EVR-5086 Assignments

Adyan Rios

2025-10-07

Table of contents

Introduction

Although the EVR-5086 class is being taught using Python, my prior experience is with R. I am also fond of sharing my work on GitHub. I have learned how GitHub pages combined with Quarto and R Studio are an extraordinary resource for developing and maintaining lab notebooks. To get better at using these tools (and the reproducibility and accessibility of my future research) I have created a html quarto book and pdf to show my work associated with the course assignments.

Set Up

I started by creating a GitHub account (username: arios101-fiu). Then, I created a GitHub repository with a gitignore and readme.md. Initially, the repository was called EVR-5086-Assignement1, but I updated it to (EVR-5086-Assignments). I cloned the repository into R Studio, thereby creating a R project. I copied in a _quarto.yml and index.qmd files from another project. I updated the files, rendered, committed, and pushed. Next, I turned on GitHub pages and updated the URLs in the yml and repository.

1 Assignment 1 – Calculus Review

EVR-5086 Fall 2025

Assignment 1 - Calculus Review

1.1 Plot the polynomial

Below are the steps I took to complete the first part of EVR-5086 Assignment 1.

In doing this exercise in R, I started by loading the R libraries I will use in this chapter. I used {ggplot2} for plotting, and {tidyr} and {dplyr} for data wrangling.

```r
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(ggplot2, tidyr, dplyr)
```

Next, I defined the variables and created the vectors I will need for the plot.

```r
# Define variables
a <- 1
n <- 1
b <- 1
p <- 2
c <- 1
q <- 3

# Create x vector from -1 to 1
x <- seq(from = -1, to = 1, by = 0.1)

# Calculate a value of y for each value of x
y <- (a * (x^n)) + (b * (x^p)) + (c * (x^q))
```

```r
# Calculate the analytical derivatives for each value of x
dy_dx <- (a * n * (x^(n - 1))) + (b * p * (x^(p - 1))) + (c * q * (x^(q - 1)))

# Calculate the numerical derivatives between each value of x
deltay <- diff(y)
deltax <- diff(x)
deltay_deltax <- deltay / deltax

# For plotting purposes, derive the midpoint across the original values of x
deltax_vec <- x[-length(x)] + deltax / 2
```

My next goal was to unite all of the vectors into a long data format. I did this by creating a data frame, then pivoting the data to only have the values that will be plotted on the x and y axis, as well as a label. Later, I will use my “linetype” label to define line types as well as the colors and shapes in my plot.

```r
# Build data frames and rename variables for plot
plot_prep <- data.frame(x, y, dy_dx) |>
  dplyr::rename(Polynomial = y, "Analytical derivative" = dy_dx)

# Wrangle to long data format and bind in numerical derivative
plot_tidy <- plot_prep |>
  tidyr::pivot_longer(!x, names_to = "linetype", values_to = "y") |>
  dplyr::bind_rows(data.frame(x = deltax_vec, y = deltay_deltax,
                              linetype = "Numerical derivative"))
```

Lastly, I create the plot and reflect on the observations and limitations of the numerical derivative.

Figure 1.1 shows that the numerical derivative, shown as red open circles, is very similar to the analytical derivative, shown as a blue solid line. The good match we see relates to the scale over which we calculated the numerical derivative compared to the scale of the rate of change in the polynomial. When calculating the numerical derivative, we can get the average rate of change between two points.

Note that for the analytical derivative we are only providing the plot with information associated with x values ranging -1 to 1, in steps of 0.1. Meanwhile, the numerical derivative is plotted at the midpoints of our original segments, with x values ranging from -0.95 to 0.95. Including the numerical derivatives in the appropriate position relative to the curved lines plotted between our analytical derivatives results in the overlay of the points and the line.

If the numerical derivative had a significantly lower resolution (e.g. just -1 and 1), it would not match well, and would be just one point, at x = 0, above the “U” shaped line representing the analytical derivative. Although such a wide spacing is extreme to consider, it helps to emphasize that grid spacing and location plotted are important considerations when working with numerical derivatives.

```r
# Plot the analytically derivative as a solid line
# and the numerical derivative as open symbols
polynomial_plot <- ggplot(data = plot_tidy,
                          aes(x = x, y = y, color = linetype)) +
  geom_point(
    data = dplyr::filter(plot_tidy, linetype == "Numerical derivative"),
    shape = 21, stroke = 1.25
  ) +
  geom_line(
    data = dplyr::filter(plot_tidy, linetype != "Numerical derivative")
  ) +
  theme(legend.title = element_blank()) +
  scale_color_manual(values = c(4, 2, 1)) +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  labs(
    title = "Plot of polynomial with analytical and numerical derivatives"
  )

polynomial_plot
```

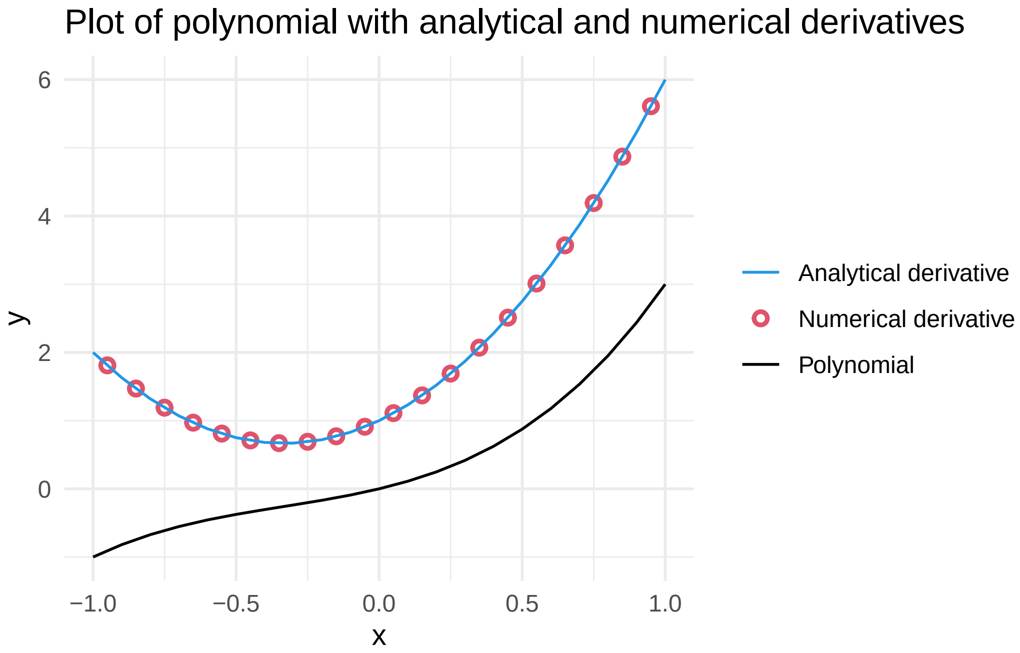


Figure 1.1: Polynomial defined by values provided in EVR-5086 Assignment 1 \n (black line), along with analytical derivative (blue line) and numerical \n derivative (red open circle).

1.2 Solve the 2-D Laplace in Excel

I created a 28 by 28 grid of the 2-D Laplace Equation. I included three internal "boundary values"; one high value of 4 and two low values of -2 and -3. The two low values were near each other compared to their respective distances to the high value. I allowed excel to iteratively calculate for 10,000 iterations with a minimum change of 0.0001. I saved the file as a CSV file after including explicit zeros surrounding the formulas. The dimensions of my data were 30 by 30. I rounded to four significant digits to see if stagnation areas would be more evident by avoiding calculating of extremely small differences.

1.2.1 Read in and plot contours using Python

Start by turning on Python in R. This requires the package {reticulate} in R which embeds a Python session within the R session. The function py_require() is also used to declare Python packages that will be used in the R session.

```r
# Check if libraries are installed; install if not.
if (!require("reticulate")) install.packages("reticulate")
```

```
Loading required package: reticulate
```

```r
# Load reticulate
library(reticulate)

# Ensures matplotlib package is available in the current session
if (!py_module_available("matplotlib")) py_require(c("matplotlib"))
```

The rest of the assignment is run in Python. First, I import the numpy and matplotlib.pyplot packages and read in the CSV file that I had created in excel. To prepare the data for plotting, I create two arrays using np.linspace() and combine them into a 30 x 30 grid of x and y coordinates using np.meshgrid(). Finally, the partial derivatives for h with respect to x and y are calculated using np.gradient().

```python
# Import packages
import numpy as np
import matplotlib.pyplot as plt

# Load csv file from excel
h = np.loadtxt('tripole.csv',delimiter=',')

# Create a grid of x and y coordinates
x_vec = np.linspace(-1.5, 1.4, 30)
y_vec = np.linspace(-1.5, 1.4, 30)
X, Y = np.meshgrid(x_vec, y_vec)

# Calculate gradient/partial derivatives
[dhdy, dhdx] = np.gradient(h, y_vec, x_vec)

# Round to 4 significant figures
dhdy4 = np.round(dhdy, 4)
dhdx4 = np.round(dhdx, 4)
```

Figure 1.2 recreates the surface plot that perviously had been explored in Excel. The x- and y-axis range from -1.5 to 1.4, while the h-axis ranges from -3 to 4. Figure 1.3 shows a contour map with flow vectors. Finally, Figure 1.4 provides a similar plot to Figure 1.3, but with streamlines instead of arrows.

Reviewing the contours, flow vectors, and streamlines I did not identify stagnation points. When I selected the two low points, I was expecting a stagnation “saddle effect”. However, the proximity and the similarity in values I used did not result in a stagnation area. Interestingly, the majority of the surface plotted consisted of extensive areas of very low gradients. Figure 1.4 shows that the streams would run beyond the edges across approximately 60% of the plotted grid.

The following three Python code chunks created Figure 1.2, Figure 1.3 and Figure 1.4, respectively.

1.2.2 Surface plot

```
fig = plt.figure(figsize = [4, 4], dpi = 300) #Create empty figure
ax = plt.axes(projection = '3d') # Create plot region
ax.set_title(" " * 20 + 'Surface plot'+ " " * 20)  # Include plot title and pad space for h-a
ax.set_xlabel('x-axis') # Include x-axis label
ax.set_ylabel('y-axis')  # Include y-axis label
ax.set_zlabel('h-axis', rotation = 90) # Include vertical h-axis label
surf = ax.plot_surface(X,Y,h) # Plot surface
plt.show() # Render plot
```

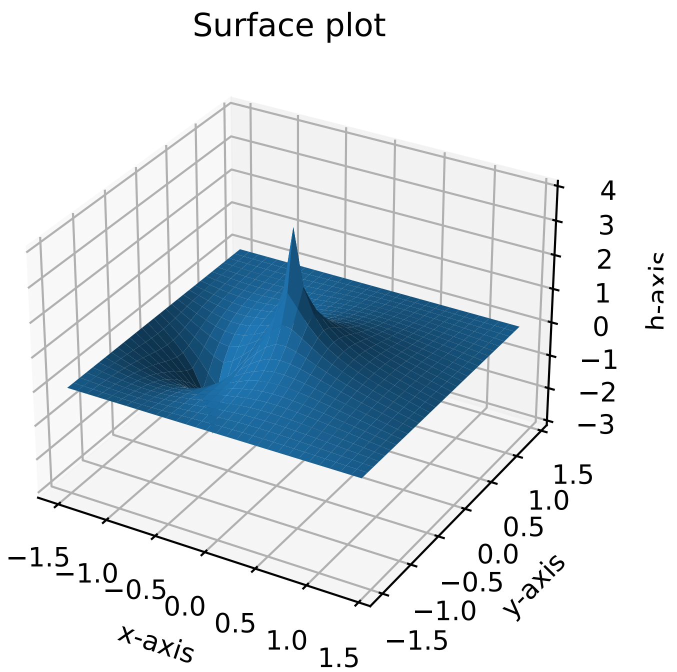


Figure 1.2: Plot of vector arrows using Python. The vectors indicate strength and direction of the negative gradient. The vectors are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.

```
plt.close('all') # Prevent accidental overplotting onto an old figure
```

1.2.3 Plot contour map and flow vectors

```
# Plot contour map and flow vectors
plt.contourf(X, Y, h) # Draw contours for h on grid coordinates (x,Y)
cbar = plt.colorbar() # Add colorbar
cbar.set_label('Ground water potential surface (h)') # Include lable on colorbar
plt.axis('equal'); # Force equal scaling on x and y
```

```
(np.float64(-1.5), np.float64(1.4), np.float64(-1.5), np.float64(1.4))
```

```
plt.title('Contour map and flow vectors') # Include plot title
plt.xlabel('X-axis') # Include x-axis label
plt.ylabel('Y-axis') # Include y-axis label
qplt = plt.quiver(X, Y, -dhdx4, -dhdy4, scale = 360) # Draw vector arrows; note: large scales
plt.show() # Render plot
```

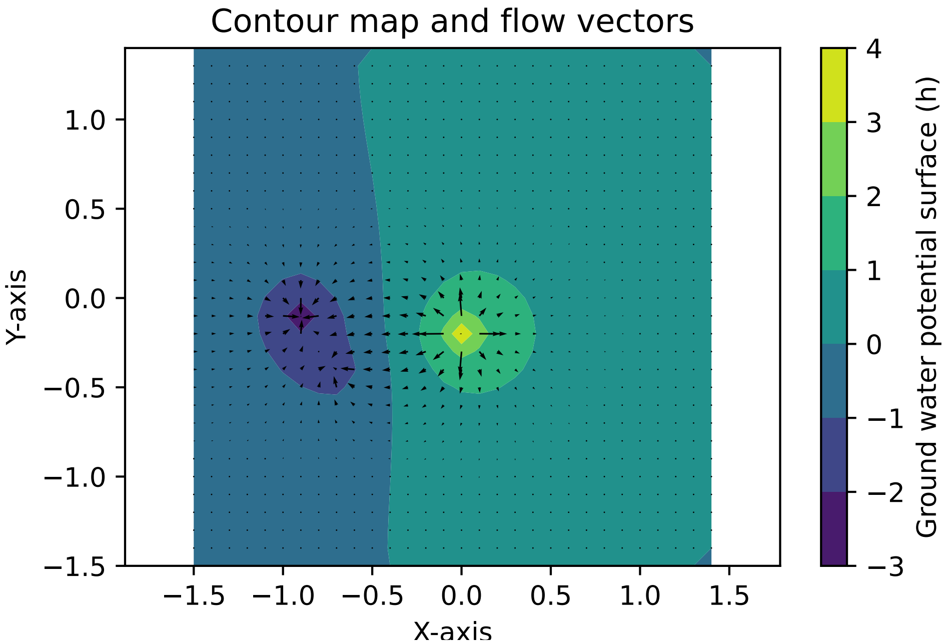


Figure 1.3: Plot of vector arrows using Python. The vectors indicate strength and direction of the negative gradient. The vectors are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.

```
plt.close('all') # Prevent accidental overplotting onto an old figure
```

1.3 Plot streamlines instead of arrows in Section 1.2.3

```
# Plot contour map streamlines
plt.contourf(X, Y, h) # Draw contours for h on grid coordinates (x,Y)
cbar = plt.colorbar() # Add colorbar
```

```
cbar.set_label('Ground water potential surface (h)') # Include lable on colorbar
plt.axis('equal'); # Force equal scaling on x and y
```

```
(np.float64(-1.5), np.float64(1.4), np.float64(-1.5), np.float64(1.4))
```

```
plt.title('Contour map and streamlines') # Include plot title
plt.xlabel('X-axis') # Include x-axis label
plt.ylabel('Y-axis') # Include y-axis label
plt.streamplot(X, Y, -dhdx4, -dhdy4) # Draw streamlines
```

```
<matplotlib.streamplot.StreamplotSet object at 0x0000022790B556D0>
```

```
plt.show() # Render plot
```

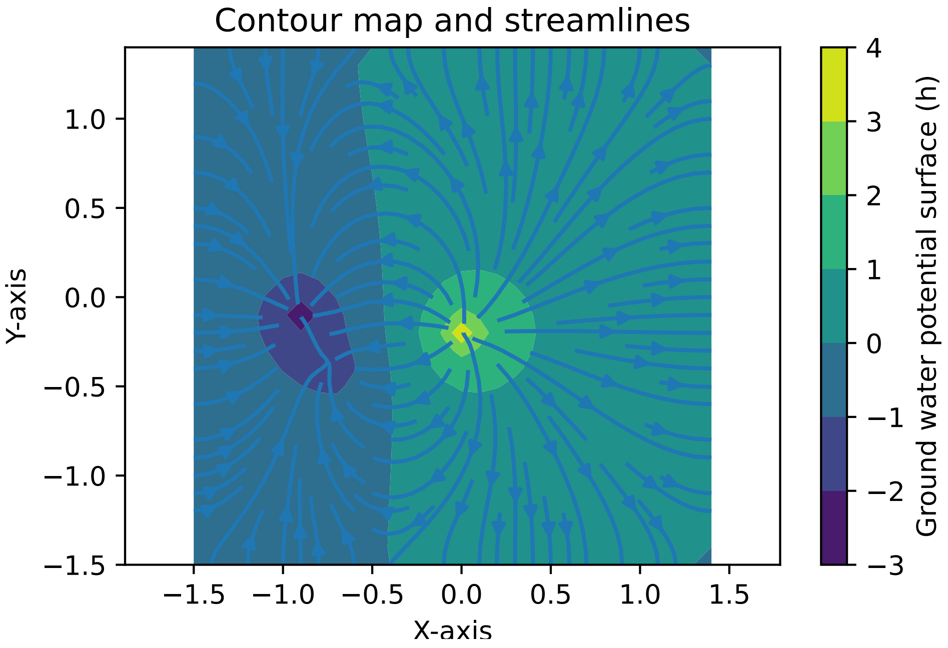


Figure 1.4: Plot of streamlines using Python. The streamlines are displayed over the contours of a tri-pole solution with a high value of 4 and lows of -2 and -3. The two low values were near each other compared to their respective distances to the high value.

```
plt.close('all') # Prevent accidental overplotting onto an old figure
```

1.4 Links to Colab and GitHub

Draft code on Colab

Quarto book chapter on GitHub

2 Assignment 2 - Reading On-line Data and Visualizing Hurricane Tracks

EVR-5086 Fall 2025

Assignment 2 - Reading On-line Data and Visualizing Hurricane Tracks

2.1 Data Retrieval and Parsing

The purpose of this code is to access and format the HURDAT2 dataset into an analysis-ready format. Since I expect to explore these data further, I did not want to perform wrangling and formatting only on filtered subsets. Instead, I extracted the header information for each storm record, expanded it, and attached it to each corresponding data line.

There are many possible approaches to this task, including base R methods. I chose to use tidyr and dplyr because their piping syntax allows me to write modular code without overwriting objects or cluttering the global environment. While I could have created a single long script, I prefer a modular workflow where each section has a clear intention. For example, this part of the workflow focuses only on data retrieval and formatting.

At the end of Section 2.1, I save the resulting data frame as an .RDS file. I prefer RDS over RData because RDS requires explicit object naming when loaded, which supports better coding practices and clearer, more reproducible code.

The steps of the code are annotated in the code chunks, and the first chunk defines and loads all the required libraries used in Section 2.1. One limitation of my approach is that effective use or contribution requires familiarity with GitHub, RStudio, and Quarto.

```r
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(here, curl, tidyr, dplyr, lubridate)
```

```r
# Specify the source and output file name and location
url <- "https://www.nhc.noaa.gov/data/hurdat/hurdat2-1851-2024-040425.txt"
destfile <- here("assignment2","hurdat.txt")

# Download and save the dataset
curl_download(url = url, destfile = destfile)
```

```r
# Read in data and differentiate between headers and data lines
lines <- readLines("hurdat.txt")                     # Read text file

# Prep headers
storm_headers <- lines[grepl("^AL", lines)]          # Keep only storm headers
header_parts <- strsplit(storm_headers, ",")         # Split based on ","
header_matrix <- do.call(rbind, header_parts)        # Convert to matrix
header_df <- as.data.frame(header_matrix)            # Convert to data frame
colnames(header_df) <- c("storm_id", "name", "rows") # Name columns

# Repeat each header based on the 'rows' column in tidyr
header_expand <- header_df |>
  mutate(rows = as.numeric(rows)) |>
  uncount(rows)

# Prep data
storm_data <- lines[!grepl("^AL", lines)] # Keep only data
hurdat_parts <- strsplit(storm_data, ",")   # Split based on ","
hurdat_matrix <- do.call(rbind, hurdat_parts) # Convert to matrix
hurdat_df <- as.data.frame(hurdat_matrix)     # Convert to to data frame

hurdat_fields <- c("yyyymmdd", "hhmm", "record", "status",
                 "lat_hemi", "lon_hemi", "wind", "pressure",
                 "ne34", "se34", "sw34", "nw34",
                 "ne50", "se50", "sw50", "nw50",
                 "ne64", "se64", "sw64", "nw64", "radius")

colnames(hurdat_df) <- hurdat_fields # Name columns

# Build analysis ready data set
hurdat_ar <- header_expand |>
  bind_cols(hurdat_df) |> # Glue together storm id and name with data
  mutate(
    yyyymmdd = ymd(yyyymmdd),                   # Tell R this is a date
    hhmm = strptime(hhmm, format = "%H%M"),     # Tell R this is a time
    hhmm = format(hhmm, "%H:%M"),
    lat = as.numeric(substr(lat_hemi, 1, nchar(lat_hemi)-1)), # Remove "S"
    lon = as.numeric(substr(lon_hemi, 1, nchar(lon_hemi)-1)), # Remove "W"
    lat_hemi = substr(lat_hemi, nchar(lat_hemi), nchar(lat_hemi)),
    lon_hemi = substr(lon_hemi, nchar(lon_hemi), nchar(lon_hemi)),
    lat = if_else(lat_hemi == "S", -lat, lat), # Make lat negative if "S"
    lon = if_else(lon_hemi == "W", -lon, lon)  # Make lon negative if "W"
  ) |>
  mutate_at(c(9:25), as.numeric) |>                  # Make data numeric
  mutate(across(where(is.numeric), ~na_if(., -999))) # Replace NAs

# Save formatted data to read into next quarto environment
saveRDS(hurdat_ar, file = here("assignment2", "hurdat.rds"))
```

2.2 Data Visualization

The data visualization has several exciting features. Because the data set was already formatted into an analysis-ready structure, this part of the assignment focuses only on visualizing a given storm ID. It begins with defining and loading the packages used later in the code. The leaflet and webshot2 packages were new to me. Leaflet allows me to create an interactive map, similar to folium in Python.

Since I am rendering my report to both HTML and PDF, I needed different approaches for each format. In HTML, I was able to embed the leaflet map directly as an htmlwidget. For PDF output, I learned to use conditional content so the widget only displays in HTML, and a static snapshot (generated with webshot2) is included in the PDF. For both versions, I provided a figure caption and alt text to improve clarity and accessibility.

To add more complexity and depth to the track visualization, I incorporated a color scale representing wind speed. This makes the map more informative and highlights storm intensity changes along its path. I think interactive widgets can serve as a useful precursor to fully developed applications for data visualization. I am excited about the continued advancements in interactive figures which allow non-coders to explore and interact with data in more meaningful ways. Although I did not implement dynamic selection in the HTML rendering of this assignment, I looked into some of the latest developments in Quarto Dashboards. For now, I set up an optional user input similar to what we did in Python. When the R code is run interactively, the user is prompted to provide a storm name; otherwise, a default storm ID is used to ensure the code still runs smoothly during rendering.

```
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(here, stringr, leaflet, webshot2, dplyr)
```

```r
default_name = "AL092021"

# If interactive (R console / RStudio), ask the user
if (interactive()) {
  storm_id <- readline(
    prompt = paste0("Enter a storm ID using ALnnyyy format [default = ",   default_name, "]:
    )
  if (storm_id == "") storm_id <- default_name
} else {
  # If running non-interactively (e.g., knitting to PDF/HTML), use default
  storm_id <- default_name
}
```

```r
# Read in data
hurdat_ar <- readRDS(here("assignment2", "hurdat.Rds"))

# Create a reusable function
track_storm <- function(dat, storm_id, zoom = 4,
                            init_location = c(20, -50)) {
  # Filter and order the points for the selected storm
  storm <- dat |>
    filter(storm_id == !!storm_id, !is.na(lat), !is.na(lon)) |>
    mutate(status = str_trim(status)) |>
    arrange(yyyymmdd, hhmm)

  if (nrow(storm) == 0) stop("No points found for this storm_id.")

  # Build popup: date + time + status (e.g., "1851-06-25 00:00 - HU")
  popup_txt <- paste0(
    format(storm$yyyymmdd, "%Y-%m-%d"), " ", storm$hhmm,
    " - ", storm$status
  )

  # Definecolor range
  pal <- colorNumeric(
    palette = "YlOrRd",   # yelloe = weak winds, red = strong winds
    domain = storm$wind   # The range of wind speeds
  )

  # Create map
  m <- leaflet(storm) |>
  addTiles() |>
  addPolylines(lng = ~lon, lat = ~lat, color = "blue",
               weight = 2.5, opacity = 1) |>
  addCircleMarkers(
    lng = ~lon,
    lat = ~lat,
    color = ~pal(wind),       # marker color by wind
    radius = 5,               # size of marker
    stroke = FALSE,
    fillOpacity = 0.8,
    popup = ~paste0(format(yyyymmdd, "%Y-%m-%d"), " ", hhmm,
                    "<br>Wind: ", wind, " kt",
                    "<br>Status: ", status)
  ) |>
  addLegend(
    "bottomright",
    pal = pal,
    values = ~wind,
    title = "Wind (kt)",
    opacity = 1
  )

  file_html <- here("assignment2", paste0(storm_id, "_map.html"))
  htmlwidgets::saveWidget(m, file_html, selfcontained = TRUE)
  m
}

leaflet_png <- function(m) {
  file_png = here("assignment2", "hurricane_tracks_map.png")
  html_tmp <- here("assignment2", "hurricane_tracks_map_tmp.html")
  htmlwidgets::saveWidget(m, html_tmp, selfcontained = TRUE)
  webshot2::webshot(html_tmp, file = file_png, vwidth = 1400,
                    vheight = 900, zoom = 1)
  return(file_png)
}
```

```
m <- track_storm(hurdat_ar, storm_id = storm_id)
png_file <- leaflet_png(m)
knitr::include_graphics(png_file)
```

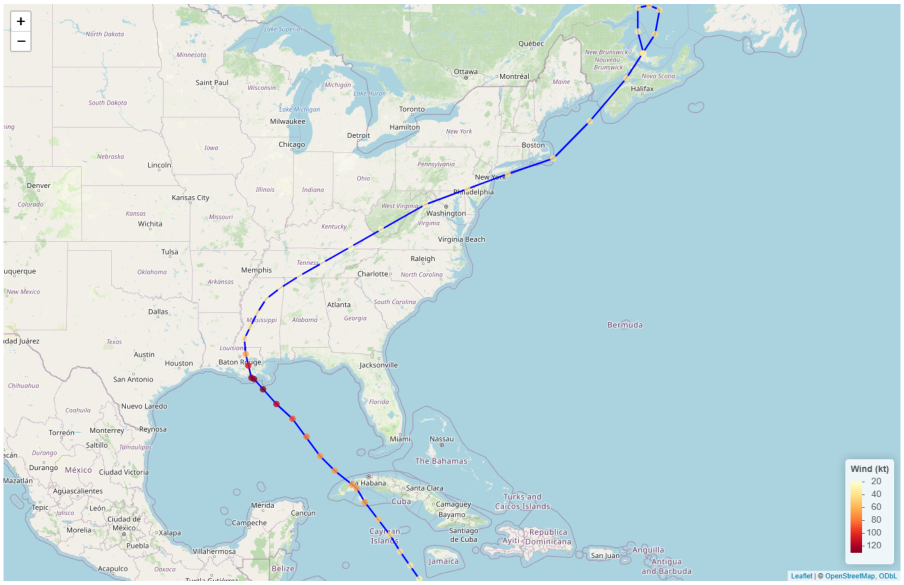


Figure 2.1: Static storm track AL092021 (PDF).

2.3 Links to Colab and GitHub

Assignment 2 Google Colab

Quarto book chapter on GitHub

3 Assignment 3 - Normal distributions and the Galton board

EVR-5086 Fall 2025

Assignment 3 - Normal distributions and the Galton board

For this assignment follows the readings and exercises in “Risk Analysis in the Earth Sciences: A Lab Manual with Exercises in R” Version 3.

3.1 Exercise 1

In this exercise I modified the provided lab0_sample.R to produce a histogram based on 10^4 samples from a normal distribution with mean 5 and standard deviation 2 Figure 3.1. I then produced two histograms for the standard normal distribution which has a mean 0 and standard deviation 1. The first was based on 10^4 samples (Figure 3.2) and the second one included only 10 samples (Figure 3.3). In addition to the modifications outlined by the exercise, I included a legend on each plot, and I dynamically read sample size into the y-axis label. I organized my code using variable names that differed by the subscript (1-3) associated with each of the three respective histograms. A main difference between Figure 3.1 and the other two histograms is how the distribution is centered around the mean of 5 and has a wider range of values compared to Figure 3.2. These differences are driven by the differences in the respective means and standard deviations. Since the first two histograms have a large number of samples being drawn from the normal distribution function rnorm(), they resulting plots show the expected bell curve shape. However, since the third histogram (Figure 3.3) is based on only 10 samples, we do not see the definition of the bell curve at all. When only using 10 samples, the standard normal distribution results in a smaller range of values than it did with 10^4 samples. When the number of samples is large, there are more opportunities for values to be sampled. I explore the related probability density in the next part of this exercise.

In the final part of this first exercise, I plotted the formula for the normal distribution provided in EVR-5086 Assignment 3 (Figure 3.4). I noticed a small difference between the equation in the assignment and the one on Wikipedia, which I included in the same plot as a second curve in blue. Overall, exploring these formulas helped me better understand how normal distribution's characteristic symmetric bell curve shape is driven in its formula by $e^{-x}2$. As shown in the examples in Figure 3.4 larger the value in the exponential term of the natural log, the wider the distribution. In this exercise I do not need to use the full formula that is on Wikipedia because, when mean is 0 and the standard deviation is 1, some constant multipliers simplify to 1. In the R code, I used text() and expression() to include color-coded mathematical expressions in the top-left corner of the plot.

3.1.1 Modify code from lab0_sample.R

```r
# Modified code from lab0_sample.R

# Set values
num_1 <- 10^4   # number of random draws
mu_1 <- 5       # mean of normal distribution to draw from
sigma_1 <- 2    # standard deviation of normal distribution

# Sample randomly from a normal distribution
x_1 <- rnorm(n = num_1, mean = mu_1, sd = sigma_1)

# Plot the results as a histogram
hist_1 <- hist(
  x_1,                 # Vector for the histogram
  main = "Adyan Rios", # Set title to my name
  xlab = "Variable",
  ylab = paste0("Frequency (n = ", sprintf("%.2e", num_1), ")"),
  xaxt = "n"           # Turn off x-axis values
)

# Add custom x-axis ticks
axis(side = 1, at = hist_1$breaks)

# Indicate mean
abline(v = mu_1, lwd = 2, col = "red")

# Indicate mean ± SD
abline(v = c(mu_1 + sigma_1, mu_1 - sigma_1),
       lwd = c(2), lty = 2, col = "blue")
```

```
# Add a legend
legend("topleft", legend = c("Mean", "Mean ± SD"), lwd = 2,
       col = c("red", "blue"), lty = c(1, 2), bty = "n")
```

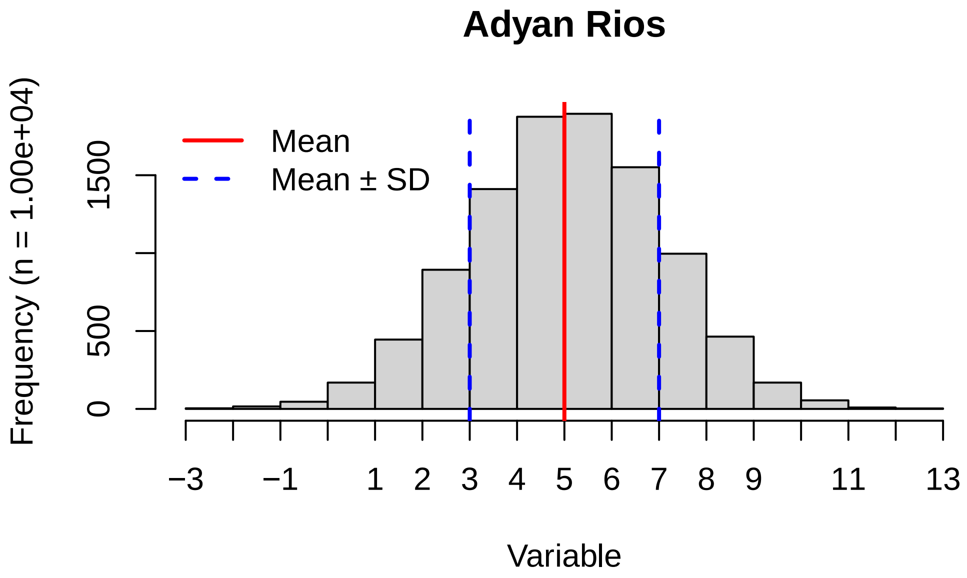


Figure 3.1: Histogram generated from 1.00e+04 random samples from a normal distribution associated with mean 5 and standard deviation 2. The mean is indicated by a vertical red line. Blue dashed vertical lines show the mean minus the standard deviation and the mean plus the standard deviation.

3.1.2 The Standard normal distribution

```
# Set values
num_2 <- 10^4    # number of random draws
mu_2 <- 0        # mean of normal distribution to draw from
sigma_2 <- 1     # standard deviation of normal distribution

# Sample randomly from a normal distribution
x_2 <- rnorm(n = num_2, mean = mu_2, sd = sigma_2)

# Plot the results as a histogram
```

```
hist_2 <- hist(
  x_2,                 # Vector for the histogram
  main = "The Standard normal distribution", # Set title
  xlab = "Variable",
  ylab = paste0("Frequency (n = ", sprintf("%.2e", num_2), ")"),
  xaxt = "n"           # Turn off x-axis values
)

# Add custom x-axis ticks
axis(side = 1, at = hist_2$breaks)

# Indicate mean
abline(v = mu_2, lwd = 2, col = "red")

# Indicate mean ± SD
abline(v = c(mu_2 + sigma_2, mu_2 - sigma_2),
       lwd = c(2), lty = 2, col = "blue")

# Add a legend
legend("topleft", legend = c("Mean", "Mean ± SD"), lwd = 2,
       col = c("red", "blue"), lty = c(1, 2), bty = "n")
```

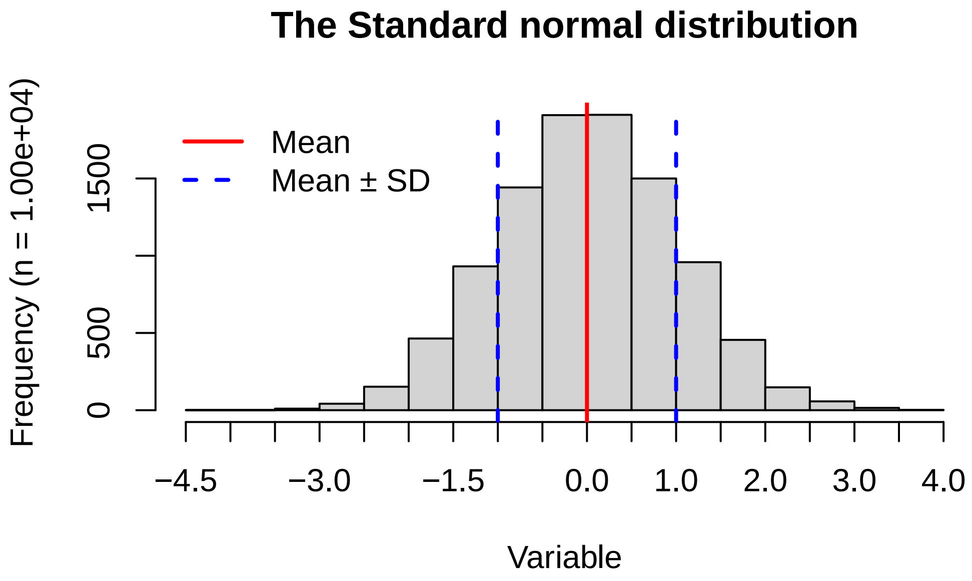


Figure 3.2: Histogram generated from 1.00e+04 random samples from a normal distribution associated with mean 0 and standard deviation 1. The mean is indicated by a vertical red line. Blue dashed vertical lines show the mean minus the standard deviation and the mean plus the standard deviation.

3.1.3 The Standard normal distribution with only 10 samples

```
# Set values
num_3 <- 10   # number of random draws
mu_3 <- 0       # mean of normal distribution to draw from
sigma_3 <- 1    # standard deviation of normal distribution

# Sample randomly from a normal distribution
x_3 <- rnorm(n = num_3, mean = mu_3, sd = sigma_3)

# Plot the results as a histogram
hist_3 <- hist(
  x_3,                      # Vector for the histogram
  main = "The Standard normal distribution", # Set title
  xlab = "Variable",
  ylab = paste0("Frequency (n = ", sprintf("%.2e", num_3), ")"),
  xaxt = "n"              # Turn off x-axis values
)

# Add custom x-axis ticks
axis(side = 1, at = hist_3$breaks)

# Indicate mean
abline(v = mu_3, lwd = 2, col = "red")

# Indicate mean ± SD
abline(v = c(mu_3 + sigma_3, mu_3 - sigma_3),
       lwd = c(2), lty = 2, col = "blue")

# Add a legend
legend("topleft", legend = c("Mean", "Mean ± SD"), lwd = 2,
       col = c("red", "blue"), lty = c(1, 2), bty = "n")
```

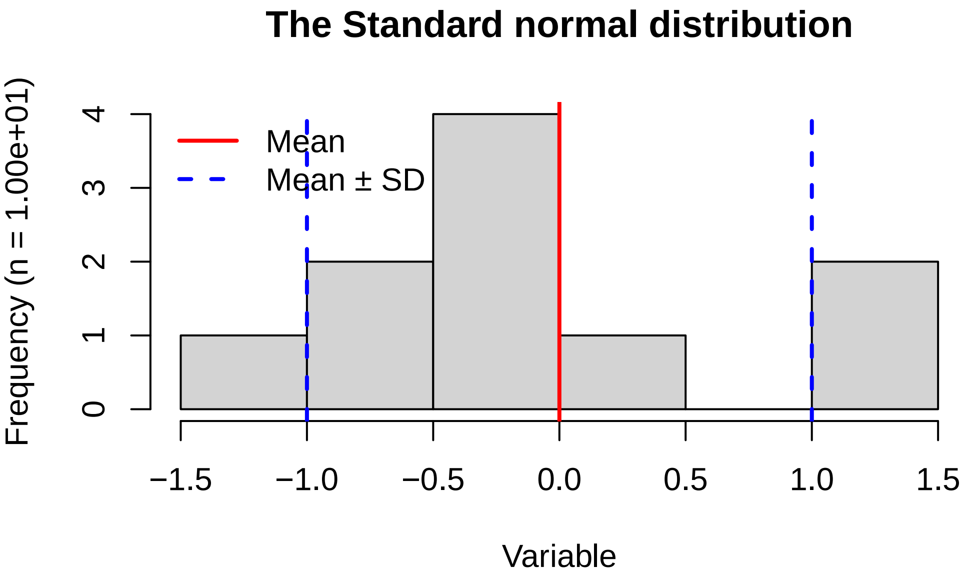


Figure 3.3: Histogram generated from 1.00e+01 random samples from a normal distribution associated with mean 0 and standard deviation 1. The mean is indicated by a vertical red line. Blue dashed vertical lines show the mean minus the standard deviation and the mean plus the standard deviation.

3.1.4 Plotting the normal distribution from Wikipedia

```r
a <- seq(-3, 3, length.out = 100)

plot(a, 1 / (sqrt(2 * 3.14)) * exp(-(a^2)), col = "red", type = "o",
     main = "Probability density function",
     xlab = "Variable x", # Label x-axis
     ylab = "f(x)")       # Label x-axis
points(a, 1 / (sqrt(2 * 3.14)) * exp( -(a^2) / 2), col = "blue")
text(-2, 0.35, expression(f(x) == 1 / (sqrt(2 %*% 3.14)) * exp( -(x^2) / 2)),
     col = "blue", cex = 0.6)
text(-2, 0.3, expression(f(x) == 1 / (sqrt(2 %*% 3.14)) * exp( -(x^2))),
     col = "red", cex = 0.6)
```

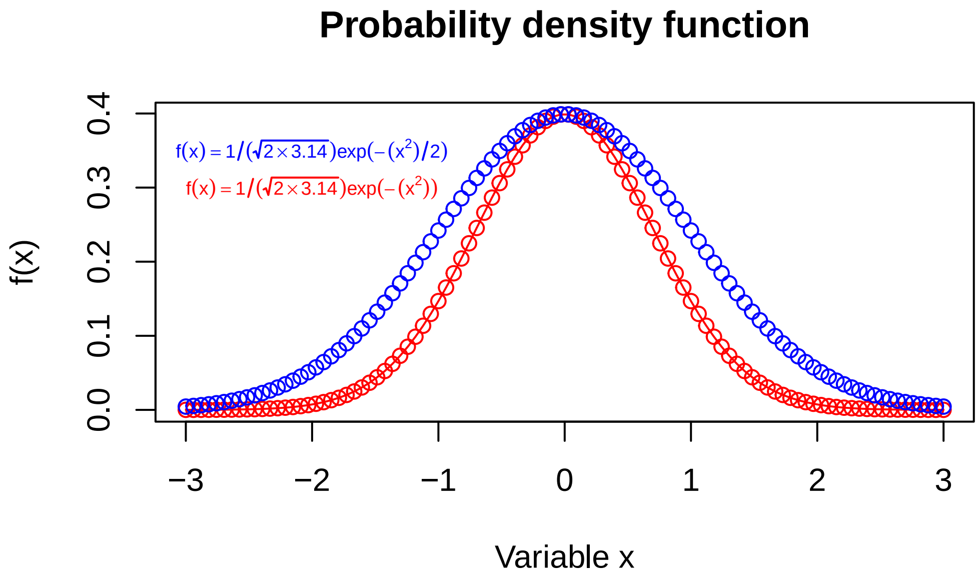


Figure 3.4: Probability density plot for the normal distribution. The standard normal distribution (mean 0 and standard deviation 1) is shown by the curve in blue. Meanwhile, a normal distribution is also plotted in red with a slightly simplified equation.

3.2 Exercise 2

In this part of the assignment, I plotted sea-level data and temperature rates of change over time. To save time when re-running the code, I added if statements that only download the data if they do not exist locally.

I ran into a couple early challenges. First, working with .txt files was a little challenging because I am used to .csv files, which usually already have descriptive column names and straightforward data frame structures. In trying a few options for how to read in the sea level data, I learned how to automatically ignore the comments beginning with "%" which streamline my code so I did not have to hard-code the lines to read. The next challenge was the way the time variable. I had not worked with decimal dates before, but I used date_decimal() from the lubridate to extract the dates into a format that was easier for me to interpret In reproducing and saving the three-panel plot (Atmospheric CO (top), global mean surface-air temperature anomaly (middle), and global mean sea level anomaly (bottom)), I followed the code provided in lab1_sample.R. I then reproduced it in a multi-panel plot that renders directly in this document (Figure 3.5).

To answer how much atmospheric carbon dioxide concentrations, global mean temperatures, and sea levels changed between 1900 and the early part of the present century I subset each data set into "early" (1900-1910) and "recent" (2000-2010) and compared the mean values. The atmospheric carbon increased by 25.64 ppm (73%). Temperature and sea level both fluctuate between seasonally, but similar to the change seen for CO2, they also increased over those 100 years. The mean temperature increased by 0.93 degrees, and the mean sea level increased by 194 mm.

The last part of Exercise 2 involved calculating the rates of temperature change. I checked the data source to clarify why I was using column 14 and why the value was being divided by 100. I learned that the anomalies are stored as hundredths of a degree. Also, column 14 reflects information summarized across the 12 months, which are also provided in separate columns. Initially, I plotted the rate of change on the same axis as the original data series (Figure 3.6) using a red line for the rate and adding a legend. I also produced separate panels to more easily compare them (Figure 3.7). The results show that prior to 1970 there are fewer years with extreme increases or decreases. After the 1970, the dips and peaks are consistently more extreme in both direction. Although there are a few strong dips in the later half of the time series, summing over the values, indicated that the increases are more extreme overall. This corresponds with the upward trend observed in the global mean temperature anomaly plot, where the cyclical nature of the series is also evident.

3.2.1 Sea level anomaly data

```r
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(here, lubridate, ggplot2)
```

```r
# Create a folder for storing downloaded files
if (!file.exists(here("assignment3/data"))) {
  dir.create(here("assignment3/data"))
}

# Download and read in the sea level anomaly data from Jevrejeva et al. (2014)
if (!file.exists(here("assignment3/data/jevrejeva2014_gmsl.txt"))) {
  download.file(
    "https://psmsl.org/products/reconstructions/gslGPChange2014.txt",
    here("assignment3/data/jevrejeva2014_gmsl.txt")
  )
}

# Read in and ignore lines with comments (starting with %)
sl.data <-  read.table(here("assignment3/data/jevrejeva2014_gmsl.txt"),
                       comment = "%")

# Assign column names
colnames(sl.data) <-
  c("time", "rate_mm", "rate_err_mm", "gmsl_mm", "gmsl_err_mm")

# Format date
sl.data$date_decimal <- lubridate::date_decimal(sl.data$time)
```

```r
# Conditionally download files used in lab1_sample.R
if (!file.exists(here("assignment3/data/co2_mm_mlo.txt"))) {
  download.file(
    "ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt",
    here("assignment3/data/co2_mm_mlo.txt")
  )
}

if (!file.exists(here("assignment3/data/law2006.txt"))) {
  download.file(
    "ftp://ftp.ncdc.noaa.gov/pub/data/paleo/icecore/antarctica/law/law2006.txt",
    here("assignment3/data/law2006.txt")
  )
}

if (!file.exists(here("assignment3/data/GLB.Ts+dSST.txt"))) {
  download.file(
    "http://data.giss.nasa.gov/gistemp/tabledata_v3/GLB.Ts+dSST.txt",
    here("assignment3/data/GLB.Ts+dSST.txt")
  )
}

# Read in the CO2 data
loa.co2.data <- read.table(here("assignment3/data/co2_mm_mlo.txt"),
                           skip = 57, header = FALSE)
law.co2.data <- read.table(here("assignment3/data/law2006.txt"),
                           skip = 183, nrows = 2004,
                           header = FALSE)

# Read in the GISS temperature data
begin.rows <- c(9, 31, 53, 75, 97, 119, 141)
num.rows <- c(19, 20, 20, 20, 20, 20, 14)
temp.data <- matrix(NA, nrow = sum(num.rows), ncol = 20)
temp.data[1: num.rows[1], ] <- as.matrix(
  read.table("data/GLB.Ts+dSST.txt", skip = begin.rows[1],
             nrows = num.rows[1], header = FALSE)
)
for (i in 2: length(begin.rows)) {
  temp.data[(sum(num.rows[1: i- 1])+ 1): sum(num.rows[1: i]), ] <-
    as.matrix(read.table("data/GLB.Ts+dSST.txt", skip = begin.rows[i],
                         nrows = num.rows[i], header = FALSE))
}
```

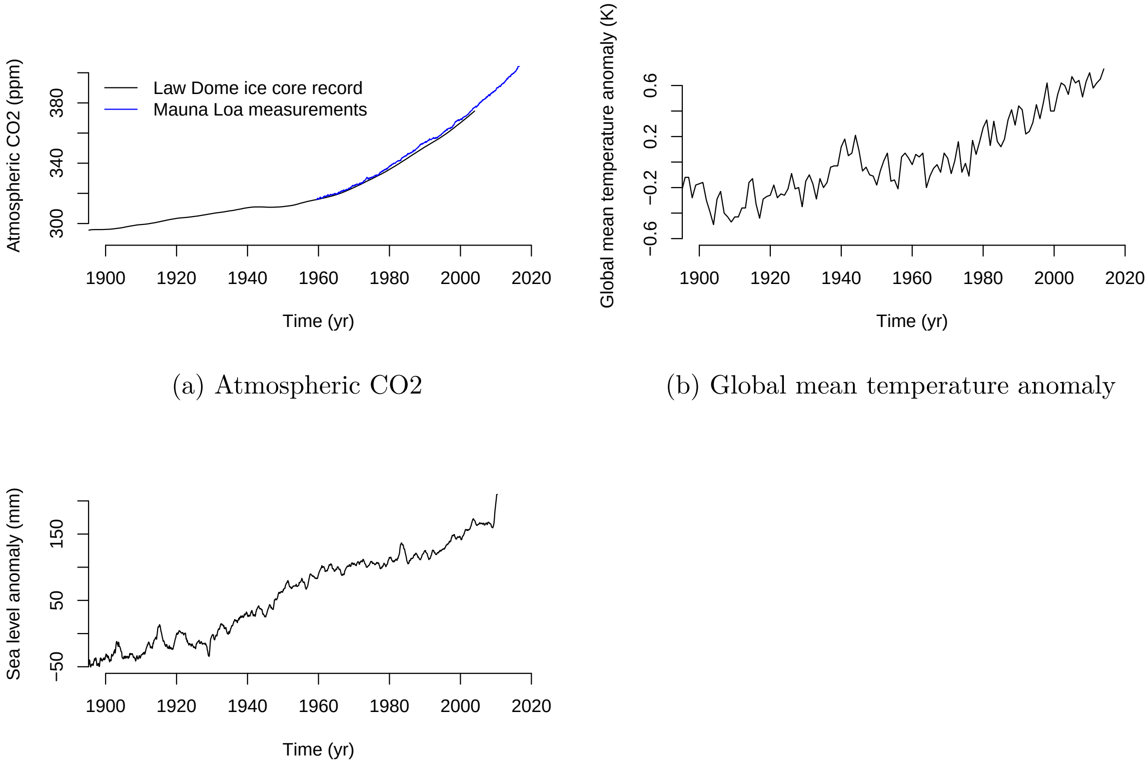
```r
# Create a folder to store figues as pdfs
if (!file.exists(here("assignment3/figures"))) {
  dir.create(here("assignment3/figures"))
}

# Plot
pdf(here("assignment3/figures/lab1_sample_plot2.pdf"),
    width = 4.5, height = 6)
par(mfrow = c(3, 1), cex = 0.66)
plot(law.co2.data[, 1], law.co2.data[, 6], type = "l", xlim = c(1900, 2020),
     ylim = c(290, 400), bty = "n", xlab = "Time (yr)",
     ylab = "Atmospheric carbon dioxide (ppm)")
```

```
lines(loa.co2.data[, 3], loa.co2.data[, 5], type = "l", col = "blue")
legend("topleft", c("Law Dome ice core record", "Mauna Loa measurements"),
       col = c("black", "blue"), lwd = 1, bty = "n")
plot(temp.data[, 1], temp.data[, 14]/ 100, type = "l", xlim = c(1900, 2020),
     ylim = c(-0.6, 0.7), bty = "n", xlab = "Time (yr)",
     ylab = "Global mean temperature anomaly (K)")
plot(sl.data$time, sl.data$gmsl_mm , type = "l", xlim = c(1900, 2020),
      ylim = c(-50, 200), bty = "l", xlab = "Time (yr)",
     ylab = "Sea level anomoly (mm)")

# Close the device and make the return value invisible
invisible(dev.off())
```

```
# Re run code to print in Quarto html and pdf

# CO2
plot(law.co2.data[, 1], law.co2.data[, 6],
     type = "l", xlim = c(1900, 2020), ylim = c(290, 400),
     bty = "n", xlab = "Time (yr)", ylab = "Atmospheric CO2 (ppm)")
lines(loa.co2.data[, 3], loa.co2.data[, 5], type = "l", col = "blue")
legend("topleft",
       c("Law Dome ice core record", "Mauna Loa measurements"),
       col = c("black", "blue"), lwd = 1, bty = "n")

# Temperature anomaly
plot(temp.data[, 1], temp.data[, 14] / 100,
     type = "l", xlim = c(1900, 2020), ylim = c(-0.6, 0.7),
     bty = "n", xlab = "Time (yr)",
     ylab = "Global mean temperature anomaly (K)")

# Sea level anomaly
plot(sl.data$time, sl.data$gmsl_mm,
     type = "l", xlim = c(1900, 2020), ylim = c(-50, 200),
     bty = "n", xlab = "Time (yr)",
     ylab = "Sea level anomaly (mm)")
```

3.2.2 lab1_sample.R Question 1

By how much have atmospheric carbon dioxide concentrations, global mean temperatures, and sea levels changed between 1900 and the early part of the present century?



(a) Atmospheric CO2

(b) Global mean temperature anomaly

(c) Sea level anomaly (mm)

Figure 3.5: Atmospheric CO (top), global mean surface-air temperature anomaly (middle), and global mean sea level anomaly (bottom), 1900–~2015.

```
# Create filtered subsets of each data set to calculate differences
early_co2 <- law.co2.data[law.co2.data$V1 >= 1900 & law.co2.data$V1 < 1910, ]
recent_co2 <- law.co2.data[law.co2.data$V1 >= 2000 & law.co2.data$V1 < 2010, ]
early_temp <- temp.data[temp.data[, 1] >= 1900 & temp.data[, 1] < 1910, ]
recent_temp <-temp.data[temp.data[, 1] >= 2000 & temp.data[, 1] < 2010, ]
early_sea <- sl.data[floor(sl.data$time) >= 1900 & floor(sl.data$time) < 1910, ]
recent_sea <- sl.data[floor(sl.data$time) >= 2000 & floor(sl.data$time) < 2010, ]

# Calculate percent changes (c) or percent differences (pc)
pc_co2 <- round((mean(recent_co2[, 6]) - mean(early_co2[, 6])) /
                  mean(early_co2[, 6]), 4)*100 #25.64 percent
c_co2 <- mean(recent_co2[, 6]) - mean(early_co2[, 6]) #73.2 percent
c_temp <- mean(recent_temp[, 14]/100) - mean(early_temp[, 14]/100) #0.93 degrees
c_sea <- mean(recent_sea$gmsl_mm) - mean(early_sea$gmsl_mm) #194 mm
```

3.2.3 Rates of temperature change

```
# Rate of change
dT_dt_1 <- diff(temp.data[, 14])/100 / diff(temp.data[, 1])
midpoint_t <- temp.data[-length(temp.data[, 1]), 1] + .5

# GISS records sometimes use 100 to represent the average temperature
par(mar = c(5, 7, 4, 2) + 0.1)
plot(temp.data[, 1], temp.data[, 14] / 100, type = "l", xlim = c(1900, 2020),
     ylim = c(-0.6, 0.9), bty = "n", xlab = "Time (yr)",
     ylab = "Global mean temperature anomaly (K)
     and annual rate of change", lwd = 1.5)
lines(midpoint_t, dT_dt_1, type = "l", col = "red", lwd = 1.5)
# Add a legend
legend("topleft",
       legend = c("Global mean temperature anomaly (K)",
                  "Annual rate of temperature change"),
       lwd = 1.5, col = c("black", "red"), lty = 1, bty = "n", cex = 0.8)

# Second y-axis for the rate
# plot(temp.data[, 1], temp.data[, 14] / 100,
#      type = "l", xlim = c(1900, 2020),
#      ylim = c(-0.6, 0.7), bty = "n", xlab = "Time (yr)",
#      ylab = "Global mean temperature anomaly (K)", lwd = 1.5)
# par(new = TRUE) # Prepare for second axis
# plot(midpoint_t, dT_dt_1, type = "l",
#      xlim = c(1900, 2020), ylim = c(-0.5, 0.5),
#      col = "red", xaxt = "n", yaxt = "n", xlab = "", ylab = "", , lwd = 1.5)
# axis(side = 4, col = "red", col.axis = "red")
# mtext("Annual rate of change", side = 4, line = 3, col = "red")
```

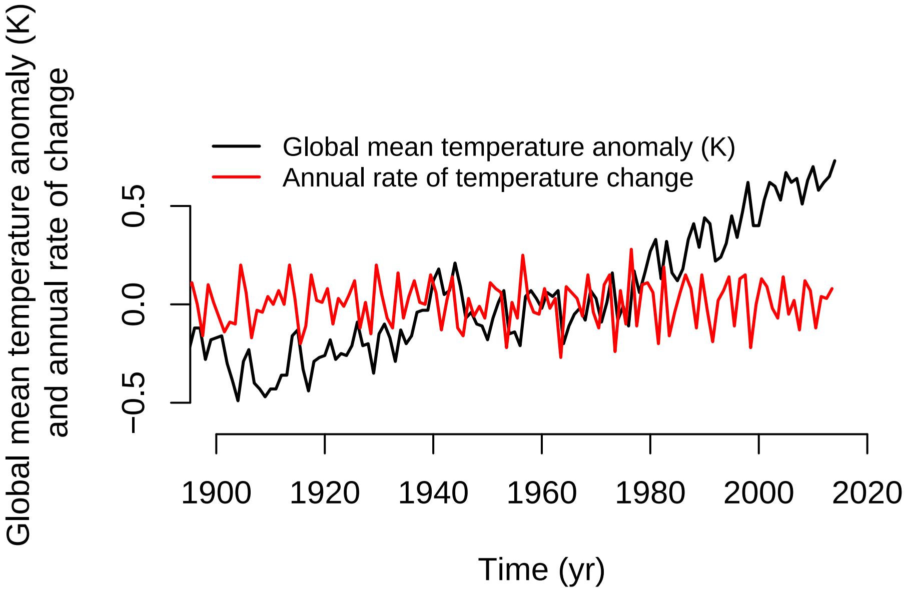
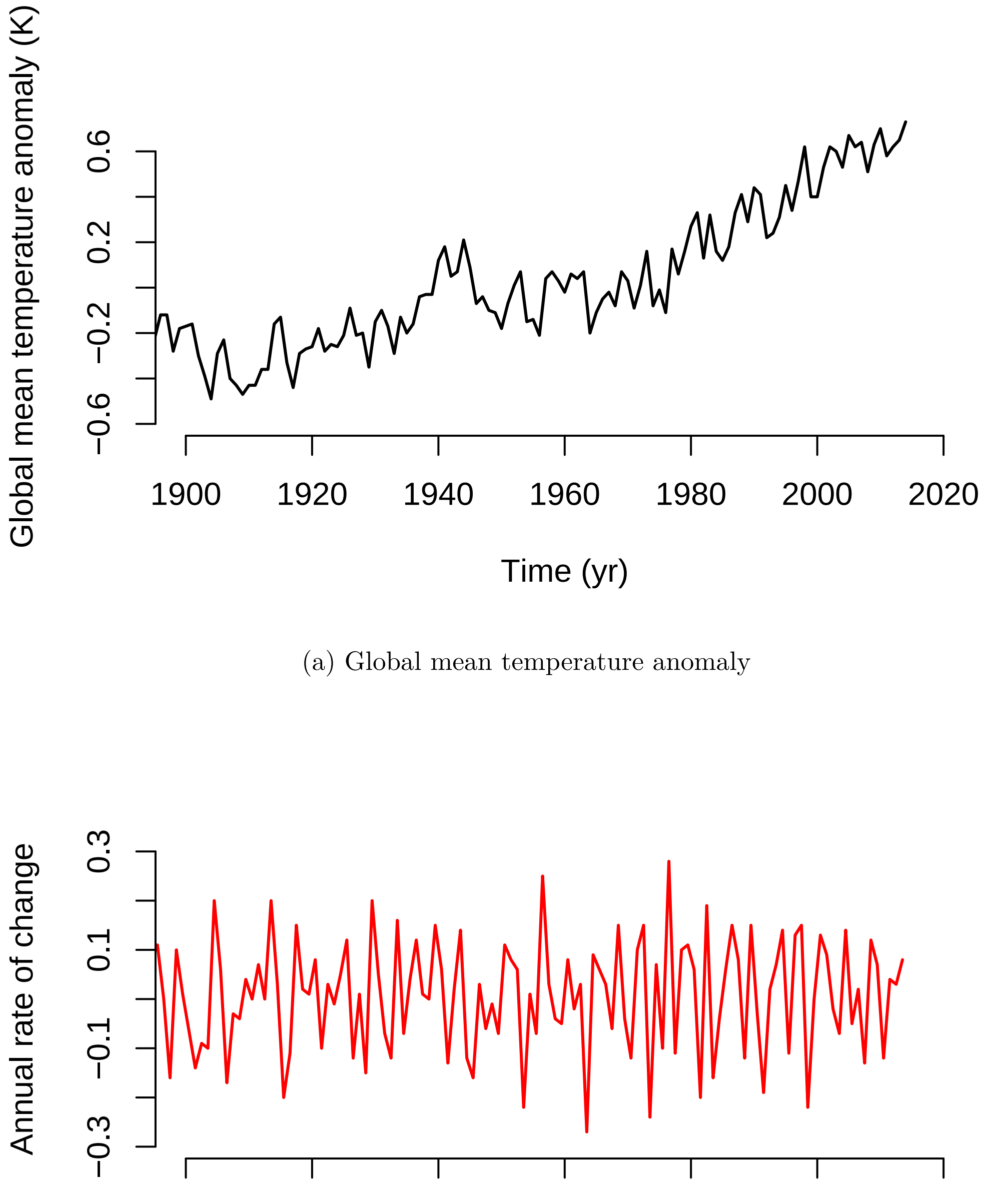


Figure 3.6: Overalaid plots of global mean temperature anomaly (black) and its annual rate of change (red).

```
plot(temp.data[, 1], temp.data[, 14] / 100, type = "l", xlim = c(1900, 2020),
     ylim = c(-0.6, 0.7), bty = "n", xlab = "Time (yr)",
     ylab = "Global mean temperature anomaly (K)", lwd = 1.5)

plot(midpoint_t, dT_dt_1, type = "l", col = "red", lwd = 1.5,
     xlim = c(1900, 2020),, ylim = c(-0.3, 0.3), bty = "n",
     xlab = "Time (yr)", ylab = "Annual rate of change")
```



(a) Global mean temperature anomaly

(b) Annual rate of change

Figure 3.7: Global mean temperature anomaly (black) and its annual rate of change (red).

3.3 Exercise 3

To prepare for this exercise, I included example code that was provided in the lab manual. I left it in because I reuse parts of it later, including the sampling examples to identify resulting "Galton Board" bins, and the code to generate histograms and Q-Q plots. For the actual exercise further down in this chapter, I created a function I could reuse for each variation across different number of balls. Using the function with intentional return vectors allowed me not to worry about clearing existing variables or figures from memory. I ran the function with 10 bins, and with 10, 20, 30, 100, and 10^4 balls. With 10 and 20 balls the histogram and Q-Q plot do not look normal. At about 30 balls the histogram and Q-Q plot begin to look approximately normal (Figure 4.3). Once the number of balls is large, as in the runs with 10^2 (Figure 4.4) and very large with 10^4 balls (Figure 4.5), the histogram looks normal but the Q-Q plot shows a slight departure. This may relate to the discrete and bounded nature of the bin outcomes, while the Q-Q plot may be better suited for continuous data.

3.3.1 Implement the Galton Board

```
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(animation)
```

```
# Set number of balls and rows following the example code
n.balls <- 200
n.rows <- 15

# ani.options(nmax = n.balls + n.rows - 2)
# quincunx(balls = n.balls, layers = n.rows)
```

```
# Follow example code to identify the resulting bin
path <- sample(x = c(-0.5, 0.5), size = (n.rows - 1), replace = TRUE)
print(path)
```

```
 [1] -0.5 -0.5 -0.5 -0.5 -0.5  0.5  0.5  0.5 -0.5 -0.5 -0.5  0.5 -0.5  0.5
```

```
bin <- sum(path)
print(bin)
```

```
[1] -2
```

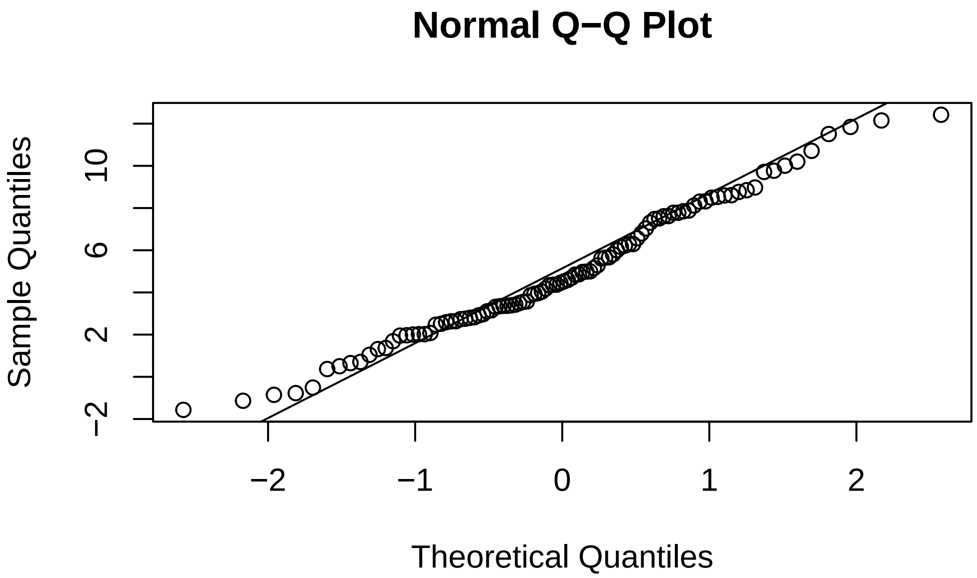
```
# Example of a for loop
n.times <- 3
for (i in 1:n.times) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
```

```
# Another example of a for loop
n.times <- 5
output <- rep(1, n.times)
for(i in 3:n.times){
  output[i] <- sum(output[(i-2):(i-1)])
}
print(output)
```

```
[1] 1 1 2 3 5
```

```
# Example of how to make the Q-Q plot
norm.vals <- rnorm(100, mean = 5, sd = 3)
qqnorm(norm.vals)
qqline(norm.vals)
```


Normal Q−Q Plot
Sample Quantiles
−2
2
6
10
−2
−1
0
1
2
Theoretical Quantiles

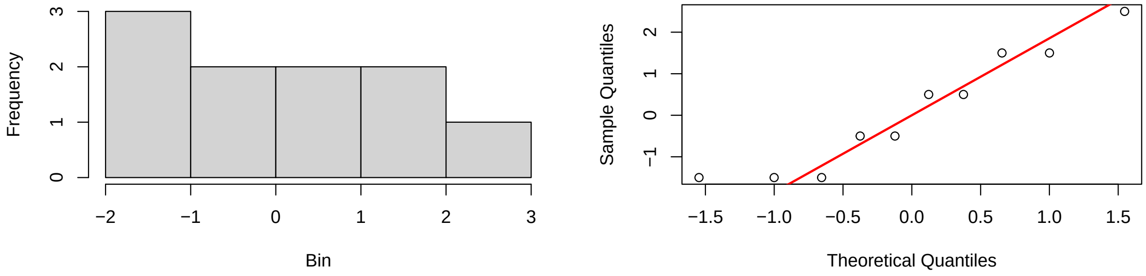
4 Exercise

- Use comments to explain what the code does and who wrote it
- Clears existing variables from memory and close open figures
- Set values for the number of balls to drop and the number of rows of pins
- Create vector output, initially populated with NAs
- Use a loop to run through values (1:n.balls), determine and store results
- Make a histogram of outputs

```r
# Develop a function to conduct the exercise
run_board <- function(n_rows, n_balls){
  output <- rep(NA, n_balls) # Create vector for output
  for (i in 1: n_balls){ # Loop over given number of balls
    path_i <- sample(x = c(-0.5, 0.5),size = (n_rows - 1), replace = TRUE)
    output[i] <- sum(path_i) # Sum over samples to obtain bin
  }
  return(output)
}
```

```r
# Repeatedly run the function with different n_balls
run5 <- run_board(n_rows = 10, n_balls = 10)

hist(run5, main = "",
     xlab = "Bin", ylab = "Frequency") # Create histogram
qqnorm(run5, main = "") # Create Q-Q plot
qqline(run5, col = "red", lwd = 2)
```

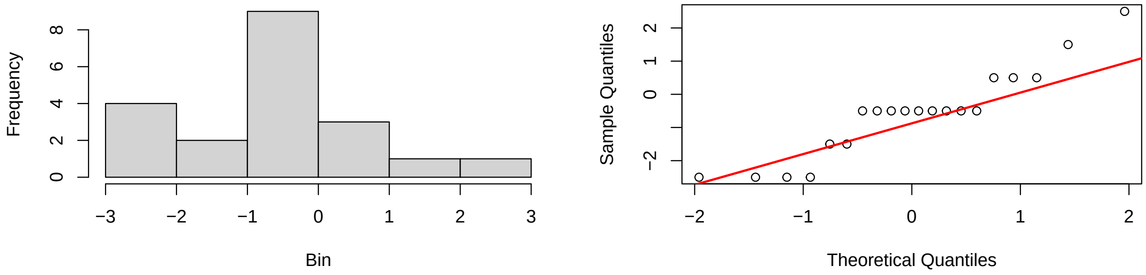
```r
# Repeatedly run the function with different n_balls
run5 <- run_board(n_rows = 10, n_balls = 20)

hist(run5, main = "",
     xlab = "Bin", ylab = "Frequency") # Create histogram
qqnorm(run5, main = "") # Create Q-Q plot
qqline(run5, col = "red", lwd = 2)
```



(a) Histogram

(b) Q-Q Plot

Figure 4.1: Results of replicating the Galton board dynamics with 10 rows and 10 balls.

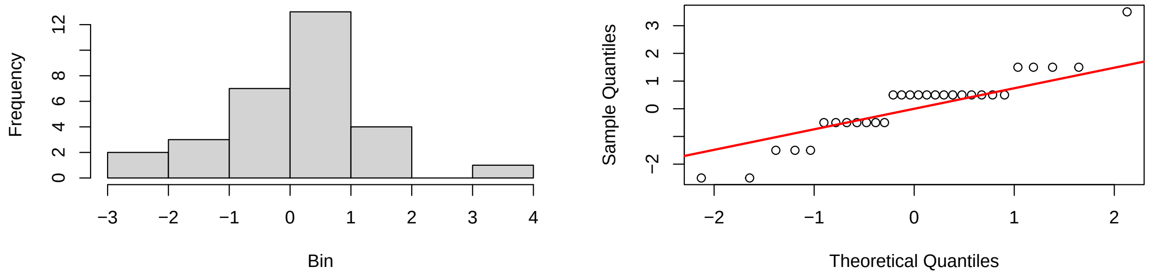


(a) Histogram

(b) Q-Q Plot

Figure 4.2: Results of replicating the Galton board dynamics with 10 rows and 20 balls.

```
# Repeatedly run the function with different n_balls
run5 <- run_board(n_rows = 10, n_balls = 30)

hist(run5, main = "",
     xlab = "Bin", ylab = "Frequency") # Create histogram
qqnorm(run5, main = "") # Create Q-Q plot
qqline(run5, col = "red", lwd = 2)
```



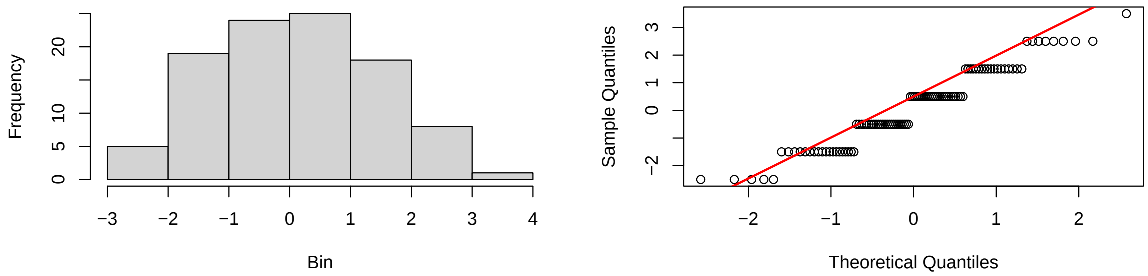
(a) Histogram

(b) Q-Q Plot

Figure 4.3: Results of replicating the Galton board dynamics with 10 rows and 30 balls.

```
# Repeatedly run the function with different n_balls
run5 <- run_board(n_rows = 10, n_balls = 10^2)

hist(run5, main = "",
     xlab = "Bin", ylab = "Frequency") # Create histogram
qqnorm(run5, main = "") # Create Q-Q plot
qqline(run5, col = "red", lwd = 2)
```

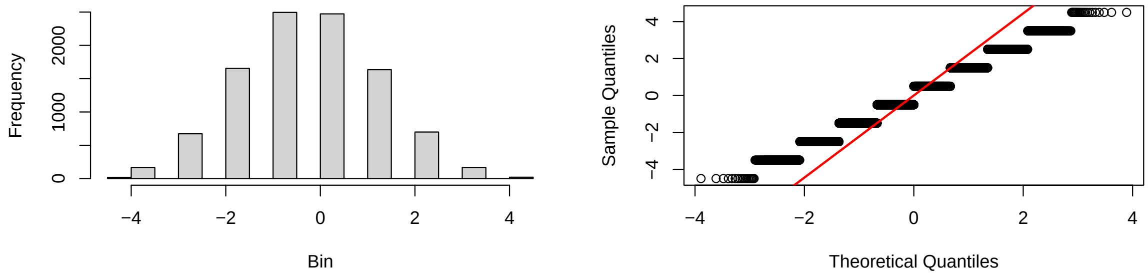
```
# Repeatedly run the function with different n_balls
run5 <- run_board(n_rows = 10, n_balls = 10^4)

hist(run5, main = "",
     xlab = "Bin", ylab = "Frequency") # Create histogram
qqnorm(run5, main = "") # Create Q-Q plot
qqline(run5, col = "red", lwd = 2)
```



(a) Histogram

(b) Q-Q Plot

Figure 4.4: Results of replicating the Galton board dynamics with 10 rows and 10ˆ2 balls.



(a) Histogram

(b) Q-Q Plot

Figure 4.5: Results of replicating the Galton board dynamics with 10 rows and 10ˆ4 balls.

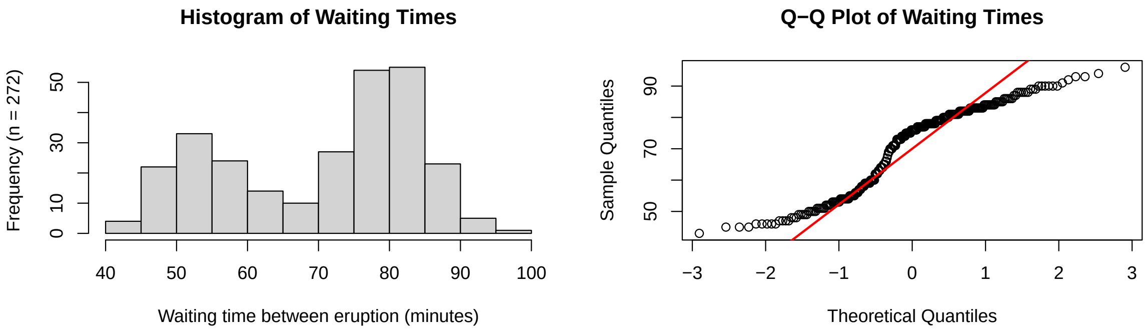
4.1 Exercise 4

Based on the histograms and Q–Q plots, the waiting time between Old Faithful eruptions is not approximately normally distributed (Figure 5.5). The histogram is clearly bimodal, and the Q–Q plot shows systematic departures from the red reference line, indicating a lack of correspondence between the theoretical normal quantiles and the sample quantiles. By contrast, based on the shape of the histogram and the close alignment with the red line in the Q–Q plot in Figure 4.7, the sepal length data for Iris setosa appears to be approximately normally distributed.

4.1.1 Generate histograms and Q-Q plots

4.1.1.1 Old Faithful eruptions

```r
# Histogram of waiting times
hist(faithful$waiting,
     main = "Histogram of Waiting Times",
     ylab = paste0("Frequency (n = ", length(faithful$waiting), ")"),
     xlab = "Waiting time between eruption (minutes)")

# Q-Q plot of waiting times
qqnorm(faithful$waiting,
       main = "Q-Q Plot of Waiting Times")
qqline(faithful$waiting, col = "red", lwd = 2)
```



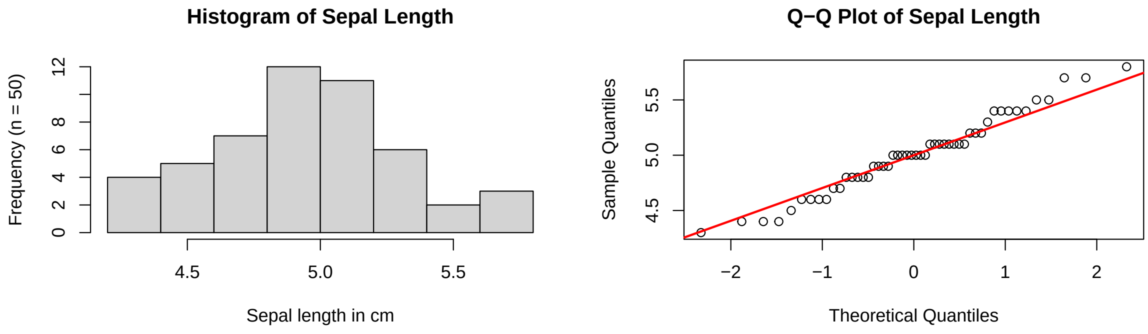
(a) Histogram

(b) Q-Q Plot

Figure 4.6: Histogram and Q-Q plot for the waiting time between eruptions of the Old Faithful geyser in yellowstone national park, WY.

4.1.1.2 Sepal length of setosa irises

```
# Histogram of waiting times
hist(iris3[, "Sepal L.", "Setosa"],
     main = "Histogram of Sepal Length",
     ylab = paste0("Frequency (n = ",
                   length(iris3[, "Sepal L.", "Setosa"]), ")"),
     xlab = "Sepal length in cm")

# Q-Q plot of waiting times
qqnorm(iris3[, "Sepal L.", "Setosa"],
       main = "Q-Q Plot of Sepal Length")
qqline(iris3[, "Sepal L.", "Setosa"], col = "red", lwd = 2)
```



(a) Histogram

(b) Q-Q Plot

Figure 4.7: Histogram and Q-Q plot of sepal length of setosa irises in cm.

4.2 Links to Colab and GitHub

Assignment 2 Google Colab

Quarto book chapter on GitHub

5 Assignment 4

EVR-5086 Fall 2025

Assignment 4

To complete this assignment in R, I used the following packages:

- here: here() enables easy file referencing
- dplyr: functions for data manipulation
- tidyr: functions for reshaping data
- flextable: formatting tables
- ggplot2: for creating plots
- moments: functions for kurtosis and skewness
- fBasics: dagoTest() for the D'Agostino normality test

```
# Check if libraries are installed; install if not.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(here, dplyr, flextable,  ggplot2, tidyr, moments, fBasics)
```

5.1 Exercise 1

To start, I read in MIA_J-D_T_Precip_inches.csv and assigned descriptive column names. Instead of computing the time series for one month at a time, I used dplyr::mutate() to divide each month's column by the total column. This results in a data frame of the fraction of the annual total by year and month. For the remainder of the assignment, I focus on the month of April. Figure 5.1 shows the fraction of annual rainfall in April in Miami, Florida, from 1906 to 2022. Table 5.1 reports the descriptive statistics, rounded to four significant digits, for the fraction of annual rainfall occurring in the month of April.

```r
# Read in and subset MIA_J-D_T_Precip_inches.csv
mia_rain <- read.csv(here("assignment4", "data", "MIA_J-D_T_Precip_inches.csv"),
                   header = FALSE)

# Add descriptive column names
names(mia_rain) <- c("year", "jan", "feb", "mar", "apr", "may", "jun",
                     "jul", "aug", "sep", "oct", "nov", "dec", "total")

# Compute monthly fraction by dividing all months by the total
mia_rain_fraction <- mia_rain %>%
      mutate(across(-c("year", "total"), ~ round(. / total, 4)))
```

```r
#Plot the time series for April's fraction of the annual total for each year
ggplot(mia_rain_fraction, aes(x = year, y = apr)) +
  geom_line() +
  labs(
    title = "Fraction of annual rainfall during April for Miami, Florida",
    subtitle = "Comparison between 1906 and 2022",
    x = "Year",
    y = "Fraction of annual rainfall during April"
  )
```

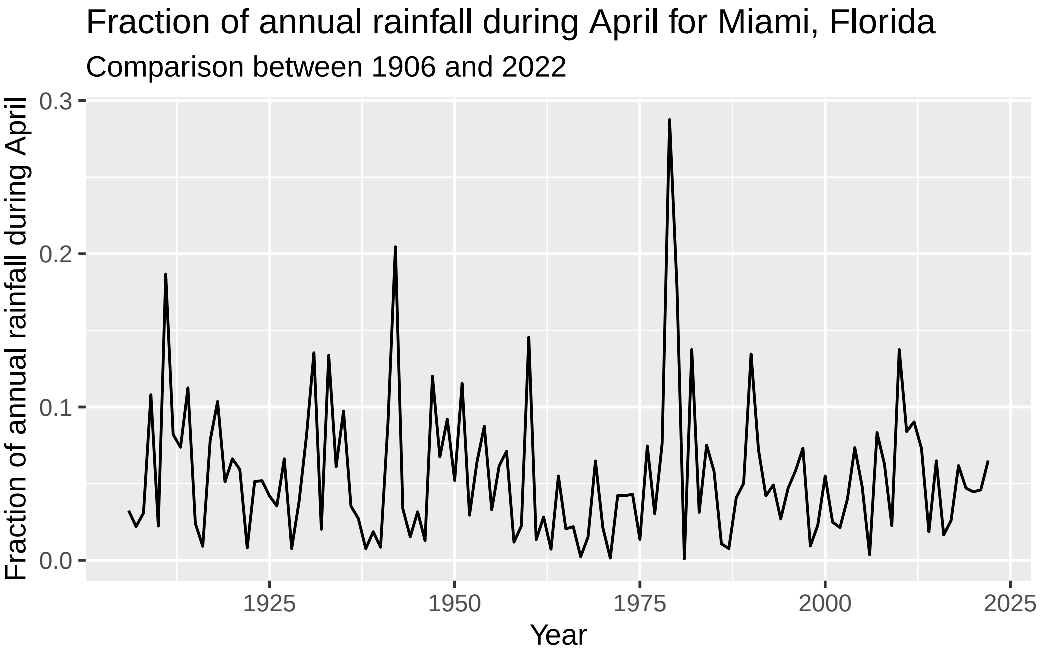


Figure 5.1: Fraction of annual rainfall during the month of April for the City of Miami, Florida, from 1906 to 2022.

5.2 Exercise 2

Next, ahead of reshaping the data, I created factor levels to be able to rearrange the data by month of the year and computed the number of unique years in the data. In preparing the data for analysis, I reformatted the data, assigned factor levels, computed the log10-transformed data. In using pipes to reshape and add variables to the data, I reduced the number of intermediate objects created. Similarly with group_by() and summarize(), I was able to calculate various summary statistics at the same time for both the fraction and the log10 fraction of Miami rainfall by month.

The comparison of the histograms and Q-Q plots in Figure 5.2 show that the log10 transformed data appears more normal than the original data, but still show departures from the Q-Q line.

```r
# Prep month levels for efficient ordering
month_levels <- c("jan", "feb", "mar", "apr", "may", "jun",
                  "jul", "aug", "sep", "oct", "nov", "dec")

# Calculate n
n_years <- n_distinct(mia_rain_fraction, "year")

# Prep data for upcoming analyses
mia_rain_fraction_tidy <- mia_rain_fraction |>
  # Remove total
  select(-total) |>
  # Reformat to tidy format (long)
  pivot_longer(
    cols = !year,
    names_to = "month",
    values_to = "fraction"
  ) |>
  # Log transform monthly fractions and define month factor levels
  mutate(
    month = factor(month, levels = month_levels, ordered = TRUE),
    log10_fraction = log10(fraction)
  ) |>
  # Reformat to tidy format (long)
  pivot_longer(
    cols = !c(year, month),
    names_to = "stat",
    values_to = "values"
  ) |>
  # Sort by stat and year
  arrange(stat, year, month)

# Compute descriptive statistics for mia_rain_fraction
mia_rain_fraction_summary <- mia_rain_fraction_tidy |>
  # Run calculation on unique combinations of year and stat
  group_by(month, stat) |>
  # Calculate summary stats
  summarise(min = round(min(values), 4),
            q1 = round(quantile(values, 0.25), 4),
            median = round(median(values), 4),
            mean = round(mean(values), 4),
            q3 = round(quantile(values, 0.75), 4),
            max = round(max(values), 4),
            variance = round(var(values), 4),
            sd = round(sd(values), 4),
            sk = round(skewness(values), 4),
            ku = round(kurtosis(values), 4),
            .groups = "drop") |>
  # Sort data by stat and month
  arrange(stat, month)
```

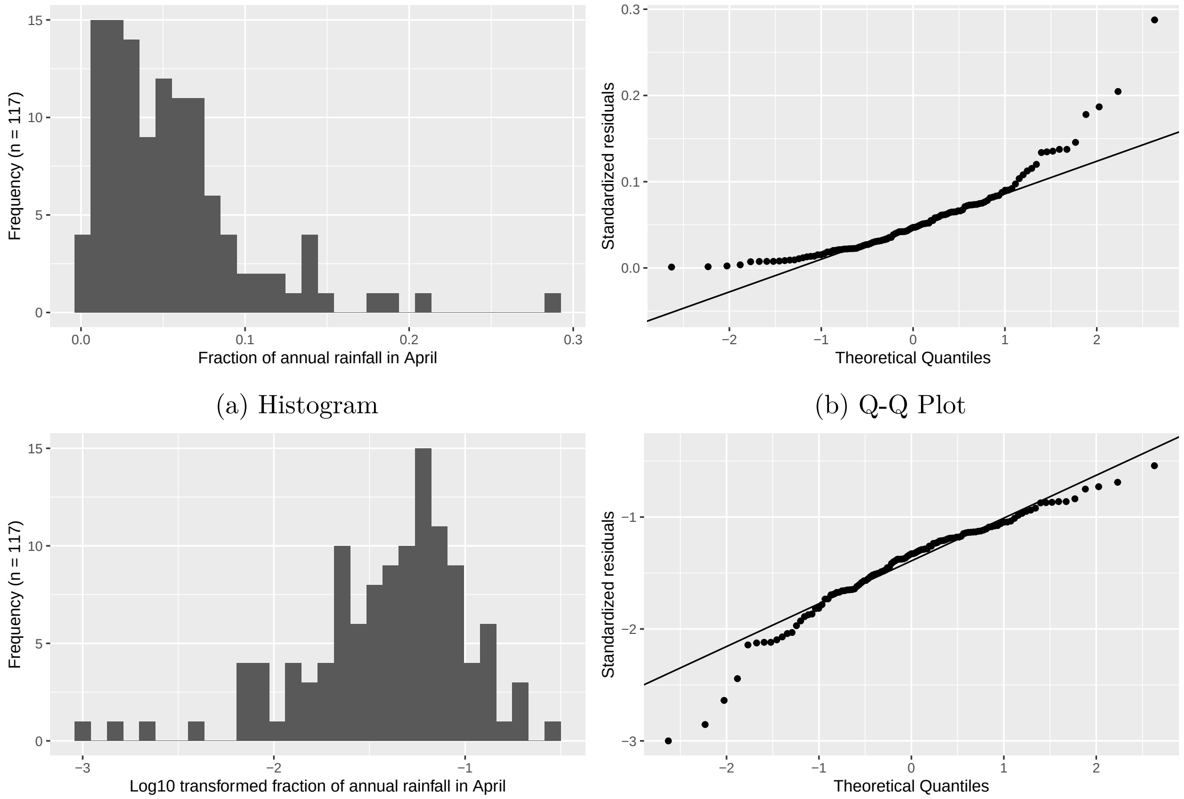
```r
# Rearrange the summary statistics to print nicely for just April
apr_summary <- mia_rain_fraction_summary |>
  # Pivot the data so that the summary statistics have a long format
  pivot_longer(-c(month, stat), names_to = "metric", values_to = "value")|>
  # Pull out the stat into respective columns
  pivot_wider(names_from = stat, values_from = value) |>
  # Filter to just April
  filter(month == "apr") |>
  # Remove the month column
  select(-month) |>
  # Print reformatted and filtered data as a flextable
  flextable() |>
  # Include a blank first column header
  set_header_labels(metric = "") |>
  # Use simple format and autofit
  theme_booktabs() |>
  fontsize(size = 9, part = "all") |>
  autofit()
```

```
# Print summary table
apr_summary
```

Table 5.1: Descriptive statistics for the fraction of annual rainfall during April for Miami, Florida between 1906 and 2022, and its log transformation.

	fraction	log10_fraction
min	0.0010	-3.0000
q1	0.0224	-1.6498
median	0.0470	-1.3279
mean	0.0560	-1.4140
q3	0.0735	-1.1337
max	0.2876	-0.5412
variance	0.0021	0.1850
sd	0.0463	0.4301
sk	1.8372	-1.0215
ku	5.0577	1.6156

```r
# Create histogram
mia_rain_fraction_tidy |>
  filter(month == "apr", stat == "fraction") |>
  ggplot(aes(x = values)) +
  geom_histogram(bins = 30) +
  xlab("Fraction of annual rainfall in April") +
  ylab(paste0("Frequency (n = ", n_years, ")"))

# Create Q-Q plot
mia_rain_fraction_tidy |>
  filter(month == "apr", stat == "fraction") |>
  ggplot(aes(sample = values)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")

# Create histogram for log10 tranformed data
mia_rain_fraction_tidy |>
  filter(month == "apr", stat == "log10_fraction") |>
  ggplot(aes(x = values)) +
  geom_histogram(bins = 30) +
  xlab("Log10 transformed fraction of annual rainfall in April") +
  ylab(paste0("Frequency (n = ", n_years, ")"))

# Create Q-Q plot for log10 transformed data
mia_rain_fraction_tidy |>
  filter(month == "apr", stat == "log10_fraction") |>
  ggplot(aes(sample = values)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")
```



(a) Histogram

(b) Q-Q Plot

(c) Histogram (Log10 transformed)

(d) Q-Q Plot (Log10 transformed)

Figure 5.2: Histogram (a) and Q-Q plot (b) for the fraction of annual rainfall during April for Miami, Florida between 1906 and 2022, and histogram (c) and Q-Q plot (d) of the corresponding log transformed data.

5.3 Exercise 3

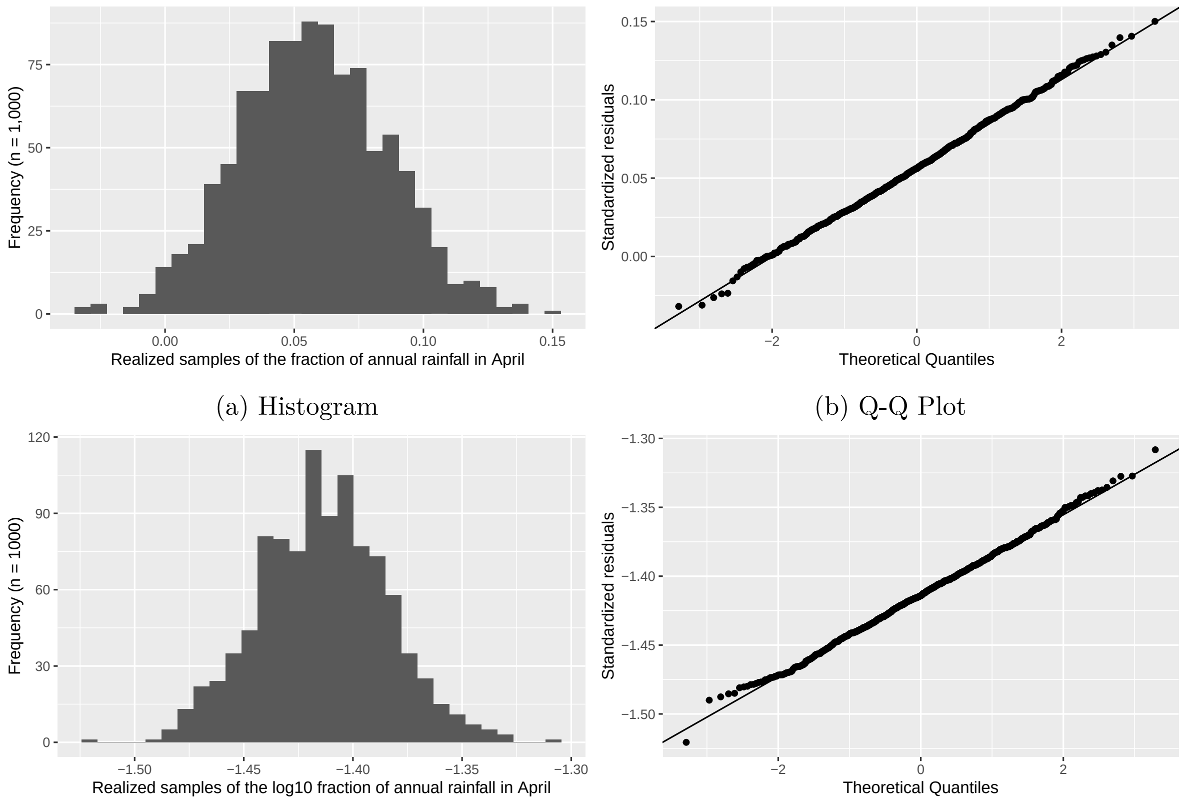
I used the descriptive statistics from the previous exercise to generate samples from a known normal distribution. I compared 1,000 versus 21 samples. Figure 5.3 shows how the Q-Q Q-Q plot improves with more samples. Meanwhile, in Figure 5.4 we can see how much worse the Q-Q plot looks with fewer samples!

```r
# Generate realizations of 1000 samples from known normal distribution
apr_rain_fraction_sample_1000 <- round(
  rnorm(n = 1000,
        mean = mia_rain_fraction_summary |>
          filter(stat == "fraction",
                 month == "apr") |>  pull(mean),
        sd = mia_rain_fraction_summary$sd[1]), 4
)

# Generate realization of 21 samples from known normal distribution
apr_rain_fraction_sample_21 <- round(
  rnorm(n = 21,
        mean = mia_rain_fraction_summary |>
          filter(stat == "fraction",
                 month == "apr") |>  pull(mean),
        sd = mia_rain_fraction_summary$sd[1]), 4
)

# Generate realization of 1000 samples from known normal log10 distribution
apr_rain_fraction_log10_sample_1000 <- round(
  rnorm(n = 1000,
        mean = mia_rain_fraction_summary |>
          filter(stat == "log10_fraction",
                 month == "apr") |>  pull(mean),
        sd = mia_rain_fraction_summary$sd[1]), 4
)

# Generate realization of 21 samples from known normal log10 distribution
apr_rain_fraction_log10_sample_21 <- round(
  rnorm(n = 21,
        mean = mia_rain_fraction_summary |>
          filter(stat == "log10_fraction",
                 month == "apr") |>  pull(mean),
        sd = mia_rain_fraction_summary$sd[1]), 4
)
```

```r
# Create histogram with 1000 samples
ggplot() +
  aes(x = apr_rain_fraction_sample_1000) +
  geom_histogram(bins = 30) +
  xlab("Realized samples of the fraction of annual rainfall in April") +
  ylab("Frequency (n = 1,000)")

# Create Q-Q plot with 1000 samples
ggplot() +
  aes(sample = apr_rain_fraction_sample_1000) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")

# Create histogram with 1000 log10 samples
ggplot() +
  aes(x = apr_rain_fraction_log10_sample_1000) +
  geom_histogram(bins = 30) +
  xlab("Realized samples of the log10 fraction of annual rainfall in April") +
  ylab("Frequency (n = 1000)")

# Create Q-Q plot with 1000 log10 samples
ggplot() +
  aes(sample = apr_rain_fraction_log10_sample_1000) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")
```

Here we plot samples from the known normal distribution with only 21 samples.

```r
# Create histogram with 21 samples
ggplot() +
  aes(x = apr_rain_fraction_sample_21) +
  geom_histogram(bins = 30) +
  xlab("Realized samples of the fraction of annual rainfall in April") +
  ylab("Frequency (n = 21)")

# Create Q-Q plot with 21 samples
ggplot() +
  aes(sample = apr_rain_fraction_sample_21) +
```



(a) Histogram

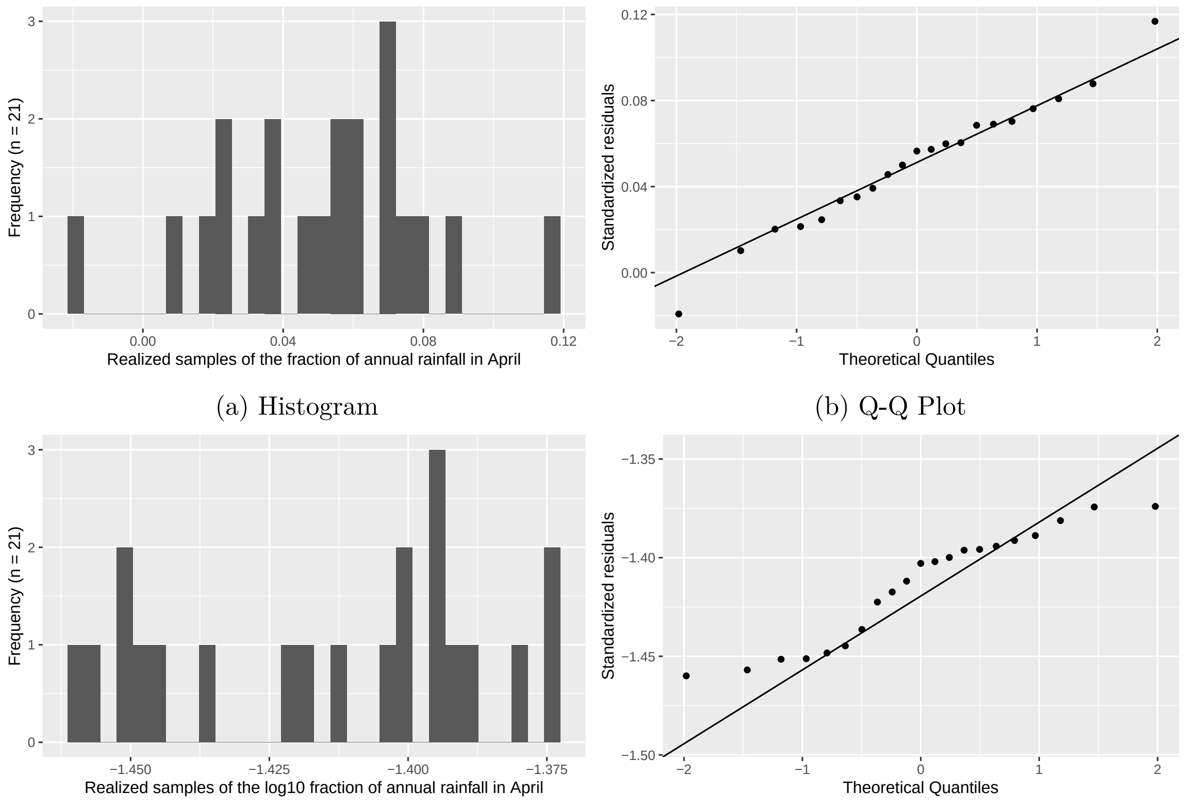
(b) Q-Q Plot

(c) Histogram (Log10 transformed)

(d) Q-Q Plot (Log10 transformed)

Figure 5.3: Histogram (a) and Q-Q plot (b) for 1000 samples realized from a normal distribution based on the fraction of annual rainfall during April for Miami, Florida between 1906 and 2022, and histogram (c) and Q-Q plot (d) of the corresponding log transformed data.

```
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")

# Create histogram with 21 log10 samples
ggplot() +
  aes(x = apr_rain_fraction_log10_sample_21) +
  geom_histogram(bins = 30) +
  xlab("Realized samples of the log10 fraction of annual rainfall in April") +
  ylab("Frequency (n = 21)")

# Create Q-Q plot with 21 log10 samples
ggplot() +
  aes(sample = apr_rain_fraction_log10_sample_21) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")
```



(a) Histogram

(b) Q-Q Plot

(c) Histogram (Log10 transformed)

(d) Q-Q Plot (Log10 transformed)

Figure 5.4: Histogram (a) and Q-Q plot (b) for 21 samples realized from a normal distribution based on the fraction of annual rainfall during April for Miami, Florida between 1906 and 2022, and histogram (c) and Q-Q plot (d) of the corresponding log transformed data.

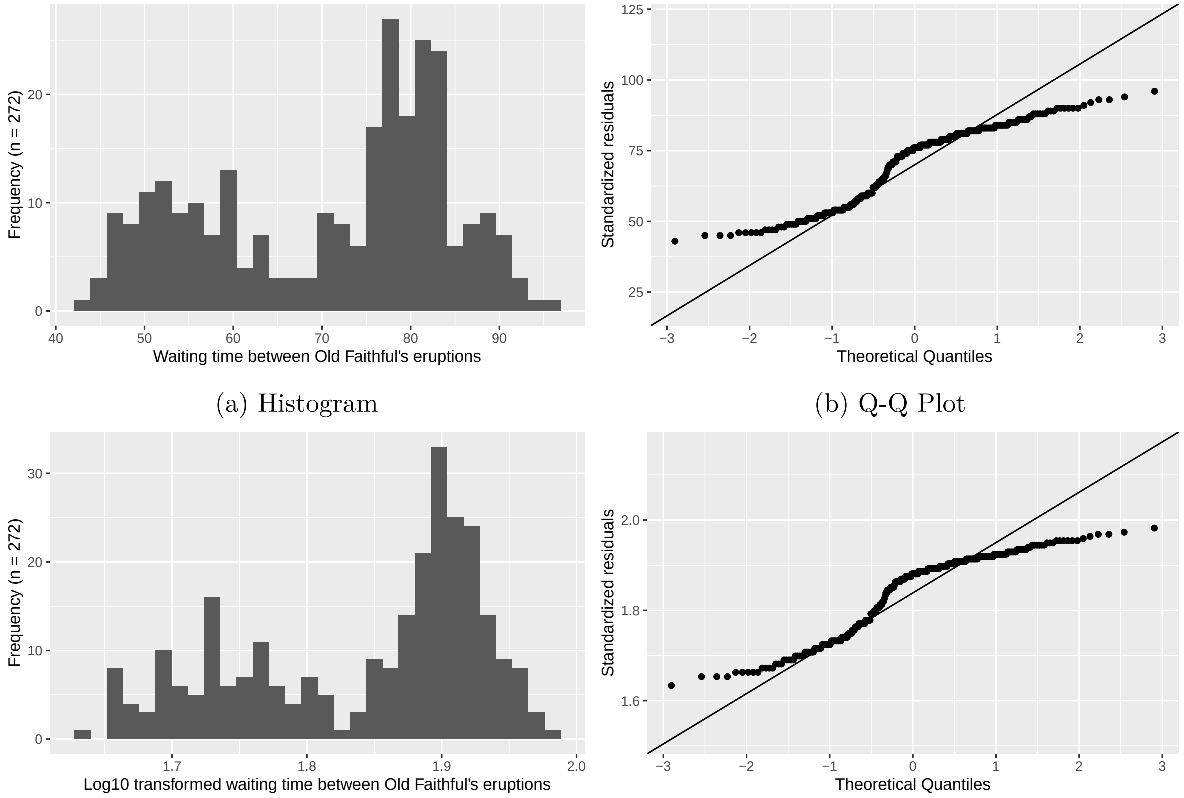
5.4 Exercise 4

I used dagoTest() to run the D'Agostino normality test on:

1. the fraction of annual rainfall for April
2. the log10 transformed fraction of annual rainfall for April
3. 1,000 realized samples from a normal distribution
4. 1,000 realized samples from a log10 transformed normal distribution
5. the waiting time between Old Faithful's eruptions
6. the log10 transformed waiting time between Old Faithful's eruptions

In Figure 5.5, I include histograms and Q-Q plots for the waiting time between Old Faithful's eruptions. Table 5.2 provides key values for the normality tests listed above. In reading more about the test, I was reminded that the null hypothesis of the data being drawn from a normally distributed population is rejected when the p-value is less than the significance level (0.05). Following this decision rule, all of the tests listed above were rejected, except the samples that were realized from a normal distribution (numbers 3 and 4 above). This result aligns with the Q-Q plots shown previously, where all of them show departures from the expected theoretical quantiles, except the 1,000 realized samples that generated using a normal distribution (Figure 5.3).

```r
# Calculate sample size
n_eruptions <- length(faithful$waiting)

# Create histogram
faithful |>
  ggplot(aes(x = waiting)) +
  geom_histogram(bins = 30) +
  xlab("Waiting time between Old Faithful's eruptions") +
  ylab(paste0("Frequency (n = ", n_eruptions, ")"))

# Create Q-Q plot
faithful |>
  ggplot(aes(sample = waiting)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")

# Create histogram for log10 transformed data
faithful |>
  ggplot(aes(x = log10(waiting))) +
  geom_histogram(bins = 30) +
  xlab("Log10 transformed waiting time between Old Faithful's eruptions") +
  ylab(paste0("Frequency (n = ", n_eruptions, ")"))

# Create Q-Q plot for log10 transformed data
faithful |>
  ggplot(aes(sample = log10(waiting))) +
  stat_qq() +
  stat_qq_line() +
  xlab("Theoretical Quantiles") +
  ylab("Standardized residuals")
```



(a) Histogram

(b) Q-Q Plot

(c) Histogram (Log10 transformed)

(d) Q-Q Plot (Log10 transformed)

Figure 5.5: Histogram (a) and Q-Q plot (b) for the waiting time between Old Faithful's eruptions, and histogram (c) and Q-Q plot (d) of the corresponding log transformed data.

```r
# Normal tests on rainfall data, sample realization, and Old Faithful
dago_mia_rain <- dagoTest(mia_rain_fraction$apr)
dago_mia_rain_log10 <- dagoTest(log10(mia_rain_fraction$apr))
dago_mia_rain_sample_1000 <- dagoTest(apr_rain_fraction_sample_1000)
dago_mia_rain_log10_sample_1000 <- dagoTest(apr_rain_fraction_log10_sample_1000)
dago_faithful <- dagoTest(faithful$waiting)
dago_faithful_log10 <- dagoTest(log10(faithful$waiting))
```

```r
# Collect results in a tibble for table output
dagoTest_results <- tibble(
  test = c(
    "April fraction",
    "April log10(fraction)",
    "April Normal sample (n = 1000)",
    "April log10 Normal sample (n = 1000)",
    "Old Faithful waiting",
    "log10(Old Faithful waiting)"
  ),
  chi_square = c(
    round(dago_mia_rain@test$statistic[1], 4),
    round(dago_mia_rain_log10@test$statistic[1], 4),
    round(dago_mia_rain_sample_1000@test$statistic[1], 4),
    round(dago_mia_rain_log10_sample_1000@test$statistic[1], 4),
    round(dago_faithful@test$statistic[1], 4),
    round(dago_faithful_log10@test$statistic[1], 4)
  ),
  p_value = c(
    round(dago_mia_rain@test$p.value[1], 4),
    round(dago_mia_rain_log10@test$p.value[1], 4),
    round(dago_mia_rain_sample_1000@test$p.value[1], 4),
    round(dago_mia_rain_log10_sample_1000@test$p.value[1], 4),
    round(dago_faithful@test$p.value[1], 4),
    round(dago_faithful_log10@test$p.value[1], 4)
  )
)
```

```
# Flextable
flextable(dagoTest_results) |>
  set_header_labels(test = "", chi_square = " ²", p_value = "p-value") |>
  colformat_num(j = c("chi_square", "p_value"), digits = 4) |>
  theme_booktabs() |>
  fontsize(size = 9, part = "all") |>
  autofit()
```

Table 5.2: Chi-squared and p-value statistics from various normality tests conducted using the D'Agostino-Pearson omnibus normality test.

	χ^2	p-value
April fraction	59.3249	0.0000
April log10(fraction)	24.3791	0.0000
April Normal sample (n = 1000)	0.8834	0.6429
April log10 Normal sample (n = 1000)	0.9314	0.6277
Old Faithful waiting	109.2417	0.0000
log10(Old Faithful waiting)	55.5880	0.0000

5.5 Exercise 5

I also computed the ranks and return periods for April rainfall. When preparing the data for the plot, I noticed that it was necessary for the ranking to have a negative sign to generate a descending ranking. Lastly, in my first attempt of the plot with the logarithmic x-axis ticks, I realized I was accidentally doing log10 twice. When I use scale_x_log10 to adjust the x-axis scale, I noticed I need to read in the non-transformed data. That way the axis values in Figure 5.6 reflect the actual return periods in years, but the spacing is logarithmic.

```r
# Rank April rainfall and compute return period
apr_rank <- mia_rain |>
  select(apr) |>
  # sort largest to smallest
  arrange(desc(apr)) |>
  # Create rank and compute return period
  mutate(
    rank = row_number(),
    recurrence = (n_years + 1) / rank,
    log10_recurrence = log10(recurrence)
  )

# Plot ranked rainfall amounts as a function of the log10 of the return period
ggplot(apr_rank, aes(x = recurrence, y = apr)) +
  geom_point() +
  scale_x_log10(
    breaks = c(1, 10, 100),
    labels = c("1", "10", "100")
  ) +
  labs(x = "Return period (years, log scale)", y = "Ranked April rainfall (inches)")
```

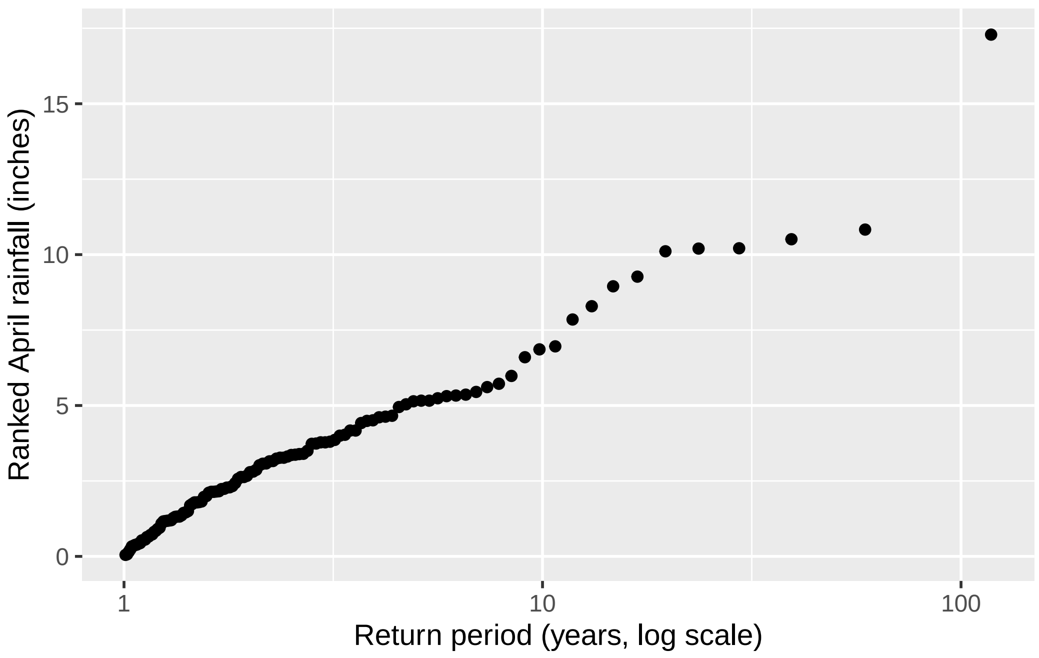


Figure 5.6: Ranked rainfall ammounts for the month of April as a function of the log10 of the retunperiod in years.