MATH 1060 Lab 2 Cumulative Distribution

In this lab, we will construct a histogram of cumulative frequency (probability) using the beaver data we used in Lab 1.

In the following, built-in R commands are in typewriter font (e.g., `command`), and variables in italic (*variable*) in order to make it clear what they are. You may use different variable names than the ones I use on this manual. (Actually, if you are a novice programmer I encourage you to do so. Think of better names for the variables we use below.) Lines in boxes are to be copied to your script file and executed, line by line, so that you will understand what each line is doing.

In case something goes wrong, try restarting R and run the whole script (Menu: Session → Restart R).

1. Import the beaver data we used in Lab 1

   Line 1
   ```
   beaver <- read.csv("······/beaver.csv")
   ```
   Recall that " <- " is the assignment operator. It assigns a value to a variable on the left side. Let's see what's inside *beaver* by typing in the console "beaver". You will see columns of data values such as "day", "time", "temp", etc. The data type of the variable *beaver* is called data frame in R.

2. Now, we will extract the values in the "temp" (body temperature of beaver) column by using "$", and put the values in a vector *x*.

   Line 2
   ```
   x <- beaver$temp
   ```
   Check what's in *x* by typing "x" in the console. It will show the values in the "temp" column of beaver arranged in the vector format. The numbers on the left hand side are indices, i.e., *x*[1]=36.33, etc. You can get the value of the i th element as *x*[i].

3. "range" command will give you the minimum and the maximaum values of a given vector in the format (min, max).

   Line 3
   ```
   r <- range(x)
   ```
   Check the content of *r* in the console.

4. Define the lower and upper bounds of the histogram. Note that *r*[1] is th minimum data value, and *r*[2] is the maximum.

   Line 4
   ```
   xmin <- round(r[1], 1)−0.1
   ```
   Line 5
   ```
   xmax <- round(r[2], 1)+0.1
   ```
   Study the effect of `round` command by typing for example, `round`(1.2345, 1) in the console. Change the second argument to 0, 2, or any other integers, and see what happens. What numbers do we have in *xmin* and *xmax*?

5. Specify the number of classes of the histogram.

   Line 6
   ```
   N <- 10
   ```

6. Define class width

   Line 7
   ```
   step <- (xmax − xmin)/N
   ```
   Check if it is a reasonable number given the range of the data values.

7. Compute the values of class boundaries.

   Line 8
   ```
   boundaries <- seq(xmin, xmax, by=step)
   ```
   Try `seq`(−5, 5, `by`=0.5) in the console. Can you make a sequence of numbers from 0 to 1 with an increment of 0.25? Change the increment to 0.3. Is 1 included in the output?

8. "cut" command tells you which class each element belongs to.

   Line 9
   ```
   xcut <- cut(x, breaks=boundaries, right=FALSE)
   ```
   Examine the content of the variable *xcut*. Note that each element in *x* is replaced by an interval such as [36.2, 36.4), which means $36.2 \leq$ value < 36.4.

9. "table" command compiles a frequency table, which in turn can be displayed graphically using the "plot" command.

   Line 10
   ```
   ft <- table(xcut)
   ```

   Line 11
   ```
   plot(ft, type='h')
   ```

   Make sure to examine what's in *ft*. Note that it is a vector with unique label (in this case, interval) for each element.

10. Let's calculate the probability (relative frequency) that x is less than a given value. To calculate relative frequency, we need the total number of data values:

    Line 12
    ```
    total = sum(ft)
    ```

    `sum` command sums all the elements in the vector, e.g., `sum(seq(1, 10, 1))` = 55 [Short hand for `seq(a,b,1)` = a:b]

11. In the following we will define a function. In R, a function may be defined using "function" command as

    function_name <- `function`(*argument*){*process the argument here*}

    For example, a function to solve quadratic equations:

    cube <- `function`(x){x^3}

    Execute the above code in the console. Then, say, *cube*(3) will give you $3^3 = 27$.

12. The probability that a radomly picked data point falls in the `i`th class or any of the classes to the left of the `i`th (cumulative probability).

    Line 13
    ```
    prob <- function(i){ sum(ft[1:i])/total }
    ```

    Let's find out what *ft*[1:*i*] means. Type first "ft", and, say, "ft[1:3]", "ft[1:4]", and so on in the console. Do you see what's happening? vector[a:b] extract the ath to the bth elements of the vector.

13. Now we substitute numbers from *i*=1 to *i*=*N* in *prob*(*i*) by using the following syntax:

    Line 14
    ```
    cp <- sapply(1:N, prob)
    ```

    The function *prob* is applied to each element of the vector 1:*N*, and the result (cumulative probabilities) is given as a vector.

14. Let's plot the result:

    Line 15
    ```
    plot(boundaries[2:(N+1)], cp, type="l", main="Cumulative Probability: P(X < x)", xlab="x", ylab="Probability")
    ```

    Note that the vector *boundaries* provides the values on the x-axis. In general `plot`(x,y), where x and y are vectors of same length, generates a scatter graph of (x[1], y[1]), (x[2], y[2]), ...

    Do you see an ogive shape? This graph tells you the probability of x being smaller than a given value. E.g., $P(x < 36.6) = 0.1$, $P(x < 37.0) = 0.8$. Note in particular $P(x < 37.6) = 1.0$, which is because all the data values are smaller than 37.6.

15. Now, re-run the whole script with larger *N*, e.g. *N*=50 or 100. You should get a much smoother curve.

16. Let's try one more thing. Comment out line 2 (put "#" at the begining of the line), and type

    Line 2′
    ```
    x = rnorm(1000)
    ```

    This generates 1000 random numbers which are normally distributed with $\mu = 0$, $\sigma = 1$. Inspect *x* in the console.

17. Set *N*=100 or more. Then run the whole script. This smooth curve is called *the cumulative distribution function (CDF) of the standard normal distribution.*

Good job! Submit the script file to LearningHub.