Supplementary materials to Angourakis et al. (2025)

Andreas Angourakis

14 February, 2025

Table of contents

1 Introduction 3
1.1 About this document . . . 3
1.2 About the Weather model . . . 3
1.2.1 Parameters and hyperparameters . . . 4
1.3 Context of the Weather model within the Indus Village model . . . 5

2 Daily weather in example locations 8

3 Example of simulation outputs of the Weather model for 5 years 27

4 Demonstration of parameter variation: solar radiation and temperature 34

5 Demonstration of parameter variation: precipitation 40

6 Calibration walk through 49
6.1 Parameter estimation using `optim()` . . . 49
6.1.1 Calibrating `gen_annual_double_logistic_curve()` (deterministic function) . . . 50
6.1.2 Adding `discretise_curve()` (stochastic function) . . . 54
6.1.3 Calibrating multiple example curves to determine hyperparameters . . . 58

7 Calibration targeting weather examples 65
7.1 Preparation . . . 65
7.2 Estimation of annual cumulative precipitation hyperparameters based on weather dataset . . . 70
7.2.1 Test an isolated version of the estimation of cumulative precipitation hyperparameters using `optim` . . . 71
7.2.2 Run estimation of cumulative precipitation hyperparameters for all sites 74
7.3 Running the entire Weather model using all estimated parameters . . . 82
7.4 Creating figure . . . 87

1 Introduction

This file and all other referenced in the code can be found at the repository: https://github.com/Two-Rains/Weather-Angourakis-et-al-2025

1.1 About this document

To facilitate a deeper understanding and application of the Weather model, this resource contains all the source code for the figures presented in the related paper (Angourakis, Baudouin, and Petrie, **in submission**), including:

- Visualizing weather variables in example locations.
- Demonstrating the full model functionality.
- Visualizing parameter sensitivity for solar radiation and temperature generation.
- Visualizing parameter sensitivity for precipitation generation.
- A walk-through on the calibration of parameters.
- A calibration workflow for example locations.

These materials offer hands-on guidance for users looking to implement, calibrate, and analyse the Weather model in their own research.

1.2 About the Weather model

The Weather model is a procedural generation model designed to produce random synthetic daily weather time series with realistic characteristics, given a set of parameters. It is implemented in NetLogo and R and is computationally efficient. The Weather model generates synthetic weather time series using algorithms based on sinusoidal and double logistic functions, incorporating stochastic variation to mimic unpredictable weather patterns. It produces daily values of surface solar radiation, average/max/min temperature, and total precipitation.

More details about the two implementation at:

- ODD document for the NetLogo implementation
- ODD document for the R implementation

1.2.1 Parameters and hyperparameters

parameter	description
year_length	Number of days per year
southern_hemisphere	Whether the annual curve corresponds to values in the southern or northern
temperature - annual_max	Annual maximum of daily mean temperature
temperature - annual_min	Annual minimum of daily mean temperature
temperature - daily_fluctuation	Standard deviation in daily mean temperature
temperature - daily_lower_dev	Lower deviation from daily mean temperature
temperature - daily_upper_dev	Upper deviation from daily mean temperature
solar - annual_max	Annual maximum of daily mean solar radiation
solar - annual_min	Annual minimum of daily mean solar radiation
solar - daily_fluctuation	Standard deviation in daily mean solar radiation

hyperparameter	parameter (year)	description
	year_length	Number of days per year
annual_sum	precipitation - annual_sum_mean	Mean and
annual_sum	precipitation - annual_sum_sd	standard deviation in annual sum of precipitati
n_samples	precipitation - plateau_value_mean	Mean and
n_samples	precipitation - plateau_value_sd	standard deviation in number of random sampl
max_sample_size	precipitation - inflection1_mean	Mean and
max_sample_size	precipitation - inflection1_sd	standard deviation in maximum length of samp
plateau_value	precipitation - rate1_mean	Mean and
plateau_value	precipitation - rate1_sd	standard deviation in value in which the gap be
inflection1	precipitation - inflection2_mean	Mean and
inflection1	precipitation - inflection2_sd	standard deviation in day of year in which the f
rate1	precipitation - rate2_mean	Mean and
rate1	precipitation - rate2_sd	standard deviation in maximum rate or slope in
inflection2	precipitation - n_samples_mean	Mean and
inflection2	precipitation - n_samples_sd	standard deviation in day of year in which the s
rate2	precipitation - max_sample_size_mean	Mean and
rate2	precipitation - max_sample_size_sd	standard deviation in maximum rate or slope in

1.3 Context of the Weather model within the Indus Village model

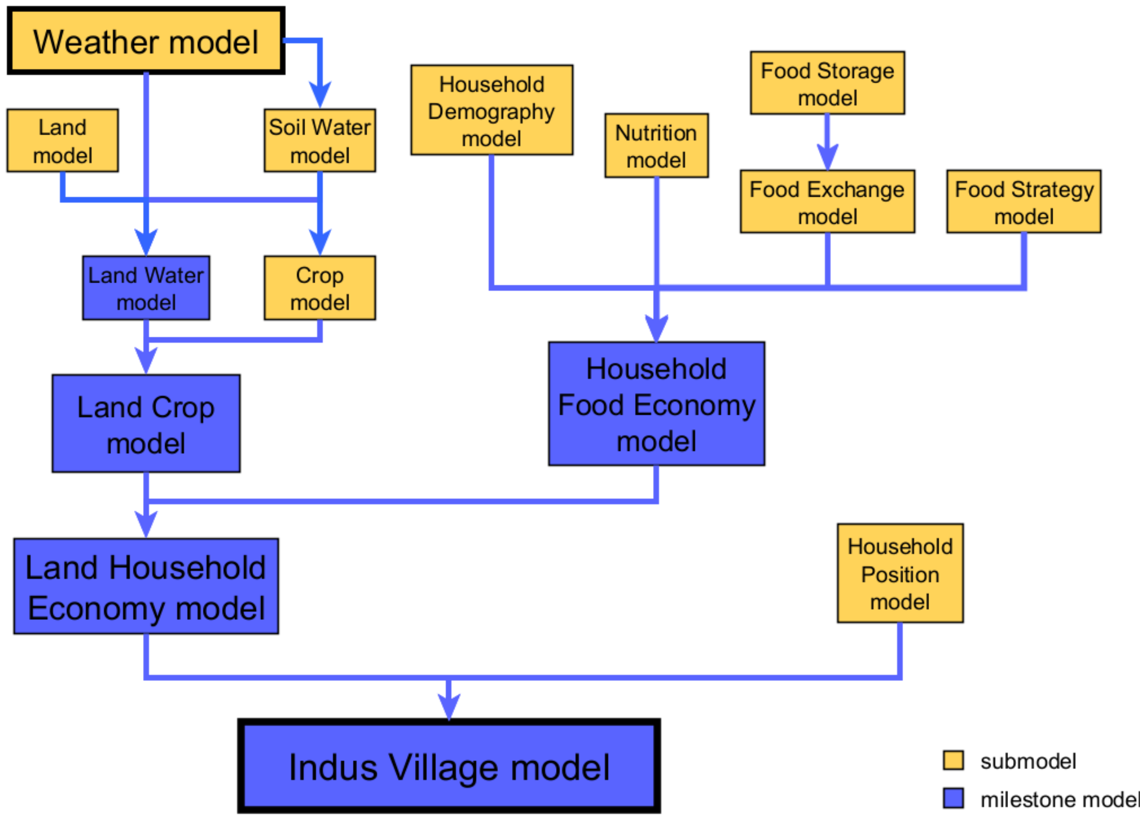


Figure 1.1: Route of model integration in the Indus Village

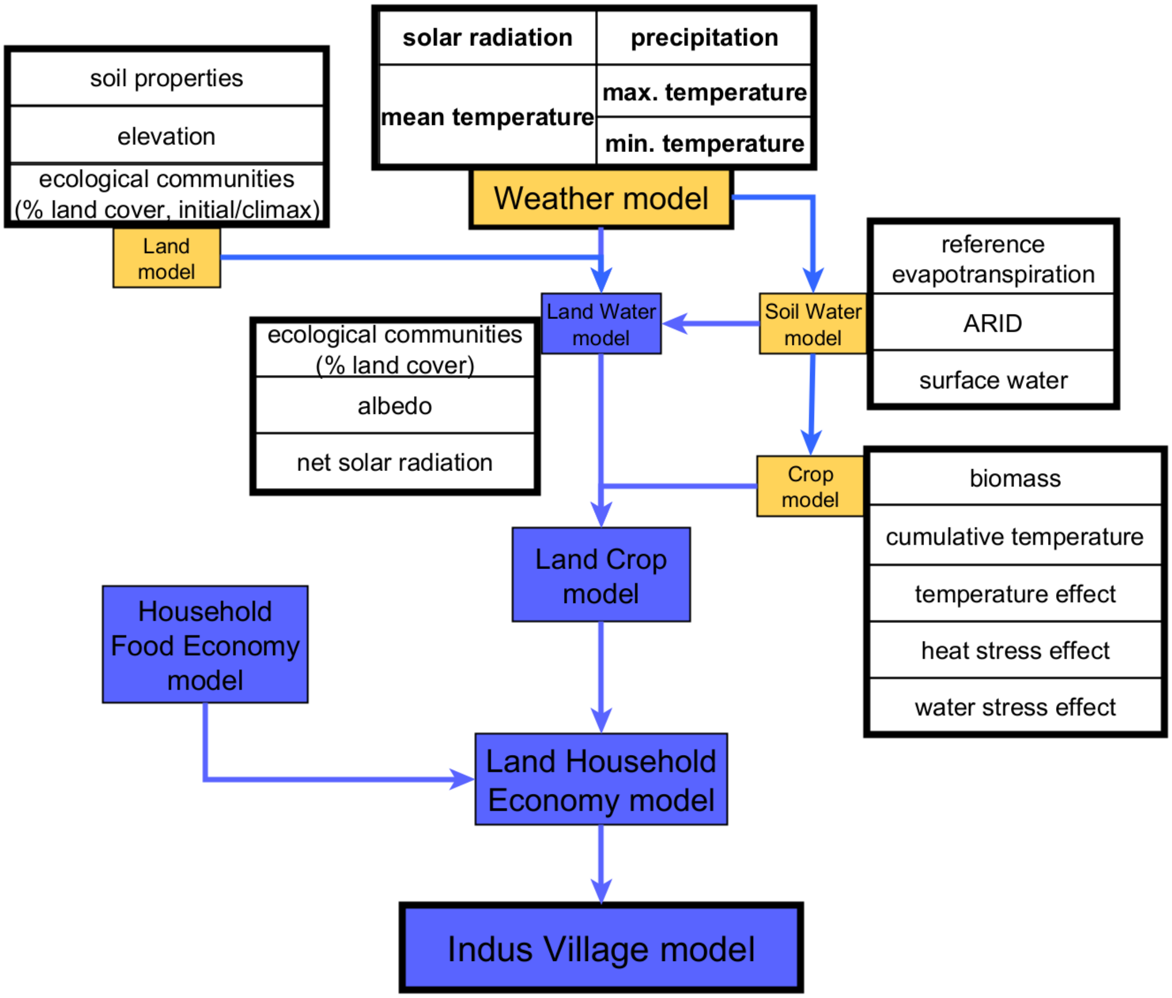


Figure 1.2: The weather variables and the key interface variables of the related models

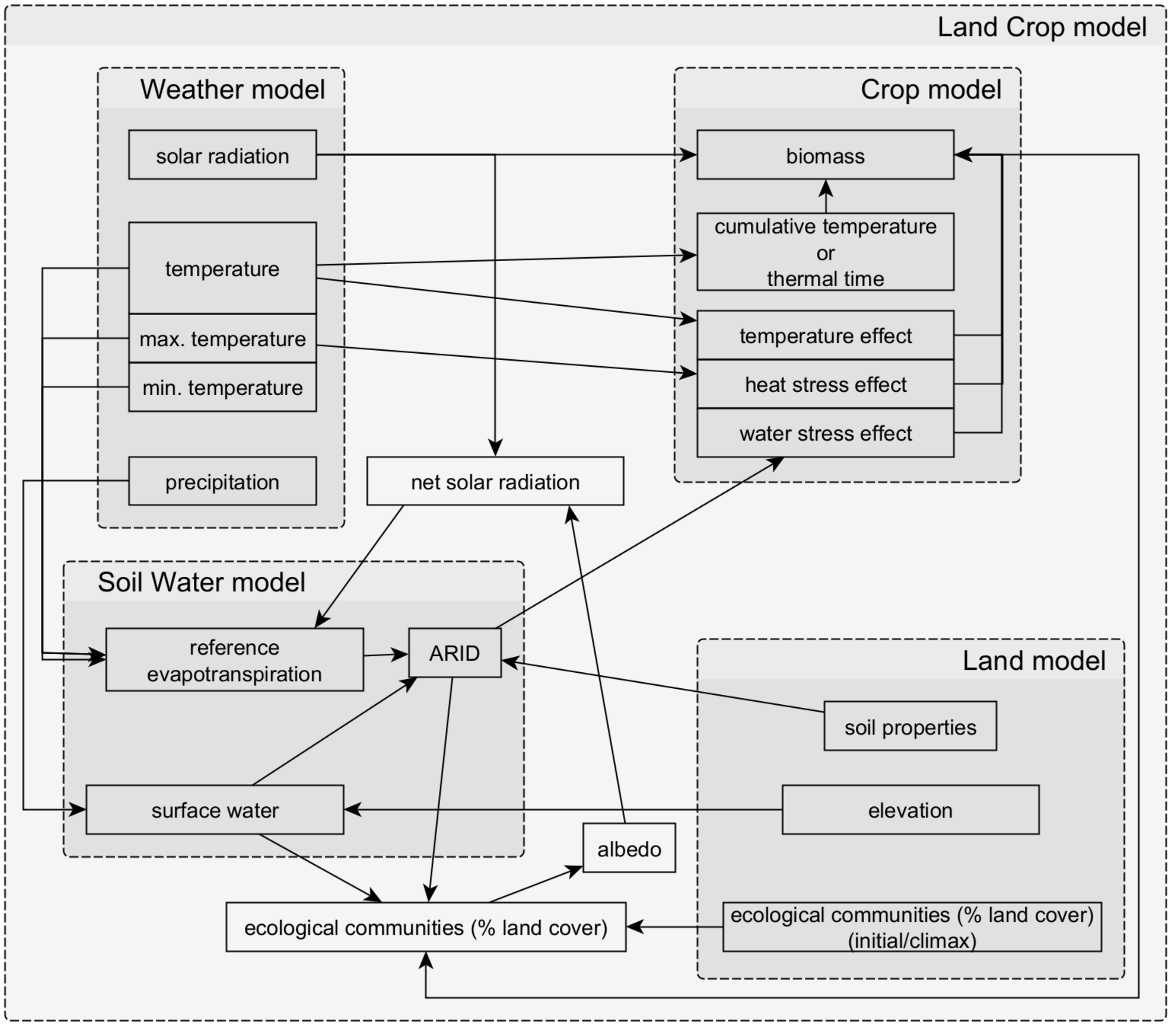


Figure 1.3: The connections between weather variables and the key interface variables of the related models

2 Daily weather in example locations

Choose file format for generated figures:

```
output_dir <- "output"
plot_file_format <- c("png", "eps")[1] # modify index number to change format
```

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
```

We use the data downloaded at NASA´s POWER access viewer (power.larc.nasa.gov/data-access-viewer/) selecting the user community 'Agroclimatology' and pin pointing the different locations between 01/01/1984 and 31/12/2007. The exact locations are:

- Rakhigarhi, Haryana, India (Latitude: 29.1687, Longitude: 76.0687)

- Irkutsk, Irkutsk Óblast, Russia (Latitude: 52.2891, Longitude: 104.2493)
- Hobart, Tasmania, Australia (Latitude: -42.8649, Longitude: 147.3441)
- Pearl Harbor, Hawaii, United States of America (Latitude: 21.376, Longitude: -157.9708)
- São Paulo, Brazil (Latitude: -23.5513, Longitude: -46.6344)
- Cambridge, United Kingdom (Latitude: 52.2027, Longitude: 0.122)
- Windhoek, Namibia (Latitude: -22.5718, Longitude: 17.0953)

We selected the ICASA Format's parameters:

- Precipitation (PRECTOT)

- Wind speed at 2m (WS2M)

- Relative Humidity at 2m (RH2M)

- Dew/frost point at 2m (T2MDEW)

- Maximum temperature at 2m (T2M_MAX)

- Minimum temperature at 2m (T2M_MIN)
- All sky insolation incident on a horizontal surface (ALLSKY_SFC_SW_DWN)
- Temperature at 2m (T2M)

and from Solar Related Parameters:

- Top-of-atmosphere Insolation (ALLSKY_TOA_SW_DWN)

```r
# Function to read and filter weather data
read_weather_data <- function(file_path) {
  data <- read.csv(file_path, skip = 18)
  data[data$YEAR %in% 1984:2007, ]
}

# Get input file paths
input_files <- list.files(path = "input", full.names = TRUE)

# Read and combine all weather data
weather <- do.call(rbind, lapply(input_files, read_weather_data))

# Define site mapping
site_mapping <- list(
  list(condition = function(x) floor(x$LAT) == 29, site = "Rakhigarhi"),
  list(condition = function(x) floor(x$LON) == 104, site = "Irkutsk"),
  list(condition = function(x) floor(x$LAT) == -43, site = "Hobart"),
  list(condition = function(x) floor(x$LAT) == 21, site = "Pearl Harbor"),
  list(condition = function(x) floor(x$LAT) == -24, site = "Sao Paulo"),
  list(condition = function(x) floor(x$LON) == 0, site = "Cambridge"),
  list(condition = function(x) floor(x$LAT) == -23, site = "Windhoek")
)

# Assign sites based on latitude and longitude
weather$Site <- NA
for (mapping in site_mapping) {
  weather$Site[mapping$condition(weather)] <- mapping$site
}

# Calculate summary statistics
years <- unique(weather$YEAR)
number_of_years <- length(years)
```

Prepare display order according to latitude:

```
# Create a function to format latitude
format_latitude <- function(lat) {
  paste(abs(round(lat, 2)), ifelse(lat < 0, "S", "N"))
}

# Create and process sites_latitude data frame
sites_latitude <- data.frame(
  Site = unique(weather$Site),
  Latitude = as.numeric(unique(weather$LAT))
)

# Sort sites_latitude by descending latitude
sites_latitude <- sites_latitude[order(-sites_latitude$Latitude), ]

# Format latitude values
sites_latitude$Latitude <- sapply(sites_latitude$Latitude, format_latitude)

# calculate easy references to sites
sites <- sites_latitude$Site
number_of_sites <- length(sites)
```

Print summary:

```
cat("Number of sites:", number_of_sites, "\n")
```

```
Number of sites: 7
```

```
cat("Sites:", paste(sites, collapse = ", "), "\n")
```

```
Sites: Irkutsk, Cambridge, Rakhigarhi, Pearl Harbor, Windhoek, Sao Paulo, Hobart
```

```
cat("Number of years:", number_of_years, "\n")
```

```
Number of years: 24
```

```
cat("Years:", paste(range(years), collapse = " - "), "\n")
```

```
Years: 1984 - 2007
```

Compute statistics for each site and day of year:

```r
# Define summary statistics function
calculate_summary <- function(data, column) {
  c(mean = mean(data[[column]], na.rm = TRUE),
    sd = sd(data[[column]], na.rm = TRUE),
    max = max(data[[column]], na.rm = TRUE),
    min = min(data[[column]], na.rm = TRUE),
    error = qt(0.975, length(data[[column]]) - 1) *
      sd(data[[column]], na.rm = TRUE) /
      sqrt(length(data[[column]])))
}

# Initialize weather_summary as a data frame
weather_summary <- data.frame(
  Site = character(),
  dayOfYear = integer(),
  solarRadiation.mean = numeric(),
  solarRadiation.sd = numeric(),
  solarRadiation.max = numeric(),
  solarRadiation.min = numeric(),
  solarRadiation.error = numeric(),
  solarRadiationTop.mean = numeric(),
  temperature.mean = numeric(),
  temperature.sd = numeric(),
  temperature.max = numeric(),
  temperature.min = numeric(),
  temperature.error = numeric(),
  maxTemperature.mean = numeric(),
  maxTemperature.max = numeric(),
  maxTemperature.min = numeric(),
  maxTemperature.error = numeric(),
  minTemperature.mean = numeric(),
  minTemperature.max = numeric(),
  minTemperature.min = numeric(),
  minTemperature.error = numeric(),
  temperature.lowerDeviation = numeric(),
  temperature.lowerDeviation.error = numeric(),
  temperature.upperDeviation = numeric(),
  temperature.upperDeviation.error = numeric(),
```

```
  precipitation.mean = numeric(),
  precipitation.max = numeric(),
  precipitation.min = numeric(),
  precipitation.error = numeric()
)

# Pre-allocate the weather_summary data frame
total_rows <- length(sites) * 366
weather_summary <- weather_summary[rep(1, total_rows), ]

# Main loop
row_index <- 1
for (site in sites) {
  for (day in 1:366) {
    temp_data <- weather[weather$Site == site & weather$DOY == day, ]

    if (nrow(temp_data) == 0) next

    weather_summary[row_index, "Site"] <- site
    weather_summary[row_index, "dayOfYear"] <- day

    # Solar radiation
    solar_summary <- calculate_summary(temp_data, "ALLSKY_SFC_SW_DWN")
    weather_summary[row_index, c("solarRadiation.mean", "solarRadiation.sd",
                                 "solarRadiation.max", "solarRadiation.min",
                                 "solarRadiation.error")] <- solar_summary

    weather_summary[row_index, "solarRadiationTop.mean"] <- mean(temp_data$ALLSKY_TOA_SW_DWN

    # Temperature
    temp_summary <- calculate_summary(temp_data, "T2M")
    weather_summary[row_index, c("temperature.mean", "temperature.sd",
                                 "temperature.max", "temperature.min",
                                 "temperature.error")] <- temp_summary

    # Max temperature
    max_temp_summary <- calculate_summary(temp_data, "T2M_MAX")
    weather_summary[row_index, c("maxTemperature.mean", "maxTemperature.max",
                                 "maxTemperature.min", "maxTemperature.error")] <- max_temp_su

    # Min temperature
    min_temp_summary <- calculate_summary(temp_data, "T2M_MIN")
```

```
    weather_summary[row_index, c("minTemperature.mean", "minTemperature.max",
                                 "minTemperature.min", "minTemperature.error")] <- min_temp_su

    # Temperature deviations
    lower_dev <- temp_data$T2M - temp_data$T2M_MIN
    upper_dev <- temp_data$T2M_MAX - temp_data$T2M

    weather_summary[row_index, "temperature.lowerDeviation"] <- mean(lower_dev, na.rm = TRUE)
    weather_summary[row_index, "temperature.lowerDeviation.error"] <- qt(0.975, length(lower_
      sd(lower_dev, na.rm = TRUE) / sqrt(length(lower_dev))

    weather_summary[row_index, "temperature.upperDeviation"] <- mean(upper_dev, na.rm = TRUE)
    weather_summary[row_index, "temperature.upperDeviation.error"] <- qt(0.975, length(upper_
      sd(upper_dev, na.rm = TRUE) / sqrt(length(upper_dev))

    # Precipitation
    precip_summary <- calculate_summary(temp_data, "PRECTOT")
    weather_summary[row_index, c("precipitation.mean", "precipitation.max",
                                 "precipitation.min", "precipitation.error")] <- precip_summa

    row_index <- row_index + 1
  }
}

# Remove any unused rows
weather_summary <- weather_summary[1:(row_index-1), ]
```

Set colours for maximum and minimum temperature:

```
max_temperature_colour = hsv(7.3/360, 74.6/100, 70/100)

min_temperature_colour = hsv(232/360, 64.6/100, 73/100)
```

Create figure:

```
# Constants
YEAR_LENGTH <- 366
SOLSTICE_SUMMER <- 172  # June 21st (approx.)
SOLSTICE_WINTER <- 355  # December 21st (approx.)

# Helper functions
```

```r
round_to_multiple <- function(x, base, round_fn = round) {
  round_fn(x / base) * base
}

create_polygon <- function(x, y1, y2, alpha = 0.5, col = "black") {
  polygon(c(x, rev(x)), c(y1, rev(y2)), col = adjustcolor(col, alpha = alpha), border = NA)
}

plot_weather_variable <- function(x, y, ylim, lwd, col = "black", lty = 1) {
  plot(x, y, axes = FALSE, ylim = ylim, type = "l", lwd = lwd, col = col, lty = lty)
}

add_confidence_interval <- function(x, y_mean, error, col, alpha = 0.5) {
  create_polygon(x, y_mean + error, y_mean, alpha, col)
  create_polygon(x, y_mean - error, y_mean, alpha, col)
}

add_min_max_interval <- function(x, y_mean, y_min, y_max, col, alpha = 0.3) {
  create_polygon(x, y_max, y_mean, alpha, col)
  create_polygon(x, y_min, y_mean, alpha, col)
}

# Main plotting function
plot_weather_summary <- function(weather_summary, sites, sites_latitude, weather) {
  # Setup plot
  num_columns <- length(sites) + 1
  num_rows_except_bottom <- 4

  layout_matrix <- rbind(
    matrix(1:(num_columns * num_rows_except_bottom), nrow = num_rows_except_bottom, ncol = nu
    c((num_columns * num_rows_except_bottom) + 1, rep((num_columns * num_rows_except_bottom)
  )

  layout(layout_matrix,
         widths = c(3, 12, rep(10, length(sites) - 2), 14),
         heights = c(3, 10, 10, 12, 2))

  # Y-axis labels
  y_labs <- c(expression(paste("solar radiation (", MJ/m^-2, ")")),
              "temperature (C)", "precipitation (mm)")

  # Calculate ranges
```

```
range_solar <- c(
  round_to_multiple(min(weather_summary$solarRadiation.min), 5, floor),
  round_to_multiple(max(weather_summary$solarRadiationTop.mean), 5, ceiling)
)
range_temp <- c(
  round_to_multiple(min(weather_summary$minTemperature.min), 5, floor),
  round_to_multiple(max(weather_summary$maxTemperature.max), 5, ceiling)
)
range_precip <- c(
  round_to_multiple(min(weather_summary$precipitation.min), 5, floor),
  round_to_multiple(max(weather_summary$precipitation.max), 5, ceiling)
)

# Plot settings
par(cex = graphic_scale, cex.axis = graphic_scale * (0.8 + axis_text_rescale))

# First column: y axis titles
for (i in 1:4) {
  par(mar = c(0, 0, 0, 0.4))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  if (i > 1) {
    text(x = 0.5, y = 0.5, font = 4,
         cex = graphic_scale * (0.78 + font_rescale),
         srt = 90,
         labels = y_labs[i-1])
  }
}

# Plot for each site
for (site in sites) {
  temp_data <- weather_summary[weather_summary$Site == site,]

  left_plot_margin <- ifelse(site == sites[1], 2, 0.1)
  right_plot_margin <- ifelse(site == sites[length(sites)], 4, 0.1)

  # Site name + latitude
  par(mar = c(0.2, left_plot_margin, 0.1, right_plot_margin))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  text(x = 0.5, y = 0.5, font = 4,
       cex = graphic_scale * (0.7 + font_rescale),
       labels = paste(site, sites_latitude$Latitude[sites_latitude$Site == site], sep = "\
```

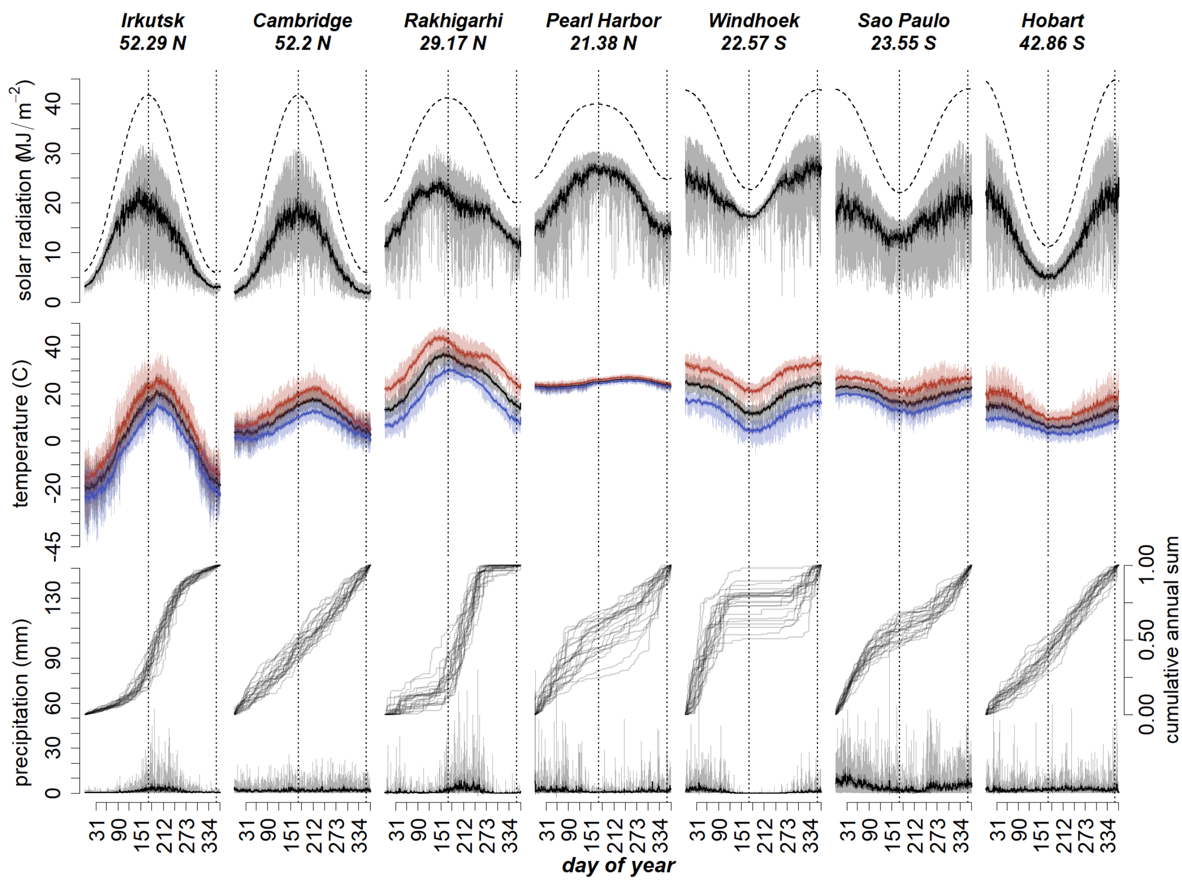
```
# Solar radiation
par(mar = c(0.1, left_plot_margin, 0.1, right_plot_margin))
plot_weather_variable(1:YEAR_LENGTH, temp_data$solarRadiation.mean, range_solar, graphic_
add_confidence_interval(1:YEAR_LENGTH, temp_data$solarRadiation.mean, temp_data$solarRadi
add_min_max_interval(1:YEAR_LENGTH, temp_data$solarRadiation.mean, temp_data$solarRadiati
lines(1:YEAR_LENGTH, temp_data$solarRadiationTop.mean, lty = 2, lwd = graphic_scale)

abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

if (site == sites[1]) {
  axis(2, at = seq(range_solar[1], range_solar[2], 5))
}

# Temperature
plot_weather_variable(1:YEAR_LENGTH, temp_data$temperature.mean, range_temp, graphic_scal
add_confidence_interval(1:YEAR_LENGTH, temp_data$temperature.mean, temp_data$temperature
add_min_max_interval(1:YEAR_LENGTH, temp_data$temperature.mean, temp_data$temperature.mir

lines(1:YEAR_LENGTH, temp_data$maxTemperature.mean, lwd = graphic_scale, col = max_temper
add_confidence_interval(1:YEAR_LENGTH, temp_data$maxTemperature.mean, temp_data$maxTemper
add_min_max_interval(1:YEAR_LENGTH, temp_data$maxTemperature.mean, temp_data$maxTemperatu

lines(1:YEAR_LENGTH, temp_data$minTemperature.mean, lwd = graphic_scale, col = min_temper
add_confidence_interval(1:YEAR_LENGTH, temp_data$minTemperature.mean, temp_data$minTemper
add_min_max_interval(1:YEAR_LENGTH, temp_data$minTemperature.mean, temp_data$minTemperatu

abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

if (site == sites[1]) {
  axis(2, at = seq(range_temp[1], range_temp[2], 5))
}

# Precipitation
par(mar = c(8, left_plot_margin, 0.1, right_plot_margin))
plot(c(1, YEAR_LENGTH), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt =

for (year in unique(weather$YEAR)) {
  site_year_data <- weather[weather$Site == site & weather$YEAR == year, ]
  lines(1:nrow(site_year_data),
        cumsum(site_year_data$PRECTOT) / sum(site_year_data$PRECTOT),
        lwd = graphic_scale,
        col = rgb(0, 0, 0, alpha = 0.2))
```

```r
    }

    if (site == sites[length(sites)]) {
      axis(4, at = seq(0, 1, 0.25))
      mtext("cumulative annual sum", 4, line = 2.5, cex = graphic_scale * (1.5 + margin_text_
    }

    par(new = TRUE, mar = c(3, left_plot_margin, 0.1, right_plot_margin))
    plot_weather_variable(1:YEAR_LENGTH, temp_data$precipitation.mean, range_precip, graphic_
    add_confidence_interval(1:YEAR_LENGTH, temp_data$precipitation.mean, temp_data$precipitat
    add_min_max_interval(1:YEAR_LENGTH, temp_data$precipitation.mean, temp_data$precipitation

    # Add solstices and axes
    abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

    if (site == sites[1]) {
      axis(2, at = seq(range_precip[1], range_precip[2], 10))
    }

    axis(1, at = cumsum(c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)), las = 2)
  }

  # Bottom row: "day of year" label
  par(mar = c(0, 0, 0, 0))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  text(x = 0.5, y = 0.7, font = 4,
       cex = graphic_scale * (0.8 + font_rescale),
       labels = "day of year")
}

# Main execution
plot_name <- file.path(output_dir, paste0("Fig1-annualWeatherVariablesExamples.", plot_file_

if (plot_file_format == "png") {
  graphic_scale <- 2
  font_rescale <- axis_text_rescale <- margin_text_rescale <- 0

  png(plot_name, width = number_of_sites * graphic_scale * 150, height = graphic_scale * 800)
} else if (plot_file_format == "eps") {
  graphic_scale = 1.2
  font_rescale = 0.1
```

```
  axis_text_rescale = -0.1
  margin_text_rescale = -0.5

  extrafont::loadfonts(device = "postscript")
  grDevices::cairo_ps(filename = plot_name ,
                      pointsize = 12,
                      width = number_of_sites * graphic_scale * 1.5,
                      height = graphic_scale * 8,
                      onefile = FALSE,
                      family = "sans"
                      )
}
plot_weather_summary(weather_summary, sites, sites_latitude, weather)
dev.off()
```

```
pdf
  2
```

```
knitr::include_graphics(plot_name)
```



Compute annual precipitation for each site and year:

```
# Initialize the result data frame
annual_precipitation <- data.frame(
  Site = character(),
  year = numeric(),
  precipitation.annual = numeric(),
  stringsAsFactors = FALSE
)

# Compute annual precipitation
for (site in sites) {
  for (year in years) {
    temp_data <- subset(weather, Site == site & YEAR == year)
    temp_data <- sum(temp_data$PRECTOT, na.rm = TRUE)

    annual_precipitation <- rbind(annual_precipitation,
```

```
                                  data.frame(Site = site,
                                             year = as.numeric(year),
                                             precipitation.annual = temp_data))
  }
}

# Clean up
rm(temp_data)
```

```
# Perform normality tests
normality_test_per_site <- lapply(sites, function(site) {
  site_data <- subset(annual_precipitation, Site == site)
  shapiro.test(site_data$precipitation.annual)
})
names(normality_test_per_site) <- sites

# Display results
print(head(annual_precipitation))
```

```
     Site year precipitation.annual
1 Irkutsk 1984               571.51
2 Irkutsk 1985               527.76
3 Irkutsk 1986               528.28
4 Irkutsk 1987               599.30
5 Irkutsk 1988               540.49
6 Irkutsk 1989               349.27
```

```
print(normality_test_per_site)
```

```
$Irkutsk

	Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.95701, p-value = 0.3813


$Cambridge

	Shapiro-Wilk normality test
```

```
data:  site_data$precipitation.annual
W = 0.98817, p-value = 0.9901


$Rakhigarhi

    Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.94736, p-value = 0.2373


$`Pearl Harbor`

    Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.90829, p-value = 0.03239


$Windhoek

    Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.93145, p-value = 0.105


$`Sao Paulo`

    Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.94245, p-value = 0.1849


$Hobart

    Shapiro-Wilk normality test

data:  site_data$precipitation.annual
W = 0.9853, p-value = 0.9704
```

Create figure:

```
# Helper functions
round_to_multiple <- function(x, base, round_fn = round) {
  round_fn(x / base) * base
}

plot_empty <- function() plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n

plot_y_axis_title <- function(label, x = 0.3, y = 0.5) {
  plot_empty()
  text(x = x, y = y, labels = label, srt = 90, font = 1, cex = graphic_scale * (0.78 + font_r
}

plot_x_axis_title <- function(label, x = 0.5, y = 0.3) {
  plot_empty()
  text(x = x, y = y, labels = label, font = 4,
       cex = graphic_scale * (0.78 + font_rescale))
}

plot_annualprecip_series <- function(precip_annual_dataframe, precip_annual_mean, precip_rang
  # Plot annual precipitation series
  par(mar = c(ifelse(is_last, 1, 0.2), 0.1, 0.1, 0.1))

  plot(precip_annual_dataframe$year, precip_annual_dataframe$precipitation.annual,
       ylim = precip_range + c(-0.1, 0.1) * diff(precip_range),
       type = 'l', lty = 1, lwd = graphic_scale, col = "black", xaxt = 'n', yaxt = 'n')

  # Add colored polygons
  polygon(c(precip_annual_dataframe$year, rev(precip_annual_dataframe$year)),
          c(pmax(precip_annual_dataframe$precipitation.annual, precip_annual_mean),
            rep(precip_annual_mean, nrow(precip_annual_dataframe))),
          col = rgb(0, 0, 0.8, alpha = 0.3), border = NA)
  polygon(c(precip_annual_dataframe$year, rev(precip_annual_dataframe$year)),
          c(pmin(precip_annual_dataframe$precipitation.annual, precip_annual_mean),
            rep(precip_annual_mean, nrow(precip_annual_dataframe))),
          col = rgb(0.8, 0, 0, alpha = 0.3), border = NA)

  abline(h = precip_annual_mean, lty = 2, col = "darkgrey")

  # Add site label and axes
  text(x = precip_annual_dataframe$year[1] - 0.03 * number_of_years,
       y = precip_range[1] + 0.04 * diff(precip_range),
```

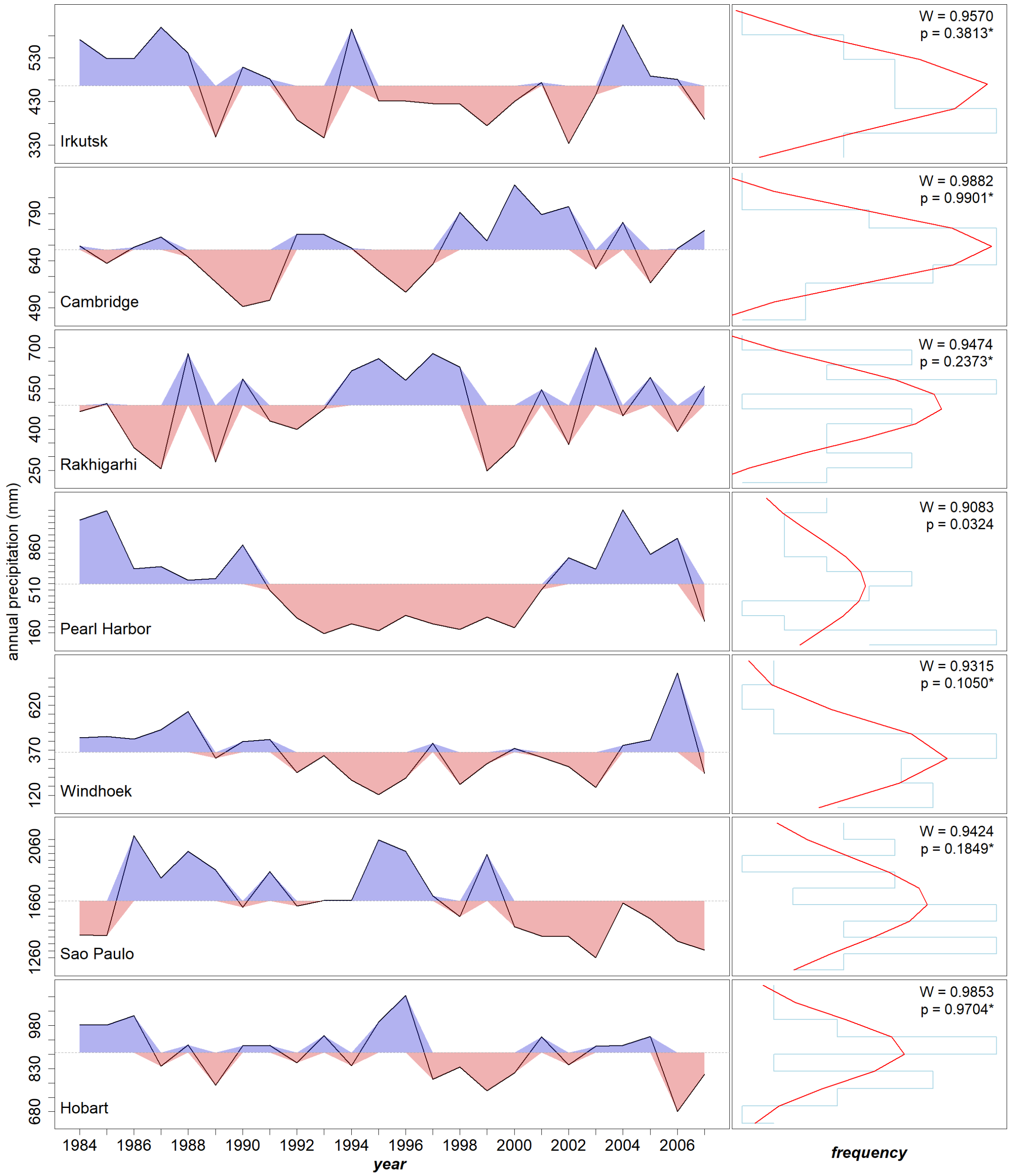
```
      labels = site_label, cex = graphic_scale * (0.8 + font_rescale), adj = 0)

  axis(2, at = seq(precip_range[1], precip_range[2], by = 50))
  if (is_last) axis(1, at = years)
}

plot_hist_and_normal <- function(precipitation_annual, is_last) {
  # Plot histogram, normal density model and Shapiro-Wilk test results
  par(mar = c(ifelse(is_last, 1, 0.2), 0.1, 0.1, 0.5))

  hist_data <- hist(precipitation_annual, breaks = 8, plot = FALSE)

  plot(hist_data$density, hist_data$mids, type = "s", lwd = 2, col = "lightblue", xaxt = 'n'

  # Add normal curve
  normal_curve <- dnorm(
    hist_data$mids,
    mean = mean(precipitation_annual, na.rm = TRUE),
    sd = sd(precipitation_annual, na.rm = TRUE))
  lines(normal_curve, hist_data$mids,
        col = "red", lwd = 2)

  # Add Shapiro-Wilk test results
  sw_test <- shapiro.test(precipitation_annual)
  text(x = 0.99 * max(hist_data$density),
      y = hist_data$breaks[1] + 0.85 * diff(range(hist_data$breaks)),
      labels = sprintf("W = %.4f\n p = %.4f%s",
                       sw_test$statistic, sw_test$p.value,
                       ifelse(sw_test$p.value > 0.05, "*", "")),
      cex = graphic_scale * (0.7 + font_rescale), adj = 1)
}

# Main plotting function
plot_annualprecip_summary <- function(annual_precipitation, sites, number_of_sites) {
  # Set up layout
  layout_matrix <- matrix(3:((2 * number_of_sites) + 4), nrow = number_of_sites + 1, ncol = 2

  layout_matrix <- cbind(c(rep(1, number_of_sites), 2), layout_matrix)

  layout(layout_matrix, widths = c(1, 12, 5), heights = c(rep(10, number_of_sites), 3))

  # Set global parameters
```

```r
  par(cex = graphic_scale, cex.axis = graphic_scale * (0.8 + axis_text_rescale))

  # Plot y-axis label
  par(mar = c(0, 0, 0, 0.4))
  plot_y_axis_title("annual precipitation (mm)")
  plot_empty()

  # Plot precipitation lines and histograms
  for (site in sites) {

    temp_data <- subset(annual_precipitation, Site == site)
    site_precipitation_mean <- mean(temp_data$precipitation.annual)
    is_last <- (site == sites[length(sites)])

    # Calculate plot ranges
    temp_range <- range(temp_data$precipitation.annual)
    temp_range <- round_to_multiple(temp_range, 10)

    # left plot
    plot_annualprecip_series(temp_data, site_precipitation_mean, temp_range, site, is_last)

    # right plot
    plot_hist_and_normal(temp_data$precipitation.annual, is_last)

    #if (is_last) axis(1, at = seq(0, round(max(hist_data$density), digits = 4), length.out =
  }
# Plot x-axis labels
plot_x_axis_title("year")
plot_x_axis_title("frequency", y = 0.7)
}

# Main execution

plot_name <- file.path(output_dir, paste0("Fig2-annualPrecipitationExamples.", plot_file_form

# Set up plot parameters and open device
if (plot_file_format == "png") {
  graphic_scale <- 2
  font_rescale <- axis_text_rescale <- margin_text_rescale <- 0
  png(plot_name, width = number_of_years * graphic_scale * 50, height = graphic_scale * numbe
} else if (plot_file_format == "eps") {
  graphic_scale <- 2
```

```
  font_rescale <- 0.2
  axis_text_rescale <- 0.2
  margin_text_rescale <- 0.2
  grDevices::cairo_ps(filename = plot_name, pointsize = 12,
                      width = number_of_years * graphic_scale * 1,
                      height = number_of_sites * graphic_scale * 4,
                      onefile = FALSE, family = "sans")
}
plot_annualprecip_summary(annual_precipitation, sites, number_of_sites)
dev.off()
```

```
pdf
  2
```

```
knitr::include_graphics(plot_name)
```


W = 0.9570
p = 0.3813*
Irkutsk
W = 0.9882
p = 0.9901*
Cambridge
W = 0.9474
p = 0.2373*
Rakhigarhi
W = 0.9083
p = 0.0324
Pearl Harbor
W = 0.9315
p = 0.1050*
Windhoek
W = 0.9424
p = 0.1849*
Sao Paulo
W = 0.9853
p = 0.9704*
Hobart
annual precipitation (mm)
year
frequency

3 Example of simulation outputs of the Weather model for 5 years

Choose file format for generated figures:

```
output_dir <- "output"
plot_file_format <- c("png", "eps")[1] # modify index number to change format
```

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
```

Initialisation using the default parametrisation, based on data from Rakhigarhi (example location, see Fig. 1):

```
SEED <- 0
YEAR_LENGTH <- 365 # ignoring leap year adjustment
NUM_YEARS <- 5
NUM_DAYS <- NUM_YEARS * YEAR_LENGTH

weather_model <- initialise_weather_model(seed = SEED, year_length = YEAR_LENGTH)
```

Show table with parameter values:

```
source("source/extract_params.R")

# Extract initial parameters
initial_params <- list(
  names = c("seed", "year_length", "albedo", "southern_hemisphere"),
  values = unlist(weather_model$PARAMS[1:4])
)

# Extract remaining parameters
remaining_params <- lapply(names(weather_model$PARAMS)[5:length(weather_model$PARAMS)],
```

```r
                            function(name) extract_params(weather_model$PARAMS[[name]], name))

# Combine all parameters
all_params <- list(
  names = c(initial_params$names, unlist(lapply(remaining_params, `[[`, "names"))),
  values = c(initial_params$values, unlist(lapply(remaining_params, `[[`, "values")))
)

# Create the table
params_values <- cbind(all_params$names, all_params$values)
row.names(params_values) <- NULL
knitr::kable(params_values,
             format = "html",
             col.names = c("parameter", "values"),
             align = c("l", "r"))
```

parameter	values
seed	0
year_length	365
albedo	0.4
southern_hemisphere	0
temperature - annual_max	40
temperature - annual_min	15
temperature - daily_fluctuation	5
temperature - daily_lower_dev	5
temperature - daily_upper_dev	5
solar - annual_max	7
solar - annual_min	3
solar - daily_fluctuation	1
precipitation - annual_sum_mean	400
precipitation - annual_sum_sd	130
precipitation - plateau_value_mean	0.1
precipitation - plateau_value_sd	0.05
precipitation - inflection1_mean	40
precipitation - inflection1_sd	20
precipitation - rate1_mean	0.15
precipitation - rate1_sd	0.02
precipitation - inflection2_mean	200
precipitation - inflection2_sd	20
precipitation - rate2_mean	0.05
precipitation - rate2_sd	0.01

parameter	values
precipitation - n__samples__mean	200
precipitation - n__samples__sd	5
precipitation - max__sample__size__mean	10
precipitation - max__sample__size__sd	3

Run model:

```
weather_model <- run_weather_model(weather_model, num_years = NUM_YEARS)
```

Set colours for maximum and minimum temperature:

```
max_temperature_colour = hsv(7.3/360, 74.6/100, 70/100)

min_temperature_colour = hsv(232/360, 64.6/100, 73/100)
```

Plot time-series:

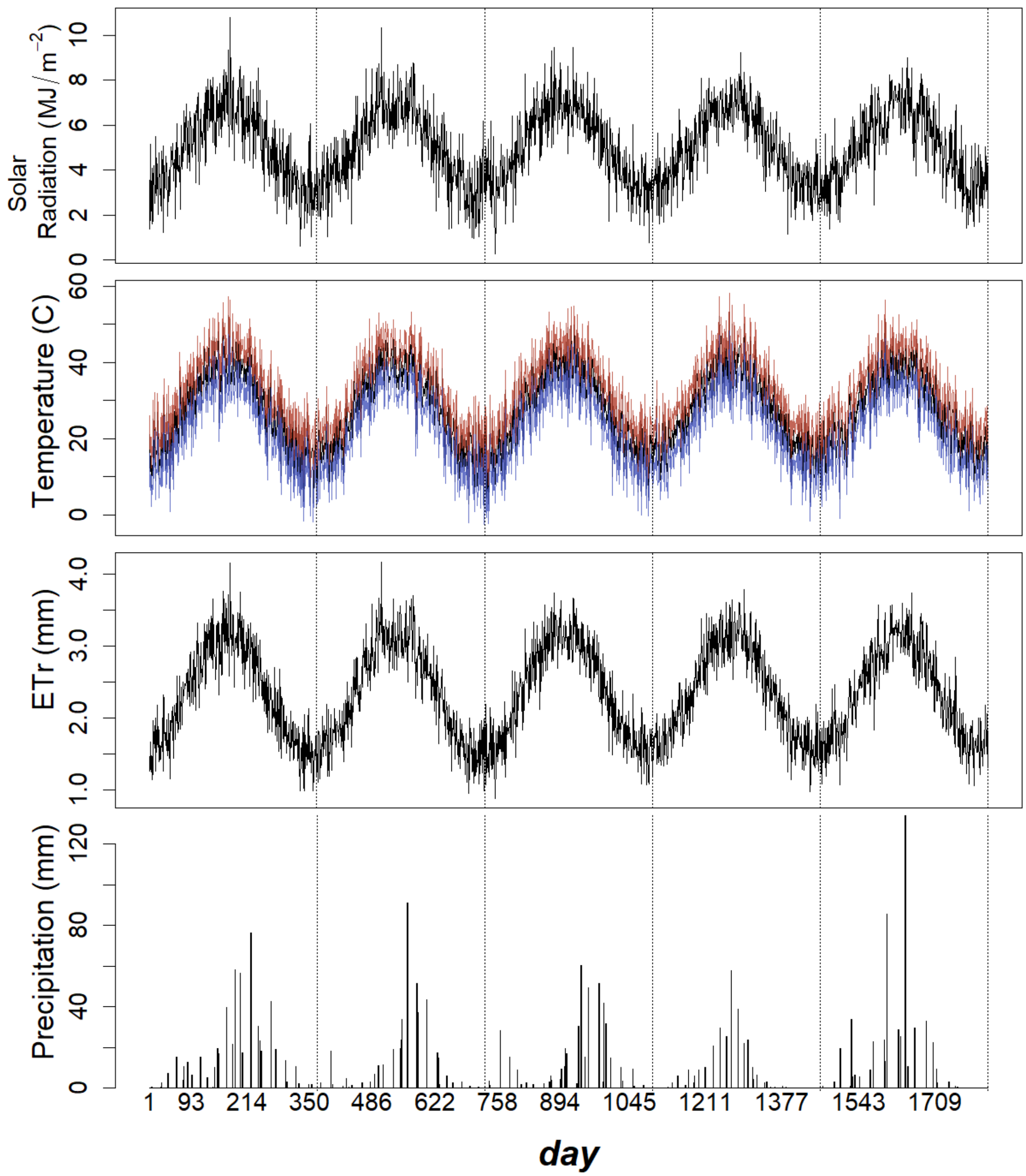
```
# Helper functions
plot_solar_radiation <- function(solar_radiation, num_days, year_length) {
    plot(1:num_days, solar_radiation,
         type = "l", xlab = "", xaxt = 'n', ylab = "")
    mark_end_years(num_days, year_length = year_length)
}

plot_temperature <- function(temperature, max_temperature_colour, min_temperature_colour, nu
    plot(1:num_days, temperature,
         type = "l", xlab = "", xaxt = 'n', ylab = "",
         ylim = c(floor(min(weather_model$daily$temperature_min)),
                  ceiling(max(weather_model$daily$temperature_max))))
    lines(1:num_days, weather_model$daily$temperature_max,
          col = adjustcolor(max_temperature_colour, alpha.f = 0.8))
    lines(1:num_days, weather_model$daily$temperature_min,
          col = adjustcolor(min_temperature_colour, alpha.f = 0.8))
    mark_end_years(num_days, year_length = year_length)
}

plot_ETr <- function(ETr, num_days, year_length) {
    plot(1:num_days, weather_model$daily$ETr, type = "l",
         ylab = "", xlab = "", xaxt = 'n')
```

```r
    mark_end_years(num_days, year_length = year_length)
}

plot_precipitation <- function(precipitation, num_days, year_length) {
    par(mar = c(2, 1, 0.1, 0.1))
    barplot(weather_model$daily$precipitation,
            ylab = "", xlab = "", xaxt = 'n')
    mark_end_years(num_days, year_length = year_length, offset = 1.2)
    abline(v = num_days * 1.2, lty = 3)
}

plot_time_axis <- function(num_days, graphic_scale, font_rescale, margin_text_rescale) {
    par(mar = c(1, 1, 0, 0.1))
    plot(c(1, num_days), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
    axis(3, at = 1:num_days, tck = 0, lwd = 0)
    mtext("day", side = 1, line = -1,
          font = 4, cex = graphic_scale * (1.7 + font_rescale + margin_text_rescale))
}

# Main plotting function
plot_weather_simulation <- function(weather_model, num_days, year_length, graphic_scale, fon
    layout(matrix(c(1:10),
                  nrow = 5, ncol = 2, byrow = FALSE),
           widths = c(1, 10),
           heights = c(10, 10, 10, 12, 2))

    y_labs <- c(expression(paste(
        "    Solar\nRadiation (", MJ/m^-2, ")")),
        "Temperature (C)", "ETr (mm)", "Precipitation (mm)")

    par(cex = graphic_scale)

    # First column
    par(mar = c(0, 0, 0, 0))
    for (i in 1:4) {
        plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
        text(x = 0.5, y = 0.5 + (i > 2) * 0.1, font = 4,
             cex = graphic_scale * (0.6 + 0.1 * (i > 1) + font_rescale),
             srt = 90,
             labels = y_labs[i])
    }
    plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
```

```r
    # Second column
    par(mar = c(0.2, 1, 0.5, 0.1), cex.axis = graphic_scale * (0.6 + axis_text_rescale))

    # 1: Solar radiation
    plot_solar_radiation(weather_model$daily$solar_radiation, num_days = num_days, year_leng
    # 2: Temperature
    plot_temperature(weather_model$daily$temperature, num_days = num_days, year_length = yea
                     max_temperature_colour = max_temperature_colour, min_temperature_colour
    # 3: Reference evapotranspiration
    plot_ETr(weather_model$daily$ETr, num_days = num_days, year_length = year_length)
    # 4: Precipitation
    plot_precipitation(weather_model$daily$precipitation, num_days = NUM_DAYS, year_length =

    # 5: x-axis title
    plot_time_axis(num_days = num_days, graphic_scale = graphic_scale, font_rescale = font_r
}

# Main execution
plot_name <- file.path(output_dir, paste0("Fig3-weather_modelExample.", plot_file_format))

if (plot_file_format == "png") {
  graphic_scale <- 2
  font_rescale <- axis_text_rescale <- margin_text_rescale <- 0
  png(plot_name, width = graphic_scale * 600, height = graphic_scale * 700)
} else if (plot_file_format == "eps") {
  graphic_scale <- 1.2
  font_rescale <- 0.1
  axis_text_rescale <- -0.1
  margin_text_rescale <- -0.5
  extrafont::loadfonts(device = "postscript")
  grDevices::cairo_ps(filename = plot_name, pointsize = 12,
                      width = graphic_scale * 6, height = graphic_scale * 7,
                      onefile = FALSE, family = "sans")
} else {
  stop("Unsupported file format")
}

plot_weather_simulation(weather_model, num_days = NUM_DAYS, year_length = YEAR_LENGTH,
                        graphic_scale = graphic_scale, font_rescale = font_rescale,
                        axis_text_rescale = axis_text_rescale, margin_text_rescale = margin_
                        max_temperature_colour = max_temperature_colour, min_temperature_colo
dev.off()
```

```
pdf
  2
```

```r
knitr::include_graphics(plot_name)
```



Solar Radiation (MJ/m^{-2})
Temperature (C)
ETr (mm)
Precipitation (mm)
day
1
93
214
350
486
622
758
894
1045
1211
1377
1543
1709

4 Demonstration of parameter variation: solar radiation and temperature

Choose file format for generated figures:

```
output_dir <- "output"
plot_file_format <- c("png", "eps")[1] # modify index number to change format
```

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
```

Set up six variations of parameter settings (i.e. `min_value`, `max_value`, `is_south_hemisphere`), assuming length of year of 365 days:

```
SEED <- 0
YEAR_LENGTH <- 365
is_southern_hemisphere_values <- c(FALSE, TRUE)

par_values_annual_sinusoid <- matrix(
  c(0.1, 1.5, 0.31,
    -0.5, 3.3, 0.73,
    1.5, 2.7, 0.06,
    2.1, 4.2, 0.25,
    -1.6, 5, 1,
    4, 4.5, 0.02),
  ncol = 3, byrow = TRUE
)

min_min_value <- min(par_values_annual_sinusoid[,1] - par_values_annual_sinusoid[,3])
max_max_value <- max(par_values_annual_sinusoid[,2] + par_values_annual_sinusoid[,3])

num_runs <- nrow(par_values_annual_sinusoid)
```

Create a colour palette for plotting:

```r
num_cold_colours <- num_runs %/% 2
num_warm_colours <- num_runs - num_cold_colours

create_color_sequence <- function(start, end, n) {
  seq(start, end, length.out = n)
}

create_color_values <- function(h_range, s_range, v_range, n) {
  cbind(
    h = create_color_sequence(h_range[1], h_range[2], n) / 360,
    s = create_color_sequence(s_range[1], s_range[2], n) / 100,
    v = create_color_sequence(v_range[1], v_range[2], n) / 100
  )
}

color_palette_values <- rbind(
  create_color_values(c(198.6, 299.4), c(61.6, 75.3), c(95.2, 76.4), num_cold_colours),
  create_color_values(c(5.15, 67.5), c(67, 77.8), c(73.7, 86.4), num_warm_colours)
)

color_palette <- apply(color_palette_values, 1, function(x) hsv(x[1], x[2], x[3]))
```

Plot curves:

```r
# Helper functions
plot_empty <- function() {
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
}

add_text <- function(x, y, label, cex_factor = 0.6, srt = 0) {
  text(x = x, y = y, labels = label, font = 4,
       cex = graphic_scale * (cex_factor + font_rescale), srt = srt)
}

# Main plotting function
plot_annual_sinusoid <- function(par_values_annual_sinusoid, is_southern_hemisphere_values, r
  layout(matrix(c(1,  2,  3, 12,
                  4,  5,  6, 12,
                  7,  8,  9, 12,
                  10, 11, 11, 12),
                nrow = 4, ncol = 4, byrow = TRUE),
         widths = c(1, 10, 10, 6),
```

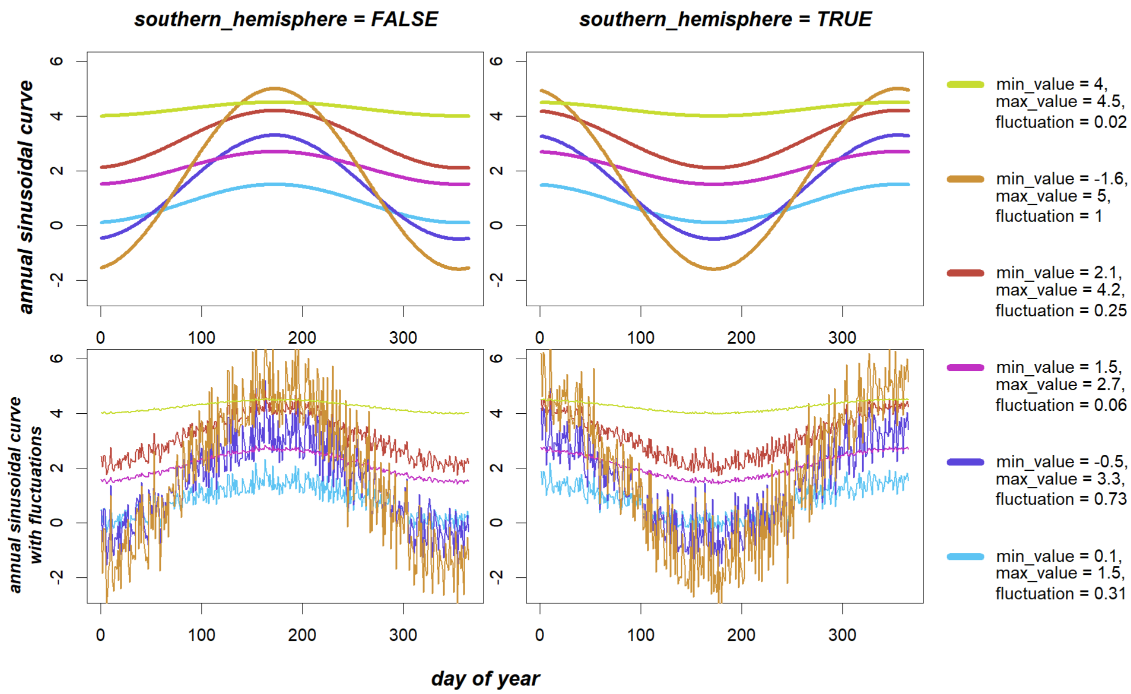
```r
       heights = c(2, 10, 10, 2))

  par(cex = graphic_scale * 1.2, mar = c(0, 0, 0, 0))

  # Titles
  plot_empty()
  for (hemisphere in c("FALSE", "TRUE")) {
    plot_empty()
    add_text(0.55, 0.5, paste("southern_hemisphere =", hemisphere))
  }

  # Y-axis titles and plots
  plot_empty()
  add_text(0.5, 0.5, "annual sinusoidal curve", srt = 90)

  par(mar = c(2, 2, 0.1, 0.1))

  for (is_southern_hemisphere in is_southern_hemisphere_values) {
    plot(c(1, YEAR_LENGTH), c(min_min_value, max_max_value), type = "n", xlab = "", ylab = ""

    for (i in 1:nrow(par_values_annual_sinusoid)) {
      curve <- gen_annual_sinusoid(
        min_value = par_values_annual_sinusoid[i, 1],
        max_value = par_values_annual_sinusoid[i, 2],
        year_length = YEAR_LENGTH,
        is_southern_hemisphere = is_southern_hemisphere)

      lines(1:length(curve), curve, col = color_palette[i], lwd = graphic_scale * 3)
    }
  }

  # Fluctuations
  par(mar = c(0, 0, 0, 0))

  plot_empty()
  add_text(0.5, 0.5, "annual sinusoidal curve\nwith fluctuations", cex_factor = 0.5, srt = 90

  par(mar = c(2, 2, 0.1, 0.1))

  for (is_southern_hemisphere in is_southern_hemisphere_values) {
    plot(c(1, YEAR_LENGTH), c(min_min_value, max_max_value), type = "n", xlab = "", ylab = ""
```

```
  for (i in 1:nrow(par_values_annual_sinusoid)) {
    curve <- gen_annual_sinusoid_with_fluctuation(
      min_value = par_values_annual_sinusoid[i, 1],
      max_value = par_values_annual_sinusoid[i, 2],
      year_length = YEAR_LENGTH,
      is_southern_hemisphere = is_southern_hemisphere,
      fluctuation = par_values_annual_sinusoid[i, 3],
      seed = SEED
    )

    lines(1:length(curve), curve, col = color_palette[i], lwd = graphic_scale * 1)
  }
}

par(mar = c(0, 0, 0, 0))

# X-axis title
plot_empty()
plot_empty()
add_text(0.5, 0.4, "day of year")

# Legend
plot(c(0, 1), c(0, nrow(par_values_annual_sinusoid) + 1), ann = F, bty = 'n', type = 'n',
x_pos <- 0.25
y_pos <- c(0.5, 0.1, -0.1)
jump <- 1

for (i in 1:nrow(par_values_annual_sinusoid)) {
  legend(x = 0, y = (y_pos[1] + jump * i),
         legend = substitute(paste("min_value = ", minValue, ","),
                             list(minValue = par_values_annual_sinusoid[i, 1])),
         col = color_palette[i], lwd = graphic_scale * 6,
         cex = graphic_scale * (0.5 + font_rescale), bty = "n")

  text(x = x_pos, y = (y_pos[2] + jump * i),
       labels = substitute(paste("max_value = ", max_value, ","),
                           list(max_value = par_values_annual_sinusoid[i, 2])),
       cex = graphic_scale * (0.5 + font_rescale), adj = 0)

  text(x = x_pos, y = (y_pos[3] + jump * i),
       labels = substitute(paste("fluctuation = ", fluctuation),
                           list(fluctuation = par_values_annual_sinusoid[i, 3])),
```

```r
         cex = graphic_scale * (0.5 + font_rescale), adj = 0)
  }
}

# Main execution
plot_name <- file.path(output_dir, paste0("Fig4-annualSinusoidCurve.", plot_file_format))

if (plot_file_format == "png") {
  graphic_scale <- 2
  font_rescale <- axis_text_rescale <- margin_text_rescale <- 0
  png(plot_name, width = graphic_scale * 1000, height = graphic_scale * 600)
} else if (plot_file_format == "eps") {
  graphic_scale <- 1.2
  font_rescale <- 0.1
  axis_text_rescale <- -0.1
  margin_text_rescale <- -0.5
  extrafont::loadfonts(device = "postscript")
  grDevices::cairo_ps(filename = plot_name, pointsize = 12,
                      width = graphic_scale * 10, height = graphic_scale * 6,
                      onefile = FALSE, family = "sans")
}
plot_annual_sinusoid(par_values_annual_sinusoid, is_southern_hemisphere_values, min_min_valu
dev.off()
```

```
pdf
  2
```

```r
knitr::include_graphics(plot_name)
```


southern_hemisphere = FALSE
southern_hemisphere = TRUE
annual sinusoidal curve
annual sinusoidal curve with fluctuations
day of year
min_value = 4, max_value = 4.5, fluctuation = 0.02
min_value = -1.6, max_value = 5, fluctuation = 1
min_value = 2.1, max_value = 4.2, fluctuation = 0.25
min_value = 1.5, max_value = 2.7, fluctuation = 0.06
min_value = -0.5, max_value = 3.3, fluctuation = 0.73
min_value = 0.1, max_value = 1.5, fluctuation = 0.31

5 Demonstration of parameter variation: precipitation

Choose file format for generated figures:

```
output_dir <- "output"
plot_file_format <- c("png", "eps")[1] # modify index number to change format
```

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
```

Set up six variations of parameter settings for the annual double logistic curve, discretisation, and annual precipitation, assuming a year length of 365 days. The random generator SEED used in discretisation is fixed:

```
SEED <- 0
YEAR_LENGTH <- 365

# Function to create parameter matrix
create_param_matrix <- function(x, ncol, nrow) {
  matrix(x, ncol = ncol, nrow = nrow, byrow = TRUE)
}

# Double logistic curve parameters
par_values_double_logistic <- create_param_matrix(c(
  0.01, 125, 0.3,  245, 0.22,
  0.15, 63,  0.55, 195, 0.6,
  0.5,  64,  0.05, 261, 0.12,
  0.45, 215, 0.01, 276, 0.39,
  0.6,  20,  0.38, 254, 0.04,
  0.85, 97,  0.24, 219, 0.17
  ), ncol = 5, nrow = 6
)
colnames(par_values_double_logistic) <- c("plateau_value", "inflection1", "rate1", "inflecti
```

```
# Discretisation parameters
par_values_discretisation <- create_param_matrix(c(
  152, 22,
  220, 10,
  240, 6,
  168, 13,
  191, 9,
  205, 17
  ), ncol = 2, nrow = 6
)
colnames(par_values_discretisation) <- c("n_samples", "max_sample_size")

annual_sum_values <- c(410, 1050, 636, 320, 1280, 745)

num_runs <- nrow(par_values_double_logistic)
```

Create a colour palette for plotting:

```
num_cold_colours <- num_runs %/% 2
num_warm_colours <- num_runs - num_cold_colours

create_color_sequence <- function(start, end, n) {
  seq(start, end, length.out = n)
}

create_color_values <- function(h_range, s_range, v_range, n) {
  cbind(
    h = create_color_sequence(h_range[1], h_range[2], n) / 360,
    s = create_color_sequence(s_range[1], s_range[2], n) / 100,
    v = create_color_sequence(v_range[1], v_range[2], n) / 100
  )
}

color_palette_values <- rbind(
  create_color_values(c(198.6, 299.4), c(61.6, 75.3), c(95.2, 76.4), num_cold_colours),
  create_color_values(c(5.15, 67.5), c(67, 77.8), c(73.7, 86.4), num_warm_colours)
)

color_palette <- apply(color_palette_values, 1, function(x) hsv(x[1], x[2], x[3]))
```

Plot curves:

```
# Helper functions
create_data_frame <- function(rows, cols) {
  data.frame(matrix(0, nrow = rows, ncol = cols))
}

create_plot <- function(x_range, y_range, ...) {
  plot(x_range, y_range, type = "n", xlab = "", ylab = "", ...)
}

add_text <- function(x, y, label, ...) {
  text(x = x, y = y, labels = label, ...)
}

draw_curve <- function(curve, color, ...) {
  lines(1:length(curve), curve, col = color, ...)
}

draw_points <- function(x, y, color, ...) {
  points(x, y, col = color, ...)
}

# Main plotting function
plot_annual_double_logistic <- function(par_values_double_logistic, par_values_discretisatio

  # Create data frames
  double_logistic_curves <- create_data_frame(YEAR_LENGTH, num_runs)
  discretised_double_logistic_curves <- create_data_frame(YEAR_LENGTH, num_runs)
  daily_precipitation <- create_data_frame(YEAR_LENGTH, num_runs)

  # Layout setup
  layout_matrix <- matrix(c(14, 14, 14, 14, 14, 17, 17,
                            1,   5,  5,  5,  5, 17, 17,
                            15, 15, 15, 15, 15, 17, 17,
                            2,   6,  6,  6,  6, 17, 17,
                            16, 16, 16, 16, 16, 17, 17,
                            3,   7,  8,  9, 10, 11, 12,
                            4,  13, 13, 13, 13, 13, 13),
                          nrow = 7, ncol = 7, byrow = TRUE)
  layout(layout_matrix,
         widths = c(2, rep(10, 6)),
         heights = c(4, 12, 4, 12, 4, 12, 1))
```

```
par(mgp = c(3, 0.4, 0), tcl = -0.4, cex = graphic_scale * 1.2)

# Y-axis titles
y_axis_titles <- c("daily cumulative value", "daily cumulative value", "daily increment")
for (i in 1:3) {
  par(mar = c(0, 0, 0, 0))
  create_plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', xaxt = 'n', yaxt = 'n')
  add_text(0.5, 0.5, y_axis_titles[i], font = 4,
           cex = graphic_scale * (0.7 + font_rescale), srt = 90)
}

# Empty plot
create_plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', xaxt = 'n', yaxt = 'n')

# Double logistic curves plot
par(mar = c(1, 1, 0.1, 1), cex.axis = graphic_scale * (0.5 + font_rescale))
create_plot(c(1, YEAR_LENGTH), c(0, 1))

for (i in 1:nrow(par_values_double_logistic)) {
  curve <- gen_annual_double_logistic_curve(
    plateau_value = par_values_double_logistic[i, 1],
    inflection1 = par_values_double_logistic[i, 2],
    rate1 = par_values_double_logistic[i, 3],
    inflection2 = par_values_double_logistic[i, 4],
    rate2 = par_values_double_logistic[i, 5],
    year_length = YEAR_LENGTH)

  draw_curve(curve, color_palette[i], lwd = graphic_scale * 3)
  draw_points(c(par_values_double_logistic[i, 2], par_values_double_logistic[i, 4]),
              c(curve[par_values_double_logistic[i, 2]], curve[par_values_double_logistic[i
              color_palette[i], pch = 19)

  double_logistic_curves[,i] <- curve
}

# Discretised double logistic plot
create_plot(c(1, YEAR_LENGTH), c(0, 1))

for (i in 1:nrow(par_values_double_logistic)) {
  curve <- discretise_curve(
    curve = double_logistic_curves[,i],
    n_samples = par_values_discretisation[i, 1],
```

```
    max_sample_size = par_values_discretisation[i, 2],
    seed = SEED)

  draw_curve(curve, adjustcolor(color_palette[i], alpha.f = 0.5), lwd = graphic_scale * 3)
  draw_points(c(par_values_double_logistic[i, 2], par_values_double_logistic[i, 4]),
              c(curve[par_values_double_logistic[i, 2]], curve[par_values_double_logistic[i
              adjustcolor(color_palette[i], alpha.f = 0.5), pch = 19)

  curve <- rescale_curve(curve)

  draw_curve(curve, color_palette[i], lwd = graphic_scale * 3)
  draw_points(c(par_values_double_logistic[i, 2], par_values_double_logistic[i, 4]),
              c(curve[par_values_double_logistic[i, 2]], curve[par_values_double_logistic[i
              color_palette[i], pch = 19)

  discretised_double_logistic_curves[,i] <- curve
}

# Daily precipitation plots
par(mar = c(2, 1, 0.1, 1), cex.axis = graphic_scale * (0.35 + axis_text_rescale))

daily_precipitation <- sapply(1:nrow(par_values_double_logistic), function(i) {
  get_increments_from_cumulative_curve(discretised_double_logistic_curves[,i]) * annual_sum
})

maxdaily_precipitation <- max(daily_precipitation)

for (i in nrow(par_values_double_logistic):1) {
  barplot(daily_precipitation[,i],
          names.arg = c("1", rep(NA, 98), "100", rep(NA, 99), "200", rep(NA, 99), "300", re
          ylim = c(0, maxdaily_precipitation),
          col = color_palette[i],
          border = color_palette[i])

  draw_points(c(par_values_double_logistic[i, 2], par_values_double_logistic[i, 4]),
              rep(maxdaily_precipitation * 0.9, 2),
              color_palette[i], pch = 19)

  abline(v = par_values_double_logistic[i, 2], col = color_palette[i], lty = 2)
  abline(v = par_values_double_logistic[i, 4], col = color_palette[i], lty = 2)
}
```

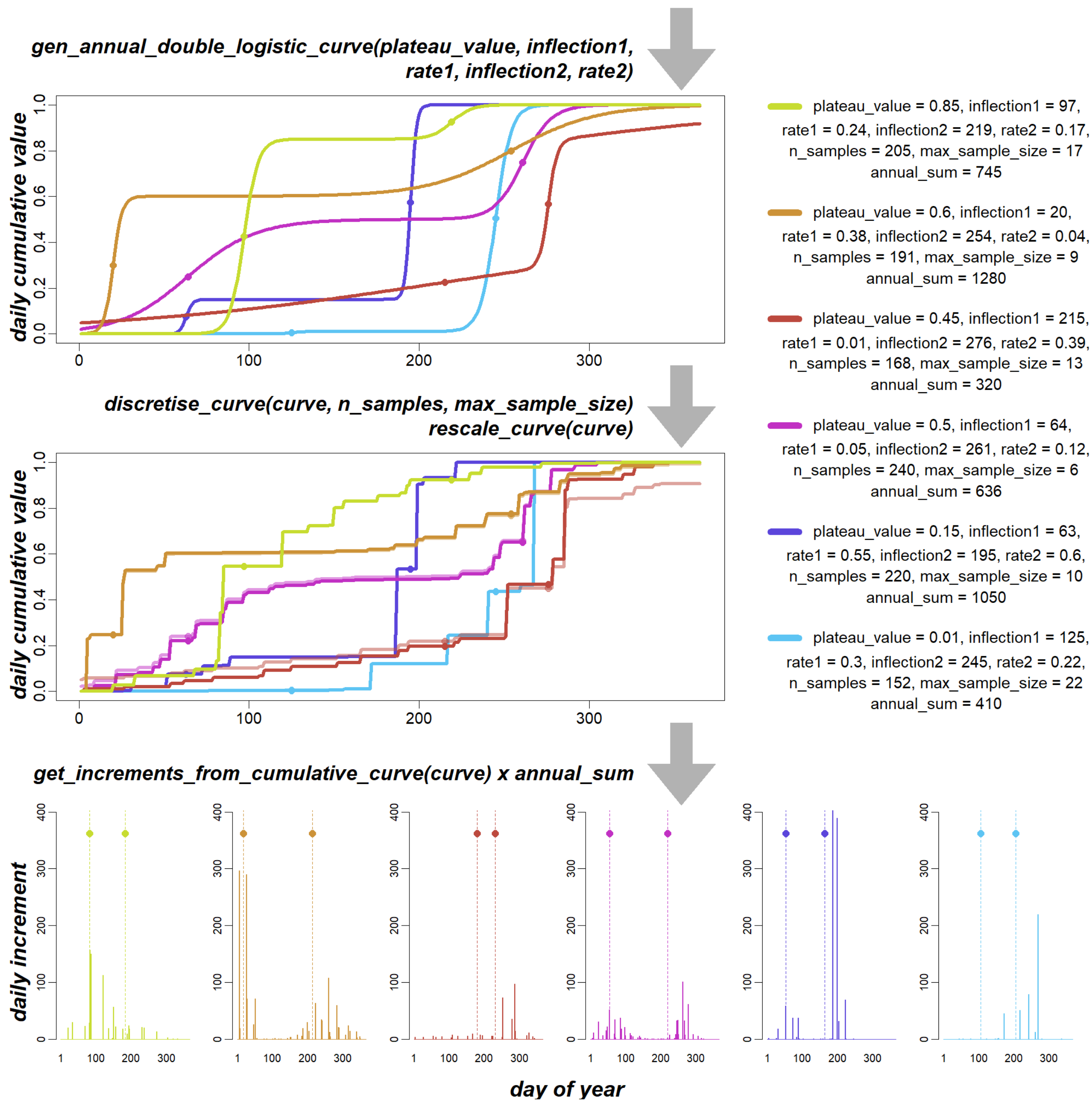
```
# X-axis title
par(mar = c(0, 0, 0, 0))
create_plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', xaxt = 'n', yaxt = 'n')
add_text(0.5, 0.4, "day of year", font = 4, cex = graphic_scale * (0.7 + font_rescale))

# Infographic bits
draw_infographic <- function(label) {
  create_plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', xaxt = 'n', yaxt = 'n')
  polygon(x = arrow_pos_x[1] + (arrow_pos_x[2] - arrow_pos_x[1]) * arrow_points_x,
          y = arrow_points_y,
          col = rgb(0,0,0, alpha = 0.3),
          border = NA)
  add_text(text_pos[1], text_pos[2],
           label, font = 4, cex = graphic_scale * (0.65 + font_rescale), adj = c(1, 0.5))
}

arrow_points_x <- c(1/3, 2/3, 2/3, 1, 0.5, 0, 1/3, 1/3)
arrow_points_y <- c(1, 1, 0.5, 0.5, 0, 0.5, 0.5, 1)
arrow_pos_x <- c(0.9, 1)
text_pos <- c(0.88, 0.4)

par(mar = c(0, 0, 0, 0))

infographic_labels <- c(
  "gen_annual_double_logistic_curve(plateau_value, inflection1,\nrate1, inflection2, rate2)
  "discretise_curve(curve, n_samples, max_sample_size)\nrescale_curve(curve)",
  "get_increments_from_cumulative_curve(curve) x annual_sum"
)

lapply(infographic_labels, draw_infographic)

# Legend
par(mar = c(0, 0, 0, 0))
create_plot(c(0, 1), c(0, nrow(par_values_double_logistic) + 1),
            ann = FALSE, bty = 'n', xaxt = 'n', yaxt = 'n')

y_pos <- c(0.5, seq(0.1, -0.3, length.out = 3))
x_pos <- 0.55
jump <- 1

for (i in 1:nrow(par_values_double_logistic)) {
  legend(x = 0,
```

```
           y = (y_pos[1] + jump * i),
           legend = substitute(
             paste("plateau_value = ", plateau_value, ", ",
                   "inflection1 = ", inflection1, ", "),
             list(plateau_value = par_values_double_logistic[i, 1],
                  inflection1 = par_values_double_logistic[i, 2])),
           col = color_palette[i],
           lwd = graphic_scale * 6, cex = graphic_scale * (0.5 + font_rescale),
           title = NULL,
           bty = "n")
    add_text(x_pos,
             (y_pos[2] + jump * i),
             substitute(
               paste("rate1 = ", rate1, ", ",
                     "inflection2 = ", inflection2, ", ",
                     "rate2 = ", rate2, ","),
               list(rate1 = par_values_double_logistic[i, 3],
                    inflection2 = par_values_double_logistic[i, 4],
                    rate2 = par_values_double_logistic[i, 5])),
             cex = graphic_scale * (0.5 + font_rescale))
    add_text(x_pos,
             (y_pos[3] + jump * i),
             substitute(
               paste("n_samples = ", n_samples, ", ",
                     "max_sample_size = ", max_sample_size),
               list(n_samples = par_values_discretisation[i, 1],
                    max_sample_size = par_values_discretisation[i, 2])),
             cex = graphic_scale * (0.5 + font_rescale))
    add_text(x_pos,
             (y_pos[4] + jump * i),
             substitute(
               paste("annual_sum = ", annual_sum),
               list(annual_sum = annual_sum_values[i])),
             cex = graphic_scale * (0.5 + font_rescale))
  }
}

# Main execution
plot_name <- file.path(output_dir, paste0("Fig5-annualDoubleLogisticCurve.", plot_file_forma

if (plot_file_format == "png") {
  graphic_scale <- 2
```

```
  font_rescale <- axis_text_rescale <-  0
  png(plot_name, width = graphic_scale * 1000, height = graphic_scale * 1000)
} else if (plot_file_format == "eps") {
  graphic_scale <- 1.2
  font_rescale <- 0.1
  axis_text_rescale <- 0.1
  extrafont::loadfonts(device = "postscript")
  grDevices::cairo_ps(filename = plot_name, pointsize = 12,
                      width = graphic_scale * 10, height = graphic_scale * 10,
                      onefile = FALSE, family = "sans")
}
plot_annual_double_logistic(par_values_double_logistic, par_values_discretisation, annual_sur
dev.off()
```

```
pdf
  2
```

```
knitr::include_graphics(plot_name)
```


gen_annual_double_logistic_curve(plateau_value, inflection1, rate1, inflection2, rate2)
daily cumulative value
discretise_curve(curve, n_samples, max_sample_size)
rescale_curve(curve)
get_increments_from_cumulative_curve(curve) x annual_sum
daily increment
day of year
plateau_value = 0.85, inflection1 = 97, rate1 = 0.24, inflection2 = 219, rate2 = 0.17, n_samples = 205, max_sample_size = 17 annual_sum = 745
plateau_value = 0.6, inflection1 = 20, rate1 = 0.38, inflection2 = 254, rate2 = 0.04, n_samples = 191, max_sample_size = 9 annual_sum = 1280
plateau_value = 0.45, inflection1 = 215, rate1 = 0.01, inflection2 = 276, rate2 = 0.39, n_samples = 168, max_sample_size = 13 annual_sum = 320
plateau_value = 0.5, inflection1 = 64, rate1 = 0.05, inflection2 = 261, rate2 = 0.12, n_samples = 240, max_sample_size = 6 annual_sum = 636
plateau_value = 0.15, inflection1 = 63, rate1 = 0.55, inflection2 = 195, rate2 = 0.6, n_samples = 220, max_sample_size = 10 annual_sum = 1050
plateau_value = 0.01, inflection1 = 125, rate1 = 0.3, inflection2 = 245, rate2 = 0.22, n_samples = 152, max_sample_size = 22 annual_sum = 410

6 Calibration walk through

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
source("source/print_parameter_comparison_table.R")
```

To generate new data based on a given dataset, we must first be able to estimate the Weather model parameters from said datasets. That is, to find the values of each parameter that can approximate the data of a given year daily series. Once this can be done for each year in the dataset, we can then estimate the hyperparameters as descriptive statistics (i.e., mean and standard deviation, minimum, maximum).

A good estimation of the parameters of the solar radiation and temperature submodels (i.e. sinusoid) can be made directly by measuring the year minimum and maximum.

However, the case of precipitation is far from trivial, given the complexity of the algorithm behind it. The workflow to estimate the parameters of the precipitation submodel deserves a demonstration.

6.1 Parameter estimation using `optim()`

Set up six variations of parameter settings of the annual double logistic curve (i.e. plateau_value, inflection1, rate1, inflection2, rate2), the discretisation producing the annual cumulative precipitation curve (i.e. n_samples, max_sample_size) and annualPrecipitation, assuming length of year of 365 days. Random generator seed used in discretisation is fixed:

```
# Fixed random seed for reproducibility
SEED <- 0

# Simulation parameters
YEAR_LENGTH <- 365

# Double logistic function parameters
params_values_double_logistic <- matrix(
  c(0.01, 125, 0.3,  245, 0.22,
```

```r
    0.15,  63, 0.55, 195, 0.6,
    0.5,   64, 0.05, 261, 0.12,
    0.45, 215, 0.01, 276, 0.39,
    0.6,   20, 0.38, 254, 0.04,
    0.85,  97, 0.24, 219, 0.17),
  nrow = 6,
  byrow = TRUE,
  dimnames = list(NULL, c("plateau_value", "inflection1", "rate1", "inflection2", "rate2"))
)

# Discretisation parameters
params_values_discretisation <- matrix(
  c(152, 22,
    220, 10,
    240,  6,
    168, 13,
    191,  9,
    205, 17),
  nrow = 6,
  byrow = TRUE,
  dimnames = list(NULL, c("n_samples", "max_sample_size"))
)

# Annual sum values
annual_sum_values <- c(410, 1050, 636, 320, 1280, 745)
```

Predefine the range of values explored for each parameter:

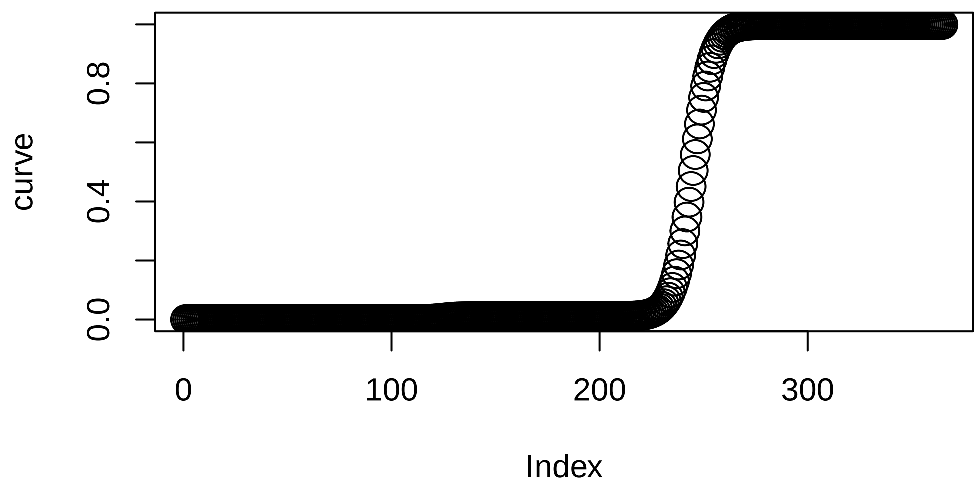
```r
params_range_lower <- c(0, 1, 0.01, 1, 0.01, 1, 3)
params_range_upper <- c(1, 365, 0.9, 365, 0.9, 365, 30)
```

6.1.1 Calibrating `gen_annual_double_logistic_curve()` (deterministic function)

Select the first set of parameter values from the `params_values_double_logistic` dataset and generate the corresponding curve with the `gen_annual_double_logistic_curve()` function. These points will represent the original state of the model that we aim to reverse engineer from the outcome curve. Plot it.

```r
original_params <- params_values_double_logistic[1, 1:5]

curve <- gen_annual_double_logistic_curve(
```

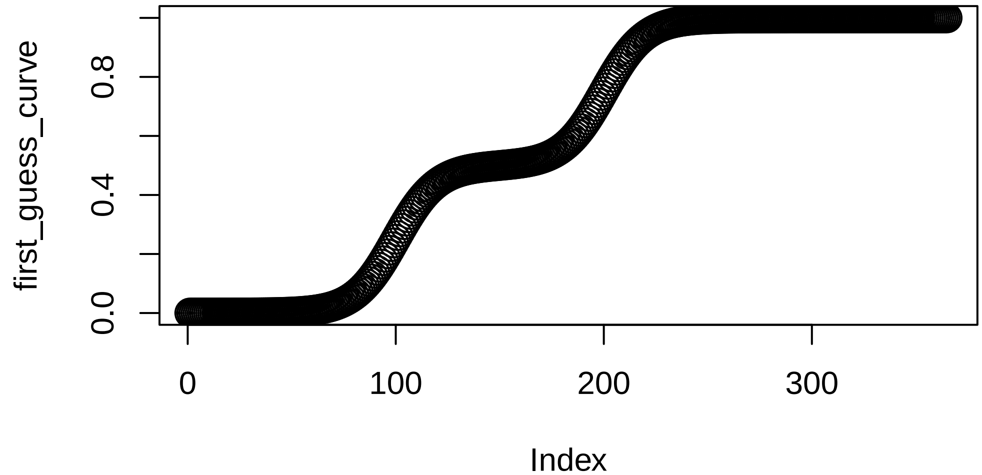
```
    plateau_value = original_params[1],
    inflection1 = original_params[2],
    rate1 = original_params[3],
    inflection2 = original_params[4],
    rate2 = original_params[5],
    year_length = YEAR_LENGTH)

plot(curve, cex = 2)
```



Define the `initial_guess` vector with your initial parameter guess values. Generate the curve using the `gen_annual_double_logistic_curve()` function with the initial guess. Plot it. Notice that our initial guess generates a somewhat "average" cumulative curve.

```
initial_guess <- c(0.5, 100, 0.1, 200, 0.1)  # Initial parameter guess

first_guess_curve <- gen_annual_double_logistic_curve(
    plateau_value = initial_guess[1],
    inflection1 = initial_guess[2],
    rate1 = initial_guess[3],
    inflection2 = initial_guess[4],
    rate2 = initial_guess[5],
    year_length = YEAR_LENGTH)
```

```
plot(first_guess_curve, cex = 2)
```



Define the `eval_objective_func()` function that calculates **the sum of squared differences between the observed data and the predicted values**, generated by the `gen_annual_double_logistic_curve()` function with a given parameter setting. Then, use the `optim()` function to estimate the best parameter values by minimizing the objective function.

NOTE: `optim()` using method "L-BFGS-B", see `?optim` or:

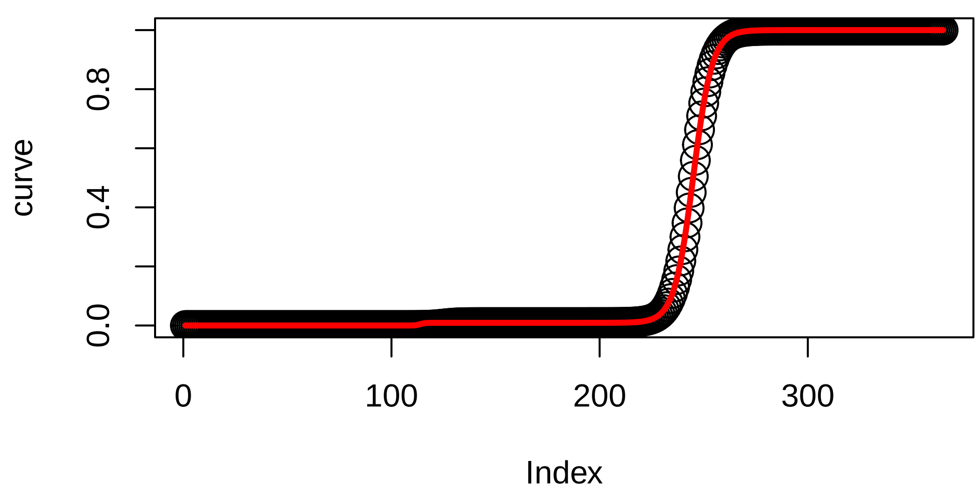
> Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing, 16, 1190–1208. doi:10.1137/0916069.

```
observed_data <- curve

# Objective function to minimize (difference between observed and predicted values or "residu
eval_objective_func <- function(params) {
  predicted_data <- gen_annual_double_logistic_curve(params[1], params[2], params[3], params
  sum((observed_data - predicted_data)^2)
}

# Use the least squares method to estimate the parameter values
```

```r
fit <- optim(initial_guess, eval_objective_func,
             method = "L-BFGS-B",
             lower = params_range_lower[1:5],
             upper = params_range_upper[1:5])

best_estimation_curve <- gen_annual_double_logistic_curve(fit$par[1], fit$par[2], fit$par[3]
```

Plot the original curve (curve) and overlay it with the curve generated using the best estimated parameter values (best_estimation_curve). The best estimated curve is shown in red.

```r
plot(curve, cex = 2)
lines(best_estimation_curve, col = 'red', lwd = 3)
```




```r
print_parameter_comparison_table(original_params, fit, params_range_upper[1:5], params_range_
```

	original	estimated	delta	range	delta (%)
plateau_value	0.01	0.0090998	0.000900	1.00	0.0900
inflection1	125.00	113.7955514	11.204449	364.00	3.0781
rate1	0.30	0.8999843	0.599984	0.89	67.4140
inflection2	245.00	244.9879056	0.012094	364.00	0.0033

	original	estimated	delta	range	delta (%)
rate2	0.22	0.2195381	0.000462	0.89	0.0519

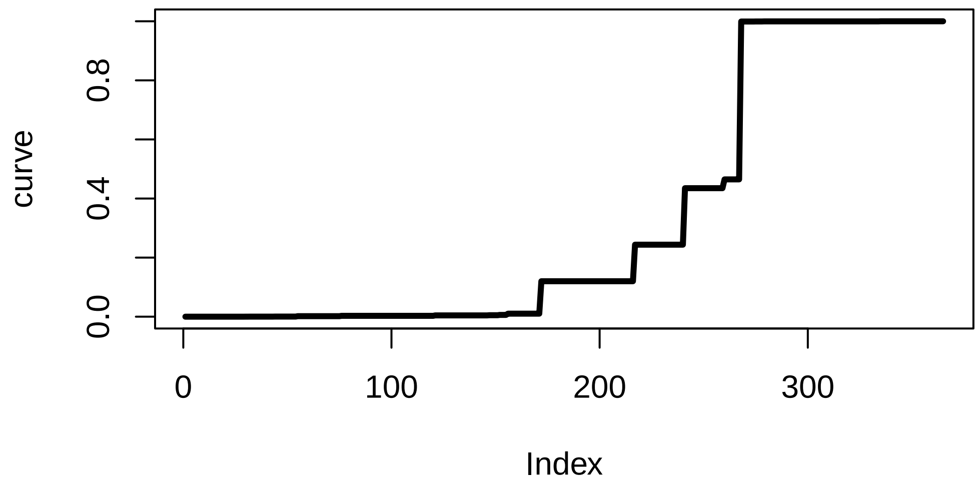
We can see that reverse engineering the parameter values of the double logistic curve is relatively straightforward. This specific curve offers a clear hint that `rate1` and `rate2` (i.e. the maximum growth rates of each logistic component) are harder to estimate when `plateau_value` is extreme .

6.1.2 Adding `discretise_curve()` (stochastic function)

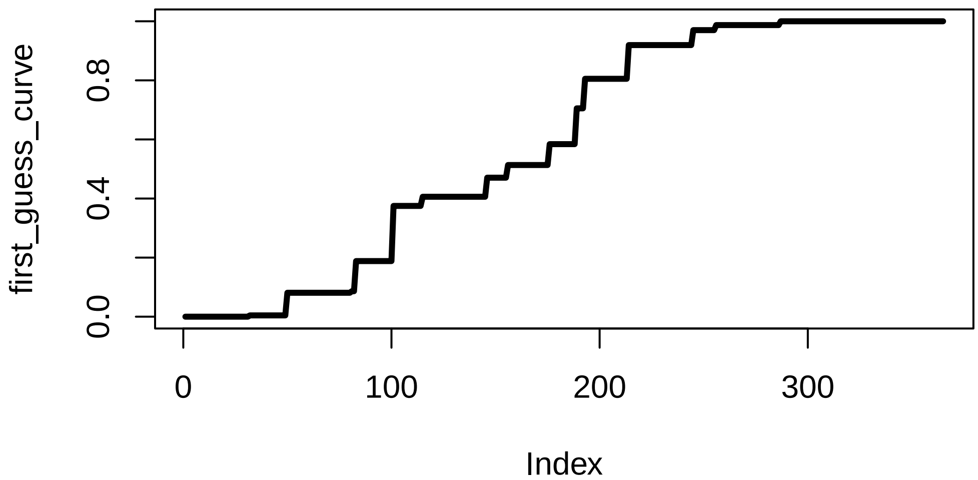
However, precipitation in the Weather model presents an additional challenge: the continuous cumulative curve is broken down into "steps" through `discretise_curve()`, which introduces **stochasticity**. We will also add `rescale_curve()` to the end of the process, in order to approach the curve that would be created by `generate_annual_precipitation()`.

Let us extend the workflow used above with `gen_annual_double_logistic_curve()` to also cover the two additional parameters of `discretise_curve()` (for now, fix `seed = 0`):

```
original_params <- c(params_values_double_logistic[1, 1:5], params_values_discretisation[1,

curve <- gen_cum_precipitation_of_year(
  plateau_value = original_params[1],
  inflection1 = original_params[2],
  rate1 = original_params[3],
  inflection2 = original_params[4],
  rate2 = original_params[5],
  year_length = YEAR_LENGTH,
  n_samples = original_params[6],
  max_sample_size = original_params[7],
  seed = SEED)

plot(curve, type = 'l', lwd = 3)
```



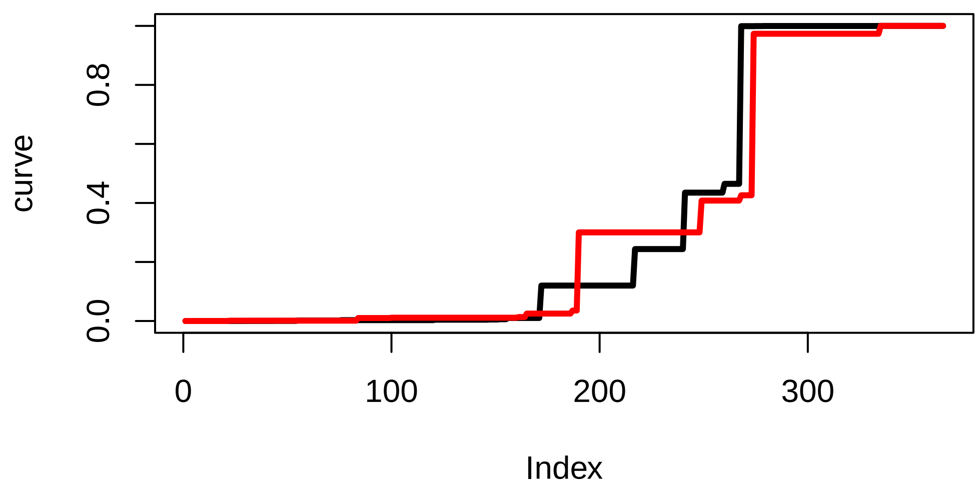

```
initial_guess <- c(0.5, 100, 0.1, 200, 0.1, 180, 15)  # Initial parameter guess

first_guess_curve <- gen_cum_precipitation_of_year(
    plateau_value = initial_guess[1],
    inflection1 = initial_guess[2],
    rate1 = initial_guess[3],
    inflection2 = initial_guess[4],
    rate2 = initial_guess[5],
    year_length = YEAR_LENGTH,
    n_samples = initial_guess[6],
    max_sample_size = initial_guess[7],
    seed = SEED)

plot(first_guess_curve, type = 'l', lwd = 3)
```




```
observed_data <- curve

# Objective function to minimize (difference between observed and predicted values)
eval_objective_func <- function(params) {
  predicted_data <- gen_cum_precipitation_of_year(
    plateau_value = params[1],
    inflection1 = params[2], rate1 = params[3],
    inflection2 = params[4], rate2 = params[5],
    year_length = YEAR_LENGTH,
    n_samples = params[6],
    max_sample_size = params[7],
    seed = SEED
    )

  sum((observed_data - predicted_data)^2)
}

# Use the least squares method to estimate the parameter values

fit <- optim(initial_guess, eval_objective_func,
             method = "L-BFGS-B",
             lower = params_range_lower,
             upper = params_range_upper)
```

```r
best_estimation_curve <- gen_cum_precipitation_of_year(
  plateau_value = fit$par[1],
  inflection1 = fit$par[2], rate1 = fit$par[3],
  inflection2 = fit$par[4], rate2 = fit$par[5],
  year_length = YEAR_LENGTH,
  n_samples = fit$par[6],
  max_sample_size = fit$par[7],
  seed = SEED
  )
```

```r
plot(curve, type = 'l', lwd = 3)
lines(best_estimation_curve, col = 'red', lwd = 3)
```




```r
print_parameter_comparison_table(original_params, fit, params_range_upper, params_range_lowe
```

	original	estimated	delta	range	delta (%)
plateau_value	0.01	0.0126065	0.002606	1.00	0.2606
inflection1	125.00	100.2542491	24.745751	364.00	6.7983
rate1	0.30	0.9000000	0.600000	0.89	67.4157
inflection2	245.00	250.3699407	5.369941	364.00	1.4753

	original	estimated	delta	range	delta (%)
rate2	0.22	0.9000000	0.680000	0.89	76.4045
n_samples	152.00	143.9652346	8.034765	364.00	2.2074
max_sample_size	22.00	30.0000000	8.000000	27.00	29.6296

Close, but a much worse fit than obtained with `gen_annual_double_logistic_curve()` only. We should take this performance in consideration going forward.

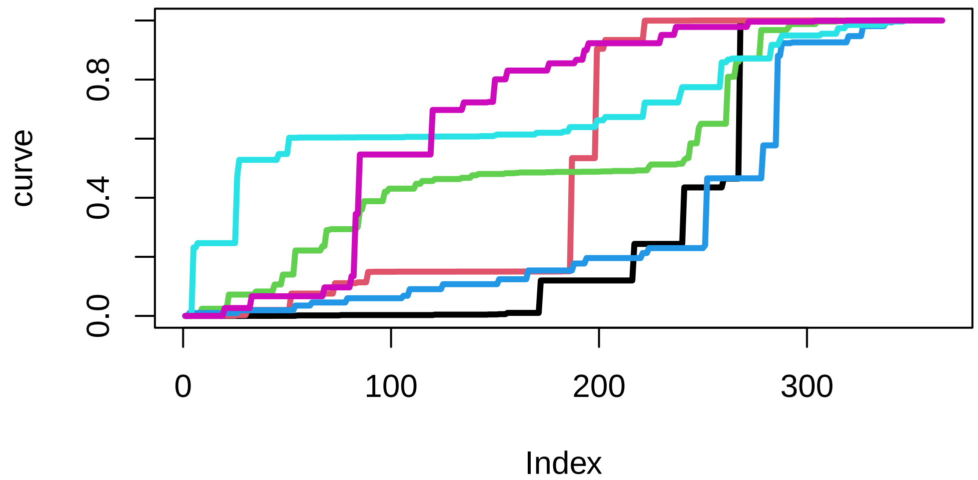
6.1.3 Calibrating multiple example curves to determine hyperparameters

Let us now apply the same workflow for estimating the hyperparameters able to generate an approximation of a sequence of year daily series.

First, generate the original dataset based on the different configurations present in `params_values_double_logistic` and `params_values_discretisation`:

```
curves <- list()
original_params_list <- list()

for (i in 1:nrow(params_values_double_logistic))
{
  original_params <- c(params_values_double_logistic[i, 1:5], params_values_discretisation[i

  curve <- gen_cum_precipitation_of_year(
    plateau_value = original_params[1],
    inflection1 = original_params[2],
    rate1 = original_params[3],
    inflection2 = original_params[4],
    rate2 = original_params[5],
    year_length = YEAR_LENGTH,
    n_samples = original_params[6],
    max_sample_size = original_params[7],
    seed = SEED
    )

  curves[[i]] <- curve
  original_params_list[[i]] <- original_params
}

plot(curves[[1]], type = 'l', col = 1, lwd = 3, ylab = 'curve')
for (i in 2:length(curves))
```

```
{
  lines(curves[[i]], col = i,  lwd = 3)
}
```



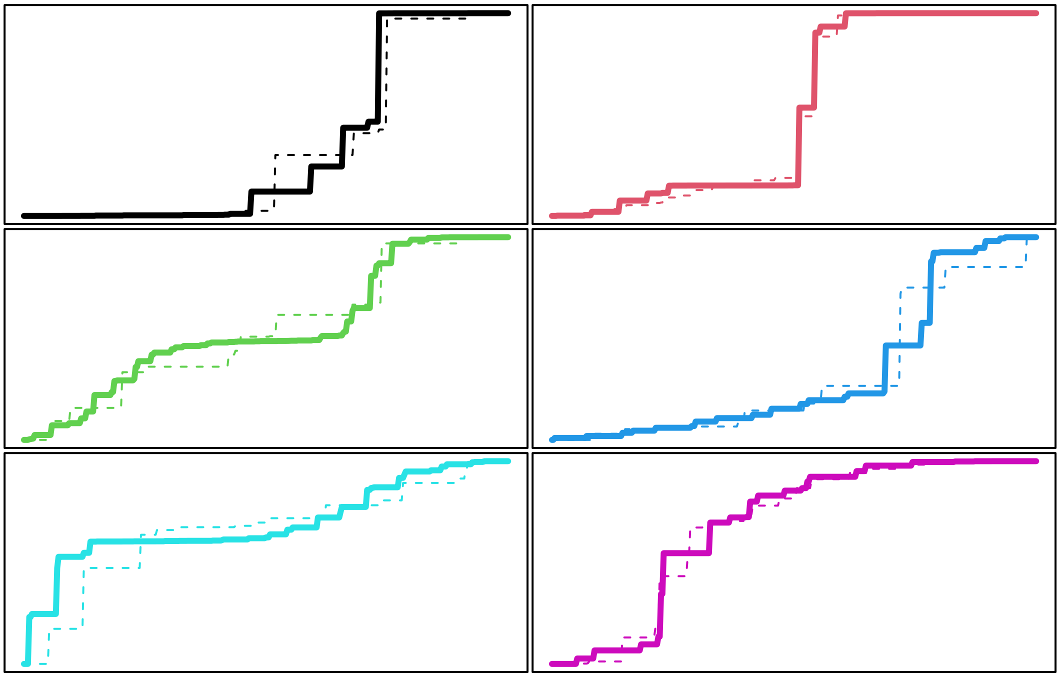
Apply `optim`, reusing `initial_guess` and `eval_objective_func`, to each curve and generate a sequence of best estimation curves:

```
best_estimation_curves <- list()
best_estimation_fits <- list()

for (i in 1:nrow(params_values_double_logistic))
{
  observed_data <- curves[[i]]

  # Use the least squares method to estimate the parameter values

  fit <- optim(initial_guess, eval_objective_func,
               method = "L-BFGS-B",
               lower = params_range_lower,
               upper = params_range_upper)

  best_estimation_curve <- gen_cum_precipitation_of_year(
    plateau_value = fit$par[1],
```

```
    inflection1 = fit$par[2], rate1 = fit$par[3],
    inflection2 = fit$par[4], rate2 = fit$par[5],
    year_length = YEAR_LENGTH,
    n_samples = fit$par[6],
    max_sample_size = fit$par[7],
    seed = SEED)

  best_estimation_curves[[i]] <- best_estimation_curve
  best_estimation_fits[[i]] <- fit
}
```

Plot original and estimated curves:

```
layout(matrix(1:6, nrow = 3, ncol = 2, byrow = TRUE))
par(mar = c(0.1, 0.1, 0.1, 0.1))
for (i in 1:length(curves)) {
  plot(curves[[i]], type = 'l', col = i, lwd = 3, xaxt = 'n', yaxt = 'n')
  lines(best_estimation_curves[[i]], col = i,  lty = 2)
}
```



Visualise the aggregate estimation quality:

```r
# Helper functions
calculate_mean_sd <- function(list_of_vectors) {
  if (length(list_of_vectors) == 0) return(list(mean = numeric(0), sd = numeric(0)))

  values_matrix <- do.call(rbind, list_of_vectors)

  list(
    mean = colMeans(values_matrix),
    sd = apply(values_matrix, 2, sd)
  )
}

get_list_params_from_fit <- function(list_of_fit_objects) {
  lapply(list_of_fit_objects, `[[`, "par")
}

# Main functions

create_parameter_comparison_summary <- function(original_params_list, fits, params_range_uppe
  original_summary <- calculate_mean_sd(original_params_list)
  estimated_summary <- calculate_mean_sd(get_list_params_from_fit(fits))

  data.frame(
    original_mean = round(original_summary$mean, digits = 4),
    original_sd = round(original_summary$sd, digits = 4),
    estimated_mean = round(estimated_summary$mean, digits = 4),
    estimated_sd = round(estimated_summary$sd, digits = 4),
    delta_mean = round(abs(original_summary$mean - estimated_summary$mean), digits = 6),
    delta_sd = round(abs(original_summary$sd - estimated_summary$sd), digits = 6),
    #range = params_range_upper - params_range_lower,
    delta_mean_percent = round(
      100 * abs(original_summary$mean - estimated_summary$mean) / (params_range_upper - param
      digits = 4
    ),
    delta_mean_percent = round(
      100 * abs(original_summary$sd - estimated_summary$sd) / (params_range_upper - params_ra
      digits = 4
    )
  )
}

print_parameter_comparison_summary_table <- function(parameter_comparison_summary) {
```

```
  knitr::kable(parameter_comparison_summary,
               format = "html",
               col.names = c(
                 "original (mean)", "original (sd)",
                 "estimated (mean)", "estimated (sd)",
                 "delta (mean)", "delta (sd)",
                 "delta (mean%)", "delta (sd%)"),
               align = c("c", "c", "c", "c", "c", "c", "c", "c"))
}

# Execution

parameter_comparison_summary <- create_parameter_comparison_summary(original_params_list, bes

print_parameter_comparison_summary_table(parameter_comparison_summary)
```

	original (mean)	original (sd)	estimated (mean)	estimated (sd)	delta (mean)	delta
plateau_value	0.4267	0.3051	0.5238	0.4174	0.097147	0.112
inflection1	97.3333	67.6481	64.9747	49.8331	32.358633	17.81
rate1	0.2550	0.2034	0.4256	0.4247	0.170560	0.221
inflection2	241.6667	29.6895	256.9584	60.9816	15.291717	31.29
rate2	0.2567	0.2050	0.4631	0.4623	0.206456	0.257
n_samples	196.0000	32.6741	159.1494	24.7352	36.850629	7.938
max_sample_size	12.8333	5.8452	21.6078	10.1365	8.774432	4.291

Although the original example curves are quite different from each other, let us assume that they correspond to cumulative precipitation of six years at a single location. This will allow us to test the `optim` calibration workflow on our ultimate target, the precipitation hyperparameters of the Weather model.

Initialise the weather model setting the precipitation hyperparameters as the mean and standard deviation of the best estimation parameter values in `parameter_comparison_summary`:

```
weather_model <- initialise_weather_model(
    seed = 0,
    precip_plateau_value_mean = parameter_comparison_summary["plateau_value", "estimated_mean
    precip_plateau_value_sd = parameter_comparison_summary["plateau_value", "estimated_sd"],
    precip_inflection1_mean = parameter_comparison_summary["inflection1", "estimated_mean"],
    precip_inflection1_sd = parameter_comparison_summary["inflection1", "estimated_sd"],
    precip_rate1_mean = parameter_comparison_summary["rate1", "estimated_mean"],
```

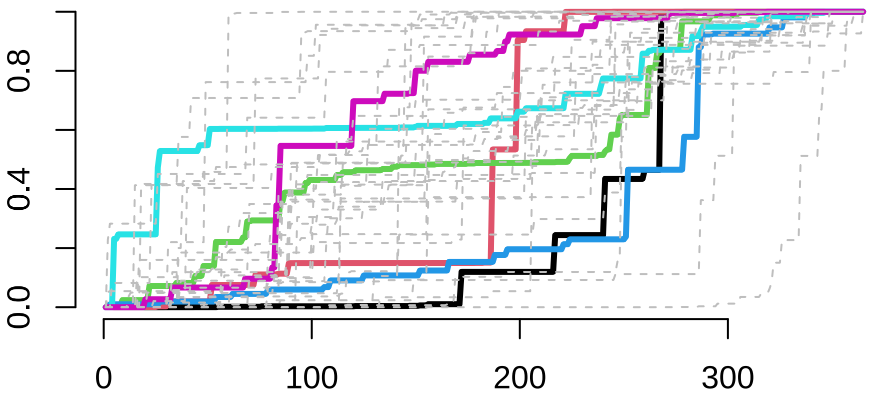
```
    precip_rate1_sd = parameter_comparison_summary["rate1", "estimated_sd"],
    precip_inflection2_mean = parameter_comparison_summary["inflection2", "estimated_mean"],
    precip_inflection2_sd = parameter_comparison_summary["inflection2", "estimated_sd"],
    precip_rate2_mean = parameter_comparison_summary["rate2", "estimated_mean"],
    precip_rate2_sd = parameter_comparison_summary["rate2", "estimated_sd"],
    precip_n_samples_mean = parameter_comparison_summary["n_samples", "estimated_mean"],
    precip_n_samples_sd = parameter_comparison_summary["n_samples", "estimated_sd"],
    precip_max_sample_size_mean = parameter_comparison_summary["max_sample_size", "estimated_
    precip_max_sample_size_sd = parameter_comparison_summary["max_sample_size", "estimated_sd
  )
```

Run the model to generate a number of cumulative curves:

```
weather_model <- run_weather_model(weather_model, num_years = 30, show_warnings = FALSE)
```

Plot original and generated curves:

```
plot(c(1, weather_model$PARAMS$year_length), c(0, 1), ann = F, bty = 'n', type = 'n', ylab =
# original curves
for (i in 1:length(curves))
{
  lines(curves[[i]], col = i,  lwd = 3)
}
# generated curves
for (year in unique(weather_model$daily$current_year))
{
  lines(1:weather_model$PARAMS$year_length,
        get_cumulative_precipitation_of_year(
          weather_model$daily$precipitation[
            weather_model$daily$current_year == year
          ]),
        col = "grey",
        lty = 2)
}
```


0.0
0.4
0.8
0
100
200
300

7 Calibration targeting weather examples

7.1 Preparation

Choose file format for generated figures:

```
output_dir <- "output"
plot_file_format <- c("png", "eps")[1] # modify index number to change format
```

Load source file containing the R implementation of the Weather model:

```
source("source/weatherModel.R")
source("source/estimate_hyperparameters_optim.R")
```

Set simulation constants:

```
SEED <- 0
YEAR_LENGTH <- 365 # ignoring leap year adjustment
SOLSTICE_SUMMER <- 172  # June 21st (approx.)
SOLSTICE_WINTER <- 355  # December 21st (approx.)
```

As a final part in this demonstration, we will extend the above process to deal with multiple instances of curves and parameter sets, generated by the same configuration of hyperparameters. We will then want to estimate those original hyperparameter values.

We use the data downloaded at NASA´s POWER access viewer (power.larc.nasa.gov/data-access-viewer/) selecting the user community 'Agroclimatology' and pin pointing the different locations between 01/01/1984 and 31/12/2007. The exact locations are:

- Rakhigarhi, Haryana, India (Latitude: 29.1687, Longitude: 76.0687)

- Irkutsk, Irkutsk Óblast, Russia (Latitude: 52.2891, Longitude: 104.2493)
- Hobart, Tasmania, Australia (Latitude: -42.8649, Longitude: 147.3441)
- Pearl Harbor, Hawaii, United States of America (Latitude: 21.376, Longitude: -157.9708)
- São Paulo, Brazil (Latitude: -23.5513, Longitude: -46.6344)
- Cambridge, United Kingdom (Latitude: 52.2027, Longitude: 0.122)

- Windhoek, Namibia (Latitude: -22.5718, Longitude: 17.0953)

We selected the ICASA Format's parameters:

- Precipitation (PRECTOT)
- Wind speed at 2m (WS2M)
- Relative Humidity at 2m (RH2M)
- Dew/frost point at 2m (T2MDEW)
- Maximum temperature at 2m (T2M_MAX)
- Minimum temperature at 2m (T2M_MIN)
- All sky insolation incident on a horizontal surface (ALLSKY_SFC_SW_DWN)
- Temperature at 2m (T2M)

and from Solar Related Parameters:

- Top-of-atmosphere Insolation (ALLSKY_TOA_SW_DWN)

```r
# Function to read and filter weather data
read_weather_data <- function(file_path) {
  data <- read.csv(file_path, skip = 18)
  data[data$YEAR %in% 1984:2007, ]
}

# Get input file paths
input_files <- list.files(path = "input", full.names = TRUE)

# Read and combine all weather data
weather <- do.call(rbind, lapply(input_files, read_weather_data))

# Define site mapping
site_mapping <- list(
  list(condition = function(x) floor(x$LAT) == 29, site = "Rakhigarhi"),
  list(condition = function(x) floor(x$LON) == 104, site = "Irkutsk"),
  list(condition = function(x) floor(x$LAT) == -43, site = "Hobart"),
  list(condition = function(x) floor(x$LAT) == 21, site = "Pearl Harbor"),
  list(condition = function(x) floor(x$LAT) == -24, site = "Sao Paulo"),
  list(condition = function(x) floor(x$LON) == 0, site = "Cambridge"),
```

```r
  list(condition = function(x) floor(x$LAT) == -23, site = "Windhoek")
)

# Assign sites based on latitude and longitude
weather$Site <- NA
for (mapping in site_mapping) {
  weather$Site[mapping$condition(weather)] <- mapping$site
}

# Calculate summary statistics
years <- unique(weather$YEAR)
number_of_years <- length(years)

# Calculate the yearly length in days
year_length_in_days <- as.integer(table(weather$YEAR) / nlevels(factor(weather$Site)))

year_length_max <- max(year_length_in_days)
```

Prepare display order according to latitude:

```r
# Create a function to format latitude
format_latitude <- function(lat) {
  paste(abs(round(lat, 2)), ifelse(lat < 0, "S", "N"))
}

# Create and process sites_latitude data frame
sites_latitude <- data.frame(
  Site = unique(weather$Site),
  Latitude = as.numeric(unique(weather$LAT))
)

# Sort sites_latitude by descending latitude
sites_latitude <- sites_latitude[order(-sites_latitude$Latitude), ]

# Format latitude values
sites_latitude$Latitude <- sapply(sites_latitude$Latitude, format_latitude)

# calculate easy references to sites
sites <- sites_latitude$Site
number_of_sites <- length(sites)
```

Compute statistics for each site and day of year:

```r
# Define summary statistics function
calculate_summary <- function(data, column) {
  c(mean = mean(data[[column]], na.rm = TRUE),
    sd = sd(data[[column]], na.rm = TRUE),
    max = max(data[[column]], na.rm = TRUE),
    min = min(data[[column]], na.rm = TRUE),
    error = qt(0.975, length(data[[column]]) - 1) *
      sd(data[[column]], na.rm = TRUE) /
      sqrt(length(data[[column]])))
}

# Initialize weather_summary as a data frame
weather_summary <- data.frame(
  Site = character(),
  dayOfYear = integer(),
  solarRadiation.mean = numeric(),
  solarRadiation.sd = numeric(),
  solarRadiation.max = numeric(),
  solarRadiation.min = numeric(),
  solarRadiation.error = numeric(),
  solarRadiationTop.mean = numeric(),
  temperature.mean = numeric(),
  temperature.sd = numeric(),
  temperature.max = numeric(),
  temperature.min = numeric(),
  temperature.error = numeric(),
  maxTemperature.mean = numeric(),
  maxTemperature.max = numeric(),
  maxTemperature.min = numeric(),
  maxTemperature.error = numeric(),
  minTemperature.mean = numeric(),
  minTemperature.max = numeric(),
  minTemperature.min = numeric(),
  minTemperature.error = numeric(),
  temperature.lowerDeviation = numeric(),
  temperature.lowerDeviation.error = numeric(),
  temperature.upperDeviation = numeric(),
  temperature.upperDeviation.error = numeric(),
  precipitation.mean = numeric(),
  precipitation.max = numeric(),
  precipitation.min = numeric(),
  precipitation.error = numeric()
```

```
)

# Pre-allocate the weather_summary data frame
total_rows <- length(sites) * 366
weather_summary <- weather_summary[rep(1, total_rows), ]

# Main loop
row_index <- 1
for (site in sites) {
  for (day in 1:366) {
    weather_site_day <- weather[weather$Site == site & weather$DOY == day, ]

    if (nrow(weather_site_day) == 0) next

    weather_summary[row_index, "Site"] <- site
    weather_summary[row_index, "dayOfYear"] <- day

    # Solar radiation
    solar_summary <- calculate_summary(weather_site_day, "ALLSKY_SFC_SW_DWN")
    weather_summary[row_index, c("solarRadiation.mean", "solarRadiation.sd",
                                 "solarRadiation.max", "solarRadiation.min",
                                 "solarRadiation.error")] <- solar_summary

    weather_summary[row_index, "solarRadiationTop.mean"] <- mean(weather_site_day$ALLSKY_TOA_

    # Temperature
    temp_summary <- calculate_summary(weather_site_day, "T2M")
    weather_summary[row_index, c("temperature.mean", "temperature.sd",
                                 "temperature.max", "temperature.min",
                                 "temperature.error")] <- temp_summary

    # Max temperature
    max_temp_summary <- calculate_summary(weather_site_day, "T2M_MAX")
    weather_summary[row_index, c("maxTemperature.mean", "maxTemperature.max",
                                 "maxTemperature.min", "maxTemperature.error")] <- max_temp_su

    # Min temperature
    min_temp_summary <- calculate_summary(weather_site_day, "T2M_MIN")
    weather_summary[row_index, c("minTemperature.mean", "minTemperature.max",
                                 "minTemperature.min", "minTemperature.error")] <- min_temp_su

    # Temperature deviations
```

```
    lower_dev <- weather_site_day$T2M - weather_site_day$T2M_MIN
    upper_dev <- weather_site_day$T2M_MAX - weather_site_day$T2M

    weather_summary[row_index, "temperature.lowerDeviation"] <- mean(lower_dev, na.rm = TRUE
    weather_summary[row_index, "temperature.lowerDeviation.error"] <- qt(0.975, length(lower_
      sd(lower_dev, na.rm = TRUE) / sqrt(length(lower_dev))

    weather_summary[row_index, "temperature.upperDeviation"] <- mean(upper_dev, na.rm = TRUE
    weather_summary[row_index, "temperature.upperDeviation.error"] <- qt(0.975, length(upper_
      sd(upper_dev, na.rm = TRUE) / sqrt(length(upper_dev))

    # Precipitation
    precip_summary <- calculate_summary(weather_site_day, "PRECTOT")
    weather_summary[row_index, c("precipitation.mean", "precipitation.max",
                                 "precipitation.min", "precipitation.error")] <- precip_summa

    row_index <- row_index + 1
  }
}

# Remove any unused rows
weather_summary <- weather_summary[1:(row_index-1), ]
```

7.2 Estimation of annual cumulative precipitation hyperparameters based on weather dataset

Declare auxiliary objects for estimating the precipitation cumulative curve with `optim`:

```
# Define the objective function for optimization
objective_function <- function(params, observed_data) {
  predicted_data <- gen_cum_precipitation_of_year(
    plateau_value = params[1],
    inflection1 = params[2], rate1 = params[3],
    inflection2 = params[4], rate2 = params[5],
    year_length = length(observed_data),
    n_samples = params[6],
    max_sample_size = params[7],
    seed = SEED
  )
```

```r
  sum((observed_data - predicted_data)^2)
}
```

7.2.1 Test an isolated version of the estimation of cumulative precipitation hyperparameters using `optim`

Prepare data for Cambridge site:

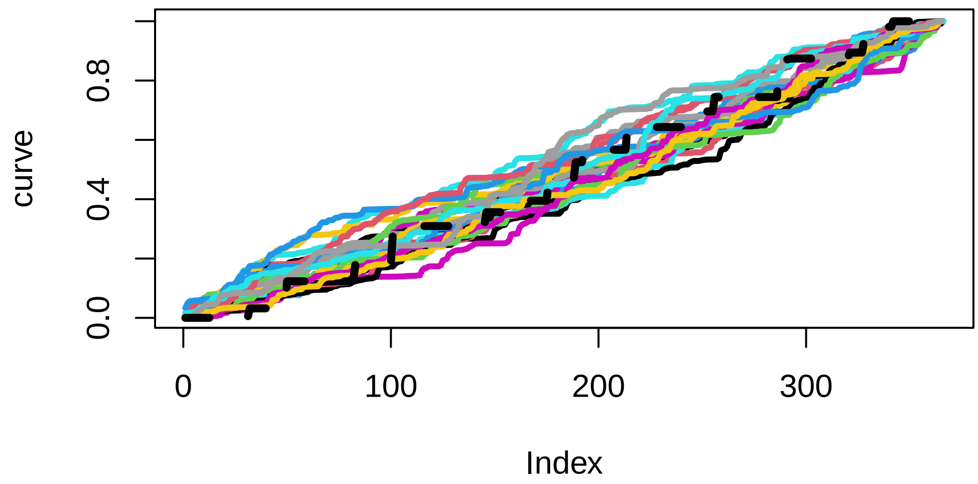
```r
cambridge_data <- subset(weather, Site == "Cambridge")
cum_precip <- get_cumulative_precipitation(
  daily_precipitation = cambridge_data$PRECTOT,
  years = cambridge_data$YEAR
)
cambridge_curves <- split(cum_precip, cambridge_data$YEAR)
```

Choose a good initial guess:

```r
cambridge_initial_guess <- c(0.5, 122, 0.005, 243, 0.005, 180, 15)

cambridge_initial_guess_curve <- gen_cum_precipitation_of_year(
    plateau_value = cambridge_initial_guess[1],
    inflection1 =  cambridge_initial_guess[2], rate1 = cambridge_initial_guess[3],
    inflection2 = cambridge_initial_guess[4], rate2 = cambridge_initial_guess[5],
    year_length = YEAR_LENGTH,
    n_samples = cambridge_initial_guess[6],
    max_sample_size = cambridge_initial_guess[7],
    seed = SEED
    )
```

Visually assess initial guess:

```r
plot(cambridge_curves[[1]], type = 'l', col = 1, lwd = 3, ylab = 'curve')
for (i in 2:length(cambridge_curves))
{
  lines(cambridge_curves[[i]], col = i,  lwd = 3)
}
lines(cambridge_initial_guess_curve, col = "black", lwd = 4, lty = 2)
```



Perform parameter estimation with best initial guess:

```
cambridge_estimation_result <- estimate_hyperparameters_optim(
  curves = cambridge_curves,
  objective_function = objective_function,
  method = "L-BFGS-B",
  lower = c(0, 1, 0.01, 1, 0.01, 1, 3),
  upper = c(1, 365, 0.9, 365, 0.9, 365, 30),
  initial_guess = cambridge_initial_guess
)
```

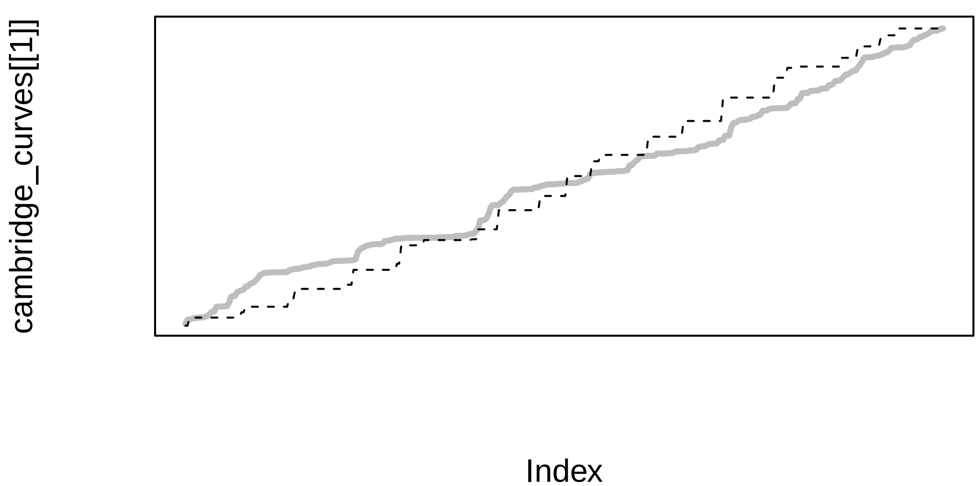
Use parameter estimations to generate curves for each year:

```
cambridge_best_estimation_curves <- list()

for (year in years)
{
  fit_year <- cambridge_estimation_result$curve_fits[[as.character(year)]]

  cambridge_best_estimation_curve <- gen_cum_precipitation_of_year(
    plateau_value = fit_year$par[1],
    inflection1 =  fit_year$par[2], rate1 = fit_year$par[3],
    inflection2 = fit_year$par[4], rate2 = fit_year$par[5],
```

```
    year_length = YEAR_LENGTH,
    n_samples = fit_year$par[6],
    max_sample_size = fit_year$par[7],
    seed = SEED
  )

  cambridge_best_estimation_curves[[as.character(year)]] <- cambridge_best_estimation_curve
}
```

Visualise fit for the first year:

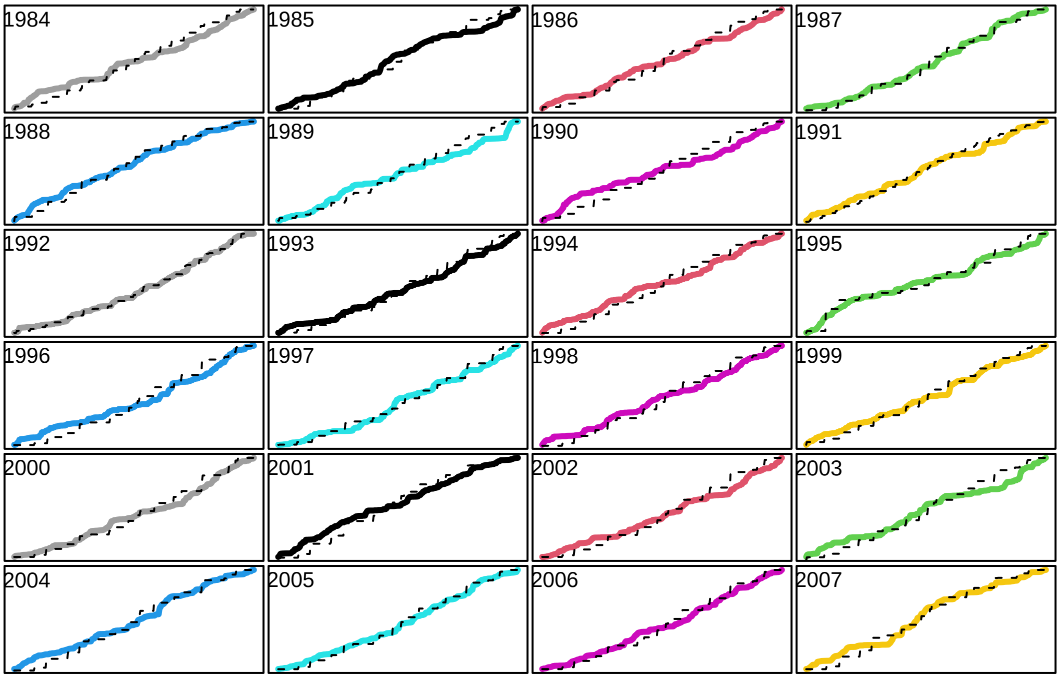
```
plot(cambridge_curves[[1]], type = 'l', col = "grey", lwd = 3, xaxt = 'n', yaxt = 'n')
lines(cambridge_best_estimation_curves[[1]],
      col = "black",
      lty = 2)
```



Visualise fit per year:

```
layout(matrix(1:length(cambridge_curves), nrow = 6, ncol = 4, byrow = TRUE))
par(mar = c(0.1, 0.1, 0.1, 0.1))
for (year in years) {
  plot(cambridge_curves[[as.character(year)]], type = 'l', col = as.character(year), lwd = 3
```

```
  lines(cambridge_best_estimation_curves[[as.character(year)]],
        col = "black",#as.character(year),
        lty = 2)
  text(as.character(year), x = 20, y = 0.9)
}
```



7.2.2 Run estimation of cumulative precipitation hyperparameters for all sites

Prepare data for all sites:

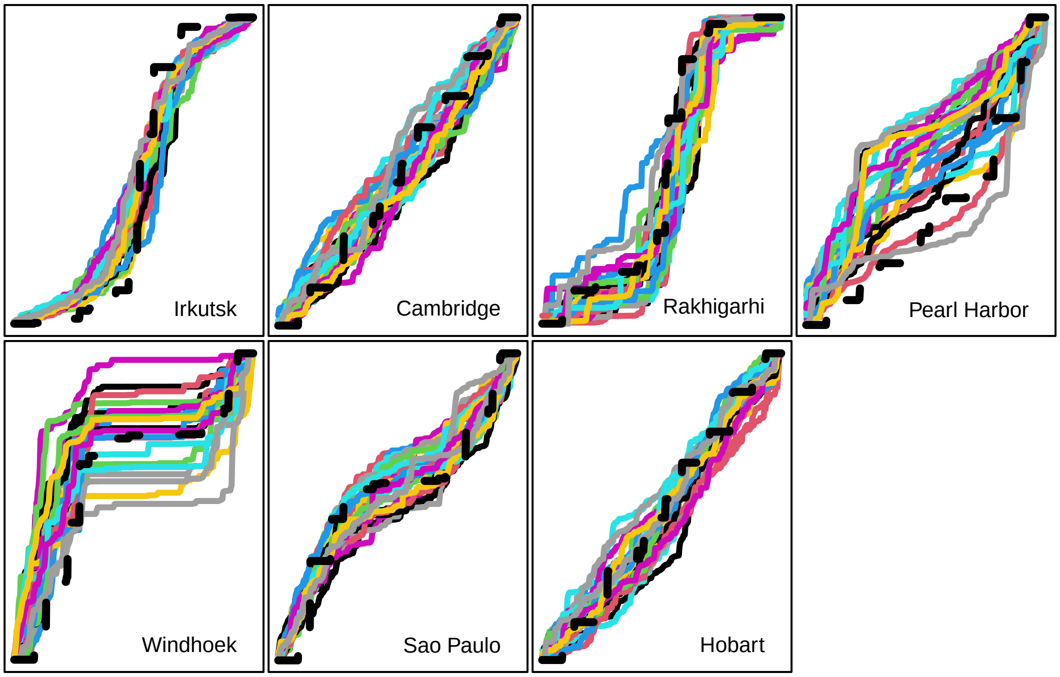
```
cum_precip_per_site <- setNames(lapply(sites, function(site){
  site_data <- subset(weather, Site == site)
  cum_precip <- get_cumulative_precipitation(
    daily_precipitation = site_data$PRECTOT,
    years = site_data$YEAR
  )
  site_curves <- split(cum_precip, site_data$YEAR)
}), sites)
```

Choose best initial guess per site:

```
initial_guesses <- setNames(lapply(sites, function(x) numeric(7)), sites)

initial_guesses[["Irkutsk"]] <- c(0.1, 60, 0.01, 200, 0.1, 180, 15)
initial_guesses[["Cambridge"]] <- c(0.5, 122, 0.005, 243, 0.005, 180, 15)
initial_guesses[["Rakhigarhi"]] <- c(0.2, 40, 0.1, 200, 0.1, 180, 15)
initial_guesses[["Pearl Harbor"]] <- c(0.8, 150, 0.005, 320, 0.1, 180, 15)
initial_guesses[["Windhoek"]] <- c(0.7, 80, 0.1, 330, 0.1, 180, 15)
initial_guesses[["Sao Paulo"]] <- c(0.6, 60, 0.1, 310, 0.1, 180, 15)
initial_guesses[["Hobart"]] <- c(0.5, 122, 0.005, 243, 0.005, 180, 15)

initial_guesses_curve <- lapply(initial_guesses, function(x) {
  gen_cum_precipitation_of_year(
    plateau_value = x[1],
    inflection1 =  x[2], rate1 = x[3],
    inflection2 = x[4], rate2 = x[5],
    year_length = YEAR_LENGTH,
    n_samples = x[6],
    max_sample_size = x[7],
    seed = SEED
  )
})
```

Visually assess initial guess:

```
layout(matrix(1:(length(sites)+1), nrow = 2, ncol = 4, byrow = TRUE))
par(mar = c(0.1, 0.1, 0.1, 0.1))

for (site in sites) {
  plot(cum_precip_per_site[[site]][[1]], type = 'l', col = 1, lwd = 3, xaxt = 'n', yaxt = 'n
  for (i in 2:length(cum_precip_per_site[[site]]))
  {
    lines(cum_precip_per_site[[site]][[i]], col = i,  lwd = 3)
  }
  lines(initial_guesses_curve[[site]], col = "black", lwd = 4, lty = 2)
  text(site, x = 340, y = 0.05, adj = 1)
}
plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
```



Perform parameter estimation for each site and year with best initial guess:

```
# Initialize an empty list to store results
estimation_results <- list()

# Iterate over all sites
for (site in sites)
{
  site_data <- subset(weather, Site == site)

  cum_precip <- get_cumulative_precipitation(
    daily_precipitation = site_data$PRECTOT,
    years = site_data$YEAR
    )

  curves <- split(cum_precip, site_data$YEAR)

  estimation_results[[site]] <- estimate_hyperparameters_optim(
    curves = curves,
    objective_function = objective_function,
    method = "L-BFGS-B",
    lower = c(0, 1, 0.01, 1, 0.01, 1, 3),
    upper = c(1, 365, 0.9, 365, 0.9, 365, 30),
```

```
    initial_guess = initial_guesses[[site]]
  )
}
```

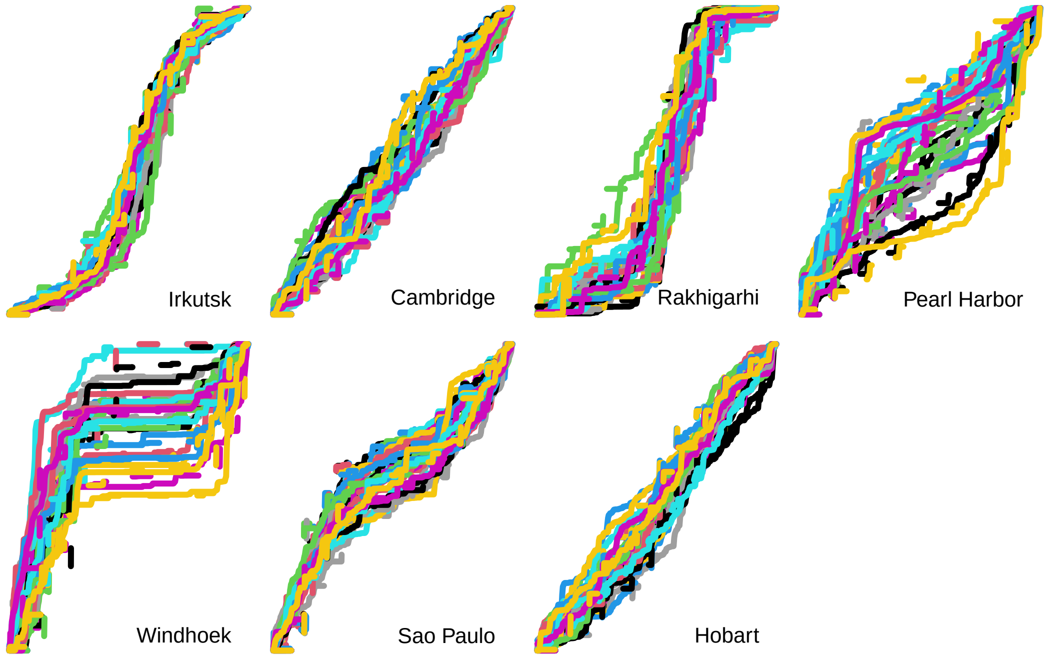
Use parameter estimations to generate curves for each site:

```
best_estimation_curves <- list()

for (site in sites)
{
  for (year in years)
  {
    fit_year <- estimation_results[[site]]$curve_fits[[as.character(year)]]

    best_estimation_curve <- gen_cum_precipitation_of_year(
      plateau_value = fit_year$par[1],
      inflection1 =  fit_year$par[2], rate1 = fit_year$par[3],
      inflection2 = fit_year$par[4], rate2 = fit_year$par[5],
      year_length = YEAR_LENGTH,
      n_samples = fit_year$par[6],
      max_sample_size = fit_year$par[7],
      seed = SEED
    )

    best_estimation_curves[[site]][[as.character(year)]] <- best_estimation_curve
  }
}
```

Visually assess fit of multiple years per site:

```
layout(matrix(1:(length(sites)+1), nrow = 2, ncol = 4, byrow = TRUE))
par(mar = c(0.1, 0.1, 0.1, 0.1))

for (site in sites) {
  plot(c(0, max(year_length_in_days)), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = '
  for (year in years)
  {
    lines(cum_precip_per_site[[site]][[as.character(year)]], col = year,  lwd = 3)
    lines(best_estimation_curves[[site]][[as.character(year)]], col = year, lwd = 3, lty = 2
  }
  text(site, x = 340, y = 0.05, adj = 1)
```

```
}
plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
```



Estimate a single parameter setting per site by averaging values per year:

```
best_estimation_fits_mean <- list()

for (site in sites)
{
  parameters_per_year <- list()
  for (year in years)
  {
    parameters_per_year[[as.character(year)]] <- estimation_results[[site]]$curve_fits[[as.cl
  }
  best_estimation_fits_mean[[site]]$mean <- apply(data.frame(parameters_per_year), 1, mean)
  best_estimation_fits_mean[[site]]$sd <- apply(data.frame(parameters_per_year), 1, sd)
}
```

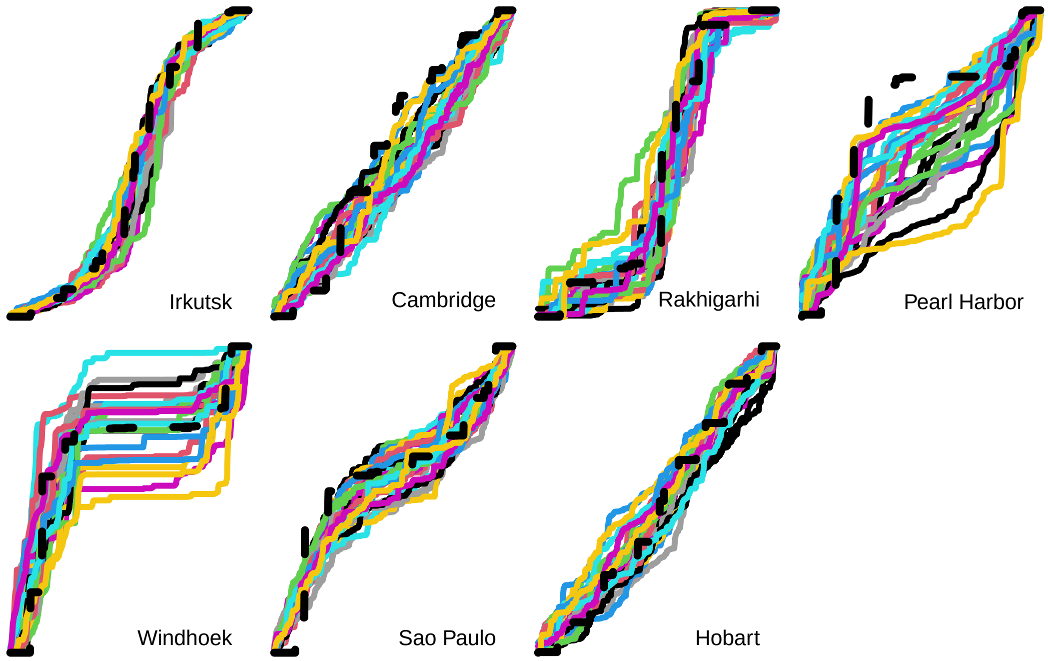
Use mean parameter estimations to generate curves for each site:

```
best_estimation_curves_mean <- list()
```

```
for (site in sites)
{
  fit_site <- best_estimation_fits_mean[[site]]$mean

  best_estimation_curve_mean <- gen_cum_precipitation_of_year(
    plateau_value = fit_site[1],
    inflection1 =  fit_site[2], rate1 = fit_site[3],
    inflection2 = fit_site[4], rate2 = fit_site[5],
    year_length = YEAR_LENGTH,
    n_samples = fit_site[6],
    max_sample_size = fit_site[7],
    seed = SEED
  )

  best_estimation_curves_mean[[site]] <- best_estimation_curve_mean
}
```

Visually assess fit of the single estimation per site:

```
layout(matrix(1:(length(sites)+1), nrow = 2, ncol = 4, byrow = TRUE))
par(mar = c(0.1, 0.1, 0.1, 0.1))

for (site in sites) {
  plot(c(0, max(year_length_in_days)), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = '
  for (year in years)
  {
    lines(cum_precip_per_site[[site]][[as.character(year)]], col = year,  lwd = 3)
  }
  lines(best_estimation_curves_mean[[site]], col = "black", lwd = 4, lty = 2)
  text(site, x = 340, y = 0.05, adj = 1)
}
plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
```




```r
best_estimation_fits_mean_table <- list()

for (site in sites)
{
  best_estimation_fits_mean_table[[site]] <- paste0(
    round(best_estimation_fits_mean[[site]]$mean, digits = 4),
    " (&PlusMinus;", round(best_estimation_fits_mean[[site]]$sd, digits = 4), ")")
}
best_estimation_fits_mean_table <- as.data.frame(best_estimation_fits_mean_table,
                  row.names = c("plateau value", "inflection1", "rate1", "inflection2", "rat

knitr::kable(best_estimation_fits_mean_table)
```

	Irkutsk	Cambridge	Rakhigarhi	Pearl.Harbor	Windhoek	Sao.Paulo	Hobart
plateau value	0.0935 (±0.1015)	0.3902 (±0.2624)	0.2267 (±0.1762)	0.783 (±0.19)	0.814 (±0.1738)	0.5705 (±0.1077)	0.3987 (±0.281)
inflection1	60.909 (±4.4393)	116.5566 (±24.6686)	34.8833 (±11.7029)	69.9943 (±69.6337)	27.9262 (±26.9712)	43.2431 (±23.0397)	122.0008 (±0.0077)
rate1	0.1268 (±0.2551)	0.0474 (±0.1816)	0.5323 (±0.3584)	0.1046 (±0.2554)	0.3787 (±0.3929)	0.3253 (±0.3593)	0.01 (±0)
inflection2	224.3544 (±9.2733)	262.4333 (±44.5082)	210.5208 (±11.3796)	325.012 (±12.8186)	333.2413 (±11.3875)	319.3132 (±14.9556)	253.1712 (±34.4429)

	Irkutsk	Cambridge	Rakhigarhi	Pearl.Harbor	Windhoek	Sao.Paulo	Hobart
rate2	0.0352 (±0.0279)	0.01 (±0)	0.3347 (±0.4006)	0.3834 (±0.4011)	0.4163 (±0.3817)	0.0292 (±0.0095)	0.0106 (±0.0012)
n_samples	178.7062 (±3.5977)	181.7599 (±1.9596)	178.646 (±10.3818)	179.9609 (±9.3249)	176.6509 (±19.3775)	182.5667 (±3.5898)	181.4085 (±4.7813)
max_sample_size	15.0996 (±3.9922)	13.0235 (±2.2968)	17.2914 (±4.5612)	13.2541 (±6.3016)	17.0167 (±5.7825)	16.5374 (±1.8574)	13.6198 (±2.5149)

This approach seems not to work well on `rate1` and `rate2` standard deviations, which are estimated too high within the relative scale of a logistic rate. For example, Windhoek gets 0.3787 (±0.3929), according to which a normal probability distribution would cover most of the 0-1 range. A similar problem occurs with the Pearl Harbor's and Windhoek's `inflection1` standard deviation.

Since the purpose is to test the potential fit of the Weather model, not the optimisation approach, we proceed to divide by a third all instances of standard deviations.

```
sd_adjustment <- 0.3

for (site in sites)
{
  best_estimation_fits_mean[[site]]$sd[3] <- best_estimation_fits_mean[[site]]$sd[3] * sd_adj
  best_estimation_fits_mean[[site]]$sd[5] <- best_estimation_fits_mean[[site]]$sd[5] * sd_adj
}
best_estimation_fits_mean[["Pearl Harbor"]]$sd[2] <- best_estimation_fits_mean[["Pearl Harbo
best_estimation_fits_mean[["Windhoek"]]$sd[2] <- best_estimation_fits_mean[["Windhoek"]]$sd[
```

```
best_estimation_fits_mean_table <- list()

for (site in sites)
{
  best_estimation_fits_mean_table[[site]] <- paste0(
    round(best_estimation_fits_mean[[site]]$mean, digits = 4),
    " (&PlusMinus;", round(best_estimation_fits_mean[[site]]$sd, digits = 4), ")")
}
best_estimation_fits_mean_table <- as.data.frame(best_estimation_fits_mean_table,
                     row.names = c("plateau value", "inflection1", "rate1", "inflection2", "rate

knitr::kable(best_estimation_fits_mean_table)
```

	Irkutsk	Cambridge	Rakhigarhi	Pearl.Harbor	Windhoek	Sao.Paulo	Hobart
plateau value	0.0935 (±0.1015)	0.3902 (±0.2624)	0.2267 (±0.1762)	0.783 (±0.19)	0.814 (±0.1738)	0.5705 (±0.1077)	0.3987 (±0.281)
inflection1	60.909 (±4.4393)	116.5566 (±24.6686)	34.8833 (±11.7029)	69.9943 (±20.8901)	27.9262 (±8.0914)	43.2431 (±23.0397)	122.0008 (±0.0077)
rate1	0.1268 (±0.0765)	0.0474 (±0.0545)	0.5323 (±0.1075)	0.1046 (±0.0766)	0.3787 (±0.1179)	0.3253 (±0.1078)	0.01 (±0)
inflection2	224.3544 (±9.2733)	262.4333 (±44.5082)	210.5208 (±11.3796)	325.012 (±12.8186)	333.2413 (±11.3875)	319.3132 (±14.9556)	253.1712 (±34.4429)
rate2	0.0352 (±0.0084)	0.01 (±0)	0.3347 (±0.1202)	0.3834 (±0.1203)	0.4163 (±0.1145)	0.0292 (±0.0028)	0.0106 (±4e-04)
n_samples	178.7062 (±3.5977)	181.7599 (±1.9596)	178.646 (±10.3818)	179.9609 (±9.3249)	176.6509 (±19.3775)	182.5667 (±3.5898)	181.4085 (±4.7813)
max_sample_size	15.0996 (±3.9922)	13.0235 (±2.2968)	17.2914 (±4.5612)	13.2541 (±6.3016)	17.0167 (±5.7825)	16.5374 (±1.8574)	13.6198 (±2.5149)

7.3 Running the entire Weather model using all estimated parameters

Calculate yearly summary statistics matching parameter inputs for each example location:

```
# Define summary function for a single site
calculate_site_summary <- function(site_data) {
  # Daily aggregated statistics
  daily_temp_mean <- aggregate(site_data$T2M, by = list(site_data$DOY), FUN = mean)
  daily_temp_sd <- aggregate(site_data$T2M, by = list(site_data$DOY), FUN = sd)
  daily_solar_mean <- aggregate(site_data$ALLSKY_SFC_SW_DWN, by = list(site_data$DOY), FUN =
  daily_solar_sd <- aggregate(site_data$ALLSKY_SFC_SW_DWN, by = list(site_data$DOY), FUN = s

  # Yearly precipitation aggregation
  annual_sum <- aggregate(site_data$PRECTOT, by = list(site_data$YEAR), FUN = sum)

  # Return computed values as a named list
  list(
    temp_annual_max = max(daily_temp_mean$x, na.rm = TRUE),
    temp_annual_min = min(daily_temp_mean$x, na.rm = TRUE),
    temp_daily_fluctuation = mean(daily_temp_sd$x, na.rm = TRUE),
    temp_daily_lower_dev = mean(site_data$T2M - site_data$T2M_MIN, na.rm = TRUE),
    temp_daily_upper_dev = mean(site_data$T2M_MAX - site_data$T2M, na.rm = TRUE),
    solar_annual_max = max(daily_solar_mean$x, na.rm = TRUE),
```

```
    solar_annual_min = min(daily_solar_mean$x, na.rm = TRUE),
    solar_daily_fluctuation = mean(daily_solar_sd$x, na.rm = TRUE),
    precip_annual_sum_mean = mean(annual_sum$x, na.rm = TRUE),
    precip_annual_sum_sd = sd(annual_sum$x, na.rm = TRUE)
  )
}

# Apply the function across sites
annual_weather_summary <- lapply(split(weather, weather$Site), calculate_site_summary)

# Convert the list of summaries into a data frame
annual_weather_summary_df <- do.call(rbind, annual_weather_summary)
#annual_weather_summary_df <- cbind(Site = names(annual_weather_summary), annual_weather_summ

# Ensure the data frame structure is consistent
annual_weather_summary_df <- as.data.frame(annual_weather_summary_df)
#rownames(annual_weather_summary_df) <- NULL
```

Initialise experiments per site using annual summary statistics and estimated yearly cumulative precipitation parameters of example locations as parameter inputs:

```
weather_model_runs <- list()

for (site in sites)
{
  estimation_optim <- best_estimation_fits_mean[[site]]

  weather_model_runs[[site]] <- initialise_weather_model(
    year_length = year_length_in_days,
    seed = SEED,
    albedo = 0.4,
    is_southern_hemisphere = weather[weather$Site == site,"LAT"][1] < 0,
    temp_annual_max = annual_weather_summary_df$temp_annual_max[[site]],
    temp_annual_min = annual_weather_summary_df$temp_annual_min[[site]],
    temp_daily_fluctuation = annual_weather_summary_df$temp_daily_fluctuation[[site]],
    temp_daily_lower_dev = annual_weather_summary_df$temp_daily_lower_dev[[site]],
    temp_daily_upper_dev = annual_weather_summary_df$temp_daily_upper_dev[[site]],

    solar_annual_max = annual_weather_summary_df$solar_annual_max[[site]],
    solar_annual_min = annual_weather_summary_df$solar_annual_min[[site]],
    solar_daily_fluctuation = annual_weather_summary_df$solar_daily_fluctuation[[site]],
```

```
    precip_annual_sum_mean = annual_weather_summary_df$precip_annual_sum_mean[[site]],
    precip_annual_sum_sd = annual_weather_summary_df$precip_annual_sum_sd[[site]],

    precip_plateau_value_mean = estimation_optim$mean[1],
    precip_plateau_value_sd = estimation_optim$sd[1],
    precip_inflection1_mean = estimation_optim$mean[2],
    precip_inflection1_sd = estimation_optim$sd[2],
    precip_rate1_mean = estimation_optim$mean[3],
    precip_rate1_sd = estimation_optim$sd[3],
    precip_inflection2_mean = estimation_optim$mean[4],
    precip_inflection2_sd = estimation_optim$sd[4],
    precip_rate2_mean = estimation_optim$mean[5],
    precip_rate2_sd = estimation_optim$sd[5],
    precip_n_samples_mean = estimation_optim$mean[6],
    precip_n_samples_sd = estimation_optim$sd[6],
    precip_max_sample_size_mean = estimation_optim$mean[7],
    precip_max_sample_size_sd = estimation_optim$sd[7]
  )
}
```

Run experiments:

```
for (site in sites)
{
  weather_model_runs[[site]] <-
    run_weather_model(weather_model_runs[[site]], number_of_years)
}
```

Create a data frame containing the daily summary statistics of simulations comparable to the one for the real data:

```
# Function to calculate summary statistics for a single day's data
calculate_daily_summary <- function(day_data) {
  # Solar radiation
  solar_mean <- mean(day_data$solar_radiation, na.rm = TRUE)
  solar_sd <- sd(day_data$solar_radiation, na.rm = TRUE)
  solar_max <- max(day_data$solar_radiation, na.rm = TRUE)
  solar_min <- min(day_data$solar_radiation, na.rm = TRUE)
  solar_error <- qt(0.975, df = max(length(day_data$solar_radiation) - 1, 1)) *
                 solar_sd / sqrt(length(day_data$solar_radiation))
```

```r
# Temperature
temp_mean <- mean(day_data$temperature, na.rm = TRUE)
temp_sd <- sd(day_data$temperature, na.rm = TRUE)
temp_max <- max(day_data$temperature, na.rm = TRUE)
temp_min <- min(day_data$temperature, na.rm = TRUE)
temp_error <- qt(0.975, df = max(length(day_data$temperature) - 1, 1)) *
              temp_sd / sqrt(length(day_data$temperature))

# Max temperature
max_temp_mean <- mean(day_data$temperature_max, na.rm = TRUE)
max_temp_max <- max(day_data$temperature_max, na.rm = TRUE)
max_temp_min <- min(day_data$temperature_max, na.rm = TRUE)
max_temp_error <- qt(0.975, df = max(length(day_data$temperature_max) - 1, 1)) *
                  sd(day_data$temperature_max, na.rm = TRUE) /
                  sqrt(length(day_data$temperature_max))

# Min temperature
min_temp_mean <- mean(day_data$temperature_min, na.rm = TRUE)
min_temp_max <- max(day_data$temperature_min, na.rm = TRUE)
min_temp_min <- min(day_data$temperature_min, na.rm = TRUE)
min_temp_error <- qt(0.975, df = max(length(day_data$temperature_min) - 1, 1)) *
                  sd(day_data$temperature_min, na.rm = TRUE) /
                  sqrt(length(day_data$temperature_min))

# Deviations
lower_dev <- mean(day_data$temperature - day_data$temperature_min, na.rm = TRUE)
lower_dev_error <- qt(0.975, df = max(length(day_data$temperature_min) - 1, 1)) *
                   sd(day_data$temperature - day_data$temperature_min, na.rm = TRUE) /
                   sqrt(length(day_data$temperature_min))

upper_dev <- mean(day_data$temperature_max - day_data$temperature, na.rm = TRUE)
upper_dev_error <- qt(0.975, df = max(length(day_data$temperature_max) - 1, 1)) *
                   sd(day_data$temperature_max - day_data$temperature, na.rm = TRUE) /
                   sqrt(length(day_data$temperature_max))

# Precipitation
precip_mean <- mean(day_data$precipitation, na.rm = TRUE)
precip_max <- max(day_data$precipitation, na.rm = TRUE)
precip_min <- min(day_data$precipitation, na.rm = TRUE)
precip_error <- qt(0.975, df = max(length(day_data$precipitation) - 1, 1)) *
                sd(day_data$precipitation, na.rm = TRUE) /
                sqrt(length(day_data$precipitation))
```

```r
  # Combine results into a named list
  list(
    solarRadiation.mean = solar_mean,
    solarRadiation.sd = solar_sd,
    solarRadiation.max = solar_max,
    solarRadiation.min = solar_min,
    solarRadiation.error = solar_error,
    temperature.mean = temp_mean,
    temperature.sd = temp_sd,
    temperature.max = temp_max,
    temperature.min = temp_min,
    temperature.error = temp_error,
    maxTemperature.mean = max_temp_mean,
    maxTemperature.max = max_temp_max,
    maxTemperature.min = max_temp_min,
    maxTemperature.error = max_temp_error,
    minTemperature.mean = min_temp_mean,
    minTemperature.max = min_temp_max,
    minTemperature.min = min_temp_min,
    minTemperature.error = min_temp_error,
    temperature.lowerDeviation = lower_dev,
    temperature.lowerDeviation.error = lower_dev_error,
    temperature.upperDeviation = upper_dev,
    temperature.upperDeviation.error = upper_dev_error,
    precipitation.mean = precip_mean,
    precipitation.max = precip_max,
    precipitation.min = precip_min,
    precipitation.error = precip_error
  )
}

# Process data for all sites and days
weather_summary_sim <- do.call(rbind, lapply(sites, function(site) {
  site_data <- as.data.frame(weather_model_runs[[site]]$daily)
  do.call(rbind, lapply(1:max(year_length_in_days), function(day) {
    day_data <- site_data[site_data$current_day_of_year == day,]
    as.data.frame(list(
      Site = site,
      day_of_year = day,
      calculate_daily_summary(day_data)
      ))
  }))
```

```r
}))

# Convert to a data frame
weather_summary_sim <- as.data.frame(weather_summary_sim)
```

7.4 Creating figure

Set colours for real and simulated data:

```r
realDataColour = hsv(200/360, 62/100, 63/100) # teal

simulatedDataColour = hsv(24/360, 79/100, 89/100) # orange
```

Create figure:

```r
# Helper functions
round_to_multiple <- function(x, base, round_fn = round) {
  round_fn(x / base) * base
}

create_polygon <- function(x, y1, y2, alpha = 0.5, col = "black") {
  polygon(c(x, rev(x)), c(y1, rev(y2)), col = adjustcolor(col, alpha = alpha), border = NA)
}

plot_weather_variable <- function(x, y, ylim, lwd, col = "black", lty = 1) {
  plot(x, y, axes = FALSE, ylim = ylim, type = "l", lwd = lwd, col = col, lty = lty)
}

add_confidence_interval <- function(x, y_mean, error, col, alpha = 0.5) {
  create_polygon(x, y_mean + error, y_mean, alpha, col)
  create_polygon(x, y_mean - error, y_mean, alpha, col)
}

add_min_max_interval <- function(x, y_mean, y_min, y_max, col, alpha = 0.3) {
  create_polygon(x, y_max, y_mean, alpha, col)
  create_polygon(x, y_min, y_mean, alpha, col)
}

# Main plotting function
```

```
plot_weather_summary_comparison <- function(weather_summary, sites, sites_latitude, weather)
  # Setup plot
  num_columns <- length(sites) + 1
  num_rows_except_bottom <- 4

  layout_matrix <- rbind(
    matrix(1:(num_columns * num_rows_except_bottom), nrow = num_rows_except_bottom, ncol = nu
    c((num_columns * num_rows_except_bottom) + 1, rep((num_columns * num_rows_except_bottom)
  )

  layout(layout_matrix,
         widths = c(3, 12, rep(10, length(sites) - 2), 14),
         heights = c(3, 10, 10, 12, 2))

  # Y-axis labels
  y_labs <- c(expression(paste("solar radiation (", MJ/m^-2, ")")),
              "temperature (C)", "precipitation (mm)")

  # Calculate ranges
  range_solar <- c(
    round_to_multiple(min(
      min(weather_summary$solarRadiation.min),
      min(weather_summary_sim$solarRadiation.min)),
      5, floor),
    round_to_multiple(max(
      max(weather_summary$solarRadiation.max),
      40),
      #max(weather_summary_sim$solarRadiation.max)),
      ## an outlier in Sao Paulo brings it to c. 46 and does not show with the polygon
      5, ceiling)
  )
  range_temp <- c(
    round_to_multiple(min(
      min(weather_summary$minTemperature.min),
      min(weather_summary_sim$minTemperature.min)),
      5, floor),
    round_to_multiple(max(
      max(weather_summary$maxTemperature.max),
      max(weather_summary_sim$maxTemperature.max)),
      5, ceiling)
  )
  range_precip <- c(
```

```
  round_to_multiple(min(
    min(weather_summary$precipitation.min),
    min(weather_summary_sim$precipitation.min)),
    5, floor),
  round_to_multiple(max(
    max(weather_summary$precipitation.max),
    max(weather_summary_sim$precipitation.max)),
    5, ceiling)
)

# Plot settings
par(cex = graphic_scale, cex.axis = graphic_scale * (0.8 + axis_text_rescale))

# First column: y axis titles
for (i in 1:4) {
  par(mar = c(0, 0, 0, 0.4))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  if (i > 1) {
    text(x = 0.5, y = 0.5, font = 4,
         cex = graphic_scale * (0.78 + font_rescale),
         srt = 90,
         labels = y_labs[i-1])
  }
}

# Plot for each site
for (site in sites) {
  weather_site <- weather[weather$Site == site,]
  weather_model_site <- weather_model_runs[[site]]$daily
  weather_summary_site <- weather_summary[weather_summary$Site == site,]
  weather_summary_site_sim <- weather_summary_sim[weather_summary_sim$Site == site,]

  left_plot_margin <- ifelse(site == sites[1], 2, 0.1)
  right_plot_margin <- ifelse(site == sites[length(sites)], 4, 0.1)

  # Site name + latitude
  par(mar = c(0.2, left_plot_margin, 0.1, right_plot_margin))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  text(x = 0.5, y = 0.5, font = 4,
       cex = graphic_scale * (0.7 + font_rescale),
       labels = paste(site, sites_latitude$Latitude[sites_latitude$Site == site], sep = "\n
```

```r
# Solar radiation

# original data
par(mar = c(0.1, left_plot_margin, 0.1, right_plot_margin))
plot_weather_variable(1:year_length_max, weather_summary_site$solarRadiation.mean,
                      range_solar, graphic_scale,
                      col = adjustcolor(realDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max, weather_summary_site$solarRadiation.mean,
                        weather_summary_site$solarRadiation.error,
                        adjustcolor(realDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site$solarRadiation.mean,
                     weather_summary_site$solarRadiation.min,
                     weather_summary_site$solarRadiation.max,
                     adjustcolor(realDataColour, alpha.f = 0.5))

# simulations
lines(1:year_length_max, weather_summary_site_sim$solarRadiation.mean,
      lwd = graphic_scale,
      col = adjustcolor(simulatedDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site_sim$solarRadiation.mean,
                        weather_summary_site_sim$solarRadiation.error,
                        adjustcolor(simulatedDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site_sim$solarRadiation.mean,
                     weather_summary_site_sim$solarRadiation.min,
                     weather_summary_site_sim$solarRadiation.max,
                     adjustcolor(simulatedDataColour, alpha.f = 0.5))

# solstices and axes
#lines(1:year_length_max, weather_summary_site$solarRadiationTop.mean, lty = 2, lwd = gra

abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

if (site == sites[1]) {
  axis(2, at = seq(range_solar[1], range_solar[2], 5))
}

# Temperature

# original data
```

```
plot_weather_variable(1:year_length_max, weather_summary_site$temperature.mean,
                      range_temp, graphic_scale,
                      col = adjustcolor(realDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site$temperature.mean,
                        weather_summary_site$temperature.error,
                        adjustcolor(realDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site$temperature.mean,
                     weather_summary_site$temperature.min,
                     weather_summary_site$temperature.max,
                     adjustcolor(realDataColour, alpha.f = 0.5))

lines(1:year_length_max, weather_summary_site$maxTemperature.mean,
      lwd = graphic_scale,
      col = adjustcolor(realDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site$maxTemperature.mean,
                        weather_summary_site$maxTemperature.error,
                        col = adjustcolor(realDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site$maxTemperature.mean,
                     weather_summary_site$maxTemperature.min,
                     weather_summary_site$maxTemperature.max,
                     adjustcolor(realDataColour, alpha.f = 0.5))

lines(1:year_length_max, weather_summary_site$minTemperature.mean,
      lwd = graphic_scale,
      col = adjustcolor(realDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site$minTemperature.mean,
                        weather_summary_site$minTemperature.error,
                        adjustcolor(realDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site$minTemperature.mean,
                     weather_summary_site$minTemperature.min,
                     weather_summary_site$minTemperature.max,
                     adjustcolor(realDataColour, alpha.f = 0.5))

# simulations
lines(1:year_length_max, weather_summary_site_sim$temperature.mean,
      lwd = graphic_scale,
```

```
      col = adjustcolor(simulatedDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site_sim$temperature.mean,
                        weather_summary_site_sim$temperature.error,
                        adjustcolor(simulatedDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site_sim$temperature.mean,
                     weather_summary_site_sim$temperature.min,
                     weather_summary_site_sim$temperature.max,
                     adjustcolor(simulatedDataColour, alpha.f = 0.5))

lines(1:year_length_max, weather_summary_site_sim$maxTemperature.mean,
      lwd = graphic_scale,
      col = adjustcolor(simulatedDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site_sim$maxTemperature.mean,
                        weather_summary_site_sim$maxTemperature.error,
                        col = adjustcolor(simulatedDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site_sim$maxTemperature.mean,
                     weather_summary_site_sim$maxTemperature.min,
                     weather_summary_site_sim$maxTemperature.max,
                     adjustcolor(simulatedDataColour, alpha.f = 0.5))

lines(1:year_length_max, weather_summary_site_sim$minTemperature.mean,
      lwd = graphic_scale,
      col = adjustcolor(simulatedDataColour, alpha.f = 1))
add_confidence_interval(1:year_length_max,
                        weather_summary_site_sim$minTemperature.mean,
                        weather_summary_site_sim$minTemperature.error,
                        adjustcolor(simulatedDataColour, alpha.f = 0.75))
add_min_max_interval(1:year_length_max,
                     weather_summary_site_sim$minTemperature.mean,
                     weather_summary_site_sim$minTemperature.min,
                     weather_summary_site_sim$minTemperature.max,
                     adjustcolor(simulatedDataColour, alpha.f = 0.5))

# solstices and axes
abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

if (site == sites[1]) {
  axis(2, at = seq(range_temp[1], range_temp[2], 5))
```

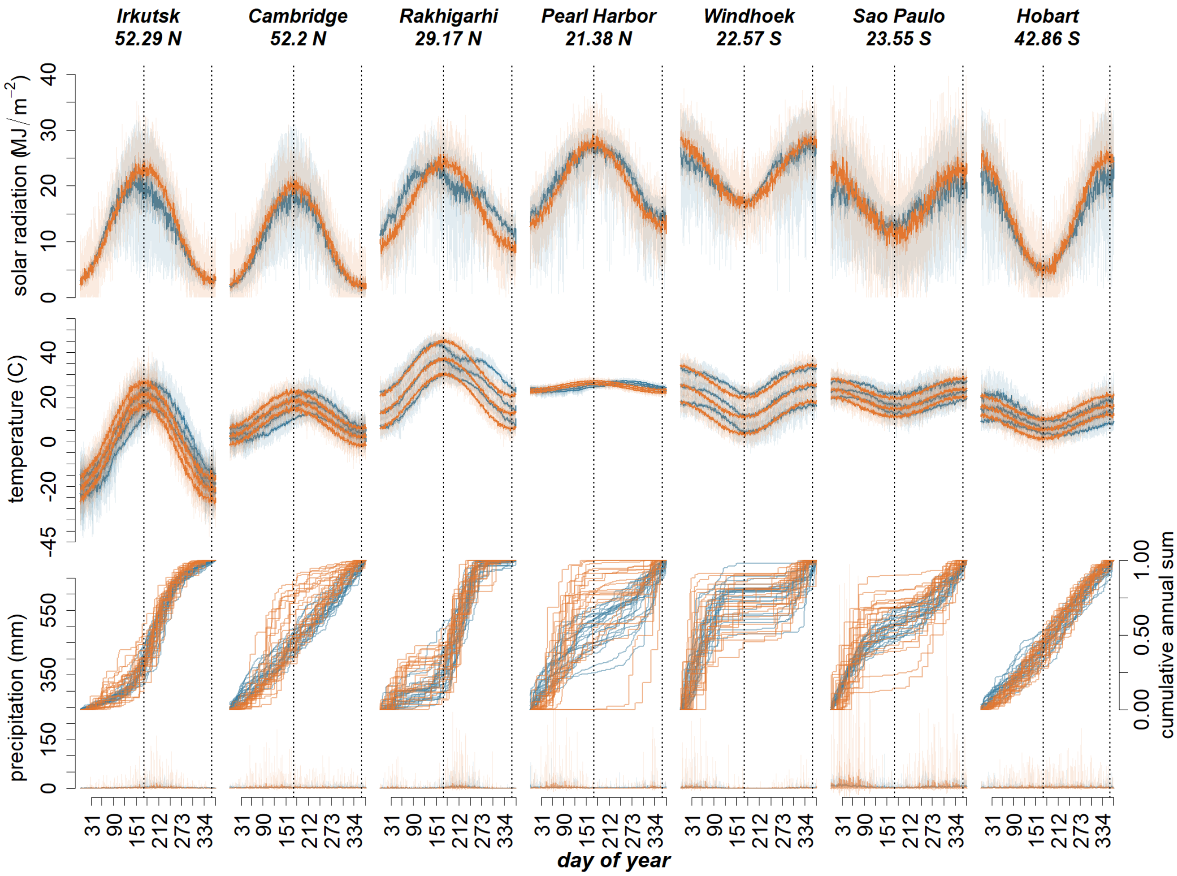
```
}

# Precipitation
par(mar = c(8, left_plot_margin, 0.1, right_plot_margin))

# cumulative precipitation
plot(c(1, year_length_max), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yax
# original data
for (year in years) {
  site_year_data <- weather_site$PRECTOT[weather_site$YEAR == year]
  lines(1:length(site_year_data),
        get_cumulative_precipitation_of_year(site_year_data),
        lwd = graphic_scale,
        col = adjustcolor(realDataColour, alpha.f = 0.5))
}
# simulation
for (year in 1:number_of_years) {
  site_year_data_sim <- weather_model_site$precipitation[weather_model_site$current_year
  lines(1:length(site_year_data_sim),
        get_cumulative_precipitation_of_year(site_year_data_sim),
        lwd = graphic_scale,
        col = adjustcolor(simulatedDataColour, alpha.f = 0.5))
}

if (site == sites[length(sites)]) {
  axis(4, at = seq(0, 1, 0.25))
  mtext("cumulative annual sum", 4, line = 2.5, cex = graphic_scale * (1.5 + margin_text_
}

# daily precipitation
par(new = TRUE, mar = c(3, left_plot_margin, 0.1, right_plot_margin))

# original data
plot_weather_variable(1:year_length_max,
                      weather_summary_site$precipitation.mean,
                      range_precip,
                      graphic_scale,
                      col = adjustcolor(realDataColour, alpha.f = 0.5))
add_confidence_interval(1:year_length_max,
                        weather_summary_site$precipitation.mean,
                        weather_summary_site$precipitation.error,
                        adjustcolor(realDataColour, alpha.f = 0.5))
```

```
    add_min_max_interval(1:year_length_max,
                         weather_summary_site$precipitation.mean,
                         weather_summary_site$precipitation.min,
                         weather_summary_site$precipitation.max,
                         adjustcolor(realDataColour, alpha.f = 0.5))

    # simulation
    lines(1:year_length_max,
          weather_summary_site_sim$precipitation.mean,
          lwd = graphic_scale,
          col = adjustcolor(simulatedDataColour, alpha.f = 0.5))
    add_confidence_interval(1:year_length_max,
                            weather_summary_site_sim$precipitation.mean,
                            weather_summary_site_sim$precipitation.error,
                            adjustcolor(simulatedDataColour, alpha.f = 0.5))
    add_min_max_interval(1:year_length_max,
                         weather_summary_site_sim$precipitation.mean,
                         weather_summary_site_sim$precipitation.min,
                         weather_summary_site_sim$precipitation.max,
                         adjustcolor(simulatedDataColour, alpha.f = 0.5))

    # solstices and axes
    abline(v = c(SOLSTICE_SUMMER, SOLSTICE_WINTER), lty = 3, lwd = graphic_scale)

    if (site == sites[1]) {
      axis(2, at = seq(range_precip[1], range_precip[2], 50))
    }

    axis(1, at = cumsum(c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)), las = 2)
  }

  # Bottom row: "day of year" label
  par(mar = c(0, 0, 0, 0))
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  plot(c(0, 1), c(0, 1), ann = FALSE, bty = 'n', type = 'n', xaxt = 'n', yaxt = 'n')
  text(x = 0.5, y = 0.7, font = 4,
       cex = graphic_scale * (0.8 + font_rescale),
       labels = "day of year")
}

# Main execution
plot_name <- file.path(output_dir, paste0("Fig6-ValidationUsingExamples.", plot_file_format))
```

```
if (plot_file_format == "png") {
  graphic_scale <- 2
  font_rescale <- axis_text_rescale <- margin_text_rescale <- 0

  png(plot_name, width = number_of_sites * graphic_scale * 150, height = graphic_scale * 800)
} else if (plot_file_format == "eps") {
  graphic_scale = 1.2
  font_rescale = 0.1
  axis_text_rescale = -0.1
  margin_text_rescale = -0.5

  extrafont::loadfonts(device = "postscript")
  grDevices::cairo_ps(filename = plot_name ,
                    pointsize = 12,
                    width = number_of_sites * graphic_scale * 1.5,
                    height = graphic_scale * 8,
                    onefile = FALSE,
                    family = "sans"
                    )
}
plot_weather_summary_comparison(weather_summary, sites, sites_latitude, weather)
dev.off()
```

```
pdf
  2
```

```
knitr::include_graphics(plot_name)
```



Irkutsk
52.29 N
Cambridge
52.2 N
Rakhigarhi
29.17 N
Pearl Harbor
21.38 N
Windhoek
22.57 S
Sao Paulo
23.55 S
Hobart
42.86 S
solar radiation (MJ/m^{-2})
temperature (C)
precipitation (mm)
cumulative annual sum
day of year