A Colorful Christmas:

about Stats and Floyd-Steinberg Dithering

R-team

December 12, 2019

Assignment Instructions

The current assingment was made in `R` version 3.6.1 and consists of four tasks for a total of 100 points. The fifth task is a bonus consisting of an extra 15 points. Behind each (sub)task you will see the number of points that can be earned. The grading model can always be modified in favor of all students.

Unless it is specified differently in any of the subtasks, you can only use functions from the core packages in `tidyverse`, the package `png`, or the functions from the libraries that automatically load with the default settings in `RStudio`.

Furthermore, load the variables that are stored in `0_data/Model_Answer_Variables.RData`. You can use these variables to check your answers, or use them to be sure you are working with the correct variables from the answers of the (previous) subtasks.

Your style of coding affects assingment grade. A correct answer is preferred above beautiful code, however, it may cost you points when very complicated code is provided as an answer at the place where basic `R` functions would suffice. Adhere to a consistent and neat programming style, e.g.,
+ https://google.github.io/styleguide/Rguide.xml,
+ http://style.tidyverse.org.

Change the name of the file `Lastname_ULCN.Rmd` accordingly (give it your real last name and ULCN number), and write down your answers in this file. Make sure you write your `R` code in `R` code chunks. When you wish or need to write text to answer one of the questions, don't use `R` comments, but just write your text outside the `R` code chunk.

Upload a report of your answers as your own clean `YOURULCN_Lastname.Rmd` file to Blackboard **before** 10.00 hours, on January 9, 2020. Make sure that we are able to knit the `.Rmd` report without any problems in this `Rproject`'s directory.

- Go to Statistical Computing with R –> Exams & Assignments –> Assignment Jaunary 09, 2020.
- Every minute later than 10:00 hours will cost you 10 out of 100 points: when you submit your `.Rmd` file at 10:05 hours spot on, it means you already lost 50 points. Files submitted after 10:05 hours will not be graded.
- To be sure, you can also e-mail your .Rmd file to `rteam@stat.leidenuniv.nl`.

Success!

the R-team.

Context Introduction

In this assignment you will code your own Floyd-Steinberg dithering algorithm:

- R.W. Floyd and L. Steinberg "An Adaptive algorithm for Spatial Grayscale", Proceedings of the Society of Information Display, Vol. 17, No. 2, 1976, pp. 75-77.

Since Leiden University does not provide access to a digital version of this article, we have used the descriptions we could find on Wikipedia and YouTube.

From Wikipedia and YouTube we should be able to find out on how to program our own Floyd-Steinberg dithering algorithm for colors as well. However, from these two sources, we do not get much information about the statistical processes on which the algorithm could rely. How many colours should we use? Therefore, we will have to apply our own idea on how to select the number of colors we would like to use for dithering of the following christmas picture:

Figure 1: The picture that we will compress with the Floyd-Steinberg algorihtm

To select the number colours that we would like to use our idea is based on the Statistical Computing lecture about permutation tests, and (very) similar to the procedure which is referred to as the Gap statistic:

- R. Tibshirani, G. Walther, and T. Hastie "Estimating the number of clusters in a data set via the gap statistic", J.R. Statist. Soc. B., Part 2, 2001, pp. 411-423.

1 About Colors: A Picture = A Data set.

There is a file called `xmas.png` in the `0_img` folder of this exam's working directory, which is a picture of 622 x 960 pixels:

Figure 2: The original xmas

Install the package `png` and then read the PNG picture into an `array` data structure with the function `png::readPNG()`, see the code chunk:

```
# file.info("0_img/xmas.png")[, c("size", "mtime")]
xmas <- png::readPNG(source = '0_img/xmas.png', native = FALSE)
xmas <- xmas[,,-4] # first three matrices only
```

1.1 Explore your PNG R object (2.5: 2.5, 98)

Notice that the class of `xmas` is an array in three dimensions, You could see it as an object consisting of three different matrices for the color intensity values of Red, Green, and Blue channels, respectively. We define a pixel in the `xmas.png` picture as the vector

$$\mathbf{x}_{ij} = \{\text{Red}_{ij}, \text{Green}_{ij}, \text{Blue}_{ij}\}^T,$$

for row i and column j in each of the color intensity values of the red, the green and the blue matrix. For example, with Red = 1, Green = 1, and Blue =1, a pixel would be white (all colors on full intensity), when all values are 0, the pixel is black (no colors).

For more information, check out the section Numeric representations which can be found on https://en.wikipedia.org/wiki/RGB_color_model

Show the Red, Green and Blue values for the pixel in row 106, and column 467. What kind of color do you expect it to have?

1.2 From an RGB array to a data.frame (10: 12.5, 87.5)

Most data analyses in `R` are conducted on a `data.frame` or `tibble`. Change your object of class `array` into a `data.frame` (or `tibble`) of 1.95966×10^5 pixels with the following six variables: `row`, `column`, `red`, `green`, and `blue`, `rgb_color`. Here, the `rgb_color` variable consists of elemens of mode `character`, corresponding to the output of the `rgb()` function.

Show that your transformed object is identical to the `xmas_df` in the `model_answers.RData` variables (present in the `0_data` folder).

1.3 Number of Unique colors in `xmas.png` (2.5: 15, 85)

Use the `data.frame` from **1.2** to show that the number of values in the character variable `rgb_color` is the same as the number of observed combinations of the variables `red`, `green`, and `blue`.

1.4 Creating A Raster To Plot the Picture in R (5: 20, 80)

Collect the character elements that represent the colors in a matrix, and force this matrix to be of class `raster`. Show that you can reproduce the `xmas.png` picture by applying `plot()` on your created `raster`.

2 A Further Understanding of the RGB Space

By default the values for the red, green and blue channels in the `xmas` object are in the interval $[0, 1]$. Note that when we multiply these values by 255, all its values end up to be integer values in the interval $[0, 255]$:

```
all((xmas * 255) %in% (0:255))
```

These RGB colors are from a standard RGB space. This is a three-dimensional space in which each dimension (Red, Green, and Blue) has a total of $2^8 = 256$ unique values. In total, $(2^8)^3 = 2^{24}$ unique colors can be created in this standard RBG Space, also referred to as a 24-Bit RGB.

For more information, take a look at the section "Regular RGB palettes" on https://en.wikipedia.org/wiki/List_of_monochrome_and_RGB_palettes

2.1 Hexadecimal identifiers for the RGB colors (5: 25 - 75)

Create your own `rgb()` function that does not rely on any other language than `R`, but still has the same input arguments `red`, `green` and `blue` and `maxValue`.

The output of your own function should be a character vector with elements of 7 characters (all starting with "#" followed by the red, blue and gree values in hexadecimal). See the "Value section" of the helpfile of `rgb()`, and in your answer you are allowed to use the following code:

```
hexadecimal <- c(0:9, LETTERS[1:6])
hdm2columns <- expand.grid(hexadecimal, hexadecimal)
channel <- paste0(hdm2columns[,2], hdm2columns[,1], sep = "")
```

Show that your own function gives the same output as `rgb()` for `red = 0.5`, `green = 0.3`, `blue = 0.7` and `maxColorValue = 1`.

Hint: if you want to use the round function `round()`, then program it yourself such that it does NOT round to the even numbers. For example, check out `round(3.5)` and `round(4.5)`.

2.2 Create Your Own Palette of RGB colors (5: 30 - 70)

Create a function with which you could find the `RGB` intensity values for the K colors from either the 3-bit, 6-bit, 9-bit, 12-bit, or 15-bit regular RGB palettes. For the colors of the specific palettes, take a look at:

https://en.wikipedia.org/wiki/List_of_monochrome_and_RGB_palettes

The input argument should be `K` (the total number of colors) or the number of bits (`n_bit = log2(K)`). The output of the function should be a list that contains a character vector with the hexadecimal color values, and a matrix of K rows with in its columns the intensity values for red, green, and blue, respectively (scaled from 0-1, or as integers in 0:255).

Your function should give an error when `K` or `n_bit` have a wrong value. The value for K should obey the following property:

$$\log_2(K) = 0 \mod 3,$$

in `R` code this means that `(log2(K) %% 3 == 0)` should evaluate to `TRUE`.

Last, show that your function returns the same objects as we have in the `ModelAnswerVariables.RData` for $\log_2(K) \in \{3, 6, 9, 12, 15\}$, referred to as `RGB_03bit`, `RGB_06bit`, `RGB_09bit`, `RGB_12bit`, `RGB_15bit`, respectively.

2.3 A Naive Approach to Color Reduction (10: 40 - 60)

Let a particular color from the regular RGB palette be denoted by

$$\mathbf{c}_k = \{\text{Red}_k, \text{Green}_k, \text{Blue}_k\}^T$$

for $k \in \{1, \dots K\}$ RGB colors.

In this assignment K can be either 8, 64, 512, 4096, or 32768, each corresponding to the 3-bit, 6-bit, 9-bit, 12-bit, and 15-bit RGB palettes. Moreover, from this part on, define the intensity values of the colors Red, Green, and Blue on the integer scale of $[0, 255]$ for both $\mathbf{c}_k$, and each

$$\mathbf{x}_{ij}$$

.

When reducing the number of colors in `xmas.png` (or the array `xmas`), we could replace each pixel $\mathbf{x}_{ij}$ in row i and column j of a picture with its "closest by" color from the $k \in \{1 \dots K\}$ colors in the particular RGB palette. We define this closest by color as follows:

$$\widehat{\mathbf{c}}_{k_{ij}} = \underset{\mathbf{c}_k}{\text{argmin}} \, ||\mathbf{x}_{ij} - \mathbf{c}_k||_2^2.$$

Note that $||\mathbf{x} - \mathbf{c}_k||_2^2$ denotes the squared ℓ_2 norm (or the squared length) of the difference between $\mathbf{x}$ and $\mathbf{c}_k$, also referred to as the squared Euclidean distance between $\mathbf{x}$ and $\mathbf{c}_k$. Thus, we replace each pixel with the color for which the squared Euclidean distance is the smallest out of the $k \in \{1, \dots, K\}$ colors.

2.3a

Compress the `xmas.png` picture into the colors from the 3-bit RGB palette (where $K = 2^3$). If needed, these colors and the RGB values can obtained from the variable `RGB_03bit` from the `Model_Answer_Variables.Rdata`.

2.3b

Use `writePNG()` or `plot()` to show your own compressed picture of `xmas.png` for the $K = 2^3$ equally spaced colors (the colors that correspond to the 8 corners of the cube that can represent the RGB space).

If you choose to work with the function `writePNG()`, then also write the needed LaTeXcode (`\includegraphics{}`) such that your picture shows in the `.pdf` file that we can create by knitting your `.Rmd` file.

3 Floyd-Steinberg dithering algorithm

Clearly, many details are lost when representing the `xmas.png` picture in the colors from the 3-bit RGB palette using the naive approach of nearest colors. Even when using 4096 colors from the 12-bit RGB palette, you could argue that some of the color transitions are (still) too roughly represented.

When we apply the Floyd-Steinberg dithering algorithm, while using the same number of colors as in the naive approach, the `xmas.png` picture will be much better represented due to a property which is called "error diffusion".

In this task we will ask you to code a specific version of the Floyd-Steinberg dithering algorithm. For the pseudo code of the algorithm, check out

https://en.wikipedia.org/wiki/Floyd-Steinberg_dithering

For a detailed lecture on the algorithm, check out

https://www.youtube.com/watch?v=0L2n8Tg2FwI.

Towards a Loss Function for the Floyd-Steinberg algorithm

It is not immediately clear what kind of loss function the Floyd-Steinberg algorithm algorithm minimizes. Moreover, we wish to use our own loss function (which determines how to code `find_closest_palette_color()`). In this section we will provide our own loss function after introducing some notation.

Let $\mathcal{R}$ be the set of the row indicators of a picture, but excluding the first row:

$$\mathcal{R} = \{2, ..., N_R\},$$

where N_R is the last row (indicating the total number of rows as well). Let $\mathcal{C}$ be the set of column indicators, but excluding the first and the last column:

$$\mathcal{C} = \{2, ..., N_C\},$$

where N_C is the indicator for the last column (indicating the total number of columns as well). Then, a diffused error $\mathbf{e}_{ij}$ in the Floyd-Steinberg can be (recursively) defined as

$$\mathbf{e}_{ij} = \mathbf{x}_{ij} + \mathbf{1}_{\mathcal{R}}(i)\left(\mathbf{1}_{\mathcal{C}}(j)\frac{1}{16}\mathbf{e}_{i-1,j-1} + \frac{5}{16}\mathbf{e}_{i-1,j} + \mathbf{1}_{\mathcal{C}}(j+1)\frac{3}{16}\mathbf{e}_{i-1,j+1}\right) + \mathbf{1}_{\mathcal{C}}(j)\frac{7}{16}\mathbf{e}_{i,j-1} - \mathbf{c}_k,$$

where the indicator function $\mathbf{1}_R$ is defined as

$$\mathbf{1}_{\mathcal{R}}(i) := \begin{cases} 1 & \text{if } i \in \mathcal{R}, \\ 0 & \text{if } i \notin \mathcal{R}, \end{cases}$$

and similarly, $\mathbf{1}_C$ is then

$$\mathbf{1}_{\mathcal{C}}(j) := \begin{cases} 1 & \text{if } j \in \mathcal{C}, \\ 0 & \text{if } j \notin \mathcal{C}. \end{cases}$$

In our version of the the Floyd-Steinberg algorithm we wish to minimize the squared length of the diffused error for each pixel $\mathbf{x}_{ij}$. Let$\widehat{\mathbf{c}}_{k_{ij}}$ be the corresponding color to pixel $\mathbf{x}_{ij}$ that minimizes the squared ℓ_2 norm of the diffused error, defined as

$$\widehat{\mathbf{c}}_{k_{ij}} = \underset{\mathbf{c}_k}{\operatorname{argmin}} \left(\left|\left| \mathbf{x}_{ij} + \mathbf{1}_{\mathcal{R}}(i) \left(\mathbf{1}_{\mathcal{C}}(j) \frac{1}{16} \mathbf{e}_{i-1,j-1} + \frac{5}{16} \mathbf{e}_{i-1,j} + \mathbf{1}_{\mathcal{C}}(j+1) \frac{3}{16} \mathbf{e}_{i-1,j+1} \right) + \mathbf{1}_{\mathcal{C}}(j) \frac{7}{16} \mathbf{e}_{i,j-1} - \mathbf{c}_k \right|\right|_2^2 \right),$$

where $||\mathbf{e}||_2^2$ denotes the squared ℓ_2 norm of the vector $\mathbf{e}$.

Then, by replacing each $\mathbf{c}_k$ with $\widehat{\mathbf{c}}_{k_{ij}}$ for the difussed error of pixel $\mathbf{x}_{ij}$, our version of the Floyd-Steinberg algorithm minimizes the loss function

$$Q_K(\mathcal{X}, \mathbf{C}_{K\times 3}) = \sum_i \sum_j ||\mathbf{e}_{ij}||_2^2\,,$$

where $\mathcal{X}$ denotes the set of all pixels, i.e.

$$\mathcal{X} = \{\mathbf{x}_{ij}\}_{\{i,\,j\}},$$

and where $\mathbf{C}$ is a matrix of size $K \times 3$ in which each row represents one of $1, \ldots, K$ transposed color vectors $\mathbf{c}_k^T$.

3.1 Programming your own Floyd-Steinberg algorithm (15: 55 - 45)

Use the pseudo-code of the Floyd-Steinberg algorithm on Wikipedia and the notation of the diffused erros to create your own function that can perform on an `array` that consists of the RGB intensities like `xmas` from the picture `xmas.png`. The function has two input arguments: an RGB array the represents the pictur, and a matrix (or data.frame) that represent a RGB palette, with its colors in the row, and the red, green, blue intensities in the columns (like `RGB_03bit$dat` from the model answer variables).

The output of your function should have three components:
1. an array of the colors with which the pixels should get replaced;
2. an array that holds your estimates of the diffused errors for each pixel;
3. the value of your loss function.

When you apply your Floyd-Steinberg dithering function on the `xmas` array to reduce the number of colors to those present in the 3-bit RGB palette, show that your answer is (almost) equal to `dither_03bit` from the model answer variables.

Hint: do not forget to dither the last row and the last column!

3.2 Plotting the Loss for 3-bit, 9-bit, and 15-bit (2.5: 57.5 - 42.5)

Plot the natural logarithm of the value of the loss-function of your Floyd-Steinberg algorithm for the 3-bit, 9-bit, and 15-bit RGB color palettes. Explain in at most 4 sentences why the value of the loss function is decreasing.

Hint: If you did not succeed in 3.1, then use `dither_03bit`, `dither_09bit`, and `dither_15bit` from the Model answer variables.

4 Statistical Computing on the Floyd-Steinberg algorithm

Suppose 'costs' (e.g. time, data storage, printing) can be saved by choosing fewer colors when diterhing pictures. Then, which of the RGB palettes would provide the optimal balance between the costs and the best representation of the picture?

Eyeballing the `xmas_dither*bit.png` dithered pictures in the `0_img` folder, your instructors would choose either `xmas_dither09bit.png` (512 colors) or the `xmas_dither12bit.png` (4096 colors) picture..., but that is not a very objective argument to use.

Since we have no idea about the statistical processes behind our data and any of theory the Floyd-Steinberg algorithm's working, we need to be creative ourselves to come up with a more objective strategy for choosing the number of colors.

What we can assume is that our data set (= picture) consists of clustered colors. For example, for the pixels to be able to represent the red christmas ball, many red pixels are clustered together. This is a property we could exploit....

4.1 Generate a Permutation of the Picture (5: 65 - 35)

Suppose we assume our picture to be just a random collection of exactly the same number of (permuted) pixels. Let us refer to this assumption as the null hypothesis (H_0). Under this assumption, our `xmas.png` picture is just as likely as any of the other $N_P!$ permuted versions of our picture, where $N_P = N_C * N_R$, the total number of pixels.

Create a function that produces a permuted replicate of the `xmas`variable, denoted by $\mathcal{X}^b$. Each pixel $\mathtt{x}_{ij}^b$ in $\mathcal{X}^b$ is a realization of a permutation over i and j of the pixels in $\mathcal{X}$. Thus, the color intensity values in the picture remain the same. In other words, permute the indices in the first and second dimension of the array `xmas` such that the values of each pixel in the third dimension remain the same.

An example of such a picture would be:

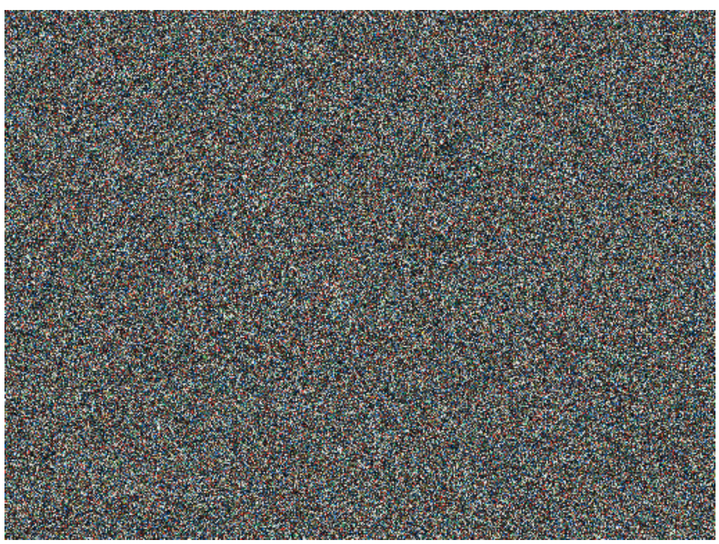

Figure 3: The xmas picture where its pixels are permuted

4.2 Log Loss of Floyd-Steinberg under H_0 (15: 80 - 20)

When we apply the Floyd-Steinberg diterhing algorithm on a permuted picture under H_0 for a certain RGB palette, we can obtain an idea of the expected value of the loss function of the Floyd-Steinberg algorithm, denoted by $E_{H_0}[Q_K]$.

For a certain integer B and a the particular number of colors K from a certain RGB palette we would estimate the expected values as

$$\widehat{E_{H_0}[Q_K]} = B^{-1} \sum_{b=1}^{B} Q_K(\mathcal{X}^b, \mathbf{C}_{K\times 3}).$$

Now, take a look at the variable `xmas_replicates_logloss` from the model answer variables. This specific variable of class `list` consisting of $B = 100$ components, where each component is a list of 5 components, representing the 3-bit, 6-bit, ... 15-bit RGB palettes. Each of these RGB palletes is again a list consisting of the a component indication the number of bits of the palette, and indication the logloss it produced on the permutation replicate $\mathcal{X}^b$ and the number of colors $\mathbf{C}_{K\times 3}$, denoted by $\log(Q(\mathcal{X}, \mathbf{C}_{K\times 3})$.

Write your own function that outputs a variable like `xmas_replicates_logloss`. Your function takes as input arguments, an array of an image (like `xmas`), a vector of values for `K_values` that can only belong to the set `2^(3 * (1:5))`, and `B` the number of replicates that need to be created.

There are two extra restrictions:
1. In your function you'll need to use the function you have created in **4.1**. If you did not succeed, then use the non-existing function `PermutePictureArray()` and just assume it is present in your global environment; 2. Use twice the function `parallel::mclapply()` to implicitly loop over each of the B permutation replicates of the image array and over each K in the vector `K_values`. Set the `mc.cores` arguments of these two functions correctly equal to `B` and `length(K_values)`.

Warning: to avoid heavy computing, there is no need to run the function. If you wish to test whether it is working fine, then only test for yourself whether it works for `B = 2`, and `K_values = c(64, 4096)`.

4.4 Visualize the Log Loss under H_0 and for our data. (10: 90 - 10)

When we have an estimate for the expected value of the loss function for each RGB palette under H_0 for $K \in \{2^3, 2^6, 2^9, 2^{12}, 2^{15}\}$ colors, we can compare these expected values with that of the value of the loss function on our real (observed) picture (`xmas.png`), Q_K for each K.

While the value of the loss becomes smaller for larger K, this may need not be the case for the difference between the (expected) loss under the null hypothesis, and that of our observed data set. Compared to noise, for some K the Floyd-Steinberg algorithm is much better at representing the picture with a clustering structure of the colors, explaining a larger difference in the loss function.

Let the difference between the expectation of the natural logarithm of the value of the loss function under H_0, on the one hand, and the observed value of the natural logarithm of the loss for our observed data, on the other hand, be denoted as the Gap statistic:

$$\mathrm{Gap}_K(\mathcal{X}, \mathbf{C}_{K\times 3})) = B^{-1} \left(\sum_{b=1}^{B} \log\left(Q_K(\mathcal{X}^b, \mathbf{C}_{K\times 3})\right) \right) - \log\left(Q_K(\mathcal{X}, \mathbf{C}_{K\times 3})\right).$$

Then, the optimal number of colors $\widehat{K}$ that we would like to choose is

$$\widehat{K} = \underset{K}{\operatorname{argmax}} \left(\mathrm{Gap}_K\right).$$

Based on this Gap statistic procedure, the optimal number of colors for the regular RGB palette to dither the `xmas.png` picture is $\widehat{K} = 2^9 = 512$.

This conclusion can also be obtained from Figure 4, where we have visualized the gaps, the expected loss, and the ± 2 standard error bars for the different values of K. Here the standard error of Gap is defined as

$$\text{SE}_{\text{Gap}_K} = \sqrt{1 + \frac{1}{B}} SD\left(Q_K(\mathcal{X}^b, \mathbf{C}_{K\times 3})\right),$$

where $SD()$ denotes the standard deviation.

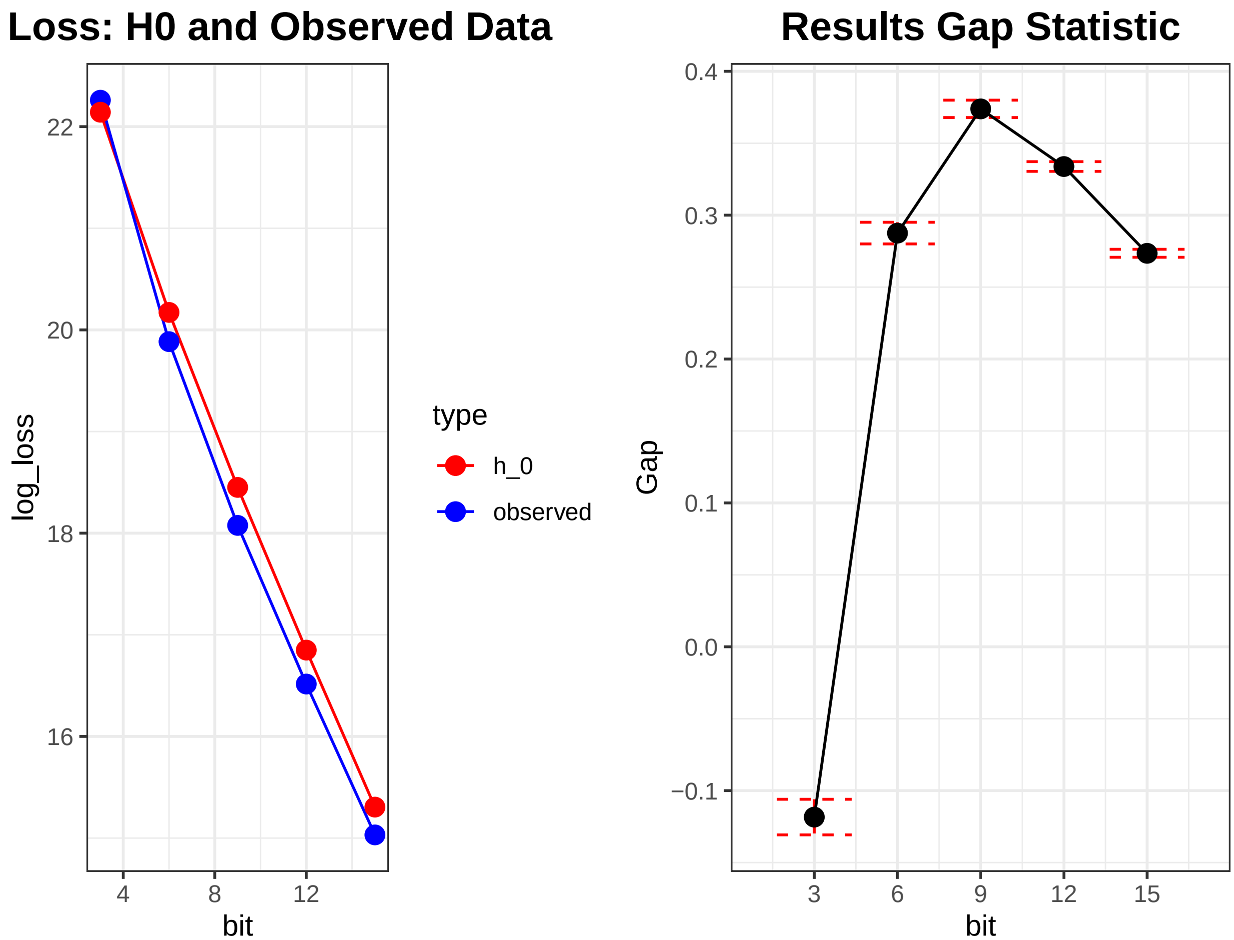

Figure 4: Results from the Gap statistic procedure

Replicate Figure 4 based on the `xmas_replicates_logloss` variable, and use functions from the packages `ggplot2` and `gridExtra`.

4.6 Alternatives (10: 100 - 0)

Could you explain why the GAP statistic is small for too small K, and also small for too large K? For your explanation, relate to the bias variance trade-off.

5. Bonus: Something new, the package `Rcpp` (15 points)

Check out the R package `Rcpp`. Use the functionality of this package to create your own Floyd-Steinberg algorithm (task **3.1**) that is much much faster.