widehat{y}^{(l)} = j\right), \qquad 1 \leq i, j \leq 2. \tag{3}$$

Write a function, called `calc_conf_mat`, that calculates the confusion matrix C for a given set of predicted classes $\widehat{y}^{(1)}, \widehat{y}^{(2)}, \ldots, \widehat{y}^{(m)}$ and true classes $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$. The `calc_conf_mat` function should have the following features:

Arguments:

1. `predicted_classes`: a numeric vector of length m, giving the values of the predicted classes $\widehat{y}^{(1)}, \widehat{y}^{(2)}, \ldots, \widehat{y}^{(m)}$.
2. `true_classes`: a numeric vector of length m, giving the values of the true classes $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$.

Computation:

The function should compute the 2×2 confusion matrix C given in Equation (3). The confusion matrix should have meaningful row and column names distinguishing the entries of the matrix.

Return:

`conf_mat`: a matrix object, giving the 2×2 confusion matrix C given in Equation (3).

(e) [*3 marks*] Further analysis by the scientists finds that the first 15 observations (i.e. those in columns 1–15) in the `ion_test` data set belong to class 1, whilst the last 15 observations (those in columns 16–30) belong to class 2. Using the `calc_conf_mat` function you wrote in part (d), plot the proportion of incorrectly classified data points for the classifications computed in part (c) over the given range of $k \in \{1, 2, \ldots, 13\}$.

(f) [*3 marks*] Run the `classify_data` function on the test data set with $k = 13$ for the following 3 distance functions:

$$f_1(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{p} |a_i - b_i|,$$

$$f_2(\boldsymbol{a}, \boldsymbol{b}) = \max_{1 \le i \le p} |a_i - b_i|,$$

$$f_3(\boldsymbol{a}, \boldsymbol{b}) = 1 - \frac{\sum_{i=1}^{p} a_i b_i}{\sqrt{\sum_{i=1}^{p} a_i^2} \sqrt{\sum_{i=1}^{p} b_i^2}}.$$

Which distance function yields the highest proportion of correctly classified data points for the `ion_test` data? Store the name of the function (`"f1"`, `"f2"`, or `"f3"`) in a variable called `best_func`.

(g) [*4 marks*] Using the predicted classes from the classification in part (f) that yielded the highest proportion of correctly classified points, plot dimensions 6 and 7 of the `ion_test` data on a scatter plot, with the two (true) classes distinguished from one another by colour, and with incorrectly classified points given by cross (x) symbols.

Hint: Several of the data points have identical values. To better visualise the data, use the `jitter` function to add random noise to the data.

(h) [*3 marks*] In this question, you will write a function, called `classify_data2`, that improves upon the function `classify_data` that you wrote in part (b), by including checks to ensure that some of the arguments passed to the function satisfy certain requirements. First, copy and paste the code you wrote for part (b) into your answer for part (h), and rename the function name to `classify_data2` in part (h).

The function `classify_data2` should check that the arguments satisfy the following three requirements:

1. The last row of `training_data` should only contain numeric values equal to 1 or 2.
2. The dimension of observations in `test_data` and `training_data` match up, i.e. the number of rows of `test_data` should be 1 less than the number of rows of `training_data`.
3. The argument `dist_func` should be a function object.

If invalid inputs are passed to this function, it should stop prematurely by using the appropriate R function and print out the associated relevant error message **exactly** as written below:

1. `Last row of training_data should only contain numeric values equal to 1 or 2.`
2. `Dimensions of test_data and training_data do not agree.`
3. `Argument dist_func should be a function object.`

For example, if requirement 2 is not satisfied, then the function should stop and print the error message 2.

Question 3: Detecting a change point in the mean of a data sequence

[*32 marks*]

Suppose we observe a data sequence of n (1-dimensional) observations $X_1, \ldots, X_n$, ordered sequentially in time, denoted as $\{X_t\}_{t=1}^n$. Change point detection is an area of statistics concerned with locating time points in a data sequence that undergo abrupt changes in its statistical properties.

The most well-studied change point detection problem involves estimating change points in the **mean** value of a data sequence. The single change in mean model for a data sequence $\{X_t\}_{t=1}^n$ is given by:

$$X_t = \begin{cases} \mu_0 + \varepsilon_t & 1 \leq t \leq \theta, \\ \mu_1 + \varepsilon_t & \theta + 1 \leq t \leq n, \end{cases} \tag{4}$$

where

- μ_0 is the mean value *before* the change,
- μ_1 is the mean value *after* the change,
- θ is the change point location,
- ε_t is an i.i.d. sequence of $N(0, \sigma^2)$ random variables.

We wish to test whether a change point has occurred, and if it has, estimate its location.

(a) [*3 marks*] Simulate an example data sequence $\{X_t\}_{t=1}^n$ following model (4), with $\theta = 150$, $n = 500$, $\mu_0 = 0$, $\mu_1 = 2$, and $\sigma^2 = 0.8$, and store it in a variable named `x`. Plot the simulated data set `x` using a line plot, and add to the plot a line $\widehat{\mu}_t$, with $t \in \{1, 2, \ldots, 500\}$, with values given by:

- $\widehat{\mu}_t = 150^{-1} \sum_{t=1}^{150} X_t$ for $1 \leq t \leq 150$,
- $\widehat{\mu}_t = 350^{-1} \sum_{t=151}^{500} X_t$ for $151 \leq t \leq 500$.

Lastly, add a vertical line at $y = 150$.

A common way to find change points in a data set is to construct a test statistic $T(k)$ that measures how likely there is to be a change at the time location k. The cumulative sum (CUSUM) test statistic is given by

$$T(k) = \left| \sqrt{\frac{k(n-k)}{n}} \left(\frac{1}{k} \sum_{t=1}^{k} X_t - \frac{1}{n-k} \sum_{t=k+1}^{n} X_t \right) \right|, \qquad 1 \leq k \leq n-1. \tag{5}$$

$T(k)$ is a scaled absolute difference in the sample means of the data to the left and right of time point k. Computing the CUSUM statistic $T(k)$ for all $k \in \{1, 2, \ldots, n-1\}$ using Equation (5), we can obtain the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$.

(b) [*7 marks*] Write a function, called `CUSUM.calc`, to compute the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$ using Equation (5). The function should have the following features:

Argument:

- `x`: a numeric vector of length n, representing the data sequence $\{X_t\}_{t=1}^n$.

Computation:

The function should compute the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$, where $T(k)$ is given in Equation (5).

By exploiting the structure of Equation (5), the computation of the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$ can be achieved in *linear* time with respect to the length of the data sequence, n. That is, it can be computed using at most Cn mathematical operations, for some constant $C > 0$. You can get 4/7 marks on this question with any correct solution, and 7/7 if your computation of the CUSUM statistic vector is linear time.

Return:

- `x.CUSUM`: a numeric vector of length $n-1$, giving the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$.

In order to decide if a change point is present in the data sequence, we need to measure the noise level σ using an estimate $\widehat{\sigma}$. One way to compute $\widehat{\sigma}$ is by using the scaled median absolute deviation (MAD) of the transformed data sequence $\{Y_t\}_{t=1}^{n-1}$, where $Y_t = (X_{t+1} - X_t)/\sqrt{2}$. Letting $\bar{Y} = \text{median}(Y_1, Y_2, \ldots, Y_{n-1})$, the noise level estimate $\widehat{\sigma}$ is given by

$$\widehat{\sigma} = 1.48 \cdot \text{MAD}\,(Y_1, Y_2, \ldots, Y_{n-1}) = 1.48 \cdot \text{median}\left(|Y_1 - \bar{Y}|, |Y_2 - \bar{Y}|, \ldots, |Y_{n-1} - \bar{Y}|\right). \tag{6}$$

(c) [*2 marks*] Write a function, called `noise.est`, to compute $\widehat{\sigma}$ given in Equation (6), **without** using the R function `mad`. The function should have the following features:

Argument:

- `x`: a numeric vector of length n, representing the data sequence $\{X_t\}_{t=1}^{n}$.

Computation

Without using the R function `mad`, the function should compute the estimate $\widehat{\sigma}$ given in Equation (6).

Return:

- `sigma.est`: a numeric value containing the estimate $\widehat{\sigma}$ given in Equation (6).

To locate the most likely location for a change point, we set $\widehat{\theta}$ as the location with the largest CUSUM test statistic value, i.e.

$$\widehat{\theta} = \text{argmax}_{1 \le k \le n-1}\{T(k)\}. \tag{7}$$

To test if the location $\widehat{\theta}$ is a genuine change point or not, one approach is to declare a change at time $\widehat{\theta}$ if

$$\widehat{\sigma}^{-1} T(\widehat{\theta}) > c, \tag{8}$$

for some positive threshold value c.

(d) [*3 marks*] Using the functions that you wrote in parts (b) and (c), write a function, called `cpt.detect`, that calculates the CUSUM statistic vector of a data sequence and the most likely location of a change point, and decides if this location is a genuine change point or not. The function should have the following features:

Arguments:

- `x`: a numeric vector of length n, representing the data sequence $\{X_t\}_{t=1}^{n}$.
- `threshold`: a positive numeric value, specifying the threshold c in Equation (8). The default value should be $\sqrt{2\log(n)}$.

Computation:

The function should compute the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$ and the noise level estimate $\widehat{\sigma}$ by calling the functions that you wrote in parts (b) and (c) where appropriate. Then, the function should compute $\widehat{\theta}$ according to Equation (7), and finally check if $\widehat{\theta}$ should be declared a change point by comparing $\widehat{\sigma}^{-1}T(\widehat{\theta})$ to the threshold c as in Equation (8).

Return:

- `x.cpt`: a list object with the following four elements:
  - `x.CUSUM`: a numeric vector of length n, the CUSUM statistic vector of `x`, $\{T(k)\}_{k=1}^{n-1}$,
  - `sigma.est`: a single numeric value, containing the estimate $\widehat{\sigma}$.
  - `cpt`: a single numeric value, the most likely location of the change point $\widehat{\theta}$ calculated via Equation (7),
  - `is.cpt`: a logical value, equal to `TRUE` if $\widehat{\theta}$ is declared a change point according to Equation (8), and `FALSE` if not.

(e) [*4 marks*] The R function `system.time` can be used to compute the running time of the `cpt.detect` function that you wrote in part (d). Use the `system.time` function to compute the average running time of the function `cpt.detect` that you wrote in part (d) over a range of data lengths n as follows:

- For each $n \in \{100, 500, 1000, 2000, 4000\}$, repeat the below process 100 times:
  - Generate a data sequence `y` of length n consisting of n $N(0, 1)$ random variables.
  - Use `system.time` to calculate the running time of `cpt.detect(y)`.
- Then, for each n, calculate the average running time of `cpt.detect(y)` over the 100 repetitions.
- Store all of the average running times in a numeric vector of length 5 named `ave_run_times`.

After you have done this, plot the average running times for the different values of n, using points that are connected by a line. Ensure that the axis labels match up with the five values of n.

To instead set the threshold c using the observed data, we can use an approach known as permutation testing. For a given number of replications R, permutation testing works as follows:

For each replication number $r \in \{1, 2, \ldots, R\}$:

- Generate a random permutation, $\{X_t^{(r)}\}_{t=1}^n$, of $\{X_t\}_{t=1}^n$, that is, a random re-arrangement of $\{X_t\}_{t=1}^n$ into a new order. In R, for a vector object `x`, this can be achieved using `sample(x)`.
- Denoting $\{T^{(r)}(k)\}_{k=1}^{n-1}$ as the CUSUM statistic vector computed on the permuted data set $\{X_t^{(r)}\}_{t=1}^n$, compute the maximum value of the CUSUM statistic $T_{\max}^{(r)} = \max_{1 \le k \le n-1}\{T^{(r)}(k)\}$, and compute $\widehat{\sigma}^{(r)}$ on the permuted data set $\{X_t^{(r)}\}_{t=1}^n$ according to equations (5) and (6).
- Store the value $A^{(r)} = \widehat{\sigma}^{(r)^{-1}} T_{\max}^{(r)}$.

Then, the threshold can be calculated as:

$$c = q_{1-\alpha}(\{A^{(r)}\}_{r=1}^R), \tag{9}$$

the $100 \times (1-\alpha)\%$-quantile of $\{A^{(r)}\}_{r=1}^R$ for a chosen significance level $\alpha \in (0, 1]$. Denoting $T_{\max} = \max_{1 \le k \le n-1}\{T(k)\}$, an approximate p-value for the test statistic can then be calculated as:

$$p = \frac{\sum_{r=1}^R \mathbb{I}\left(A^{(r)} \ge \widehat{\sigma}^{-1} T_{\max}\right)}{R}. \tag{10}$$

(f) [*5 marks*] Write a function, called `perm.test`, that uses a loop structure to generate R random permutations of the data set $\{X_t\}_{t=1}^n$, computes the value $A^{(r)}$ for each $r \in \{1, \ldots, R\}$, and computes the corresponding threshold c according to Equation (9), and approximate p-value according to Equation (10). The `perm.test` function should have the following features:

Arguments:

1. `x`: a numeric vector of length n, representing the data sequence $\{X_t\}_{t=1}^n$.
2. `x.CUSUM.max`: a single numeric value, the maximum, $\max_{1 \leq k \leq n-1}\{T(k)\}$, of the CUSUM statistic vector $\{T(k)\}_{k=1}^{n-1}$,
3. `sigma.est`: a single numeric value, containing the noise level estimate $\widehat{\sigma}$.
4. `reps`: a numeric value, giving the number of permutations, R, to perform. The default value should be 199.
5. `sig.lvl`: a numeric value between 0 and 1, giving the significance level α of the permutation test. The default value should be 0.05.

Computation:

The `perm.test` function should use a loop structure to calculate the permutation test threshold c and approximate p-value p according to Equations (9) and (10). You should loop R times in order to compute the values $\{A^{(r)}\}_{r=1}^R$, and then calculate c and p.

Return:

- `x.perm`, a list object with the following two elements:
  1. `perm.c`: a single numeric value giving the calculated threshold.
  2. `p.val`: a single numeric value giving the approximate p-value.

Hint: the `sample` function will generate a random permutation of a vector.

(g) [*3 marks*] In this question, you will write a function, called `cpt.detect2`, that improves upon the function `cpt.detect` that you wrote in part (d), by including an option for the threshold c to be calculated using the permutation test. First, copy and paste the function you wrote for part (d) into your answer for part (g), and rename the function name to `cpt.detect2` in part (g). The `cpt.detect2` should include the following extra features:

Arguments:

The function should have the same arguments as `cpt.detect`, as well as the following additional arguments:

- `threshold.type`: a character object specifying the threshold type. If equal to `"manual"`, the threshold is given by the `threshold` argument. If it is equal to `"perm"`, then the permutation test is used to calculate the threshold.
- `reps`: a numeric value, giving the number of permutations, R, to perform (if the permutation test is used). The default value should be 199.
- `sig.lvl`: a numeric value between 0 and 1, giving the significance level α of the permutation test (if the permutation test is used). The default value should be 0.05.

Computation

The function should perform the same computations as `cpt.detect`. Additionally, if the argument `threshold.type = "perm"`, the function should compute the threshold c and approximate p-value p by calling the `perm.test` function from part (f).

Return:

In addition to the four elements specified in part (d), the following two elements should also be included in the returned list object `x.cpt`:

- `threshold.type`: the user inputted value for the argument `threshold.type`,
- `threshold`: the value of the threshold, chosen manually or computed via the permutation test.

Lastly, if `threshold.type = "perm"`, then the following element should also be included in the returned list object `x.cpt`:

- `p.val`: if `threshold.type = "perm"`, the approximate p-value p that was computed using the permutation test.

(h) [*5 marks*] In this question you will use the code you've written so far to analyse data collected on the river Nile. Data in the file `nile_volume.txt` records measurements of the annual volume (in units $10^8 m^3$) of discharge from the Nile River at Aswan for the years 1871 to 1970. The measurements are of meteorological importance as evidence of a possible abrupt change in the rainfall levels around the turn of the 20th century.

Load the `nile_volume.txt` data set into your R workspace in a variable named `nile.data`. Apply your `cpt.detect2` function to the Nile data using the threshold calculated via the permutation test at significance level 0.01 and using 499 replications, and store it in a variable named `nile.cpt`. In a single figure with two plots arranged one above the other, plot on the uppermost plot:

- The Nile data, along with a line $\widehat{\mu}_t$ with $t \in \{1, 2, \ldots, n\}$ giving the estimated mean function:
  - $\widehat{\mu}_t = \widehat{\theta}^{-1} \sum_{t=1}^{\widehat{\theta}} X_t$ for $1 \leq t \leq \widehat{\theta}$,
  - $\widehat{\mu}_t = \left(n - \widehat{\theta}\right)^{-1} \sum_{t=\widehat{\theta}+1}^{n} X_t$ for $\widehat{\theta} + 1 \leq t \leq n$.
- a vertical line at the location of the most likely change point $\widehat{\theta}$.

On the lowermost plot, plot:

- The CUSUM statistic of the Nile data divided by the noise estimate $\widehat{\sigma}$. Add a vertical line at the location of the most likely change point $\widehat{\theta}$, and two horizontal lines: one at the value $y = \sqrt{2\log(n)}$, and one at the value given by `nile.cpt$threshold`.