Contents

#1.LOADING LIBRARIES AND DATASETS

```
# ============================================================
# PHASE 0 - Loading Libraries and Datasets

# 0) Package setup - install if missing, then load quietly
#     (All packages are CRAN; install once per machine.)
# ============================================================
load_if_needed <- function(pkgs) {
  missing <- setdiff(pkgs, rownames(installed.packages()))
  if (length(missing)) {
    message("Installing missing packages: ", paste(missing, collapse = ", "))
    install.packages(missing, dependencies = TRUE)
  }
  invisible(lapply(pkgs, function(p)
    suppressPackageStartupMessages(library(p, character.only = TRUE))))
}

req_pkgs <- c(
  # --- core tidy/data handling ---
  "arrow",           # Fast Parquet/Feather IO + lazy datasets
  "dplyr",           # Data wrangling (filter/mutate/summarise/join)
  "magrittr",        # Pipes (%>%, %<>%) & helpers
  "lubridate",       # Dates/times (ymd(), year(), decimal_date())
  "tidyr",           # Reshaping (pivot_longer/wider, separate, unite)
  "forcats",         # Factor tools (fct_* helpers)
  "purrr",           # Functional tools (map, pmap, safely)
  # --- modelling & viz (used later in the pipeline) ---
  "ggplot2",         # Plotting
  "scales",          # Axis/label formatting
  "mgcv",            # GAM/BAM fitting
  "gratia",          # GAM diagnostics & partial-effects
  "readr",           # Fast read/write for CSV
  "MASS",            # Misc. stats (e.g., mvrnorm)
  "patchwork",       # Compose multiple ggplots
  # --- light mapping (used in later EDA) ---
  "sf",
  "rnaturalearth",
  "rnaturalearthdata"
)

load_if_needed(req_pkgs)
```

```
## Warning: package 'arrow' was built under R version 4.4.3
```

```
## Warning: package 'magrittr' was built under R version 4.4.3
```

```
## Warning: package 'lubridate' was built under R version 4.4.3
```

```
## Warning: package 'tidyr' was built under R version 4.4.3
```

```
## Warning: package 'forcats' was built under R version 4.4.3

## Warning: package 'ggplot2' was built under R version 4.4.3

## Warning: package 'scales' was built under R version 4.4.3

## Warning: package 'mgcv' was built under R version 4.5.1

## Warning: package 'gratia' was built under R version 4.4.3

## Warning: package 'MASS' was built under R version 4.4.3

## Warning: package 'patchwork' was built under R version 4.4.3

## Warning: package 'rnaturalearth' was built under R version 4.4.3

## Warning: package 'rnaturalearthdata' was built under R version 4.4.3
```

```r
# Reproducibility niceties (keeps behaviour stable across runs)
options(stringsAsFactors = FALSE)
set.seed(20250816)

# ================================================================
# 1) Data location - point to the Parquet files
#     *Adjust DATA_DIR if your files live elsewhere.*
# ================================================================
DATA_DIR <- getwd()  # change to a fixed path if preferred, e.g. "D:/EA_open_data"

paths <- list(
  sites   = file.path(DATA_DIR, "INV_OPEN_DATA_SITE.parquet"),
  metrics = file.path(DATA_DIR, "INV_OPEN_DATA_METRICS.parquet"),
  whpt    = file.path(DATA_DIR, "R_INV_WHPT_METRICS_B.parquet"),
  taxa    = file.path(DATA_DIR, "INV_OPEN_DATA_TAXA.parquet")
)

# Small helper: fail fast with a clear message if a file is missing
assert_parquet <- function(path) {
  if (!file.exists(path)) {
    stop(
      sprintf(
        "Parquet not found:\n  %s\n\nFix: set DATA_DIR correctly or download the file.",
        normalizePath(path, winslash = "/", mustWork = FALSE)
      ),
      call. = FALSE
    )
  }
  normalizePath(path, winslash = "/", mustWork = TRUE)
}

# ================================================================
# 2) Open raw datasets (lazy) - Arrow keeps things fast & memory-light
```

```r
#     Note: nothing is read into R memory until you call `collect()`.
# ============================================================
sites_raw   <- arrow::open_dataset(assert_parquet(paths$sites))
metrics_raw <- arrow::open_dataset(assert_parquet(paths$metrics))
whpt_raw    <- arrow::open_dataset(assert_parquet(paths$whpt))
taxa_raw    <- arrow::open_dataset(assert_parquet(paths$taxa))

# Quick peek helpers (uncomment if you want to sanity-check schemas)
# dplyr::glimpse(dplyr::collect(head(sites_raw,   3)))
# dplyr::glimpse(dplyr::collect(head(metrics_raw, 3)))
# dplyr::glimpse(dplyr::collect(head(whpt_raw,    3)))
# dplyr::glimpse(dplyr::collect(head(taxa_raw,    3)))

# ------------------------------------------------------------
# From here on we keep everything 'lazy' as long as possible and
# `collect()` only when necessary. This is essential for speed and
# memory efficiency on the full EA datasets.
# ------------------------------------------------------------
```

#2. DATA CLEANING AND DATA PREPROCESSING

```r
# ============================================================================
# PHASE 1 - Configure & Clean
# Purpose: define project-wide options and build the 3
# core, cleaned tables used throughout the analysis:
#   - metrics_clean: one row per (SITE_ID, SAMPLE_DATE) visit
#   - whpt_clean   : harmonised family counts by sample
#   - sites_clean  : site metadata / coordinates / areas
# Notes:
#   • We keep only ANAA/ANLA/ANLE analysis methods (others dropped).
#   • We default to S3PO kick-sampling as recommended.
#   • We drop pre-1990 samples (method standardisation/audit began ~1990).
#   • Arrow is used for fast IO and explicit typing before collect().
# ============================================================================

# ---- 0) Project options (tweak as needed) ----
USE_S3PO   <- TRUE                          # recommended by Mike
DATE_FLOOR <- as.Date("1990-01-01")         # 1995 only if modelling issues persist
FAMILIES   <- c("Aphelocheiridae",          # 1-species families (drop Potamanthidae: too rare)
                "Brachycentridae",
                "Odontoceridae",
                "Cordulegastridae")

# ---- 1) Keep only core analysis methods in METRICS; type fixes; optional S3PO filter; de-dup ----
metrics_clean <- metrics_raw %>%
  # keep only relevant analysis methods (drop ANPA, ANCA, etc.)
  dplyr::filter(ANALYSIS_METHOD %in% c("ANAA","ANLA","ANLE")) %>%
  # type coercions done lazily in Arrow
  dplyr::mutate(
    SITE_ID     = arrow::cast(SITE_ID,   arrow::int32()),   # ensure integer keys for joins
    SAMPLE_ID   = arrow::cast(SAMPLE_ID, arrow::int32()),   # sample identifier
    SAMPLE_DATE = arrow::cast(SAMPLE_DATE, arrow::date32()) # Arrow date (faster; converted after colle
  ) %>%
```

```r
  # only columns we actually use downstream
  dplyr::select(SITE_ID, SAMPLE_ID, SAMPLE_DATE,
                SAMPLE_TYPE, SAMPLE_METHOD, ANALYSIS_TYPE, ANALYSIS_METHOD) %>%
  # bring to R for small, deterministic post-processing
  dplyr::collect() %>%
  dplyr::mutate(SAMPLE_DATE = as.Date(SAMPLE_DATE)) %>%      # convert Arrow date32 -> base Date
  dplyr::arrange(SITE_ID, SAMPLE_DATE, SAMPLE_ID) %>%        # stable order: site → date → sample
  # optional: filter to S3PO only (recommended)
  { if (USE_S3PO) dplyr::filter(., SAMPLE_METHOD == "S3PO") else . } %>%
  # date floor (standardisation/audit starts ~1990)
  dplyr::filter(SAMPLE_DATE >= DATE_FLOOR) %>%
  # handle same-day replicates: keep first per (site, date)
  dplyr::group_by(SITE_ID, SAMPLE_DATE) %>%                  # ensure one visit per day per site
  dplyr::slice(1L) %>%                                       # deterministic de-dup (earliest SAMPLE_ID
  dplyr::ungroup()

# ---- 2) WHPT harmonised family data (only needed fields; typed) ----
whpt_clean <- whpt_raw %>%
  dplyr::mutate(
    SITE_ID      = arrow::cast(SITE_ID,      arrow::int32()), # align types to metrics_clean for joins
    SAMPLE_ID    = arrow::cast(SAMPLE_ID,    arrow::int32()),
    TOTAL_NUMBER = arrow::cast(TOTAL_NUMBER, arrow::int32())  # family-level abundance/count (incl. 1 s
  ) %>%
  dplyr::select(SITE_ID, SAMPLE_ID, EQ_TAXON_UNIT, TOTAL_NUMBER) %>% # keep only fields needed later
  dplyr::collect()

# ---- 3) Sites table (minimal columns for optional mapping/covariates) ----
sites_clean <- sites_raw %>%
  dplyr::mutate(SITE_ID = arrow::cast(SITE_ID, arrow::int32())) %>%   # consistent key type
  dplyr::select(SITE_ID, REPORTING_AREA, CATCHMENT, WATER_BODY,       # area/catchment useful for summa
                FULL_EASTING, FULL_NORTHING) %>%                      # BNG coordinates for mapping
  dplyr::collect()
```

```r
# ======================================================================
# PHASE 2 - Classify samples as BIN (categorical) vs COUNT (numeric)
# ======================================================================

# ---- 0) Load TAXA minimally + type fixes (lazy -> collect once)
taxa_clean <- taxa_raw %>%
  dplyr::mutate(
    SAMPLE_ID       = arrow::cast(SAMPLE_ID,       arrow::int32()),
    TOTAL_ABUNDANCE = arrow::cast(TOTAL_ABUNDANCE, arrow::int32())
  ) %>%
  dplyr::select(SAMPLE_ID, TOTAL_ABUNDANCE) %>%
  dplyr::collect()

# ---- 1) Build per-SAMPLE_ID classification from TAXA
#    - ignore zeros and NAs when classifying
#    - placeholders define categorical bins; include '1' as a (rare) categorical value
bin_placeholders <- c(1L, 3L, 33L, 333L, 3333L, 33333L)

taxa_nonzero <- taxa_clean %>%
  dplyr::filter(!is.na(TOTAL_ABUNDANCE), TOTAL_ABUNDANCE != 0L)
```

```r
# initial classification by TAXA content
taxa_flags_initial <- taxa_nonzero %>%
  dplyr::group_by(SAMPLE_ID) %>%
  dplyr::summarise(
    all_placeholders = all(TOTAL_ABUNDANCE %in% bin_placeholders),
    any_placeholder  = any(TOTAL_ABUNDANCE %in% bin_placeholders),
    any_numeric      = any(!(TOTAL_ABUNDANCE %in% bin_placeholders)),
    .groups = "drop"
  ) %>%
  dplyr::mutate(
    # Mixed (placeholders + numerics) are treated as BIN (safe)
    data_type_taxa = dplyr::case_when(
      all_placeholders                ~ "bin",
      any_placeholder & any_numeric ~ "bin",   # weird early-2000s mixes -> bin
      TRUE                            ~ "count"
    )
  )

# ensure every SAMPLE_ID in metrics has a flag row
# samples missing from TAXA get NA here; we'll resolve via ANALYSIS_METHOD overrides
sample_flags_taxa <- metrics_clean %>%
  dplyr::select(SAMPLE_ID, ANALYSIS_METHOD) %>%
  dplyr::left_join(taxa_flags_initial, by = "SAMPLE_ID")

# ---- 2) Apply method-based overrides (project rules)
#    ANLE -> bin (always)
#    ANAA -> count (always)
#    ANLA -> use TAXA; if still NA/ambiguous, choose bin (safer)
#    Anything else should not be present (filtered earlier)
resolve_data_type <- function(analysis_method, from_taxa) {
  if (is.na(analysis_method)) return(NA_character_)
  if (analysis_method == "ANLE") return("bin")
  if (analysis_method == "ANAA") return("count")
  if (analysis_method == "ANLA") {
    # Prefer TAXA classification where available; if missing/ambiguous, choose bin
    if (!is.na(from_taxa)) return(from_taxa)
    return("bin")
  }
  # Should not occur (we filtered), but default to bin if it does
  "bin"
}

sample_flags_final <- sample_flags_taxa %>%
  dplyr::mutate(
    data_type = purrr::pmap_chr(
      list(ANALYSIS_METHOD, data_type_taxa),
      ~ resolve_data_type(..1, ..2)
    )
  ) %>%
  dplyr::select(SAMPLE_ID, data_type)

# ---- 3) Attach flags back to metrics (roster) for downstream table building
roster_flagged <- metrics_clean %>%
```

```r
  dplyr::left_join(sample_flags_final, by = "SAMPLE_ID") %>%
  # Safety: if still NA (should be rare), default to bin (conservative)
  dplyr::mutate(data_type = tidyr::replace_na(data_type, "bin"))

# --- quick sanity checks (optional) ---
# table(roster_flagged$data_type, useNA = "ifany")
# with(roster_flagged, table(ANALYSIS_METHOD, data_type))



# ---- 4) Season handling in preprocessing
# NOTE: The only change below is explicit namespacing of `filter` and `select`
#       to guarantee dplyr's versions are used (avoids masking by other packages).

# project switch: keep any off-season samples as a third level?
KEEP_OFFSEASON <- FALSE   # recommended FALSE for EA data (mostly spring/autumn)

roster_flagged <- roster_flagged %>%
  mutate(
    month    = month(SAMPLE_DATE),
    season   = case_when(
      month %in% 3:5  ~ "spring",
      month %in% 9:11 ~ "autumn",
      TRUE            ~ NA_character_
    ),
    season_f = factor(season, levels = c("spring","autumn"))
  ) %>%
  { if (!KEEP_OFFSEASON) dplyr::filter(., !is.na(season_f)) else . } %>%
  dplyr::select(
    # keep the usual roster fields + season vars
    SITE_ID, SAMPLE_ID, SAMPLE_DATE, SAMPLE_METHOD, ANALYSIS_METHOD, data_type,
    season_f, month
  )

# quick sanity check (optional)
# table(roster_flagged$season_f, useNA = "ifany")
```

```r
# ================================================
# PHASE 3 - Build modelling tables per family
# ================================================

# --- Families of interest ---
families <- c(
  "Aphelocheiridae",
  "Brachycentridae",
  "Odontoceridae",
  "Cordulegastridae"
  # Potamanthidae is too rare for now
)

# --- Function to create modelling tables for one family ---
make_family_tables <- function(family_name, roster_tbl, whpt_tbl) {
```

```r
# a) Sites where this family has ever been observed
#    - Filter to the target family in the harmonised WHPT table
#    - Keep unique SITE_IDs only and pull as vector
sites_with_fam <- whpt_tbl %>%
  dplyr::filter(EQ_TAXON_UNIT == family_name) %>%
  dplyr::distinct(SITE_ID) %>%
  dplyr::pull(SITE_ID)

# b) Restrict the visit roster to those sites
#    (This keeps all sampling occasions at sites where the family was seen at least once.)
roster_family <- roster_tbl %>%
  dplyr::filter(SITE_ID %in% sites_with_fam)

# c) Extract counts for this family (from WHPT table) and join to roster
#    - After the left_join, missing TOTAL_NUMBER means the family was not recorded -> set to 0
family_counts <- whpt_tbl %>%
  dplyr::filter(EQ_TAXON_UNIT == family_name) %>%
  dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)

fam_data <- roster_family %>%
  dplyr::left_join(family_counts, by = c("SITE_ID", "SAMPLE_ID")) %>%
  dplyr::mutate(
    TOTAL_NUMBER  = tidyr::replace_na(TOTAL_NUMBER, 0L),                # add zeros back
    taxon_present = TOTAL_NUMBER > 0,                                   # presence/absence flag
    decimal_date  = lubridate::decimal_date(SAMPLE_DATE),               # numeric time for smooths
    SITE_ID.F     = factor(SITE_ID)                                     # site factor for RE
  )

# ============================================================
# 1) Presence/Absence table
#    - Use the joined data as-is for binomial models
# ============================================================
pa_tbl <- fam_data

# ============================================================
# 2) Ordinal categorical table (merge AB4+AB3)
#    - Keep only categorical samples (data_type == "bin")
#    - Map WHPT-style placeholders to ordered classes
#    - Optionally merge highest classes to reduce sparsity
# ============================================================
ord_tbl <- fam_data %>%
  dplyr::filter(data_type == "bin") %>%
  dplyr::mutate(
    ord_class = dplyr::case_when(
      TOTAL_NUMBER == 0L   ~ "AB0",
      TOTAL_NUMBER == 1L   ~ "AB1",
      TOTAL_NUMBER == 3L   ~ "AB2",
      TOTAL_NUMBER == 33L  ~ "AB3",
      TOTAL_NUMBER >= 333L ~ "AB4",
      TRUE                 ~ NA_character_
    ),
    # Merge AB4 into AB3 bucket if desired (helps stability when AB4 is rare)
    ord_class = forcats::fct_collapse(ord_class, AB3p = c("AB3","AB4"))
```

```
    ) %>%
    dplyr::filter(!is.na(ord_class)) %>%
    dplyr::mutate(
      ord_class = factor(ord_class,
                         levels = c("AB0","AB1","AB2","AB3p"),
                         ordered = TRUE)
    )

  # ========================================================
  # 3) Censored Poisson table
  #    - Keep only numeric (count) samples
  #    - Convert exact/1-s.f. counts to integer intervals on the count scale
  #      * 0-9 as point intervals (±0.5), 10-99 etc. as decade blocks
  # ========================================================
  cpois_tbl <- fam_data %>%
    dplyr::filter(data_type == "count") %>%
    dplyr::mutate(
      lower = dplyr::case_when(
        TOTAL_NUMBER <= 9L ~ as.numeric(TOTAL_NUMBER) - 0.5,               # exact counts -> tight boun
        TRUE ~ floor(TOTAL_NUMBER / 10) * 10 - 0.5                          # decade bin lower edge
      ),
      upper = dplyr::case_when(
        TOTAL_NUMBER <= 9L ~ as.numeric(TOTAL_NUMBER) + 0.5,               # exact counts -> tight boun
        TRUE ~ floor(TOTAL_NUMBER / 10) * 10 + 9.5                          # decade bin upper edge
      )
    )

  # ========================================================
  # 4) Censored Normal table (same bounds as cpois)
  # ========================================================
  cnorm_tbl <- cpois_tbl

  # Return all four data frames for this family
  list(
    pa      = pa_tbl,
    ordinal = ord_tbl,
    cpois   = cpois_tbl,
    cnorm   = cnorm_tbl
  )
}

# --- Apply the constructor to all families ---
#     - purrr::map to iterate, then name the list by family
models_data <- purrr::map(
  families,
  make_family_tables,
  roster_tbl = roster_flagged,
  whpt_tbl   = whpt_clean
) %>%
  rlang::set_names(families)

# --- Example access (kept unchanged) ---
pa_aphe    <- models_data[["Aphelocheiridae"]]$pa
```

```r
ord_brach  <- models_data[["Brachycentridae"]]$ordinal
cp_odno    <- models_data[["Odontoceridae"]]$cpois
cn_cord    <- models_data[["Cordulegastridae"]]$cnorm
```

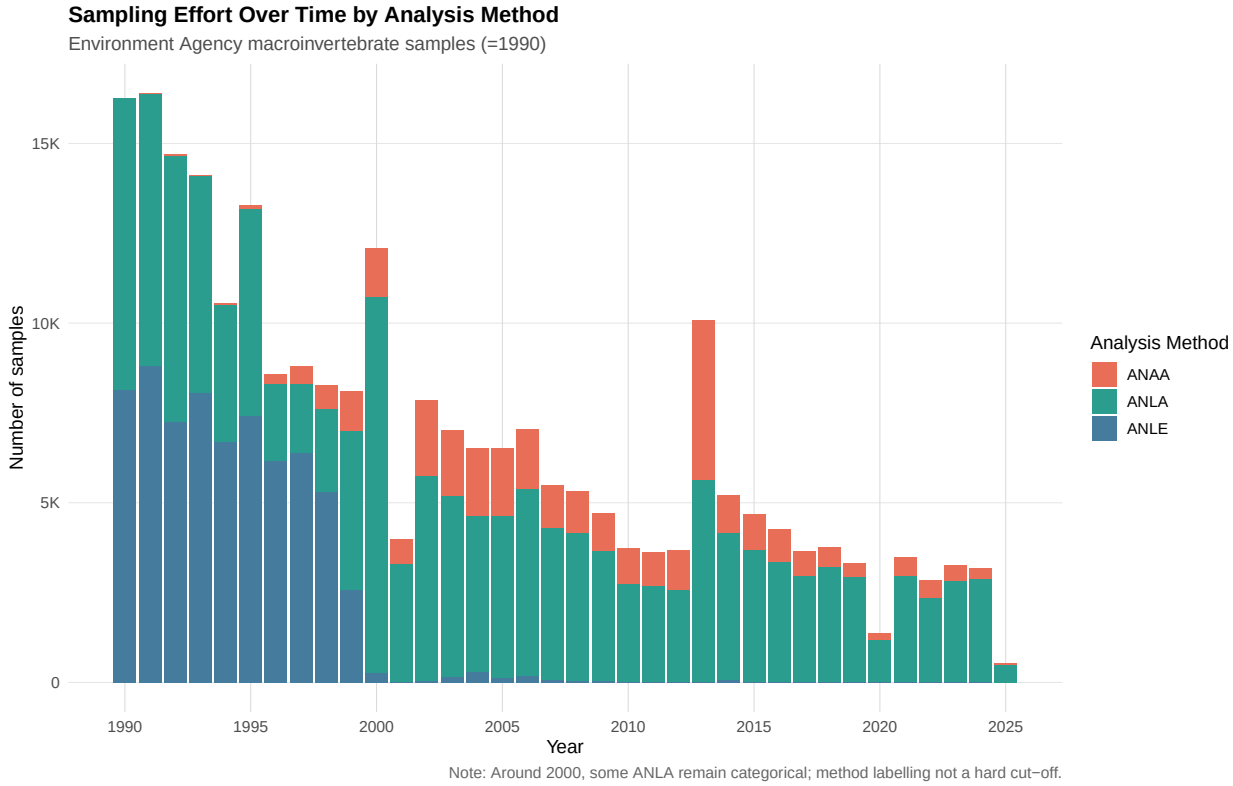
#3. EXPLORATORY DATA ANALYSIS

```r
# ------------------------------------------------------------------------
# Polished EDA plots for the EA macroinvertebrate project
# Expects: metrics_clean, whpt_clean, roster_flagged already created
# Uses: families vector (defined here if missing)
# Purpose: Produce clear, publication-ready figures directly tied to the
#          project questions (methods over time, family frequencies,
#          categorical vs numeric mix, abundance distributions, seasonality,
#          and presence rates).
# ------------------------------------------------------------------------

# If families wasn't defined earlier
if (!exists("families")) {
  families <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")
}

# ---------- Styling ----------
# Minimal, consistent house style for all EDA figures
theme_ea <- function(base_size = 12) {
  ggplot2::theme_minimal(base_size = base_size) +
    ggplot2::theme(
      panel.grid.minor = element_blank(),                 # de-clutter
      panel.grid.major.x = element_line(size = 0.25, colour = "grey85"),
      panel.grid.major.y = element_line(size = 0.25, colour = "grey90"),
      plot.title        = element_text(face = "bold", size = base_size + 2),
      plot.subtitle     = element_text(colour = "grey30"),
      plot.caption      = element_text(colour = "grey40", size = base_size - 2),
      strip.text        = element_text(face = "bold")
    )
}
# Compact numeric labels (e.g., 1.2k, 3.4M) - avoids crowded axes
short_num  <- scales::label_number(scale_cut = scales::cut_short_scale())
# Colour palettes used consistently across plots
method_cols <- c(ANAA = "#E86E58", ANLA = "#2A9D8F", ANLE = "#457B9D")
dtype_cols  <- c(bin = "#F4A261", count = "#2A9D8F")

# ---------- 1) Sampling effort over time by analysis method ----------
# What it shows: volume of sampling each year split by EA analysis method.
# Why it matters: motivates date filtering (>=1990) and shows the method mix over time.
p_effort <- metrics_clean %>%
  dplyr::mutate(year = lubridate::year(SAMPLE_DATE)) %>%
  dplyr::group_by(year, ANALYSIS_METHOD) %>%
  dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
  ggplot2::ggplot(ggplot2::aes(year, n, fill = ANALYSIS_METHOD)) +
  ggplot2::geom_col(width = 0.9) +
  ggplot2::scale_fill_manual(values = method_cols, name = "Analysis Method") +
  ggplot2::scale_x_continuous(breaks = seq(1990, lubridate::year(Sys.Date()) + 1, 5)) +
  ggplot2::scale_y_continuous(labels = short_num) +
```

```
  ggplot2::labs(
    title = "Sampling Effort Over Time by Analysis Method",
    subtitle = "Environment Agency macroinvertebrate samples ( 1990)",
    x = "Year", y = "Number of samples",
    caption = "Note: Around 2000, some ANLA remain categorical; method labelling not a hard cut-off."
  ) +
  theme_ea()
```

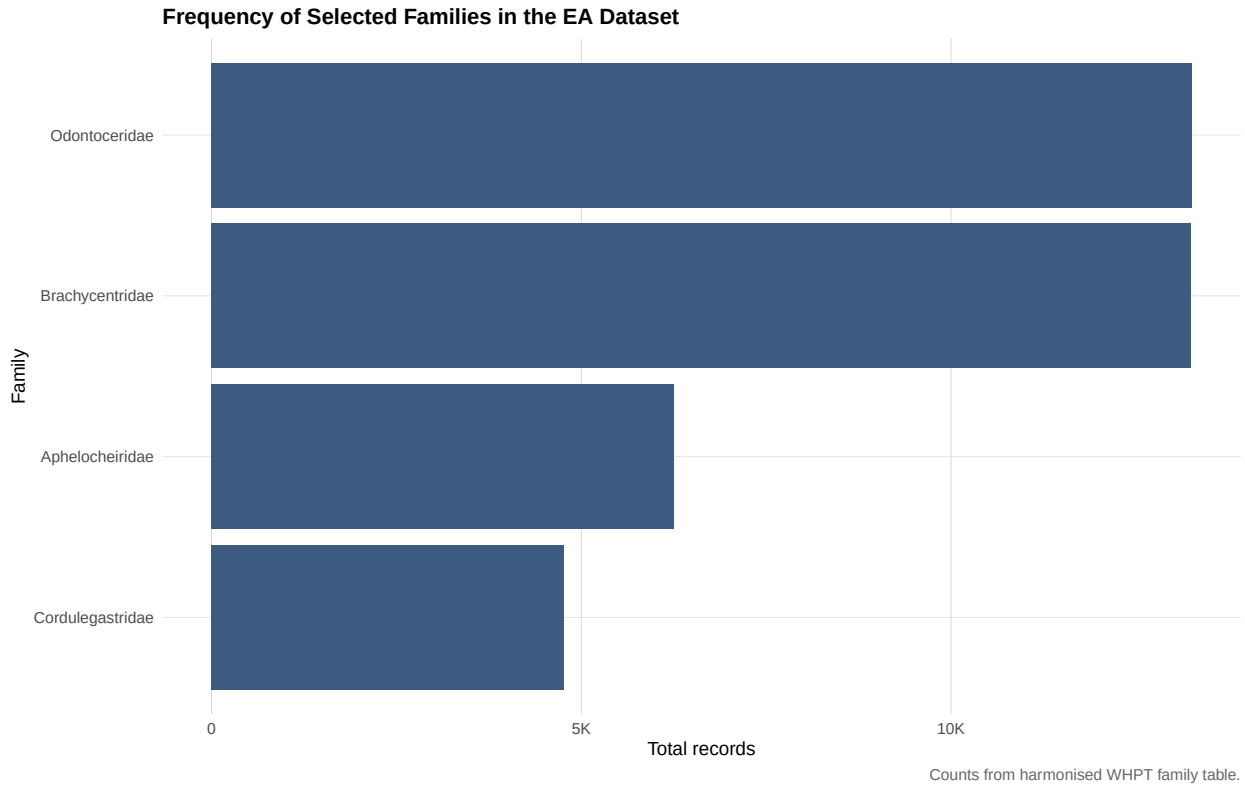
```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
print(p_effort)
```



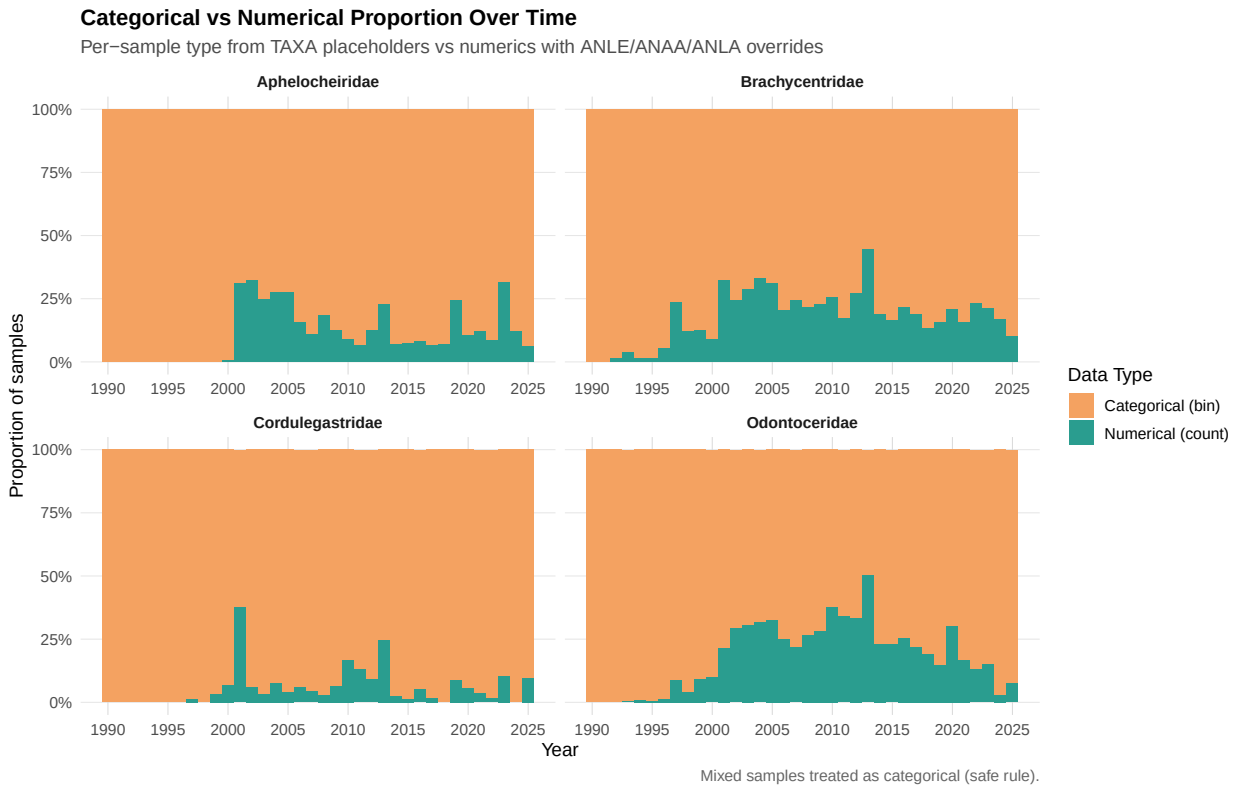

```
# ---------- 2) Frequency of selected families ----------
# What it shows: overall counts of records for the four target families.
# Why it matters: supports the choice of "not too rare, not too common" families.
p_freq <- whpt_clean %>%
  dplyr::filter(EQ_TAXON_UNIT %in% families) %>%
  dplyr::count(EQ_TAXON_UNIT, name = "n") %>%
  dplyr::mutate(EQ_TAXON_UNIT = forcats::fct_reorder(EQ_TAXON_UNIT, n)) %>%
  ggplot2::ggplot(ggplot2::aes(n, EQ_TAXON_UNIT)) +
  ggplot2::geom_col(fill = "#3D5A80") +
  ggplot2::scale_x_continuous(labels = short_num) +
```

```
  ggplot2::labs(
    title = "Frequency of Selected Families in the EA Dataset",
    x = "Total records", y = "Family",
    caption = "Counts from harmonised WHPT family table."
  ) +
  theme_ea()
print(p_freq)
```




```
# ---------- 3) Categorical vs numerical proportion over time (by family) ----------
# What it shows: yearly mix of categorical vs numeric data for each family.
# Why it matters: demonstrates the ANLA/ANLE/ANAA transition and justifies
#                 modelling strategy (ordered categorical vs censored/count).
p_dtype <- roster_flagged %>%
  dplyr::left_join(whpt_clean %>% dplyr::select(SAMPLE_ID, EQ_TAXON_UNIT), by = "SAMPLE_ID") %>%
  dplyr::filter(EQ_TAXON_UNIT %in% families) %>%
  dplyr::mutate(year = lubridate::year(SAMPLE_DATE)) %>%
  dplyr::group_by(EQ_TAXON_UNIT, year, data_type) %>%
  dplyr::summarise(n = dplyr::n(), .groups = "drop_last") %>%
  dplyr::mutate(prop = n / sum(n)) %>%
  dplyr::ungroup() %>%
  ggplot2::ggplot(ggplot2::aes(year, prop, fill = data_type)) +
  ggplot2::geom_col(position = "fill", width = 0.95) +
  ggplot2::facet_wrap(~ EQ_TAXON_UNIT, ncol = 2, scales = "free_x") +
  ggplot2::scale_fill_manual(values = dtype_cols, name = "Data Type",
                    labels = c("Categorical (bin)", "Numerical (count)")) +
  ggplot2::scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  ggplot2::scale_x_continuous(breaks = seq(1990, lubridate::year(Sys.Date()) + 1, 5)) +
```

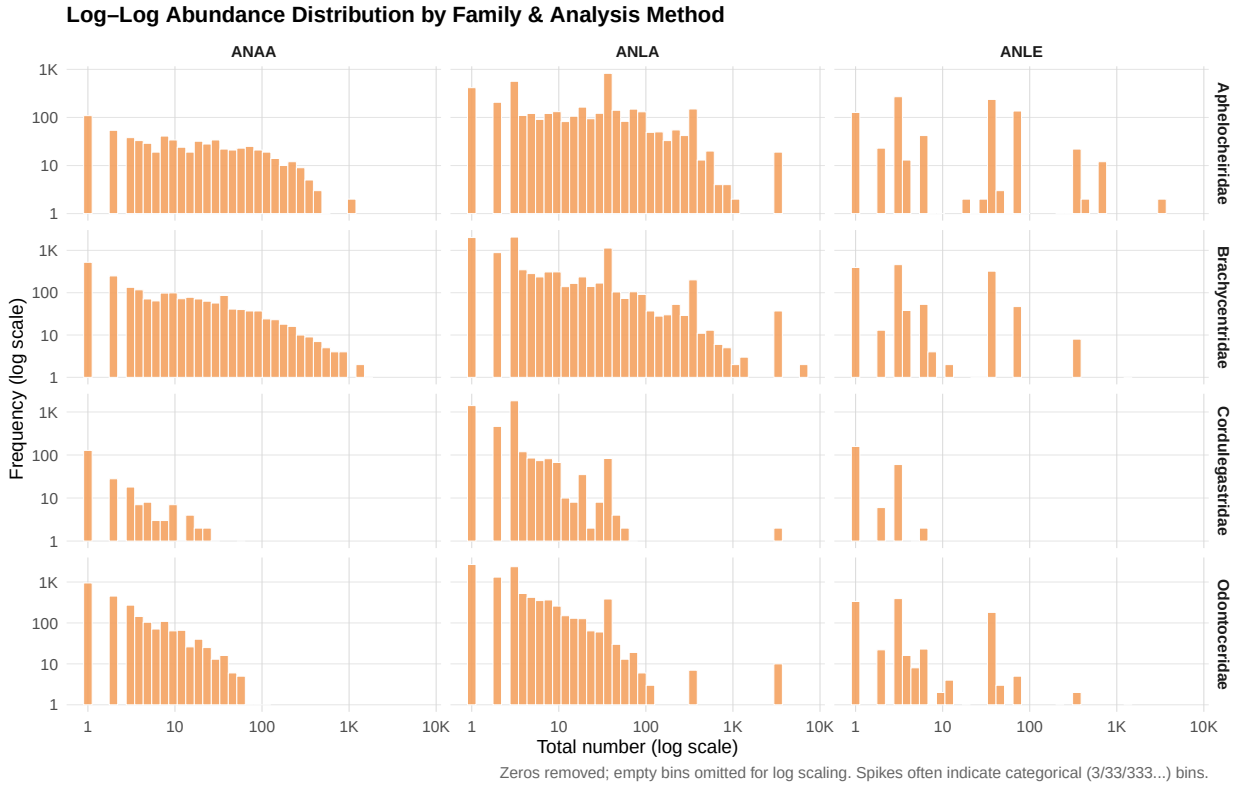
```
  ggplot2::labs(
    title = "Categorical vs Numerical Proportion Over Time",
    subtitle = "Per-sample type from TAXA placeholders vs numerics with ANLE/ANAA/ANLA overrides",
    x = "Year", y = "Proportion of samples",
    caption = "Mixed samples treated as categorical (safe rule)."
  ) +
  theme_ea()
print(p_dtype)
```




```
# ---------- 4) Log-log abundance distributions by family & method (no warnings) ----------
# What it shows: distribution of non-zero abundances by method for each family on log-log axes.
# Why it matters: highlights categorical spikes (3/33/333...) vs smoother numeric counts.
abund_df <- whpt_clean %>%
  dplyr::filter(EQ_TAXON_UNIT %in% families) %>%
  dplyr::left_join(metrics_clean %>% dplyr::select(SAMPLE_ID, ANALYSIS_METHOD), by = "SAMPLE_ID") %>%
  dplyr::filter(!is.na(ANALYSIS_METHOD), !is.na(TOTAL_NUMBER), TOTAL_NUMBER > 0)

p_hist <- ggplot2::ggplot(abund_df, ggplot2::aes(x = TOTAL_NUMBER)) +
  ggplot2::geom_histogram(
    bins = 40,
    ggplot2::aes(y = ggplot2::after_stat(ifelse(count == 0, NA, count))),  # drop empty bins before log
    alpha = 0.9, fill = "#F4A261", colour = "white", linewidth = 0.15
  ) +
  ggplot2::scale_x_log10(labels = short_num) +
  ggplot2::scale_y_log10(labels = short_num) +
  ggplot2::facet_grid(EQ_TAXON_UNIT ~ ANALYSIS_METHOD, scales = "free_y") +
  ggplot2::labs(
```

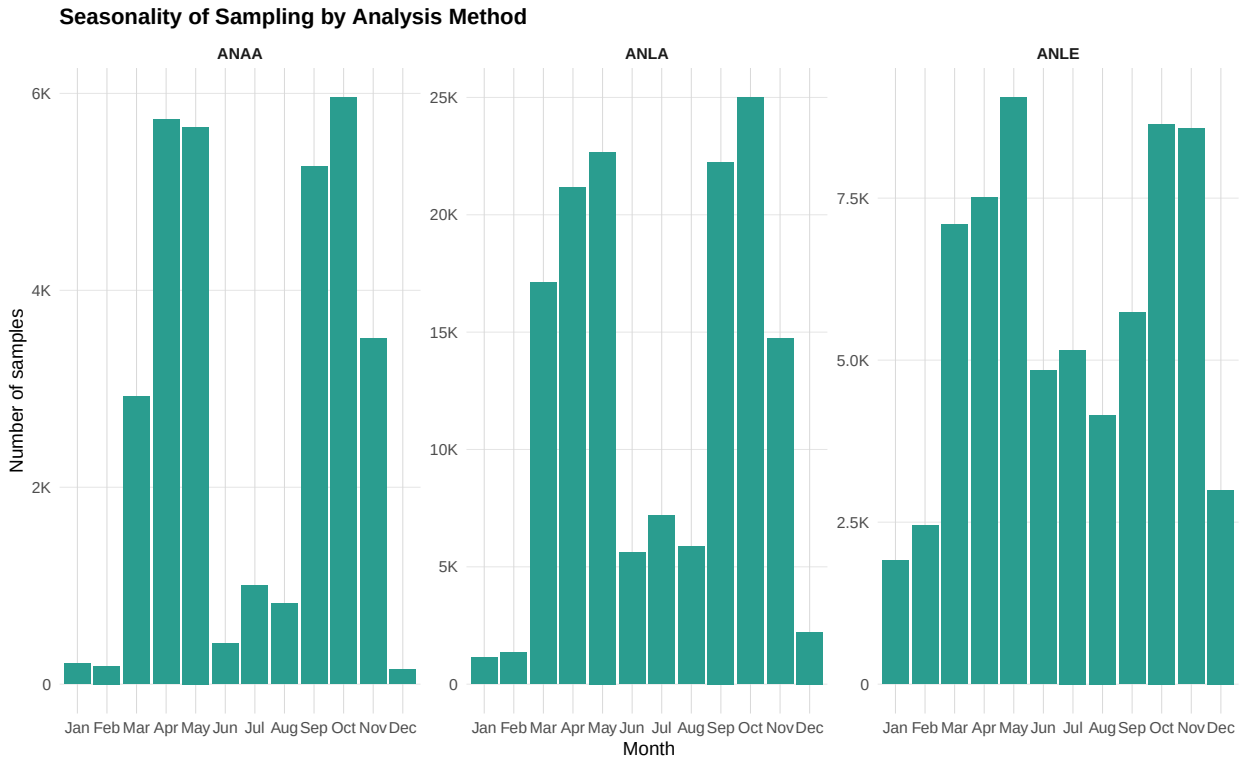
```
    title = "Log-Log Abundance Distribution by Family & Analysis Method",
    x = "Total number (log scale)", y = "Frequency (log scale)",
    caption = "Zeros removed; empty bins omitted for log scaling. Spikes often indicate categorical (3/3
  ) +
  theme_ea()
print(p_hist)
```

```
## Warning: Removed 234 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```




```
# ---------- 5) Seasonality of sampling (by method) ----------
# What it shows: months when sampling occurs, split by analysis method.
# Why it matters: validates spring/autumn focus and the season factor used in models.
p_season <- metrics_clean %>%
  dplyr::mutate(month = factor(lubridate::month(SAMPLE_DATE, label = TRUE), ordered = TRUE)) %>%
  dplyr::group_by(ANALYSIS_METHOD, month) %>%
  dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
  ggplot2::ggplot(ggplot2::aes(month, n)) +
  ggplot2::geom_col(fill = "#2A9D8F") +
  ggplot2::facet_wrap(~ ANALYSIS_METHOD, ncol = 3, scales = "free_y") +
  ggplot2::scale_y_continuous(labels = short_num) +
  ggplot2::labs(
    title = "Seasonality of Sampling by Analysis Method",
    x = "Month", y = "Number of samples",
    caption = "Sampling concentrates in spring (Mar-May) and autumn (Sep-Nov)."
  ) +
```

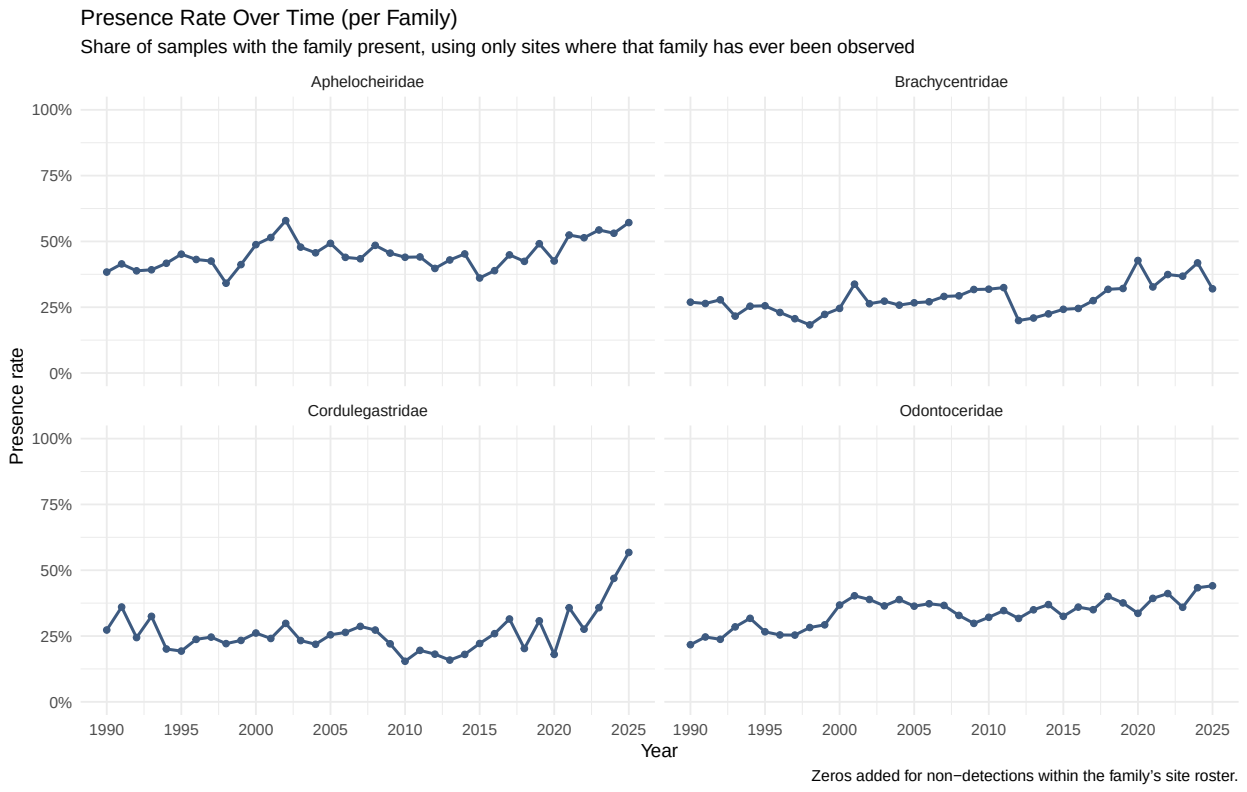
```
  theme_ea()
print(p_season)
```



Sampling concentrates in spring (Mar–May) and autumn (Sep–Nov).

```
# ---------- 6) Presence rate over time (per family) ----------
# helper: presence table for one family using site roster where family ever observed
# Rationale: compute annual presence proportion only across sites that have
#            recorded the family at least once (avoids inflating absences at
#            sites where the family never occurs).
presence_rate_tbl <- function(fam) {
  sites_with <- whpt_clean %>%
    dplyr::filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::distinct(SITE_ID) %>% dplyr::pull()

  roster <- metrics_clean %>%
    dplyr::filter(SITE_ID %in% sites_with) %>%          # only sites where fam seen at least once
    dplyr::select(SITE_ID, SAMPLE_ID, SAMPLE_DATE)

  counts <- whpt_clean %>%
    dplyr::filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)

  roster %>%
    dplyr::left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%
    dplyr::mutate(TOTAL_NUMBER = tidyr::replace_na(TOTAL_NUMBER, 0L),
           present      = TOTAL_NUMBER > 0,
           year         = lubridate::year(SAMPLE_DATE)) %>%
    dplyr::group_by(EQ_TAXON_UNIT = fam, year) %>%
```

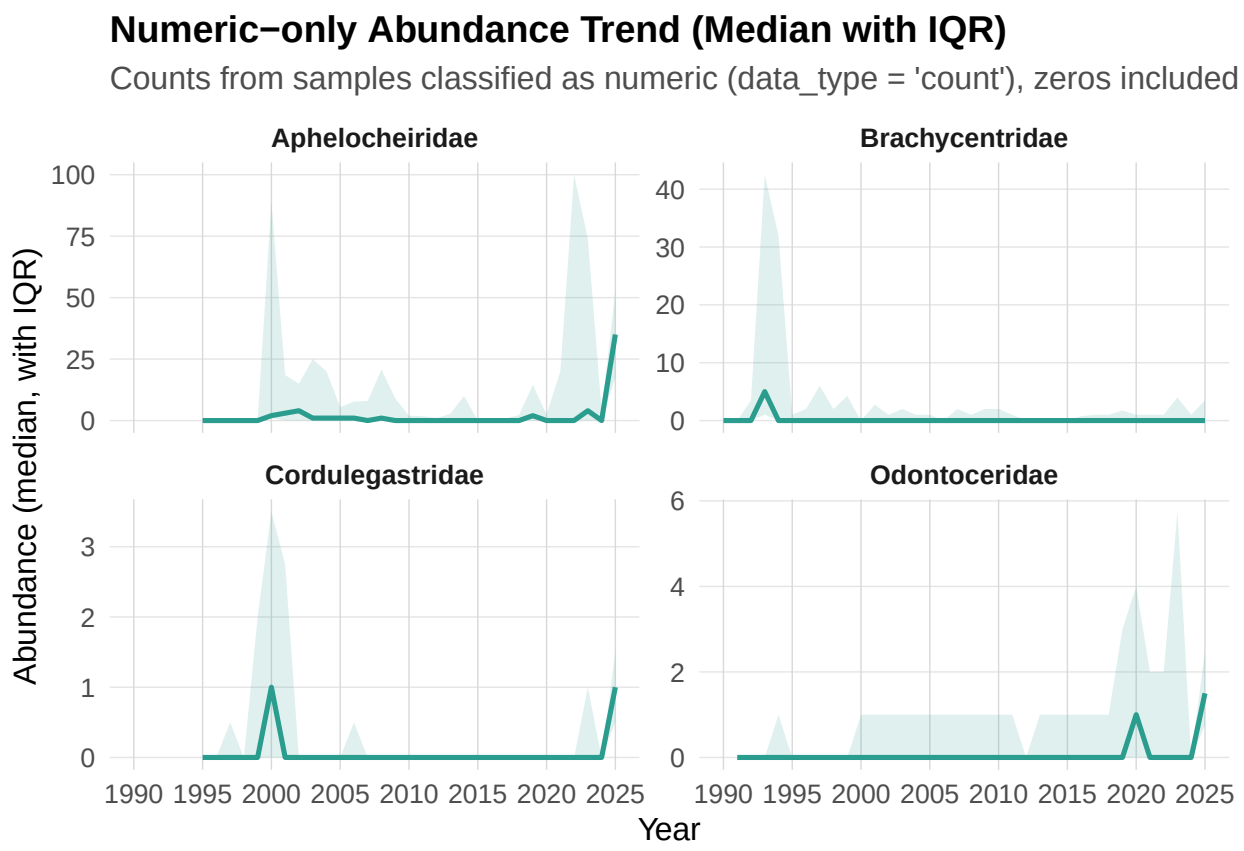
```r
    dplyr::summarise(p_present = mean(present),
              n_samples = dplyr::n(), .groups = "drop")
}

# Bind presence-rate time series for all requested families
presence_df <- dplyr::bind_rows(lapply(families, presence_rate_tbl))

# Plot annual presence proportion with 0-100% scale per panel
p_presence <- presence_df %>%
  ggplot2::ggplot(ggplot2::aes(year, p_present)) +
  ggplot2::geom_line(linewidth = 0.9, colour = "#3D5A80") +
  ggplot2::geom_point(size = 1.6, colour = "#3D5A80") +
  ggplot2::facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  ggplot2::scale_y_continuous(labels = scales::percent_format(accuracy = 1), limits = c(0, 1)) +
  ggplot2::scale_x_continuous(breaks = seq(1990, lubridate::year(Sys.Date()) + 1, 5)) +
  ggplot2::labs(
    title = "Presence Rate Over Time (per Family)",
    subtitle = "Share of samples with the family present, using only sites where that family has ever be
    x = "Year", y = "Presence rate",
    caption = "Zeros added for non-detections within the family's site roster."
  ) +
  ggplot2::theme_minimal(base_size = 12)
print(p_presence)
```




```r
# ------------------------------------------------------------------
# EDA add-ons for abundance & data-type patterns
# (Uses: whpt_clean, roster_flagged, sites_clean, families,
```

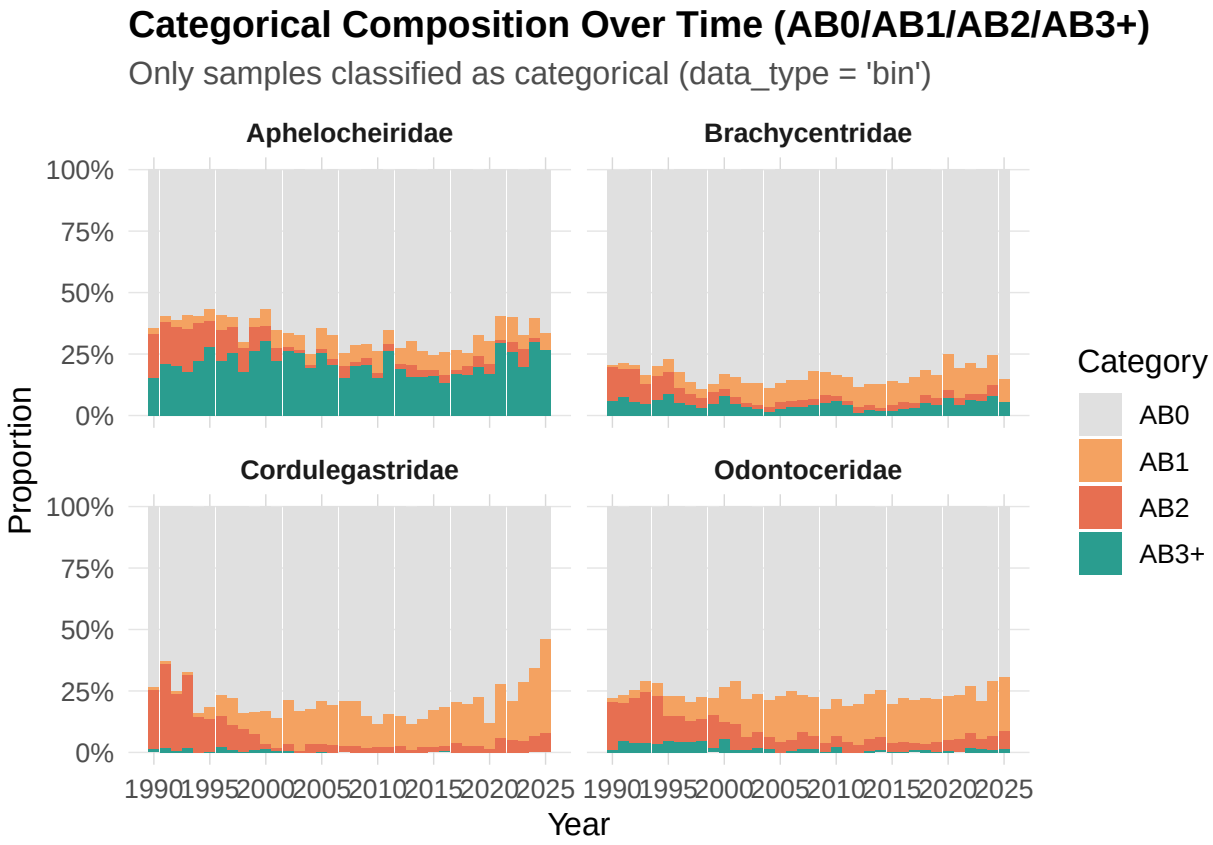
```r
#        plus `theme_ea()` and `short_num` defined earlier.)
# ---------------------------------------------------------------------

# ----------------------------
# Helpers
# ----------------------------

# Return the roster (metrics rows) for sites where a given family
# has been observed at least once (so we add zeros correctly).
family_roster <- function(fam) {
  sites_with <- whpt_clean %>%
    filter(EQ_TAXON_UNIT == fam) %>%
    distinct(SITE_ID) %>%
    pull()
  roster_flagged %>% filter(SITE_ID %in% sites_with)
}

# Join family counts to that roster and fill non-detections with 0.
# (Qualified dplyr::select to avoid conflicts with other packages.)
join_counts_zero <- function(roster_tbl, fam) {
  counts <- whpt_clean %>%
    filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)
  roster_tbl %>%
    left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%
    mutate(TOTAL_NUMBER = replace_na(TOTAL_NUMBER, 0L))
}

# ----------------------------
# 1) Numeric-only abundance trend (median + IQR) per family
#    • Uses only samples classified as numeric ("count")
#    • Adds zeros (explicit absences) within the family's site roster
# ----------------------------
num_trend <- bind_rows(lapply(families, function(fam) {
  roster <- family_roster(fam) %>% filter(data_type == "count")
  join_counts_zero(roster, fam) %>%
    mutate(year = year(SAMPLE_DATE)) %>%
    group_by(EQ_TAXON_UNIT = fam, year) %>%
    summarise(
      n   = n(),
      med = median(TOTAL_NUMBER),
      q25 = quantile(TOTAL_NUMBER, 0.25),
      q75 = quantile(TOTAL_NUMBER, 0.75),
      .groups = "drop"
    )
}))

p_num_trend <- ggplot(num_trend, aes(year, med)) +
  geom_ribbon(aes(ymin = q25, ymax = q75), alpha = 0.15, fill = "#2A9D8F") +
  geom_line(colour = "#2A9D8F", linewidth = 0.9) +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2, scales = "free_y") +
  scale_y_continuous(labels = short_num) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
```

```
  labs(
    title    = "Numeric-only Abundance Trend (Median with IQR)",
    subtitle = "Counts from samples classified as numeric (data_type = 'count'), zeros included",
    x = "Year", y = "Abundance (median, with IQR)"
  ) +
  theme_ea()
print(p_num_trend)
```



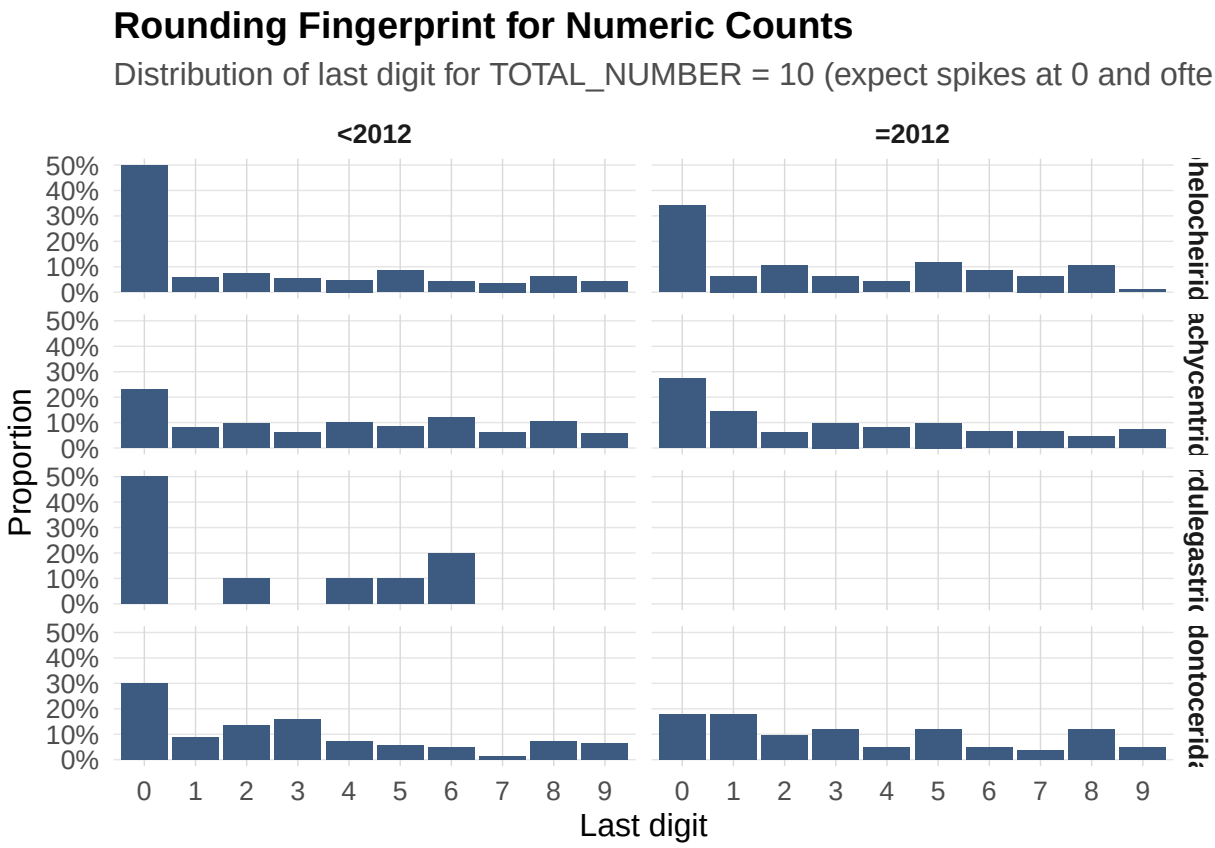

```
# ----------------------------
# 2) Categorical composition over time (AB0/AB1/AB2/AB3+)
#    • Uses only samples classified as categorical ("bin")
#    • Collapses high bins to AB3+ and shows proportions per year
# ----------------------------
cat_comp <- bind_rows(lapply(families, function(fam) {
  roster <- family_roster(fam) %>% filter(data_type == "bin")
  df <- join_counts_zero(roster, fam) %>%
    mutate(
      year = year(SAMPLE_DATE),
      ord  = case_when(
        TOTAL_NUMBER == 0L   ~ "AB0",
        TOTAL_NUMBER == 1L   ~ "AB1",
        TOTAL_NUMBER == 3L   ~ "AB2",
        TOTAL_NUMBER >= 33L  ~ "AB3+",
        TRUE                 ~ NA_character_
      )
```

```
    ) %>%
    filter(!is.na(ord)) %>%
    group_by(EQ_TAXON_UNIT = fam, year, ord) %>%
    summarise(n = n(), .groups = "drop") %>%
    group_by(EQ_TAXON_UNIT, year) %>%
    mutate(prop = n / sum(n)) %>%
    ungroup()
  df
}))

p_cat_comp <- ggplot(cat_comp, aes(year, prop, fill = ord)) +
  geom_col(width = 0.95, position = "fill") +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  scale_fill_manual(
    values = c(AB0 = "#E0E0E0", AB1 = "#F4A261", AB2 = "#E76F51", `AB3+` = "#2A9D8F"),
    name   = "Category"
  ) +
  scale_y_continuous(labels = percent_format(accuracy = 1)) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  labs(
    title    = "Categorical Composition Over Time (AB0/AB1/AB2/AB3+)",
    subtitle = "Only samples classified as categorical (data_type = 'bin')",
    x = "Year", y = "Proportion"
  ) +
  theme_ea()
print(p_cat_comp)
```



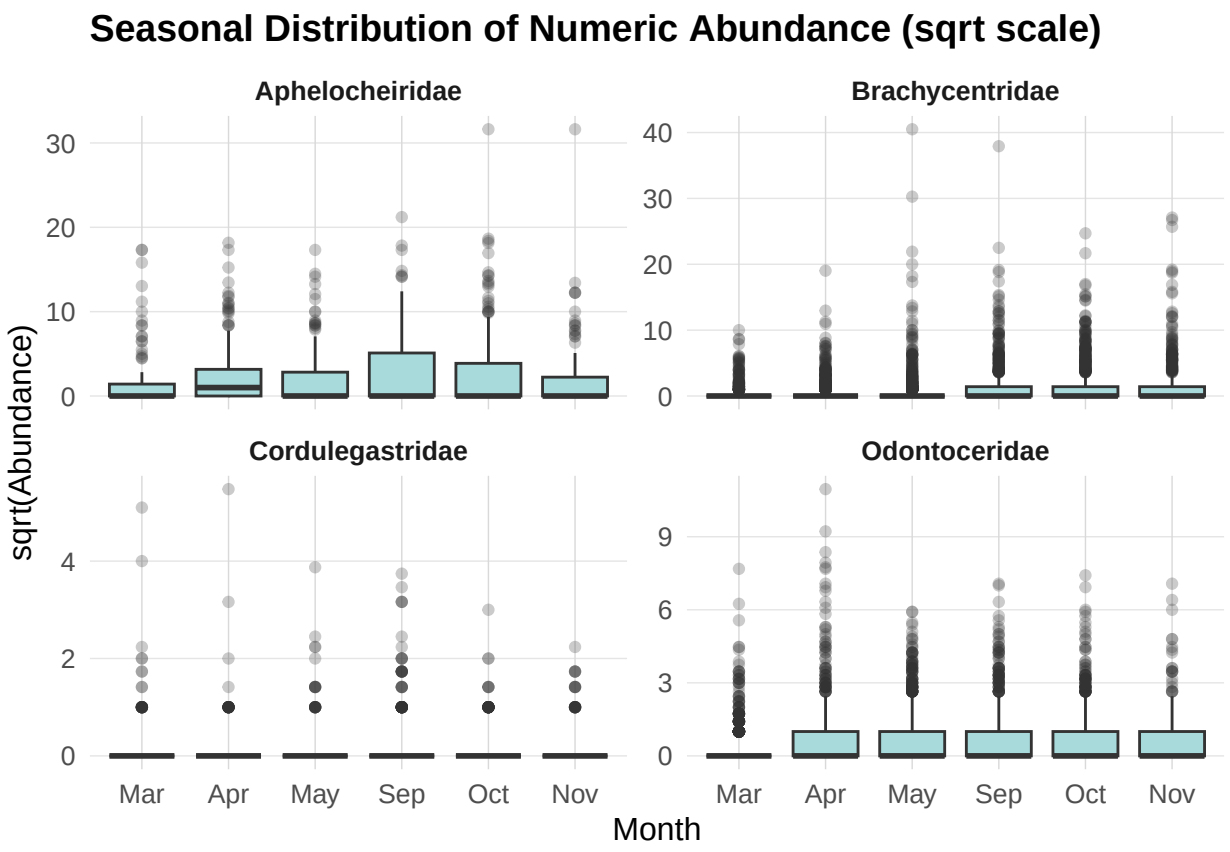

```
# ----------------------------
# 3) Rounding fingerprint for numeric counts (last digit distribution)
#    • Focus on TOTAL_NUMBER  10 where 1 s.f. rounding is typical
#    • Optional pre/post-2012 era split for visual comparison
# ----------------------------
round_fp <- bind_rows(lapply(families, function(fam) {
  roster <- family_roster(fam) %>% filter(data_type == "count")
  join_counts_zero(roster, fam) %>%
    filter(TOTAL_NUMBER >= 10) %>%                          # restrict to 1 s.f. zone
    mutate(
      last_digit = TOTAL_NUMBER %% 10,
      era        = if_else(year(SAMPLE_DATE) < 2012, "<2012", " 2012")
    ) %>%
    group_by(EQ_TAXON_UNIT = fam, era, last_digit) %>%
    summarise(n = n(), .groups = "drop") %>%
    group_by(EQ_TAXON_UNIT, era) %>%
    mutate(prop = n / sum(n)) %>%
    ungroup()
}))

p_round_fp <- ggplot(round_fp, aes(factor(last_digit), prop)) +
  geom_col(fill = "#3D5A80") +
  facet_grid(EQ_TAXON_UNIT ~ era) +
  scale_y_continuous(labels = percent_format(accuracy = 1)) +
  labs(
    title    = "Rounding Fingerprint for Numeric Counts",
```

```
    subtitle = "Distribution of last digit for TOTAL_NUMBER  10 (expect spikes at 0 and often 5)",
    x = "Last digit", y = "Proportion"
  ) +
  theme_ea()
print(p_round_fp)
```



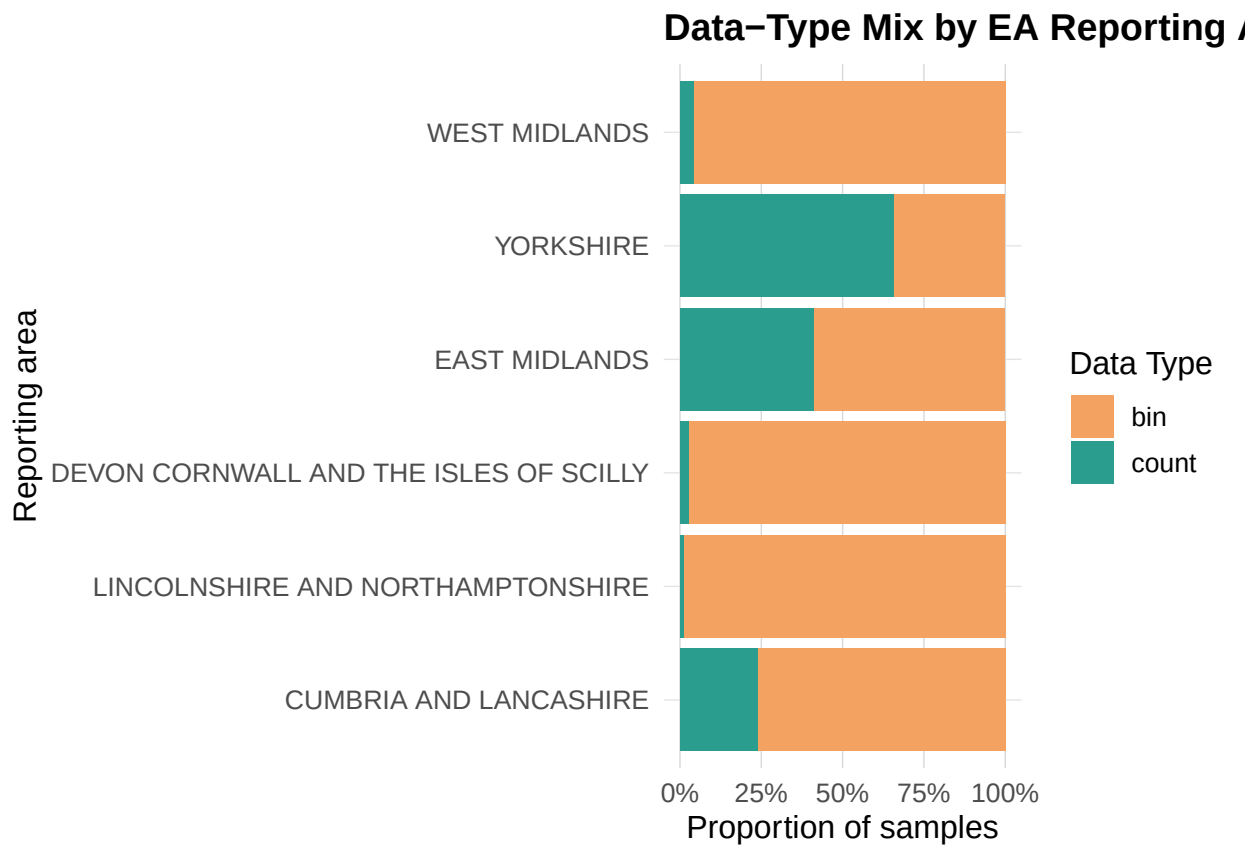

```
# ----------------------------
# 4) Seasonal abundance (numeric counts, sqrt scale)
#    • Boxplots of sqrt(count) by month for each family
# ----------------------------
season_df <- bind_rows(lapply(families, function(fam) {
  roster <- family_roster(fam) %>% filter(data_type == "count")
  join_counts_zero(roster, fam) %>%
    mutate(
      month         = factor(month(SAMPLE_DATE, label = TRUE), ordered = TRUE),
      EQ_TAXON_UNIT = fam
    )
}))

p_season_abund <- ggplot(season_df, aes(month, sqrt(TOTAL_NUMBER))) +
  geom_boxplot(outlier.alpha = 0.25, fill = "#A8DADC") +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2, scales = "free_y") +
  labs(
    title = "Seasonal Distribution of Numeric Abundance (sqrt scale)",
    x = "Month", y = "sqrt(Abundance)"
```

```
  ) +
  theme_ea()
print(p_season_abund)
```



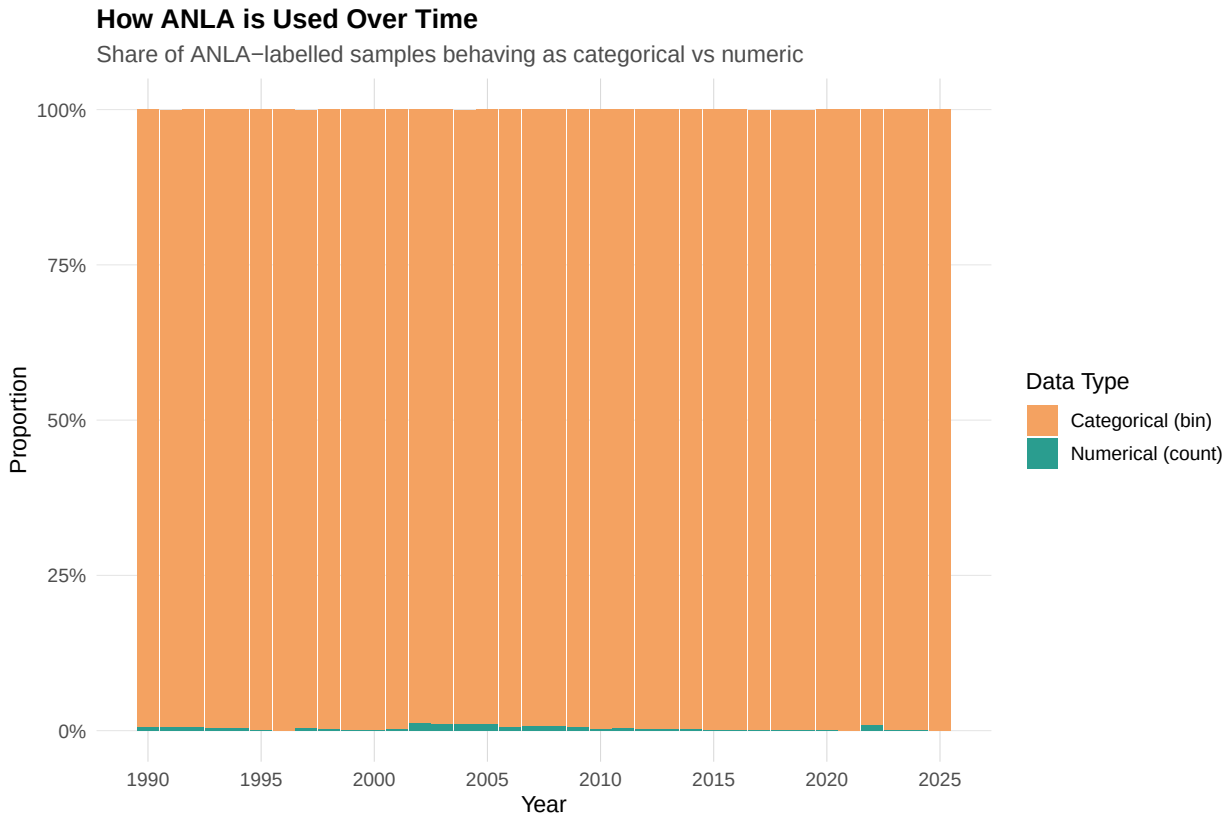

```
# ----------------------------
# 5) Regional heterogeneity: data-type mix by EA area (top 12)
#    • Shows proportion of categorical vs numeric samples by area
#    • Uses sites_clean to fetch REPORTING_AREA for each SITE_ID
# ----------------------------
area_mix <- roster_flagged %>%
  left_join(sites_clean %>% dplyr::select(SITE_ID, REPORTING_AREA), by = "SITE_ID") %>%
  filter(!is.na(REPORTING_AREA)) %>%
  count(REPORTING_AREA, data_type, name = "n") %>%
  group_by(REPORTING_AREA) %>%
  mutate(total = sum(n), prop = n / total) %>%
  ungroup() %>%
  slice_max(order_by = total, n = 12, with_ties = FALSE) %>%
  mutate(REPORTING_AREA = fct_reorder(REPORTING_AREA, total))

p_area_mix <- ggplot(area_mix, aes(REPORTING_AREA, prop, fill = data_type)) +
  geom_col(position = "fill") +
  coord_flip() +
  scale_fill_manual(values = c(bin = "#F4A261", count = "#2A9D8F"), name = "Data Type") +
  scale_y_continuous(labels = percent_format(accuracy = 1)) +
  labs(
```

```
    title = "Data-Type Mix by EA Reporting Area (Top 12 by sample volume)",
    x = "Reporting area", y = "Proportion of samples"
  ) +
  theme_ea()
print(p_area_mix)
```




```
# ============================================================
# Lightweight theme + number formatter
#   - 1) ANLA usage shift over time
#   - 2) Ordinal sparsity (ABO/AB1/AB2/AB3+) - full + zoomed views
#   - 3) Rounding fingerprint for numeric counts
#   - 4) Site-level presence map for one family
#   - 5) Same-day replicates (pre de-dup) by year
# ============================================================

# --- Short number axis formatter (e.g., 1k, 2.5k) ----------------------------
if (!exists("short_num")) short_num <- label_number(scale_cut = cut_short_scale())

# ============================================================
# 1) ANLA meaning shift through time (categorical vs numeric)
#    • Restrict to ANALYSIS_METHOD == "ANLA"
#    • Compute yearly share of bin vs count
# ============================================================
p_anla_shift <- roster_flagged %>%
  mutate(year = year(SAMPLE_DATE)) %>%
```

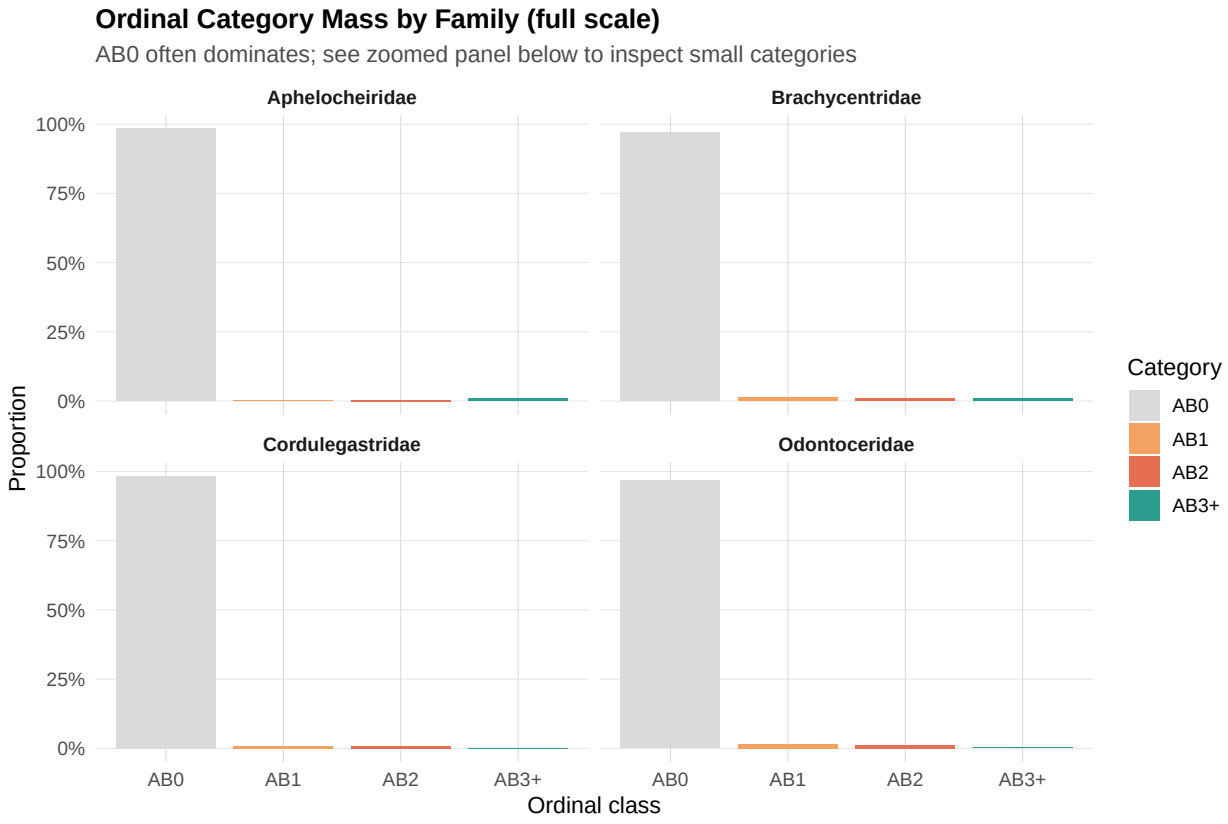
```
  filter(ANALYSIS_METHOD == "ANLA") %>%
  count(year, data_type, name = "n") %>%
  group_by(year) %>% mutate(prop = n / sum(n)) %>% ungroup() %>%
  ggplot(aes(year, prop, fill = data_type)) +
  geom_col(width = 0.95, position = "fill") +
  scale_y_continuous(labels = percent_format(accuracy = 1)) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  scale_fill_manual(values = c(bin = "#F4A261", count = "#2A9D8F"),
                    name = "Data Type",
                    labels = c("Categorical (bin)", "Numerical (count)")) +
  labs(
    title = "How ANLA is Used Over Time",
    subtitle = "Share of ANLA-labelled samples behaving as categorical vs numeric",
    x = "Year", y = "Proportion"
  ) + theme_ea()
print(p_anla_shift)
```



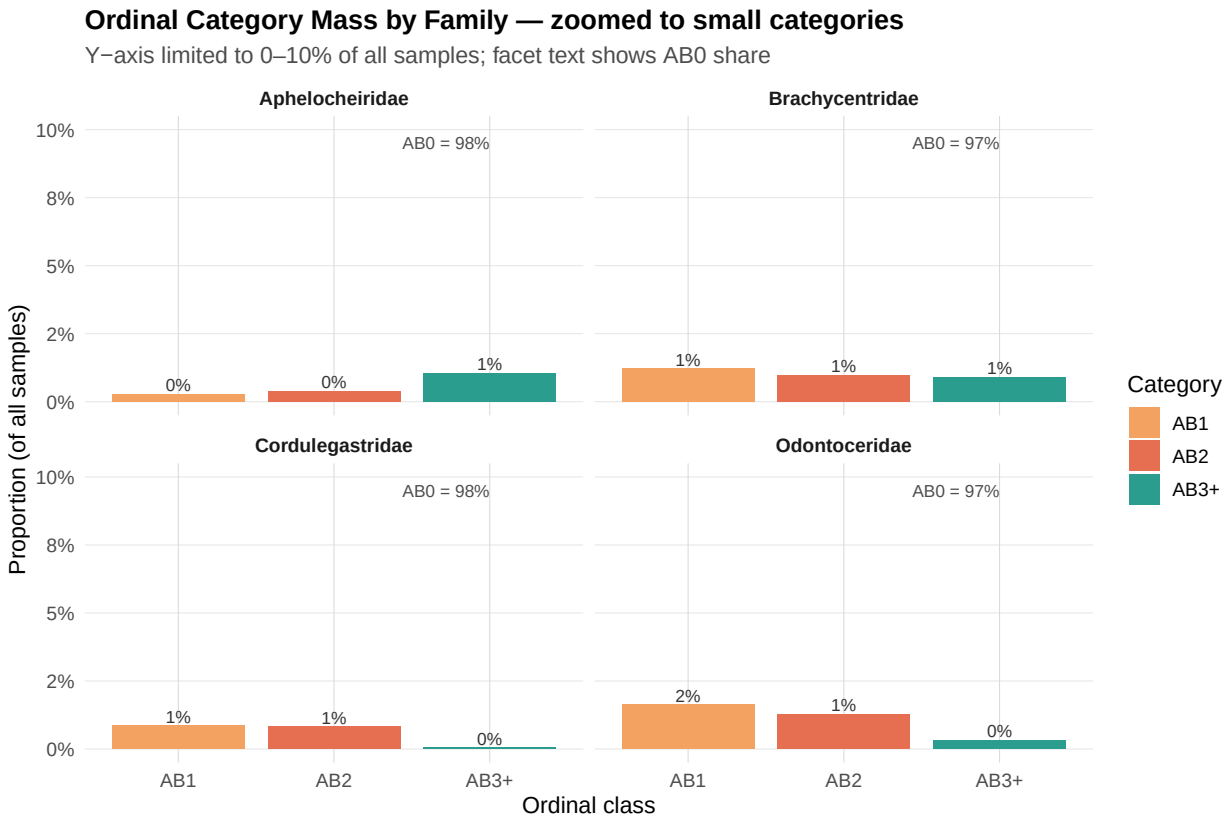

```
# ================================================================
# 2) Ordinal sparsity: AB0 / AB1 / AB2 / AB3+ by family (all years)
#    • Work on categorical samples only (data_type == 'bin')
#    • Add explicit zeros; collapse high bins to AB3+
#    Output: (A) full stacked, (B) zoomed to small categories
# ================================================================

# -- Build once: overall category proportions per family ----------------------
cat_sparsity_df <- bind_rows(lapply(families, function(fam) {
```

```r
  roster <- roster_flagged %>% filter(data_type == "bin")                # only categorical samples
  counts <- whpt_clean %>%
    filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)                      # minimal join fields
  roster %>%
    left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%                 # add counts for this family
    mutate(TOTAL_NUMBER = replace_na(TOTAL_NUMBER, 0L),                  # explicit zeros for non-dete
           ord = case_when(                                              # map raw values to ordinal b
             TOTAL_NUMBER == 0L  ~ "AB0",
             TOTAL_NUMBER == 1L  ~ "AB1",
             TOTAL_NUMBER == 3L  ~ "AB2",
             TOTAL_NUMBER >= 33L ~ "AB3+",
             TRUE ~ NA_character_
           )) %>%
    filter(!is.na(ord)) %>%                                              # drop any unexpected values
    count(EQ_TAXON_UNIT = fam, ord, name = "n")                          # counts per bin
})) %>%
  group_by(EQ_TAXON_UNIT) %>%
  mutate(prop = n / sum(n)) %>%                                          # within-family proportions
  ungroup()

# -- (A) Full stacked view ------------------------------------------------
p_cat_sparsity_better <- ggplot(cat_sparsity_df,
                                aes(x = fct_relevel(ord,"AB0","AB1","AB2","AB3+"),
                                    y = prop, fill = ord)) +
  geom_col(width = 0.85) +                                               # 100% stacked bars
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +                                # one panel per family
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  scale_fill_manual(values = c(AB0="#DADADA", AB1="#F4A261", AB2="#E76F51", `AB3+`="#2A9D8F"),
                    name = "Category") +
  labs(title = "Ordinal Category Mass by Family (full scale)",
       subtitle = "AB0 often dominates; see zoomed panel below to inspect small categories",
       x = "Ordinal class", y = "Proportion") +
  theme_ea()

# -- (B) Zoomed view (exclude AB0) ----------------------------------------
ab0_lab <- cat_sparsity_df %>%
  filter(ord == "AB0") %>%
  transmute(EQ_TAXON_UNIT, lab = paste0("AB0 = ", scales::percent(prop, 1)))  # facet annotation

p_cat_sparsity_zoom <- cat_sparsity_df %>%
  filter(ord != "AB0") %>%                                               # focus on small categories
  ggplot(aes(x = fct_relevel(ord,"AB1","AB2","AB3+"), y = prop, fill = ord)) +
  geom_col(width = 0.85) +
  geom_text(aes(label = scales::percent(prop, 1)),                        # label bars with %
            vjust = -0.25, size = 3, colour = "grey20") +
  geom_text(data = ab0_lab, aes(x = "AB3+", y = 0.095, label = lab),      # show AB0 share per facet
            inherit.aes = FALSE, hjust = 1, size = 3.2, colour = "grey30") +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  coord_cartesian(ylim = c(0, 0.10)) +                                   # zoom to 0-10%
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  scale_fill_manual(values = c(AB1="#F4A261", AB2="#E76F51", `AB3+`="#2A9D8F"),
                    name = "Category") +
```

```
  labs(title = "Ordinal Category Mass by Family - zoomed to small categories",
       subtitle = "Y-axis limited to 0-10% of all samples; facet text shows AB0 share",
       x = "Ordinal class", y = "Proportion (of all samples)") +
  theme_ea()

print(p_cat_sparsity_better)
```

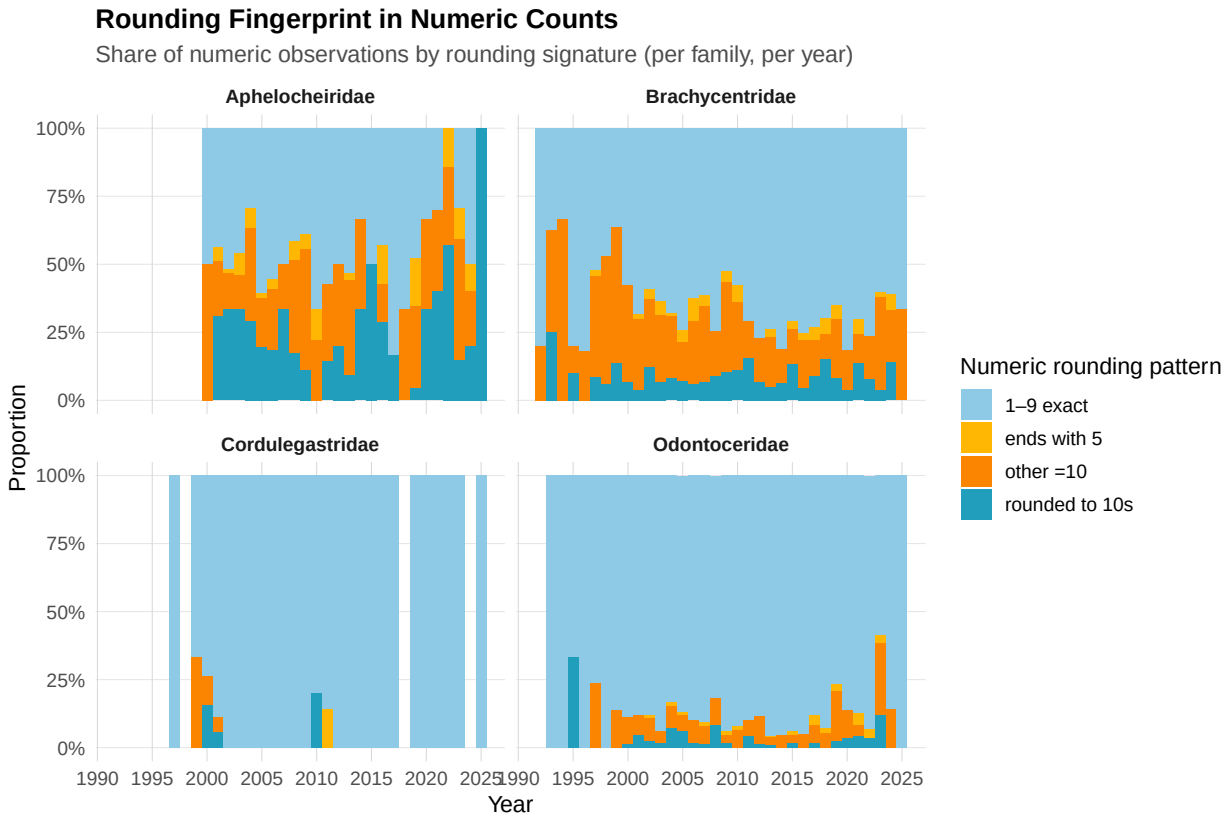



```
print(p_cat_sparsity_zoom)
```



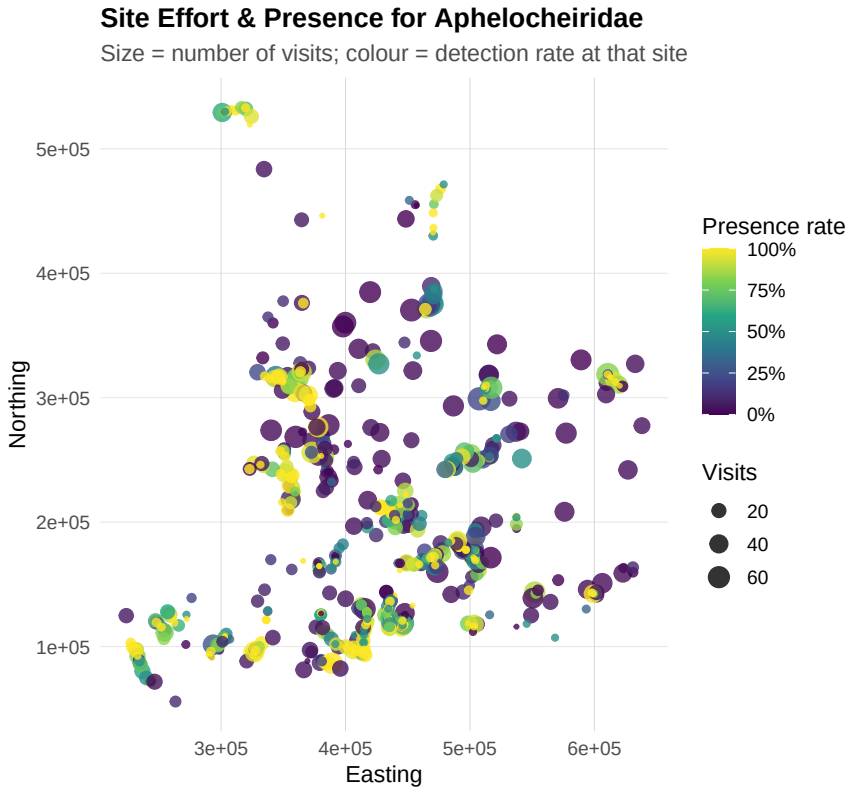

```
# ======================================================================
# 3) Rounding mix (numeric counts): 1-9 exact vs tens vs 5s vs other
#    • Work on numeric samples only (data_type == 'count')
#    • Categorise rounding signature and plot yearly proportions
# ======================================================================
p_rounding_mix <- bind_rows(lapply(families, function(fam) {
  roster <- roster_flagged %>% filter(data_type == "count")
  counts <- whpt_clean %>%
    filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)  # qualified select to avoid masking
  roster %>%
    left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%
    mutate(TOTAL_NUMBER = replace_na(TOTAL_NUMBER, 0L),
           group = case_when(
             TOTAL_NUMBER >= 1  & TOTAL_NUMBER <= 9  ~ "1-9 exact",
             TOTAL_NUMBER >= 10 & TOTAL_NUMBER %% 10 == 0 ~ "rounded to 10s",
             TOTAL_NUMBER >= 10 & TOTAL_NUMBER %% 10 == 5 ~ "ends with 5",
             TOTAL_NUMBER >= 10                           ~ "other 10",
             TRUE                                          ~ NA_character_
           ),
           year = year(SAMPLE_DATE)) %>%
    filter(!is.na(group)) %>%
    count(EQ_TAXON_UNIT = fam, year, group, name = "n") %>%
    group_by(EQ_TAXON_UNIT, year) %>% mutate(prop = n/sum(n)) %>% ungroup()
})) %>%
  ggplot(aes(year, prop, fill = group)) +
  geom_col(width = 0.95, position = "fill") +
```

```
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  scale_y_continuous(labels = percent_format(accuracy = 1)) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  scale_fill_manual(values = c("1-9 exact"="#8ecae6","rounded to 10s"="#219ebc",
                               "ends with 5"="#ffb703","other  10"="#fb8500"),
                    name = "Numeric rounding pattern") +
  labs(
    title = "Rounding Fingerprint in Numeric Counts",
    subtitle = "Share of numeric observations by rounding signature (per family, per year)",
    x = "Year", y = "Proportion"
  ) + theme_ea()
print(p_rounding_mix)
```



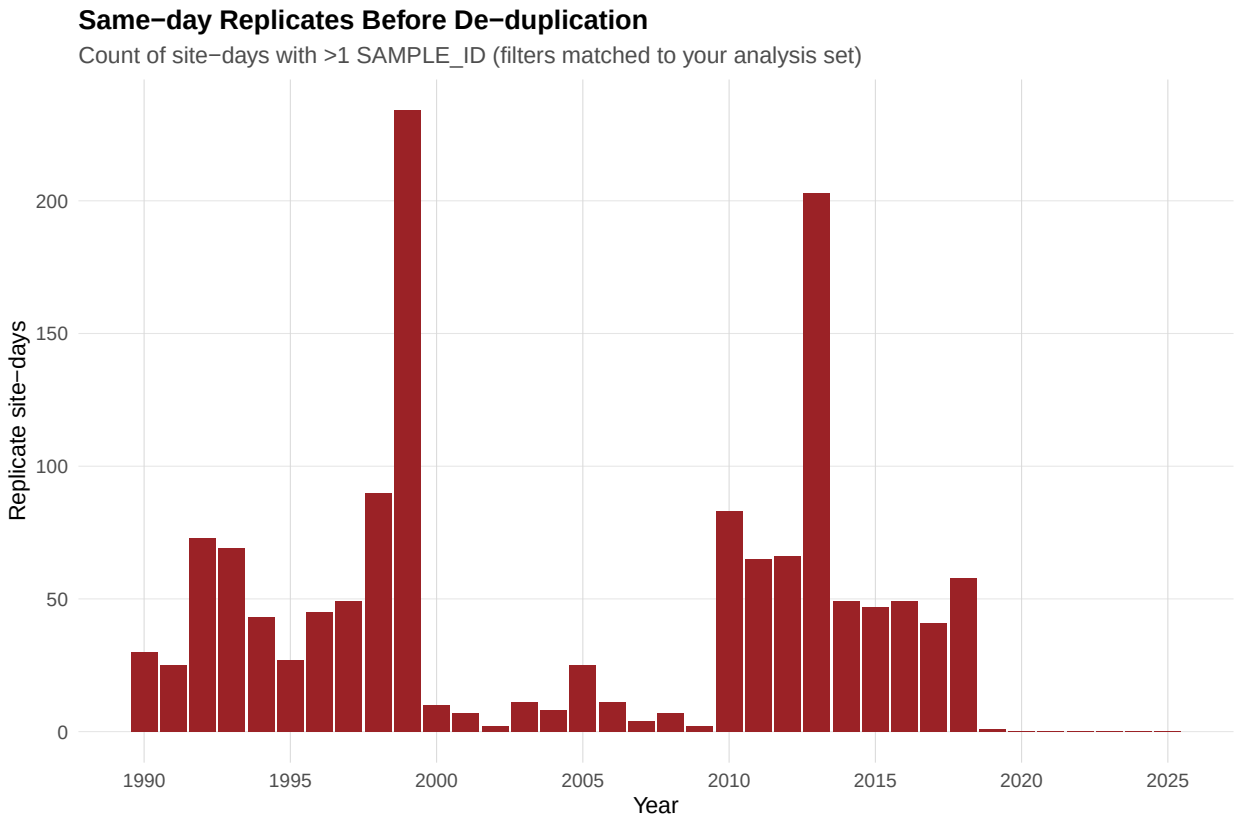

```
# ======================================================================
# 4) Site-level presence map for one family (effort & detection)
#    • Pick first family; compute visits & detection rate per site
#    • Join EA coordinates; scatter by easting/northing
# ======================================================================
fam_map <- families[1]  # pick any family to display
site_presence <- {
  sites_with <- whpt_clean %>%
    filter(EQ_TAXON_UNIT == fam_map) %>% distinct(SITE_ID) %>% pull()
  roster <- roster_flagged %>% filter(SITE_ID %in% sites_with)
  counts <- whpt_clean %>% filter(EQ_TAXON_UNIT == fam_map) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)  # qualified select
```

```
  roster %>%
    left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%
    mutate(TOTAL_NUMBER = replace_na(TOTAL_NUMBER, 0L),
           present = TOTAL_NUMBER > 0) %>%
    group_by(SITE_ID) %>%
    summarise(visits = n(), p_present = mean(present), .groups = "drop") %>%
    left_join(sites_clean %>% dplyr::select(SITE_ID, FULL_EASTING, FULL_NORTHING),  # qualified select
              by = "SITE_ID") %>%
    filter(!is.na(FULL_EASTING), !is.na(FULL_NORTHING))
}

p_map <- ggplot(site_presence, aes(FULL_EASTING, FULL_NORTHING)) +
  geom_point(aes(size = visits, colour = p_present), alpha = 0.8) +
  scale_size_continuous(range = c(0.8, 5), name = "Visits") +
  scale_colour_viridis_c(name = "Presence rate", labels = percent_format(accuracy = 1)) +
  coord_equal() +
  labs(
    title = paste0("Site Effort & Presence for ", fam_map),
    subtitle = "Size = number of visits; colour = detection rate at that site",
    x = "Easting", y = "Northing",
    caption = "British National Grid coordinates; for cartographic basemaps use {sf}."
  ) + theme_ea()
print(p_map)
```



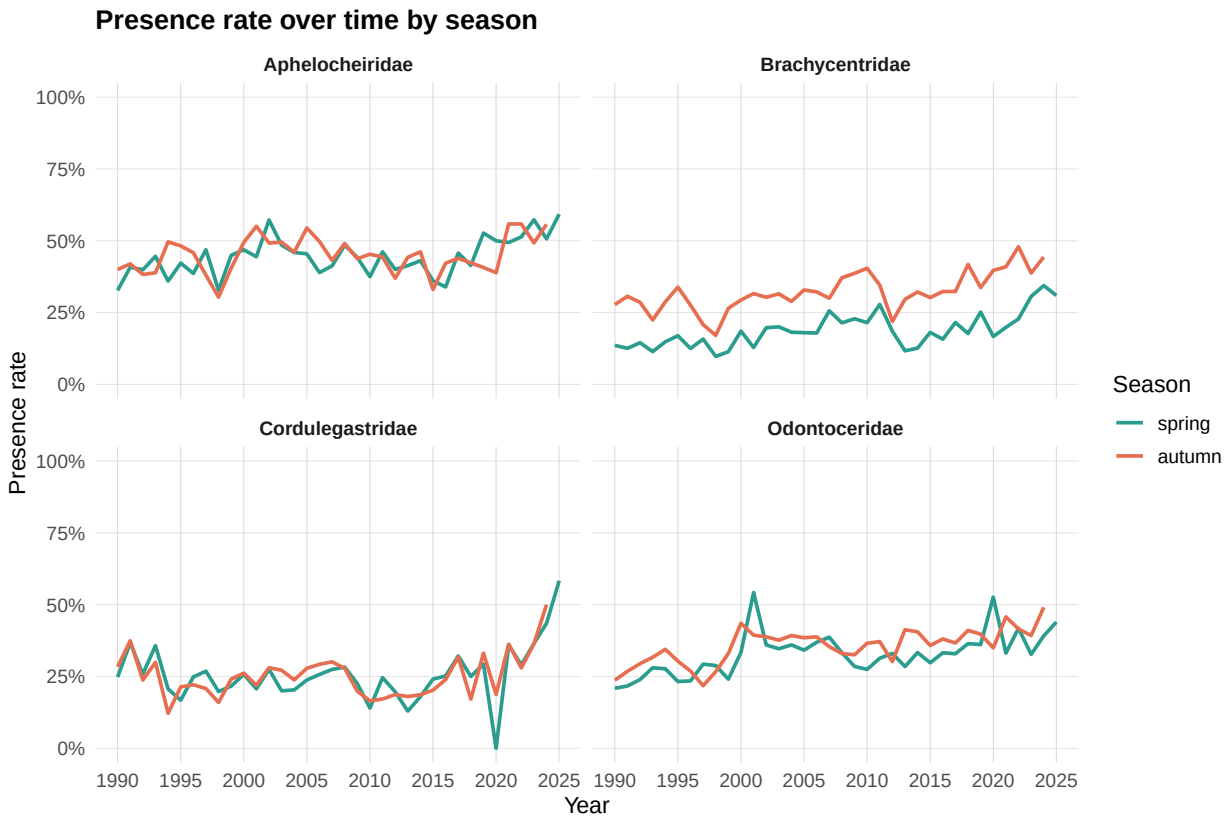

```
# ======================================================================
# 5) Same-day replicates (pre-dedup) by year
```

```r
#    • Re-derive a minimal metrics table from Arrow (as in analysis)
#    • Count days with >1 SAMPLE_ID per site/date before your de-dup
# ======================================================================
if (!exists("DATE_FLOOR")) DATE_FLOOR <- as.Date("1990-01-01")
if (!exists("USE_S3PO"))   USE_S3PO   <- TRUE

replicates_year <- {
  m_raw <- metrics_raw %>%
    dplyr::mutate(
      SITE_ID     = arrow::cast(SITE_ID,   arrow::int32()),
      SAMPLE_ID   = arrow::cast(SAMPLE_ID, arrow::int32()),
      SAMPLE_DATE = arrow::cast(SAMPLE_DATE, arrow::date32())
    ) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, SAMPLE_DATE, SAMPLE_METHOD) %>%
    dplyr::collect() %>%
    dplyr::mutate(SAMPLE_DATE = as.Date(SAMPLE_DATE)) %>%
    dplyr::filter(SAMPLE_DATE >= DATE_FLOOR) %>%
    { if (USE_S3PO) dplyr::filter(., SAMPLE_METHOD == "S3PO") else . }

  m_raw %>%
    group_by(SITE_ID, SAMPLE_DATE) %>%
    summarise(n_ids = n_distinct(SAMPLE_ID), .groups = "drop") %>%
    mutate(year = year(SAMPLE_DATE),
           is_replicate = n_ids > 1) %>%
    group_by(year) %>%
    summarise(replicate_days = sum(is_replicate),
              total_days = n(),
              prop = replicate_days / total_days,
              .groups = "drop")
}

p_reps <- ggplot(replicates_year, aes(year, replicate_days)) +
  geom_col(fill = "#9b2226") +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  scale_y_continuous(labels = short_num) +
  labs(
    title = "Same-day Replicates Before De-duplication",
    subtitle = "Count of site-days with >1 SAMPLE_ID (filters matched to your analysis set)",
    x = "Year", y = "Replicate site-days"
  ) + theme_ea()
print(p_reps)
```



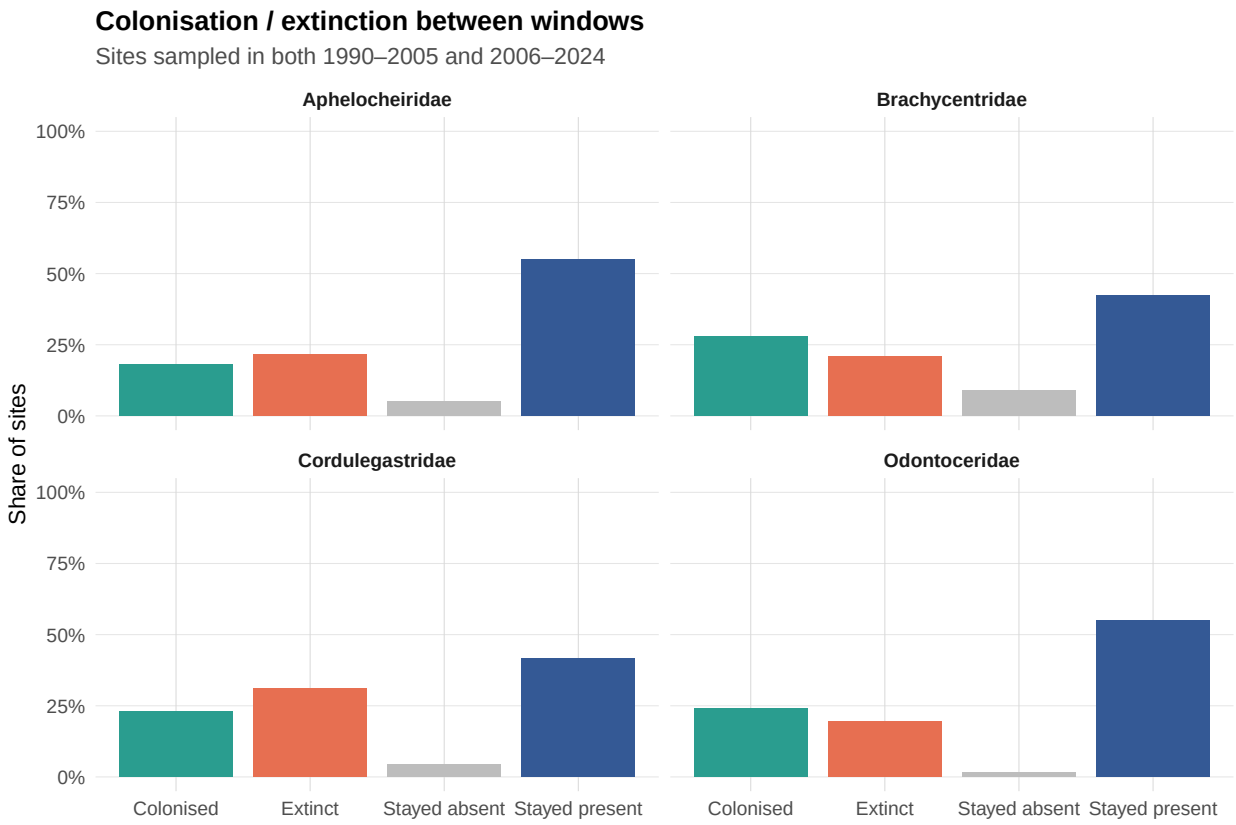

```
# ================================================================
# Three EDA plots (presence/absence focus)
#   1) Presence rate over time by season
#   2) Colonisation / extinction between windows
#   3) Zero proportion over time by season
# ------------------------------------------------
# Expects in memory:
#   - roster_flagged: deduped roster with SAMPLE_DATE, SITE_ID, SAMPLE_ID, season_f
#   - whpt_clean    : harmonised WHPT with EQ_TAXON_UNIT, SITE_ID, SAMPLE_ID, TOTAL_NUMBER
#   - families      : character vector of family names
# ================================================================

# ---- Helper: build a joined table for one family ----
# Adds explicit zeros, presence flag, year & season.
fam_join <- function(fam) {
  sites_with <- whpt_clean %>%
    dplyr::filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::distinct(SITE_ID) %>% dplyr::pull()

  roster <- roster_flagged %>%
    dplyr::filter(SITE_ID %in% sites_with) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, SAMPLE_DATE, season_f)

  counts <- whpt_clean %>%
    dplyr::filter(EQ_TAXON_UNIT == fam) %>%
    dplyr::select(SITE_ID, SAMPLE_ID, TOTAL_NUMBER)
```

```
  roster %>%
    dplyr::left_join(counts, by = c("SITE_ID","SAMPLE_ID")) %>%
    dplyr::mutate(
      TOTAL_NUMBER  = tidyr::replace_na(TOTAL_NUMBER, 0L),
      present       = TOTAL_NUMBER > 0L,
      year          = lubridate::year(SAMPLE_DATE),
      EQ_TAXON_UNIT = fam
    )
}

# ================================================================
# 1) Presence rate over time by season
#    - Site roster restricted to where the family has ever been observed
#    - Presence = share of samples with TOTAL_NUMBER > 0
# ================================================================
presence_season_df <- dplyr::bind_rows(lapply(families, fam_join)) %>%
  dplyr::group_by(EQ_TAXON_UNIT, season_f, year) %>%
  dplyr::summarise(p_present = mean(present), n = dplyr::n(), .groups = "drop")

p_presence_season <- ggplot(presence_season_df,
                            aes(year, p_present, colour = season_f)) +
  geom_line(linewidth = 0.9) +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  scale_y_continuous(labels = percent_format(accuracy = 1), limits = c(0, 1)) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  scale_colour_manual(values = c(spring = "#2A9D8F", autumn = "#E76F51"),
                      name = "Season") +
  labs(title = "Presence rate over time by season",
       x = "Year", y = "Presence rate") +
  theme_ea()

print(p_presence_season)
```

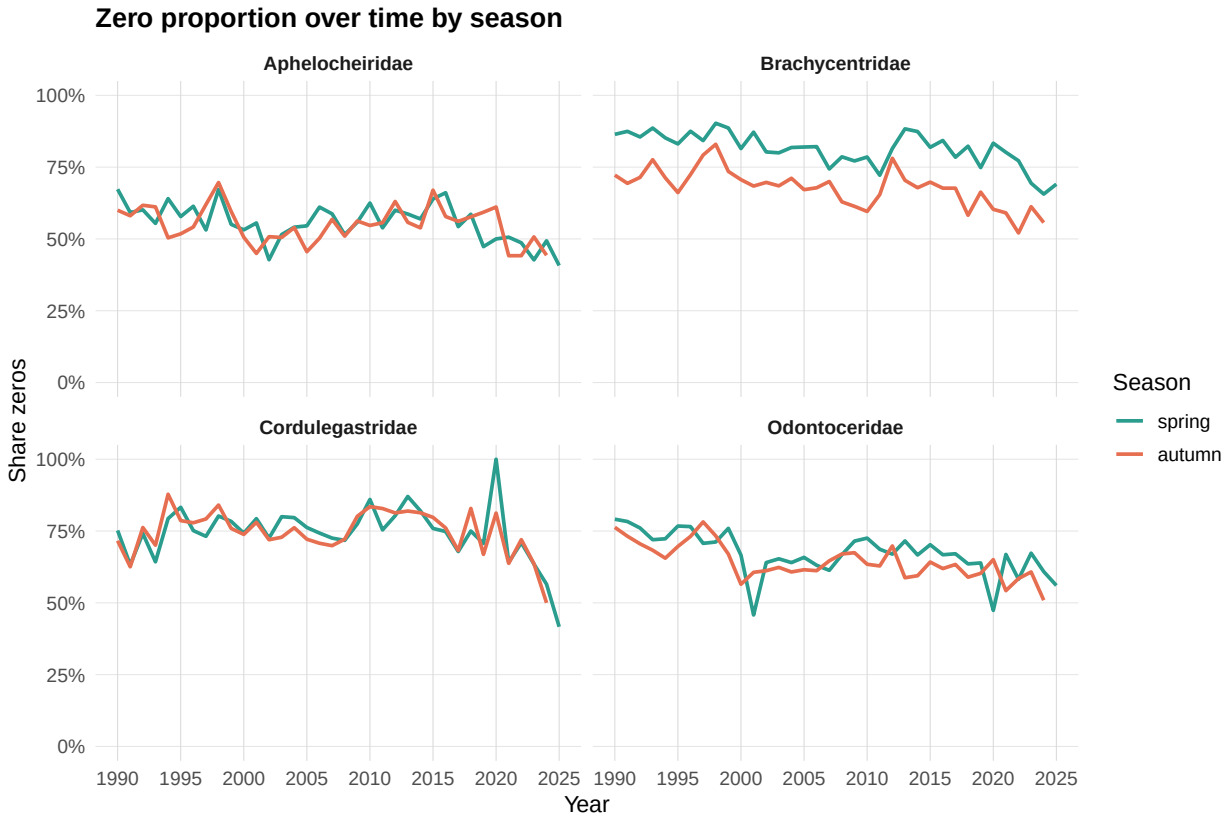



```
# ================================================================
# 2) Colonisation / extinction between windows
#    - Windows: Early = 1990-2005, Recent = 2006-2024 (inclusive)
#    - Consider sites that were sampled in BOTH windows
#    - Classification per site:
#         Colonised (0 -> 1), Extinct (1 -> 0),
#         Stayed absent (0 -> 0), Stayed present (1 -> 1)
# ================================================================
early_years  <- 1990:2005
recent_years <- 2006:2024

col_ext_df <- dplyr::bind_rows(lapply(families, fam_join)) %>%
  dplyr::mutate(period = dplyr::case_when(
    year %in% early_years  ~ "early",
    year %in% recent_years ~ "recent",
    TRUE ~ NA_character_
  )) %>%
  dplyr::filter(!is.na(period)) %>%
  dplyr::group_by(EQ_TAXON_UNIT, SITE_ID, period) %>%
  dplyr::summarise(p_any = as.integer(any(present)), .groups = "drop") %>%
  tidyr::pivot_wider(names_from = period, values_from = p_any) %>%
  # Keep sites sampled in BOTH windows
  dplyr::filter(!is.na(early), !is.na(recent)) %>%
  dplyr::mutate(class = dplyr::case_when(
    early == 0L & recent == 1L ~ "Colonised",
    early == 1L & recent == 0L ~ "Extinct",
    early == 0L & recent == 0L ~ "Stayed absent",
```

```
    early == 1L & recent == 1L ~ "Stayed present",
    TRUE ~ NA_character_
  )) %>%
  dplyr::filter(!is.na(class)) %>%
  dplyr::count(EQ_TAXON_UNIT, class, name = "n") %>%
  dplyr::group_by(EQ_TAXON_UNIT) %>%
  dplyr::mutate(prop = n / sum(n)) %>%
  dplyr::ungroup()

p_col_ext <- ggplot(col_ext_df, aes(class, prop, fill = class)) +
  geom_col(width = 0.85) +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  scale_y_continuous(labels = percent_format(accuracy = 1), limits = c(0, 1)) +
  scale_fill_manual(values = c("Colonised" = "#2A9D8F",
                               "Extinct" = "#E76F51",
                               "Stayed absent" = "#BDBDBD",
                               "Stayed present" = "#345995"),
                    guide = "none") +
  labs(title = "Colonisation / extinction between windows",
       subtitle = "Sites sampled in both 1990–2005 and 2006–2024",
       x = NULL, y = "Share of sites") +
  theme_ea()

print(p_col_ext)
```



```r
# ============================================================
# 3) Zero proportion over time by season
#    - Same denominator as (1): roster restricted to sites where family ever seen
#    - Metric = share of samples with TOTAL_NUMBER == 0
# ============================================================
zero_season_df <- dplyr::bind_rows(lapply(families, fam_join)) %>%
  dplyr::group_by(EQ_TAXON_UNIT, season_f, year) %>%
  dplyr::summarise(p_zero = mean(TOTAL_NUMBER == 0L), n = dplyr::n(), .groups = "drop")

p_zero_season <- ggplot(zero_season_df, aes(year, p_zero, colour = season_f)) +
  geom_line(linewidth = 0.9) +
  facet_wrap(~ EQ_TAXON_UNIT, ncol = 2) +
  scale_y_continuous(labels = percent_format(accuracy = 1), limits = c(0, 1)) +
  scale_x_continuous(breaks = seq(1990, year(Sys.Date()) + 1, 5)) +
  scale_colour_manual(values = c(spring = "#2A9D8F", autumn = "#E76F51"),
                      name = "Season") +
  labs(title = "Zero proportion over time by season",
       x = "Year", y = "Share zeros") +
  theme_ea()

print(p_zero_season)
```



#4. MODELLING PHASE

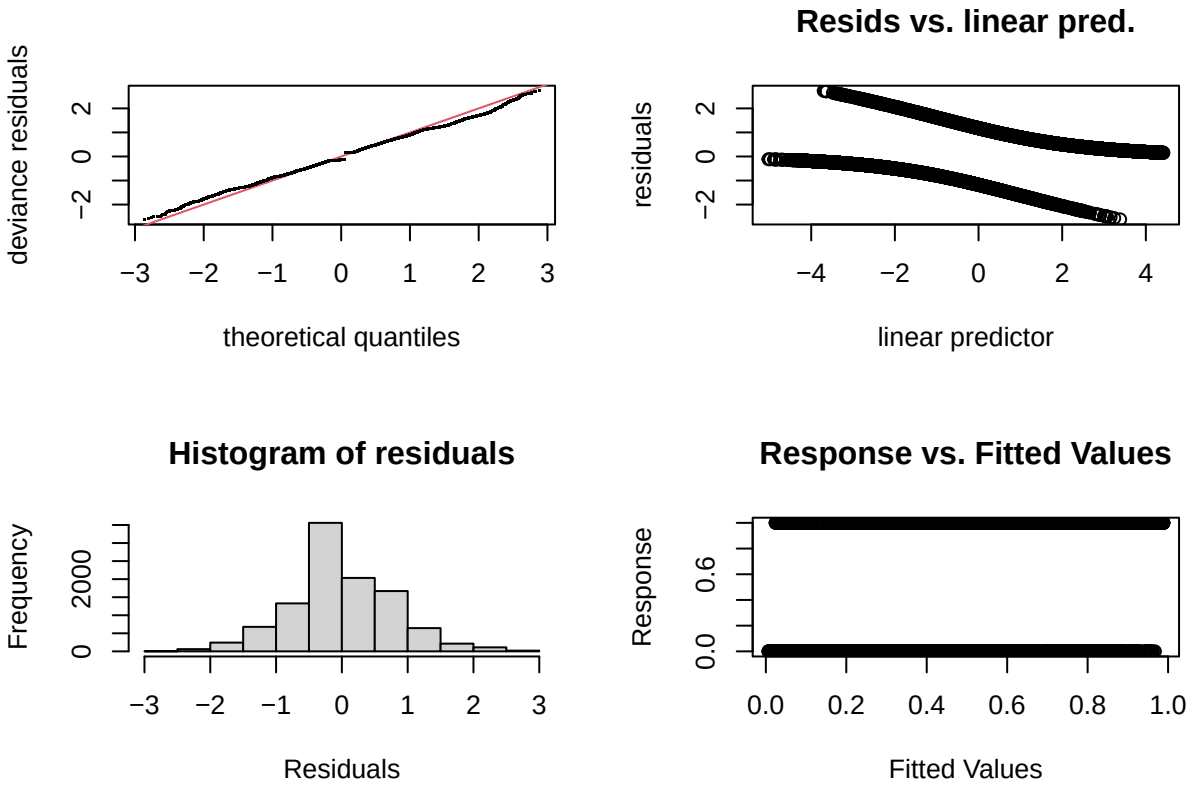
#4.1 PRESENCE/ABSENCE BINOMIAL GAMM MODEL

```r
# ================================================
# PHASE 4 - MODEL 1 : Presence/Absence GAMM (one family)

#   Goal: fit a binomial GAMM for presence/absence of a chosen family,
#         with season-specific temporal smooths and a site random intercept.
#   Inputs expected in memory: models_data (from Phase 3).
#   Output: pa_model object + quick diagnostics.
# ================================================

# -------- choose family --------
fam_name <- "Aphelocheiridae"          # <-- change to any of your families
pa_df    <- models_data[[fam_name]]$pa  # presence/absence table built in Phase 3

# -------- optional: restrict to S3PO (EA-recommended) --------
USE_S3PO <- TRUE
if (USE_S3PO && "SAMPLE_METHOD" %in% names(pa_df)) {
  pa_df <- dplyr::filter(pa_df, SAMPLE_METHOD == "S3PO")  # keep only S3PO samples if column exists
}

# -------- sensible k based on span of years --------
n_years <- dplyr::n_distinct(lubridate::year(pa_df$SAMPLE_DATE))  # number of distinct calendar years
k_time  <- min(20, max(8, round(0.6 * n_years)))                  # cap between 8 and 20, ~0.6*years

# -------- fit binomial GAMM --------
pa_model <- bam(
  taxon_present ~
    season_f +                                    # season-specific intercepts (spring vs autumn)
    s(decimal_date, by = season_f, k = k_time) + # two time smooths, one per season
    s(SITE_ID.F, bs = "re"),                      # random intercept for site (controls spatial heteroge
  family   = binomial(link = "logit"),           # presence/absence with logit link
  data     = pa_df,                              # modelling table
  method   = "fREML",                            # stable, fast REML for smoothing selection
  discrete = TRUE,                               # speed-up for large data
  select   = TRUE,    # shrink unnecessary wiggle (adds penalties to drop unneeded basis functions)
  gamma    = 1.2      # mild extra penalty to reduce overfitting
)

# -------- minimal diagnostics / outputs --------
print(summary(pa_model))       # EDFs per smooth, parametric terms, deviance explained, etc.
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## taxon_present ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.23214    0.09055   2.564  0.01036 *
## season_fautumn  0.19156    0.06148   3.116  0.00183 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df  Chi.sq p-value
## s(decimal_date):season_fspring   4.041     19  114.33  <2e-16 ***
## s(decimal_date):season_fautumn   3.520     19   73.43  <2e-16 ***
## s(SITE_ID.F)                   561.823    775 3955.86  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.609   Deviance explained = 55.7%
## fREML = 3588.7  Scale est. = 1         n = 10566
```

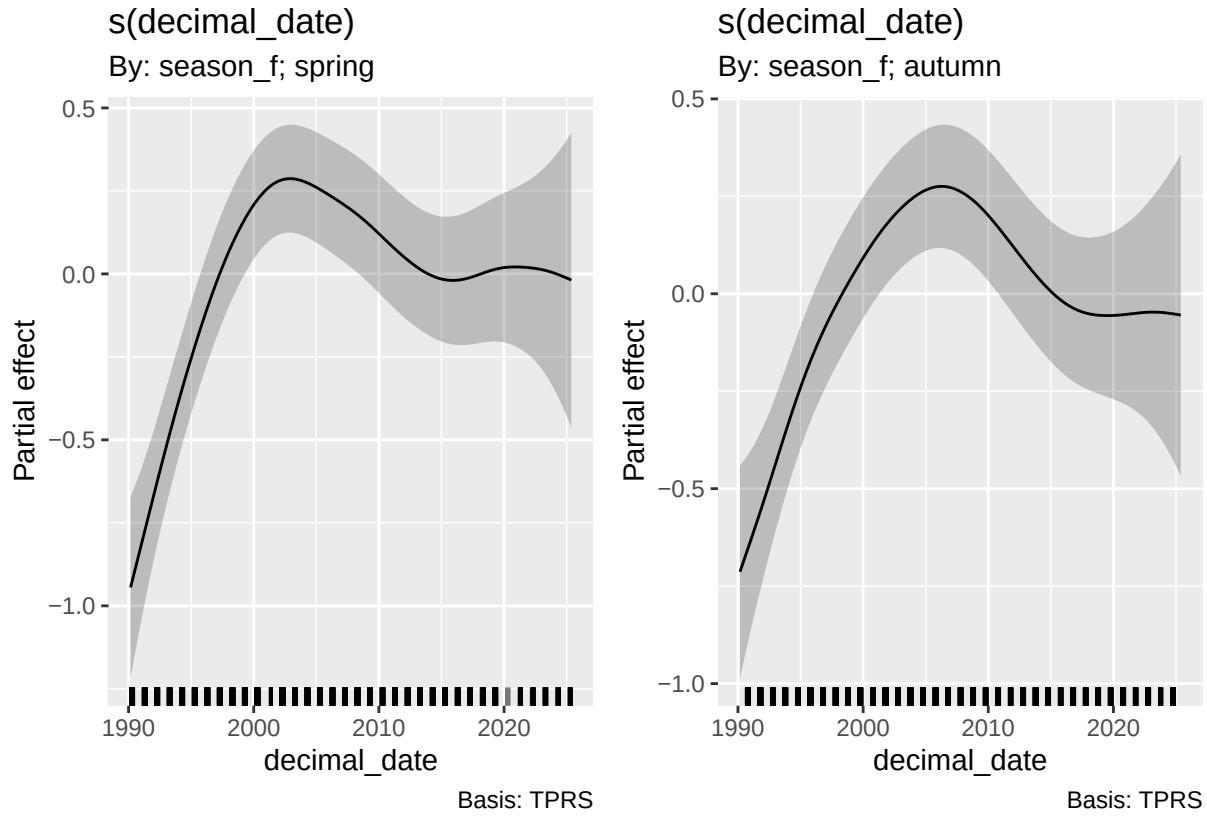
```
gam.check(pa_model)            # residual checks + k-index (are bases big enough?)
```



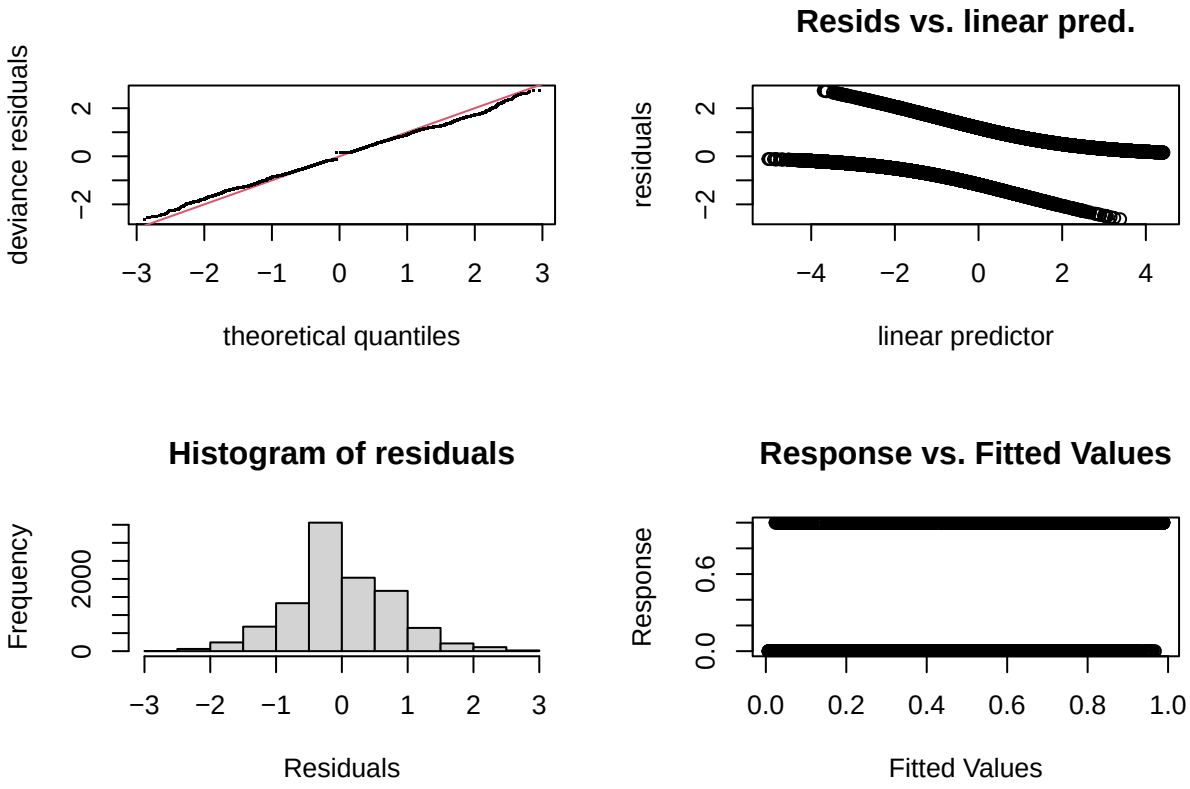

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 7.361786e-08 4.248104e-09 6.502546e-08 4.878808e-09 2.448449e-05
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.345468428 -0.061533302 -0.019899815 -0.006838829  -0.34439814
## [2,] -0.061533302  0.296948087 -0.007142537 -0.004152661  -0.05562482
## [3,] -0.019899815 -0.007142537  0.865482112 -0.136607852  -0.20293464
```

```
## [4,] -0.006838829 -0.004152661 -0.136607852  0.237233101  -0.03923195
## [5,] -0.344398142 -0.055624820 -0.202934642 -0.039231945 214.97517654
##
## Model rank =  816 / 816
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.04    0.97   0.005 **
## s(decimal_date):season_fautumn  19.00   3.52    0.97   0.005 **
## s(SITE_ID.F)                   776.00 561.82      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
draw(pa_model, select = 1:2)    # partial effect plots for the two season-specific time smooths
```




```r
# ================================================
# PA model quick Model Diagnostics
# Requires: pa_model (bam fit), pa_df (training data)
# Purpose: run lightweight checks + a small set of
#          quantitative summaries.
# ================================================

# 1) Basis/smooth adequacy
gam.check(pa_model, rep = 0)    # rep=0 avoids expensive re-smoothing; reports k-index and residual patt
```

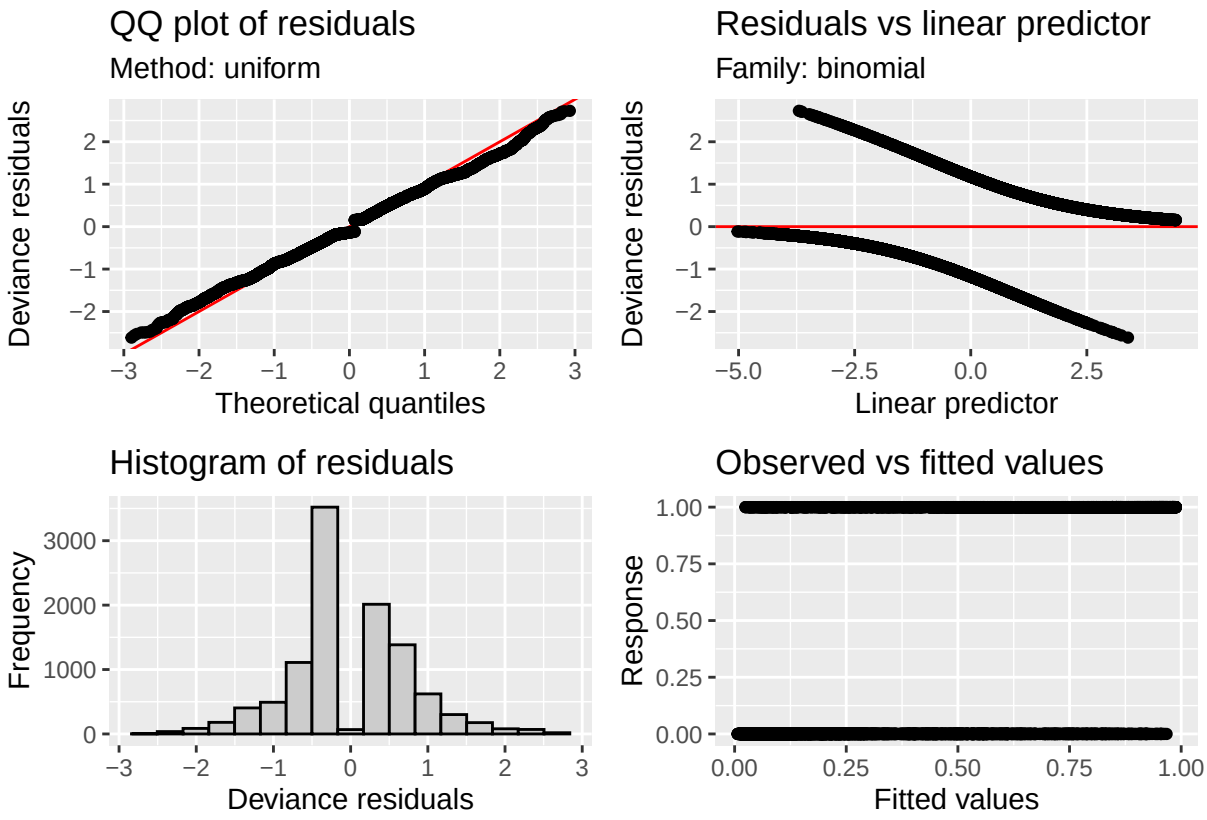



```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 7.361786e-08 4.248104e-09 6.502546e-08 4.878808e-09 2.448449e-05
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.345468428 -0.061533302 -0.019899815 -0.006838829  -0.34439814
## [2,] -0.061533302  0.296948087 -0.007142537 -0.004152661  -0.05562482
## [3,] -0.019899815 -0.007142537  0.865482112 -0.136607852  -0.20293464
## [4,] -0.006838829 -0.004152661 -0.136607852  0.237233101  -0.03923195
## [5,] -0.344398142 -0.055624820 -0.202934642 -0.039231945 214.97517654
##
## Model rank =  816 / 816
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.04    0.96  <2e-16 ***
## s(decimal_date):season_fautumn  19.00   3.52    0.96  <2e-16 ***
## s(SITE_ID.F)                   776.00 561.82      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

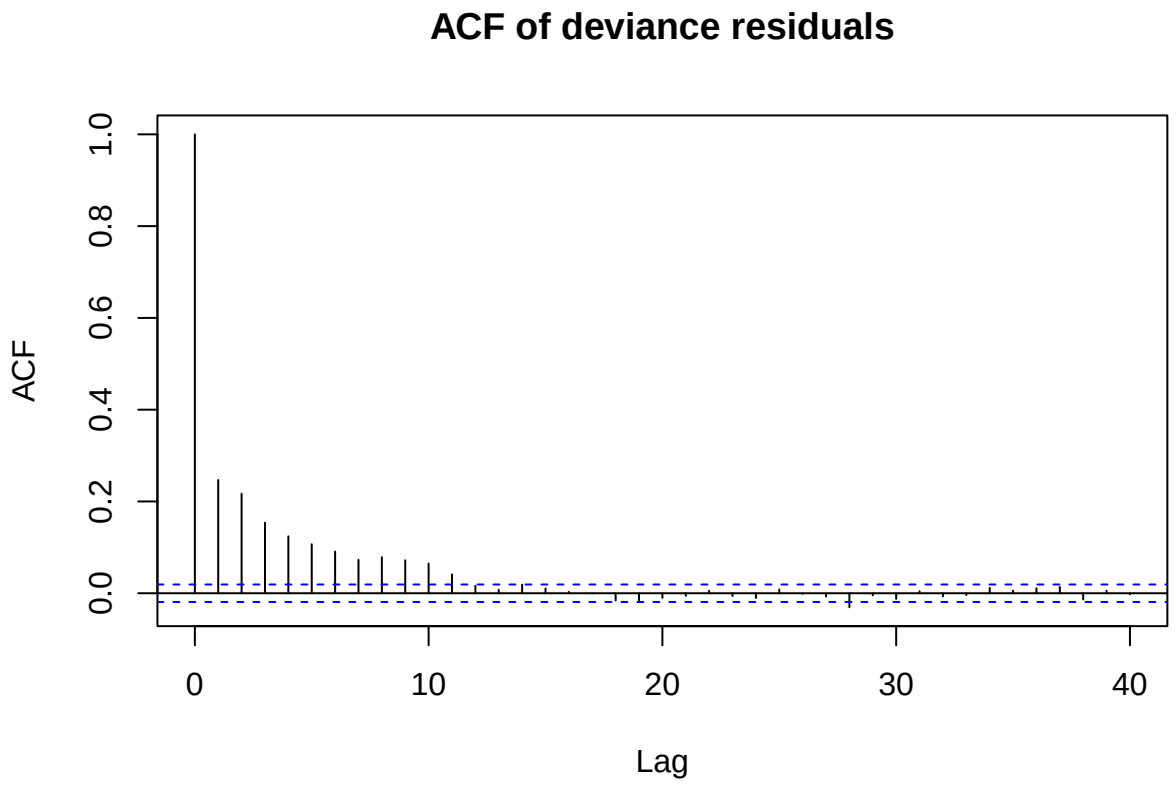
```r
# 2) Concurvity - robust to return shape
con_obj <- try(mgcv::concurvity(pa_model, full = TRUE), silent = TRUE)  # can return list or matrix dep
if (!inherits(con_obj, "try-error")) {
  if (is.list(con_obj) && !is.null(con_obj$estimate)) {
    cat("\nConcurvity (estimate, rounded):\n")
    print(round(con_obj$estimate, 3))   # typical output: para / smooth blocks
  } else {
    cat("\nConcurvity (raw object):\n")
    print(con_obj)                      # fallback for alternate structures
  }
} else {
  message("concurvity() failed or not available; skipping.")  # don't fail the whole diagnostic pass
}
```

```
##
## Concurvity (raw object):
##          para s(decimal_date):season_fspring s(decimal_date):season_fautumn
## worst       1                       0.3484258                       0.3044989
## observed    1                       0.1468689                       0.1642139
## estimate    1                       0.2561181                       0.2569928
##          s(SITE_ID.F)
## worst     1.000000000
## observed  0.095318124
## estimate  0.005511913
```

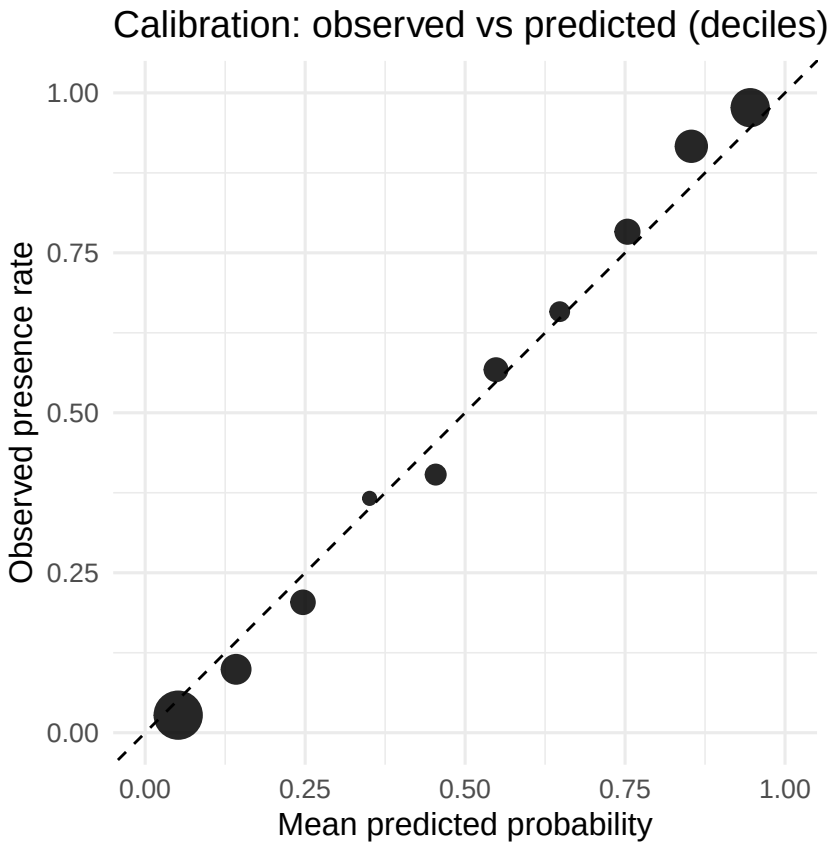
```r
# 3) Compact residual diagnostics
appraise(pa_model)  # produces QQ, residual vs fitted, histogram, scale-location (silent if device is o
```




```
# 4) Residual autocorrelation (global)
res_dev <- residuals(pa_model, type = "deviance")  # deviance residuals on the response scale
acf(res_dev[is.finite(res_dev)], na.action = na.pass,
    main = "ACF of deviance residuals")           # check for temporal dependence left over
```



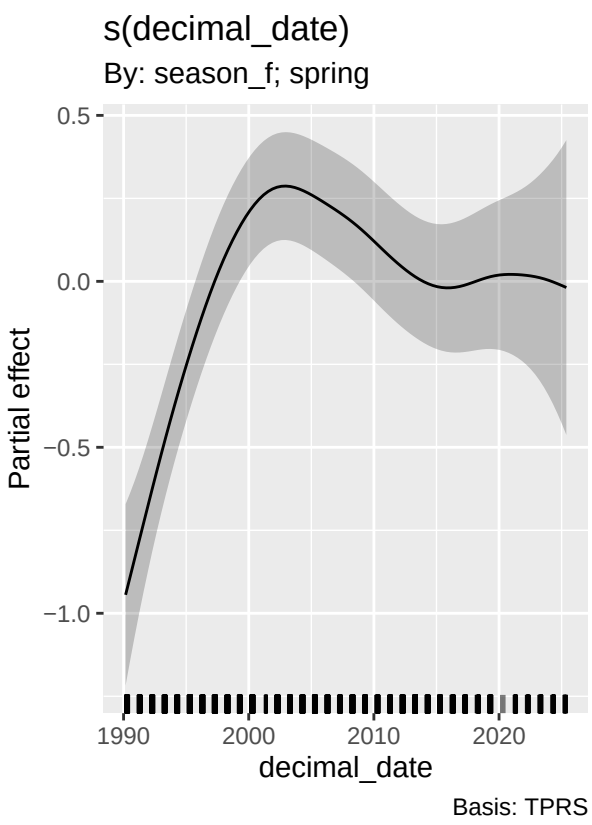

```
# 5) Calibration (observed vs predicted by decile)
cal <- tibble(
  phat = fitted(pa_model, type = "response"),      # predicted probabilities
  y    = as.numeric(pa_df$taxon_present)           # 0/1 outcome as numeric
) |>
  mutate(bin = cut(phat, breaks = seq(0, 1, by = 0.1), include.lowest = TRUE)) |>  # decile bins
  group_by(bin) |>
  summarise(p_hat = mean(phat), p_obs = mean(y), n = n(), .groups = "drop")        # mean predicted vs

ggplot(cal, aes(p_hat, p_obs, size = n)) +
  geom_point(alpha = 0.85) +                                                       # larger points
  geom_abline(slope = 1, intercept = 0, linetype = 2) +                            # perfect-calibr
  coord_equal(xlim = c(0, 1), ylim = c(0, 1)) +
  scale_size_continuous(range = c(2, 8), guide = "none") +
  labs(title = "Calibration: observed vs predicted (deciles)",
       x = "Mean predicted probability", y = "Observed presence rate") +
  theme_minimal(base_size = 12)
```

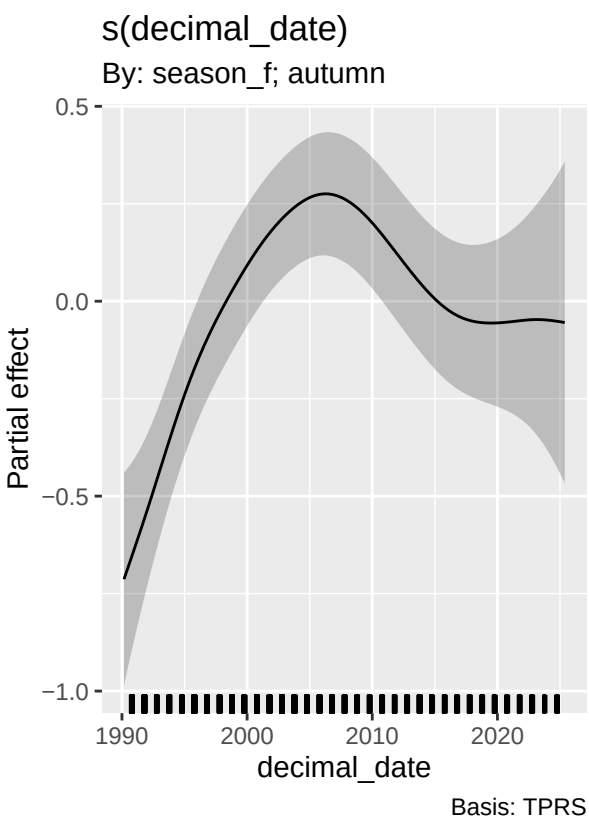



```
# 6) Quick metrics
phat  <- fitted(pa_model, type = "response")   # vector of predicted probabilities
y     <- as.numeric(pa_df$taxon_present)       # 0/1 truth
brier <- mean((y - phat)^2)                    # Brier score (lower is better)
devex <- summary(pa_model)$dev.expl            # deviance explained (fraction)
cat(sprintf("\nBrier score: %.4f | Deviance explained: %.1f%%\n",
            brier, 100 * devex))
```

```
##
## Brier score: 0.0914 | Deviance explained: 55.7%
```

```
# 7) Season-specific time smooths (spring/autumn)
draw(pa_model, select = 1:2)   # partial effects for s(decimal_date):season_fspring & :autumn
```






```r
# ================================================
# PHASE 4 - Batch PA GAMMs for 4 families
# Goal: fit the same presence/absence GAMM to each family,
#       print compact text diagnostics, and return a result list.
#       (No figures produced; gam.check plots are suppressed.)
# ================================================

# Families to run
families <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")

# Option: EA-recommended S3PO-only
USE_S3PO <- TRUE    # if TRUE, restricts to kick samples with consistent method coding

# ---- Helper: fit one family ----
fit_pa_family <- function(fam_name, md = models_data, use_s3po = TRUE) {
  # Safety: check that the models_data list has this family and a 'pa' table
  stopifnot(!is.null(md[[fam_name]]), "pa" %in% names(md[[fam_name]]))
  df <- md[[fam_name]]$pa   # training data for this family's PA model

  # Filter to S3PO if requested (only if the column exists)
  if (use_s3po && "SAMPLE_METHOD" %in% names(df)) {
    df <- dplyr::filter(df, SAMPLE_METHOD == "S3PO")
  }

  # Build 2-level season if missing (spring/autumn only)
  # Uses month(SAMPLE_DATE) to label spring = Mar-May; autumn = Sep-Nov
```

```r
if (!("season_f" %in% names(df))) {
  df <- df %>%
    mutate(m = month(SAMPLE_DATE),
           season_f = factor(if_else(m %in% 3:5, "spring", "autumn"),
                             levels = c("spring","autumn")))
}

# Sensible k from span of years (avoid over/under-smoothing)
k_time <- {
  ny <- dplyr::n_distinct(lubridate::year(df$SAMPLE_DATE))
  min(20, max(8, round(0.6 * ny)))
}

# Fit model:
# - season-specific intercepts (season_f)
# - season-varying time smooth s(decimal_date, by = season_f)
# - site random intercept s(SITE_ID.F, bs="re")
# - binomial logit PA, fREML, discrete=TRUE for speed, select+gamma to regularise
m <- bam(
  taxon_present ~
    season_f +
    s(decimal_date, by = season_f, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = binomial(link = "logit"),
  data     = df,
  method   = "fREML",
  discrete = TRUE,
  select   = TRUE,
  gamma    = 1.2
)

# Compact text summary to console: parametric terms, smooth EDFs, deviance explained
cat("\n============================\n")
cat(sprintf("Family: %s\n", fam_name))
cat("============================\n")
print(summary(m))

# gam.check without plotting to screen:
# send plots to a temp PDF device, close, then delete file
tmp <- tempfile(fileext = ".pdf")
pdf(tmp)
gc_out <- try(gam.check(m, rep = 0), silent = TRUE)  # rep=0 => no refits; k-index etc.
dev.off()
unlink(tmp)

# Quick scalar metrics on training data
phat  <- fitted(m, type = "response")   # predicted presence prob
y     <- as.numeric(df$taxon_present)   # 0/1 vector
brier <- mean((y - phat)^2)             # Brier score (lower is better)
devex <- summary(m)$dev.expl            # fraction deviance explained
aic   <- AIC(m)                         # AIC for rough comparability
nobs_ <- nobs(m)                        # effective N used
```

```r
  cat(sprintf("Brier score: %.4f | Deviance explained: %.1f%% | AIC: %.1f | N: %d\n",
              brier, 100 * devex, aic, nobs_))

  # Return a compact result object (model + meta for later use)
  list(
    family   = fam_name,
    model    = m,
    data     = df,
    k_time   = k_time,
    brier    = brier,
    dev_expl = devex,
    AIC      = aic,
    n        = nobs_
  )
}

# ---- Run for all families ----
# Produces a named list of result objects, one per family
pa_results <- lapply(families, fit_pa_family, md = models_data, use_s3po = USE_S3PO)
```

```
##
## ============================
## Family: Aphelocheiridae
## ============================
##
## Family: binomial
## Link function: logit
##
## Formula:
## taxon_present ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.23214    0.09055   2.564  0.01036 *
## season_fautumn  0.19156    0.06148   3.116  0.00183 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df  Chi.sq p-value
## s(decimal_date):season_fspring   4.041     19  114.33  <2e-16 ***
## s(decimal_date):season_fautumn   3.520     19   73.43  <2e-16 ***
## s(SITE_ID.F)                   561.823    775 3955.86  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.609   Deviance explained = 55.7%
## fREML = 3588.7  Scale est. = 1         n = 10566

##
## Method: fREML   Optimizer: perf chol
## $grad
```

```
## [1] 7.361786e-08 4.248104e-09 6.502546e-08 4.878808e-09 2.448449e-05
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.345468428 -0.061533302 -0.019899815 -0.006838829  -0.34439814
## [2,] -0.061533302  0.296948087 -0.007142537 -0.004152661  -0.05562482
## [3,] -0.019899815 -0.007142537  0.865482112 -0.136607852  -0.20293464
## [4,] -0.006838829 -0.004152661 -0.136607852  0.237233101  -0.03923195
## [5,] -0.344398142 -0.055624820 -0.202934642 -0.039231945 214.97517654
##
## Model rank =  816 / 816
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.04    0.94  <2e-16 ***
## s(decimal_date):season_fautumn  19.00   3.52    0.94  <2e-16 ***
## s(SITE_ID.F)                   776.00 561.82      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Brier score: 0.0914 | Deviance explained: 55.7% | AIC: 7572.7 | N: 10566
##
## ============================
## Family: Brachycentridae
## ============================
##
## Family: binomial
## Link function: logit
##
## Formula:
## taxon_present ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.56927    0.03856  -40.70   <2e-16 ***
## season_fautumn  1.00735    0.03018   33.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df Chi.sq p-value
## s(decimal_date):season_fspring   13.20     19  563.0  <2e-16 ***
## s(decimal_date):season_fautumn   14.15     19  655.3  <2e-16 ***
## s(SITE_ID.F)                   1604.26   2370 9029.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.364   Deviance explained = 35.2%
## fREML =  14329  Scale est. = 1         n = 39394


##
## Method: fREML   Optimizer: perf chol
```

```
## $grad
## [1]  1.999594e-07 -2.658969e-07  2.084543e-07 -1.858680e-06  6.418051e-05
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  2.589143e+00  1.451064e-07  1.991833e-03 -1.323424e-08 -6.293444e-01
## [2,]  1.451064e-07  2.658967e-07  1.754937e-09 -6.859048e-15 -7.628958e-08
## [3,]  1.991833e-03  1.754937e-09  3.151603e+00 -1.498228e-06 -2.949710e-01
## [4,] -1.323424e-08 -6.859048e-15 -1.498228e-06  1.858673e-06  1.427121e-07
## [5,] -6.293444e-01 -7.628958e-08 -2.949710e-01  1.427121e-07  5.591293e+02
##
## Model rank =  2411 / 2411
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring   19.0   13.2    0.94   0.010 **
## s(decimal_date):season_fautumn   19.0   14.1    0.94   0.005 **
## s(SITE_ID.F)                   2371.0 1604.3      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Brier score: 0.1130 | Deviance explained: 35.2% | AIC: 31741.9 | N: 39394
##
## ============================
## Family: Odontoceridae
## ============================
##
## Family: binomial
## Link function: logit
##
## Formula:
## taxon_present ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.50108    0.03213 -15.594   <2e-16 ***
## season_fautumn  0.27197    0.02777   9.792   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df Chi.sq p-value
## s(decimal_date):season_fspring    6.721     19  439.2  <2e-16 ***
## s(decimal_date):season_fautumn    6.145     19  441.8  <2e-16 ***
## s(SITE_ID.F)                   1745.518   2874 8020.3  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.328   Deviance explained = 30.9%
## fREML =  14608  Scale est. = 1         n = 33093


##
```

```
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -1.453750e-06 -3.022771e-08 -1.335667e-06 -5.601100e-08 -5.947037e-04
##
## $hess
##              [,1]          [,2]          [,3]          [,4]         [,5]
## [1,]  2.406091471  0.0305327900 -0.0111240618  0.0012838387  -0.75683856
## [2,]  0.030532790  0.0054346094  0.0003697543 -0.0006505681   0.01161503
## [3,] -0.011124062  0.0003697543  1.2064497195 -0.0521702303  -0.31892135
## [4,]  0.001283839 -0.0006505681 -0.0521702303  0.2880521863  -0.03269052
## [5,] -0.756838564  0.0116150257 -0.3189213528 -0.0326905177 566.93408836
##
## Model rank =  2915 / 2915
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'     edf k-index p-value
## s(decimal_date):season_fspring   19.00    6.72    0.93  <2e-16 ***
## s(decimal_date):season_fautumn   19.00    6.14    0.93  <2e-16 ***
## s(SITE_ID.F)                   2875.00 1745.52      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Brier score: 0.1410 | Deviance explained: 30.9% | AIC: 32588.2 | N: 33093
##
## ============================
## Family: Cordulegastridae
## ============================
##
## Family: binomial
## Link function: logit
##
## Formula:
## taxon_present ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.749668   0.047113 -15.912   <2e-16 ***
## season_fautumn  0.001418   0.045249   0.031    0.975
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df Chi.sq p-value
## s(decimal_date):season_fspring  10.593     19  153.8  <2e-16 ***
## s(decimal_date):season_fautumn   6.974     19  152.1  <2e-16 ***
## s(SITE_ID.F)                   863.488   1506 3403.8  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.316   Deviance explained = 30.6%
## fREML = 6287.1  Scale est. = 1         n = 15451
```

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -4.388990e-06 -2.214713e-07 -3.396904e-06 -4.633842e-05 -6.296068e-04
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.776167e+00 -2.166160e-01 -3.774598e-02 -8.281351e-07 -1.963713e-01
## [2,] -2.166160e-01  2.323148e-01 -4.242054e-04 -5.942001e-07  7.282518e-02
## [3,] -3.774598e-02 -4.242054e-04  2.202389e+00 -5.126238e-05 -3.900519e-01
## [4,] -8.281351e-07 -5.942001e-07 -5.126238e-05  4.633505e-05  1.332617e-05
## [5,] -1.963713e-01  7.282518e-02 -3.900519e-01  1.332617e-05  2.714421e+02
##
## Model rank =  1547 / 1547
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'     edf k-index p-value
## s(decimal_date):season_fspring   19.00   10.59    0.95   0.020 *
## s(decimal_date):season_fautumn   19.00    6.97    0.95   0.035 *
## s(SITE_ID.F)                   1507.00  863.49      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Brier score: 0.1224 | Deviance explained: 30.6% | AIC: 13953.2 | N: 15451
```

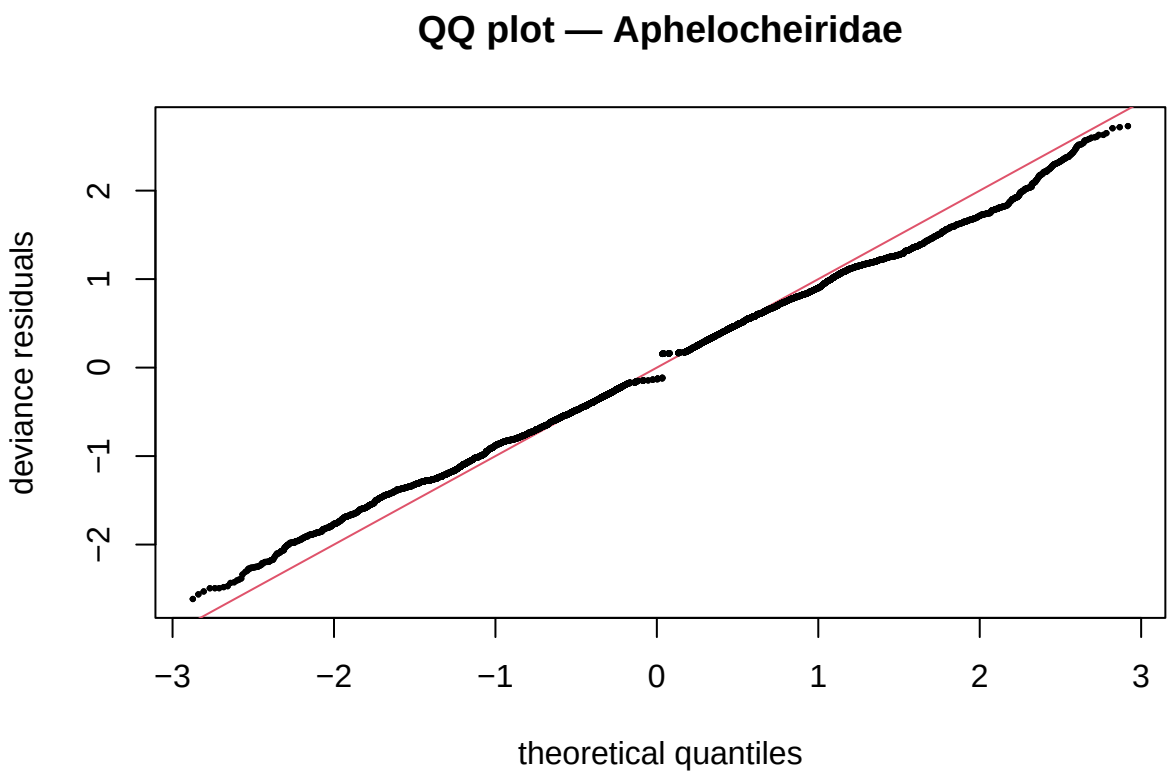
```r
names(pa_results) <- families

# ---- Optional: collect a summary table ----
# Binds core metrics across families for quick comparison
pa_summary <- do.call(rbind, lapply(pa_results, function(x) {
  data.frame(
    family   = x$family,
    n        = x$n,
    k_time   = x$k_time,
    brier    = x$brier,
    dev_expl = x$dev_expl,
    AIC      = x$AIC,
    row.names = NULL
  )
}))
print(pa_summary)
```

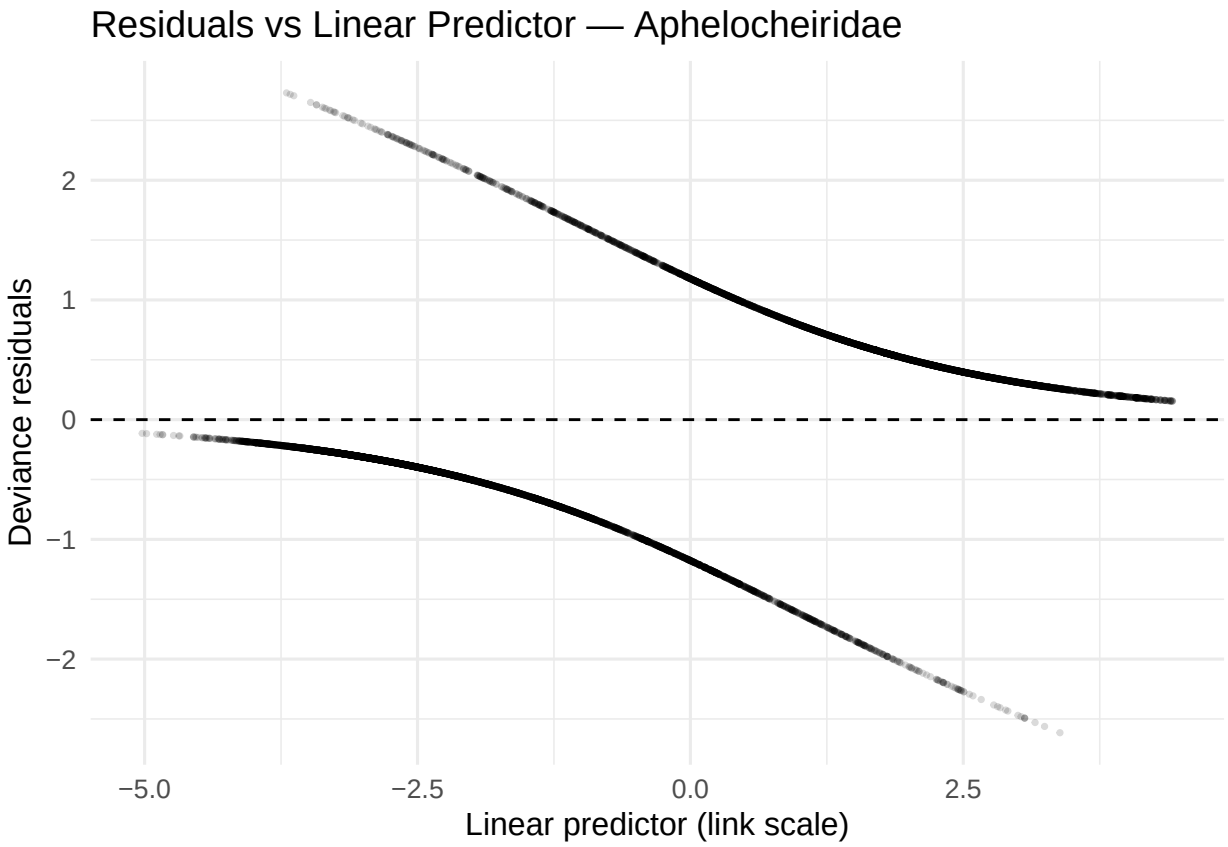
```
##                            family     n k_time      brier  dev_expl       AIC
## Aphelocheiridae   Aphelocheiridae 10566     20 0.09135607 0.5572837  7572.696
## Brachycentridae   Brachycentridae 39394     20 0.11297931 0.3516936 31741.926
## Odontoceridae       Odontoceridae 33093     20 0.14097378 0.3088072 32588.177
## Cordulegastridae Cordulegastridae 15451     20 0.12242216 0.3055677 13953.177
```

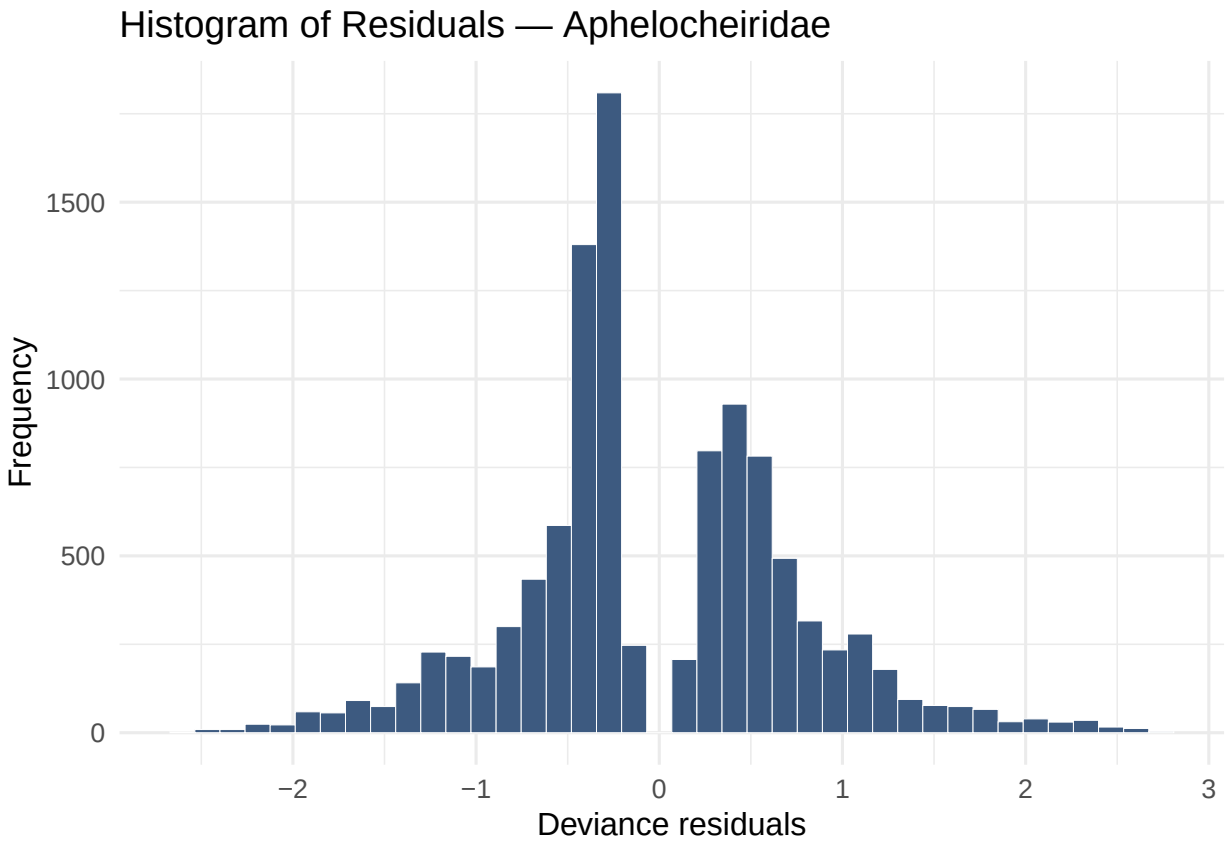
```r
# ---- Optional: save models ----
# invisible(lapply(pa_results, function(x) saveRDS(x$model, paste0("pa_model_", x$family, ".rds"))))
```

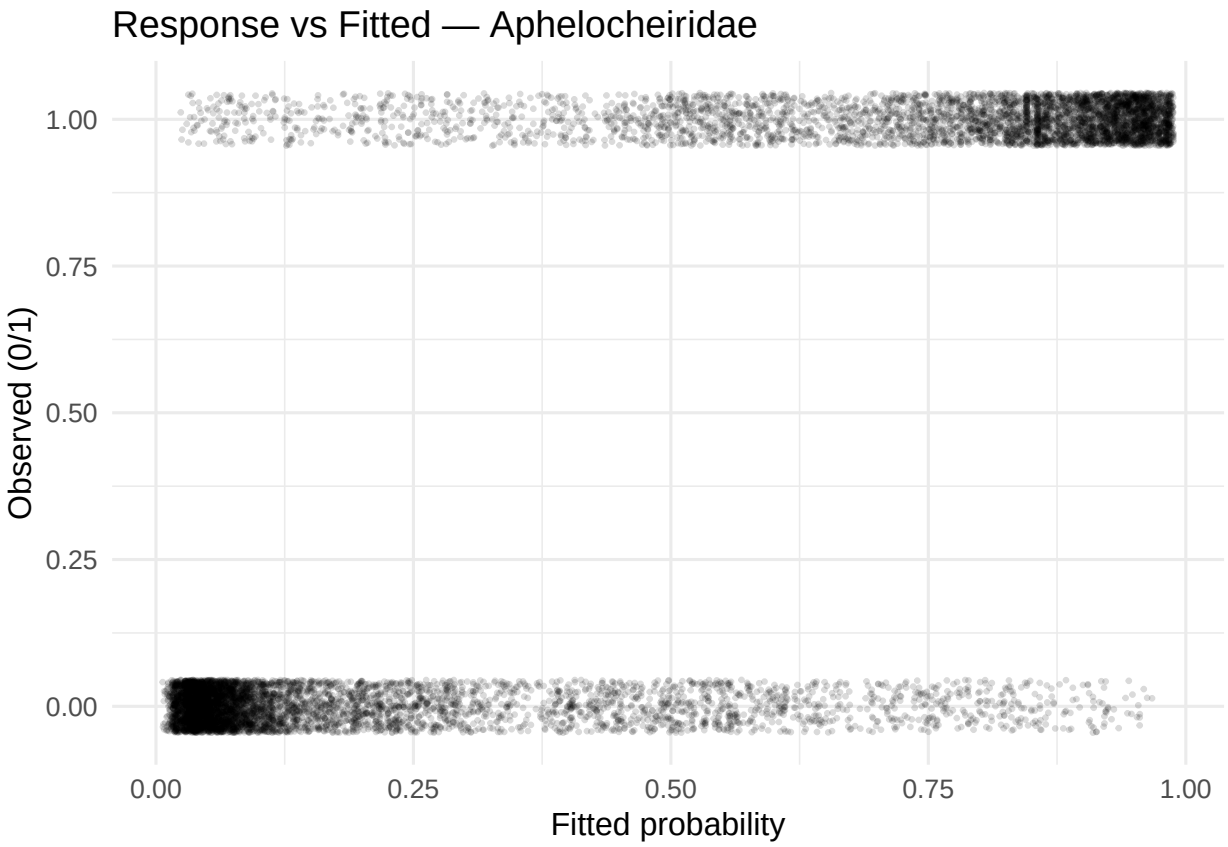
```r
# PA model quick Model Diagnostics for all families
# (Produces: QQ plot, residuals vs linear predictor, residual histogram,
# response vs fitted jitter, and season-specific time smooths.)


plot_pa_diagnostics_clean <- function(res) {
  # `res` is one element of pa_results: list(model, family, data, ...)
  m   <- res$model          # mgcv::bam fitted model
  fam <- res$family         # family name string (for titles)
  df  <- res$data           # training data used to fit

  # 1) QQ plot of residuals.
  #    rep = 0 avoids costly reference simulation; pch/cex make points subtle.
  par(mfrow = c(1,1))
  mgcv::qq.gam(m, rep = 0, pch = 19, cex = 0.3, main = paste("QQ plot -", fam))

  # Prepare a tidy frame with the key diagnostics quantities:
  eta   <- predict(m, type = "link")         # linear predictor
  resid <- residuals(m, type = "deviance")   # deviance residuals
  phat  <- fitted(m, type = "response")      # fitted probabilities on response scale
  y     <- as.numeric(df$taxon_present)      # observed 0/1
  dd    <- data.frame(eta = eta, resid = resid, phat = phat, y = y)

  # 2) Residuals vs linear predictor.
  p_res_lin <- ggplot(dd, aes(eta, resid)) +
    geom_point(alpha = 0.15, size = 0.6) +
    geom_hline(yintercept = 0, linetype = 2) +
    labs(title = paste("Residuals vs Linear Predictor -", fam),
         x = "Linear predictor (link scale)", y = "Deviance residuals") +
    theme_minimal(base_size = 12)
  print(p_res_lin)

  # 3) Histogram of residuals (distribution shape check).
  p_hist <- ggplot(dd, aes(resid)) +
    geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
    labs(title = paste("Histogram of Residuals -", fam),
         x = "Deviance residuals", y = "Frequency") +
    theme_minimal(base_size = 12)
  print(p_hist)

  # 4) Response vs Fitted: jittered points only.
  p_resp_fit <- ggplot(dd, aes(phat, y)) +
    geom_jitter(height = 0.045, width = 0, alpha = 0.15, size = 0.5) +
    labs(title = paste("Response vs Fitted -", fam),
         x = "Fitted probability", y = "Observed (0/1)") +
    theme_minimal(base_size = 12)
  print(p_resp_fit)

  # 5) Season-specific time smooths:
  #    locate smooth terms containing "decimal_date" and draw each.
  sm_ids   <- which(grepl("decimal_date", smooths(m)))
  sm_names <- smooths(m)[sm_ids]
  for (j in seq_along(sm_ids)) {
```

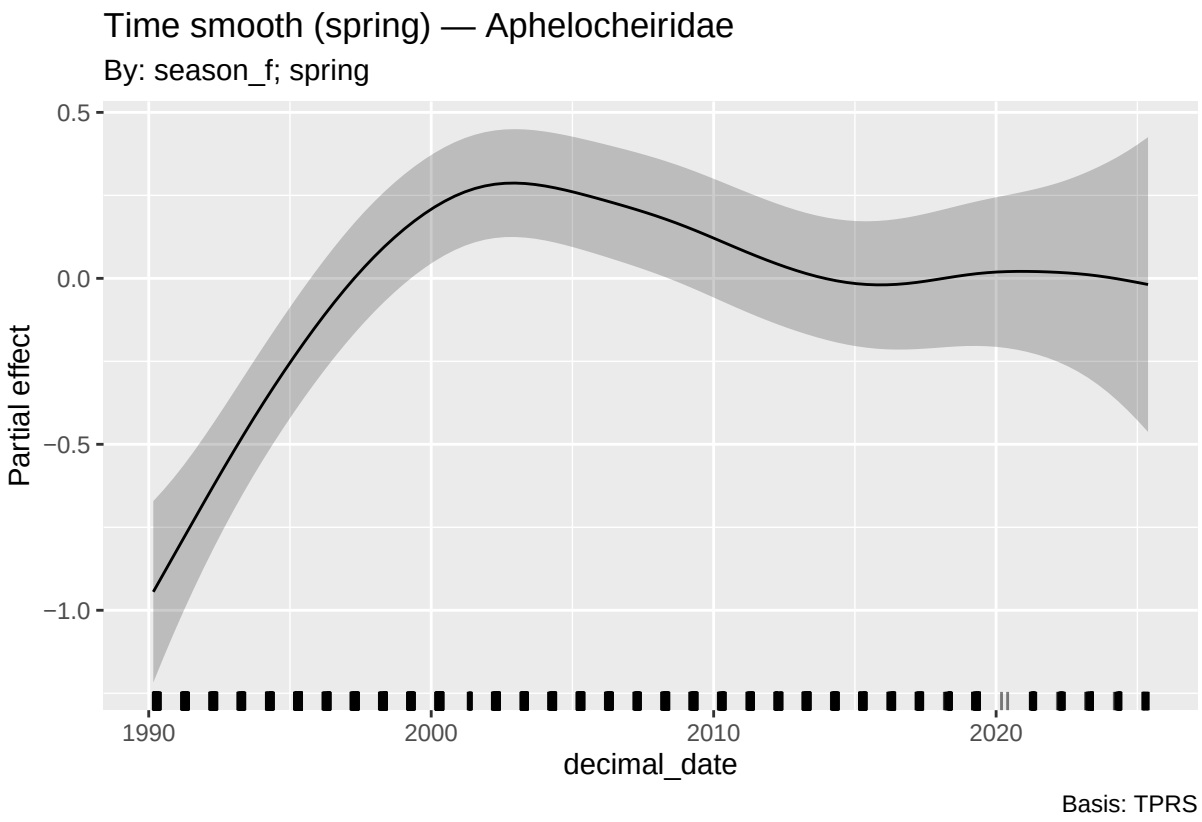
```r
    nm  <- sm_names[j]
    lab <- if (grepl("spring", nm, ignore.case = TRUE)) "spring"
           else if (grepl("autumn", nm, ignore.case = TRUE)) "autumn" else nm
    p_sm <- draw(m, select = sm_ids[j]) + ggtitle(paste("Time smooth (", lab, ") - ", fam, sep = ""))
    print(p_sm)
  }
}

# Run the diagnostic panels for every fitted PA model in `pa_results`.
# `invisible()` avoids printing the returned list of NULLs to the console.
invisible(lapply(pa_results, plot_pa_diagnostics_clean))
```

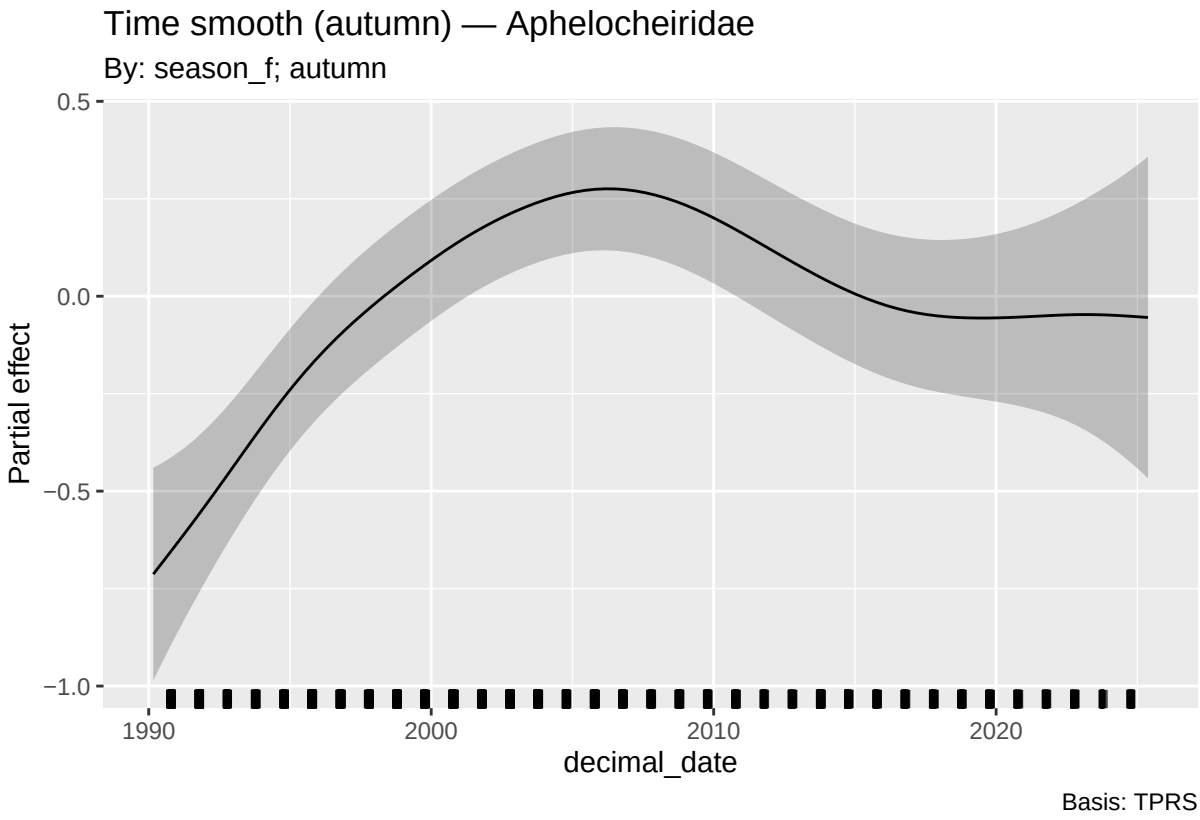


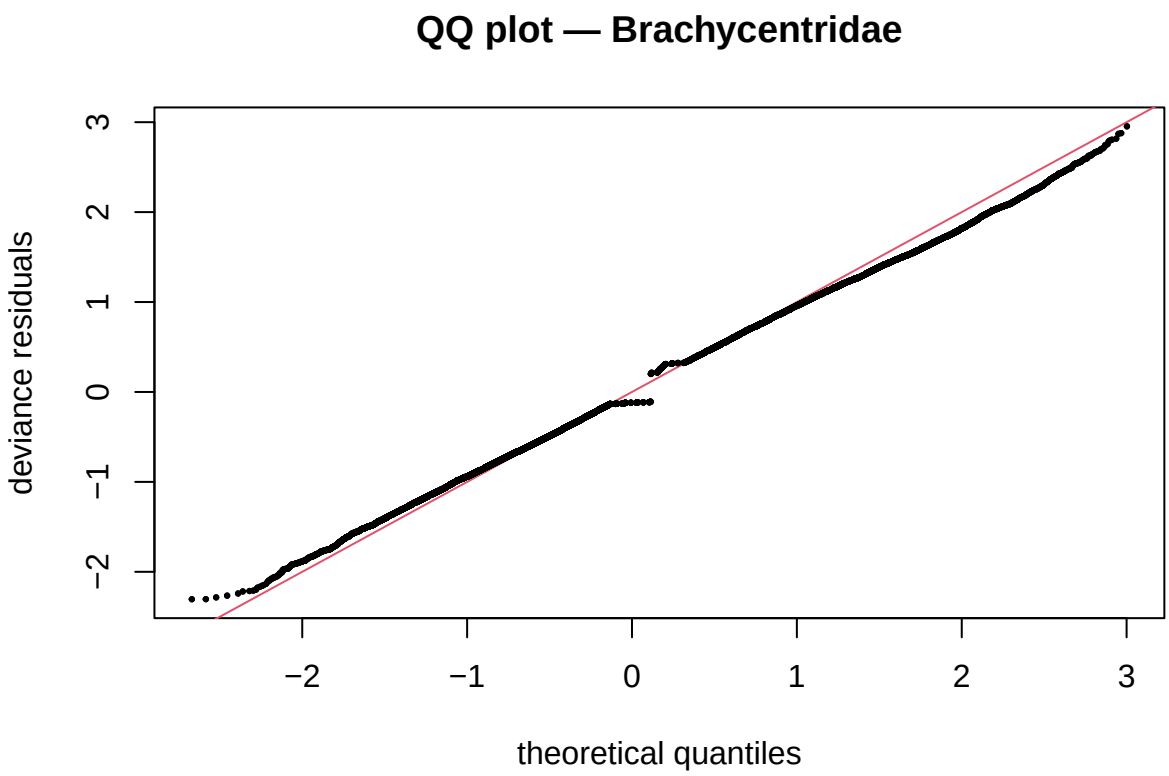

Residuals vs Linear Predictor — Aphelocheiridae
Deviance residuals
2
1
0
−1
−2
−5.0
−2.5
0.0
2.5
Linear predictor (link scale)

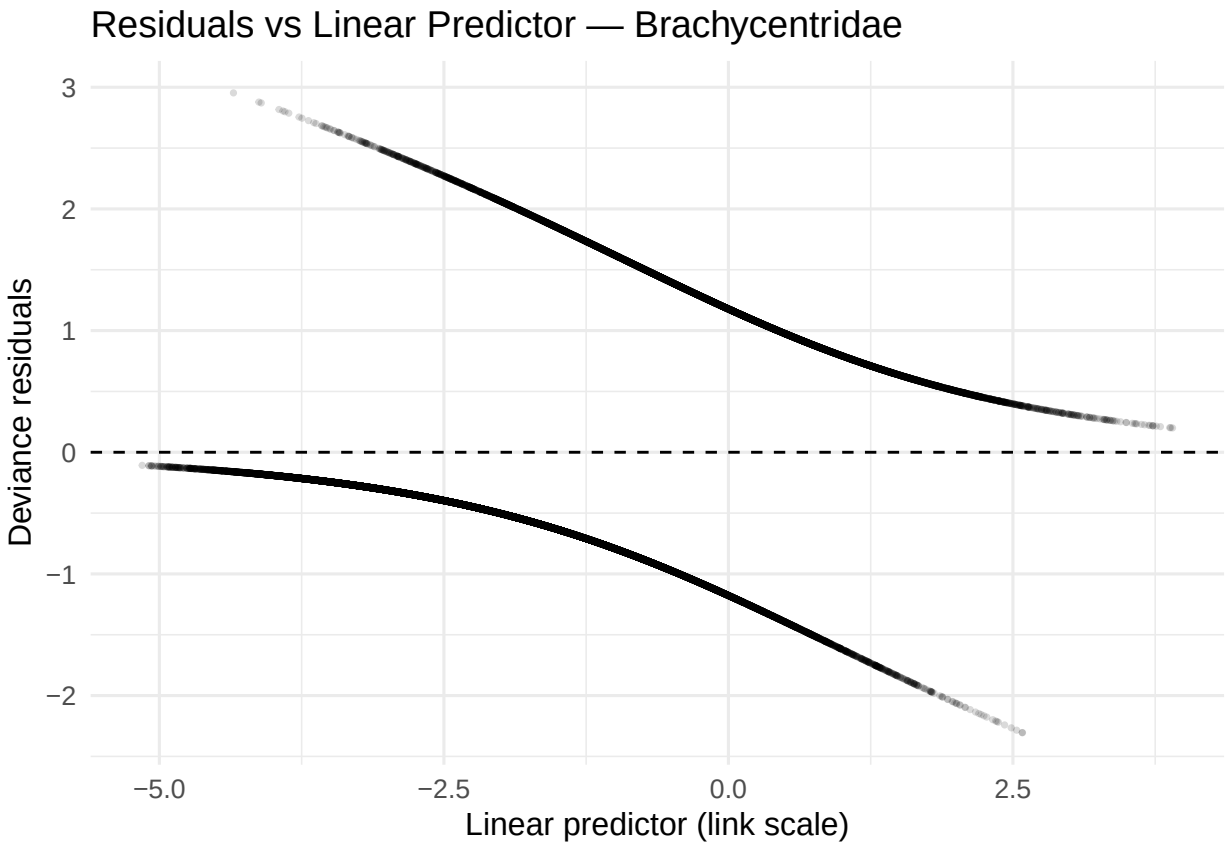

Histogram of Residuals — Aphelocheiridae
Frequency
1500
1000
500
0
−2
−1
0
1
2
3
Deviance residuals

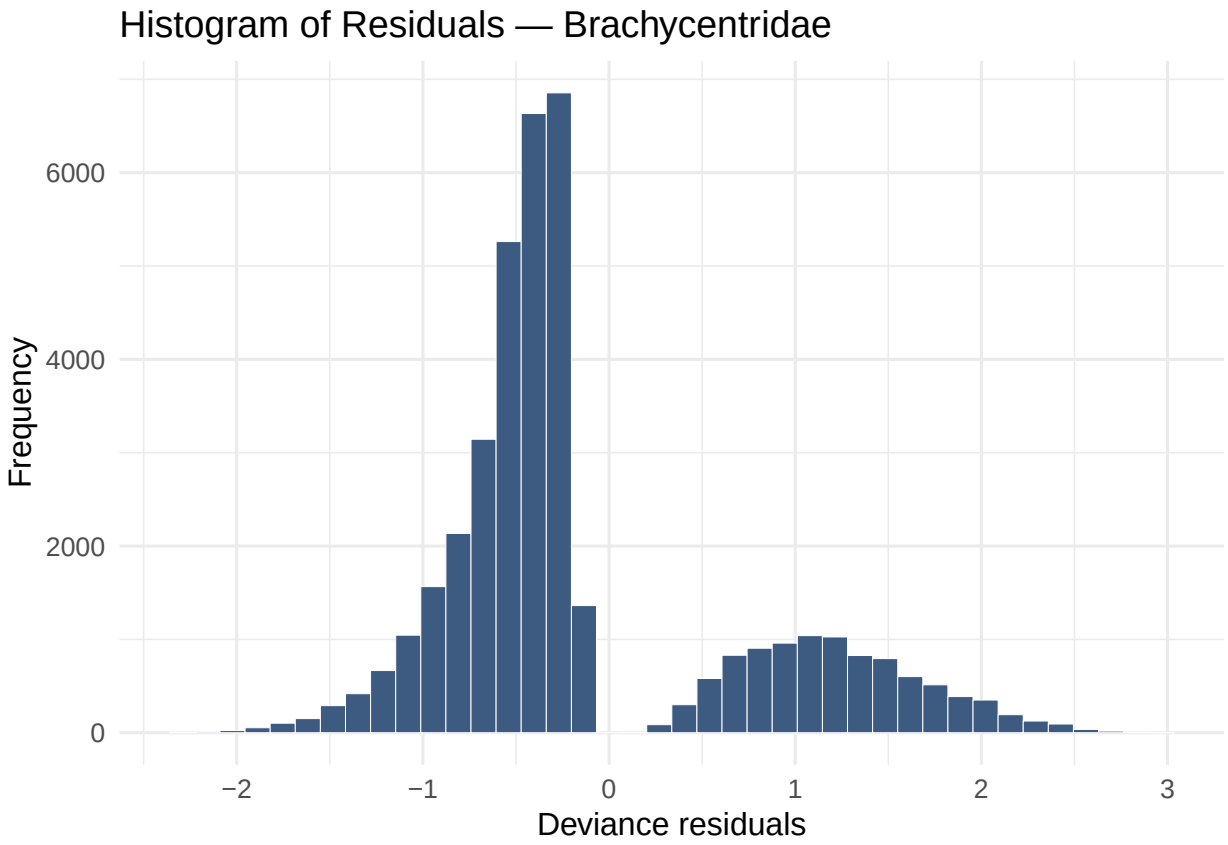

Response vs Fitted — Aphelocheiridae
Observed (0/1)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Fitted probability

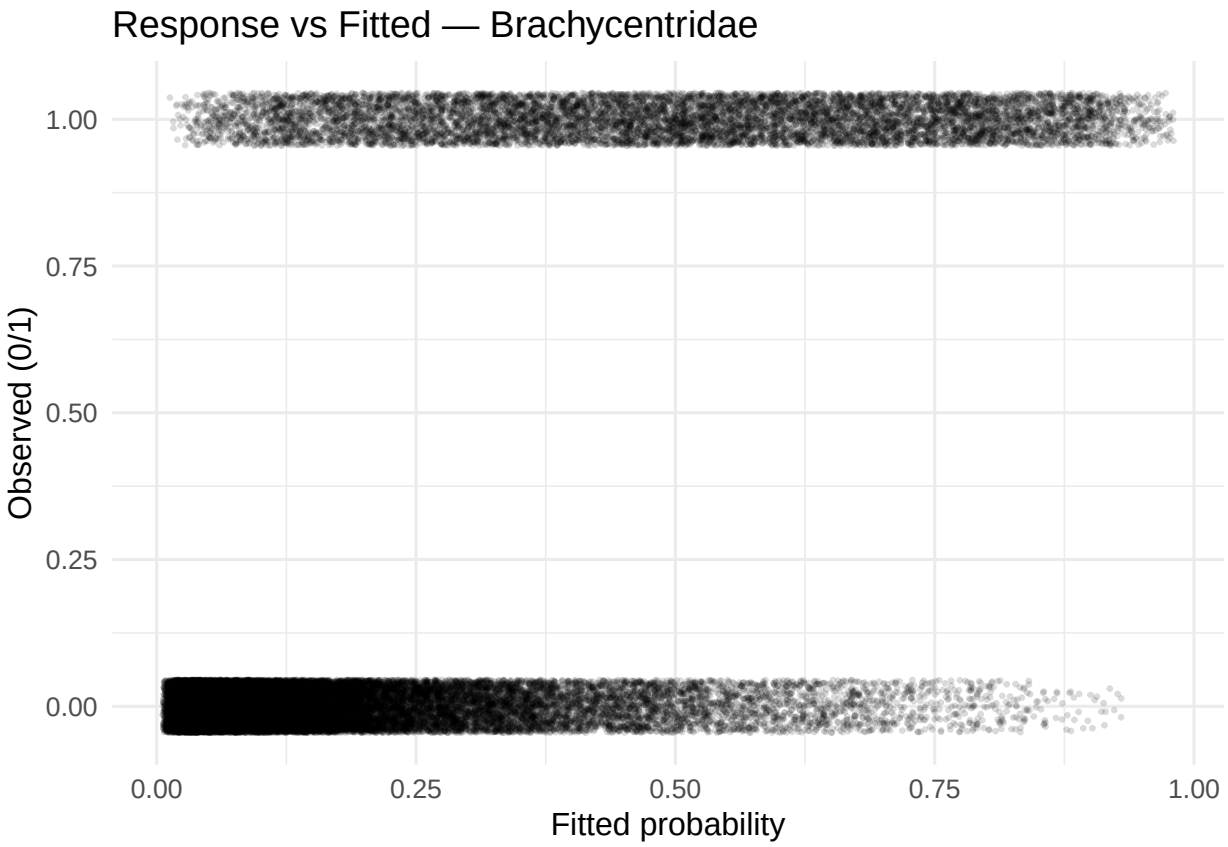

Time smooth (spring) — Aphelocheiridae
By: season_f; spring
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

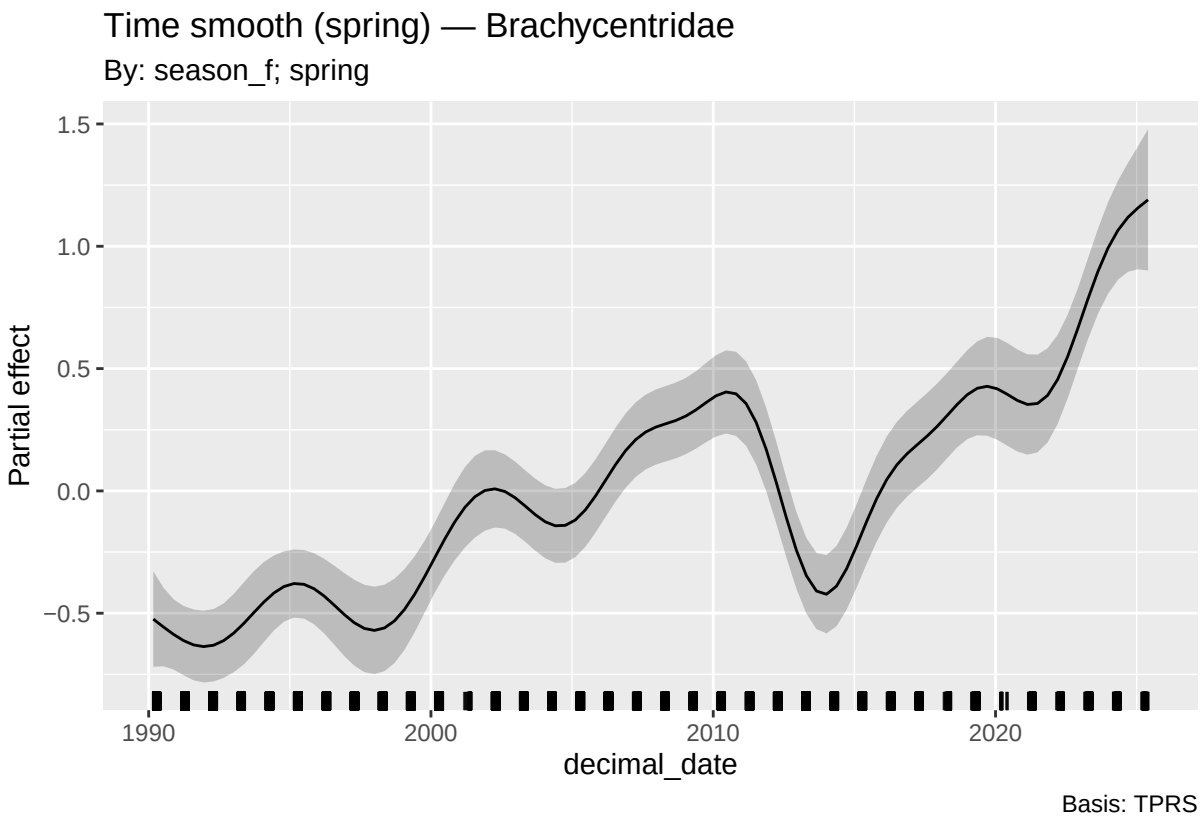

Time smooth (autumn) — Aphelocheiridae
By: season_f; autumn
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

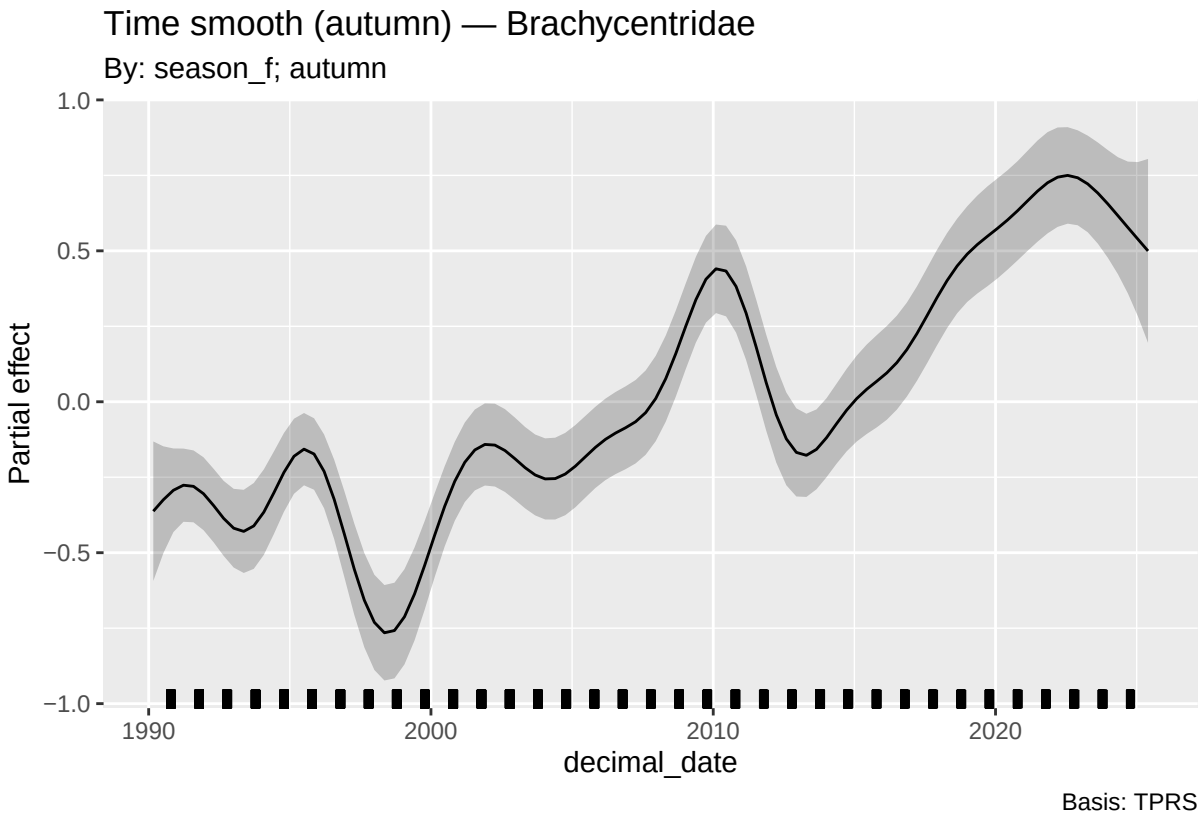

QQ plot — Brachycentridae
deviance residuals
theoretical quantiles
−2
−1
0
1
2
3

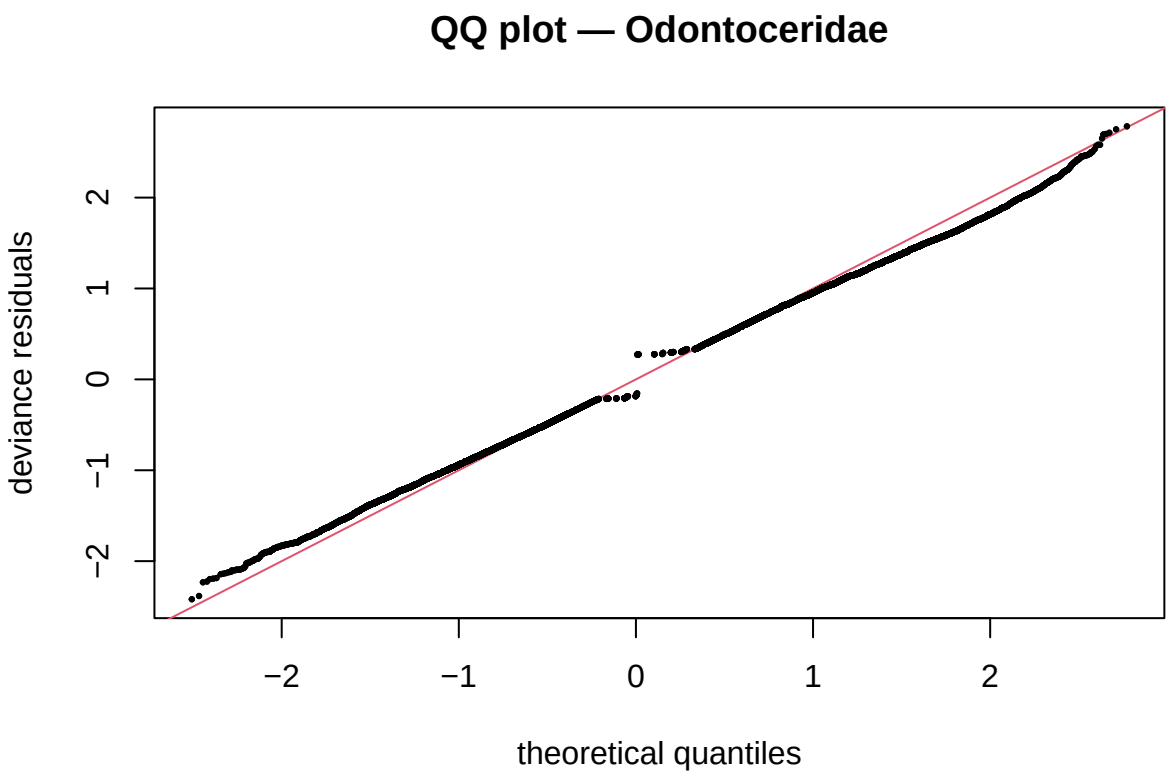

Residuals vs Linear Predictor — Brachycentridae
Deviance residuals
3
2
1
0
−1
−2
−5.0
−2.5
0.0
2.5
Linear predictor (link scale)

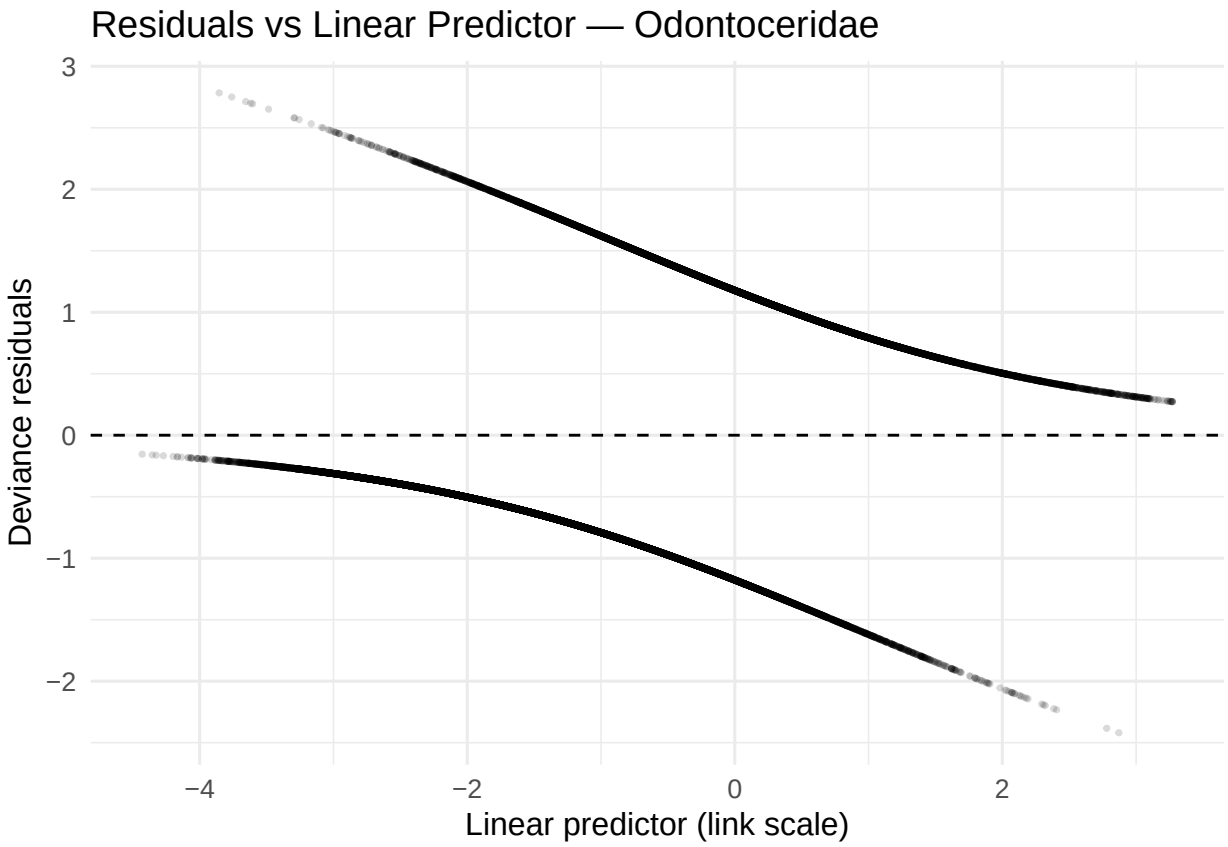

Histogram of Residuals — Brachycentridae
Frequency
6000
4000
2000
0
−2
−1
0
1
2
3
Deviance residuals

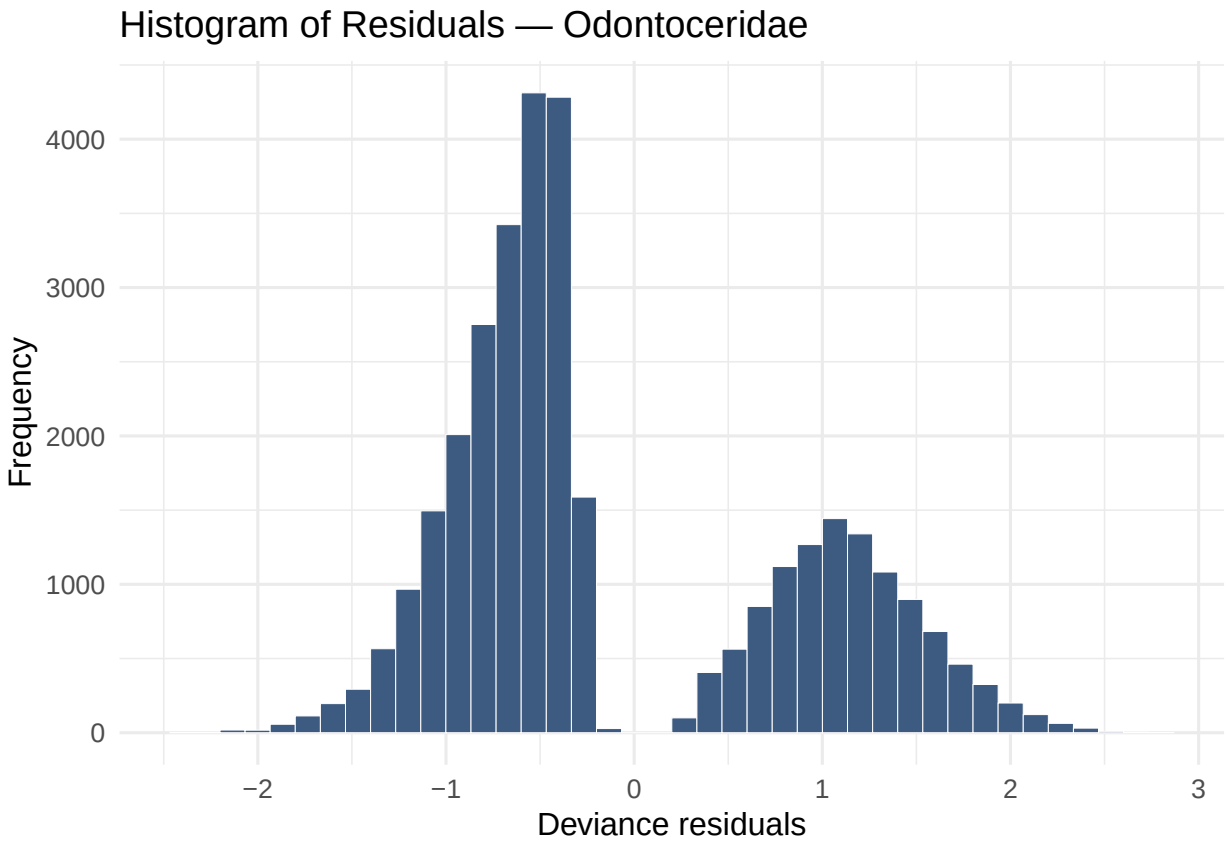

Response vs Fitted — Brachycentridae
Observed (0/1)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Fitted probability

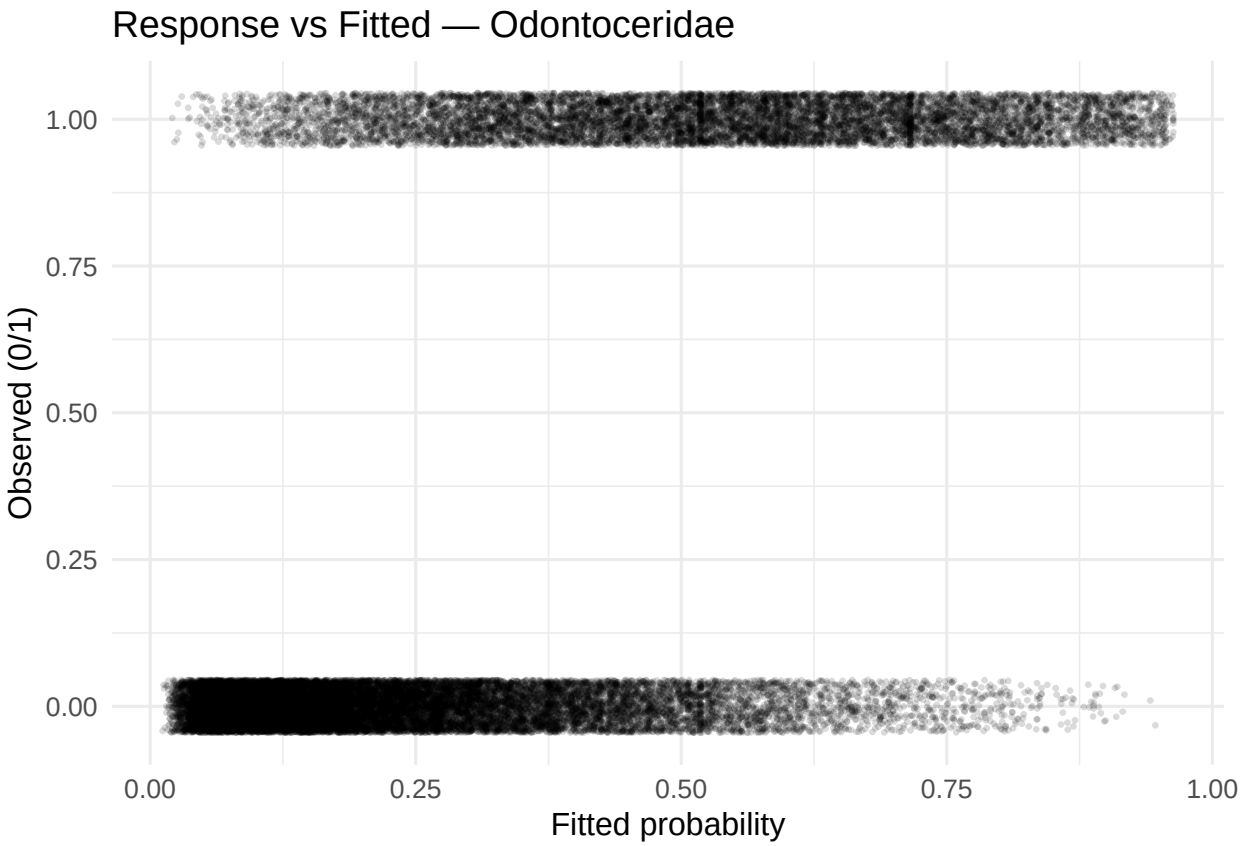

Time smooth (spring) — Brachycentridae
By: season_f; spring
Partial effect
1.5
1.0
0.5
0.0
−0.5
1990
2000
2010
2020
decimal_date
Basis: TPRS

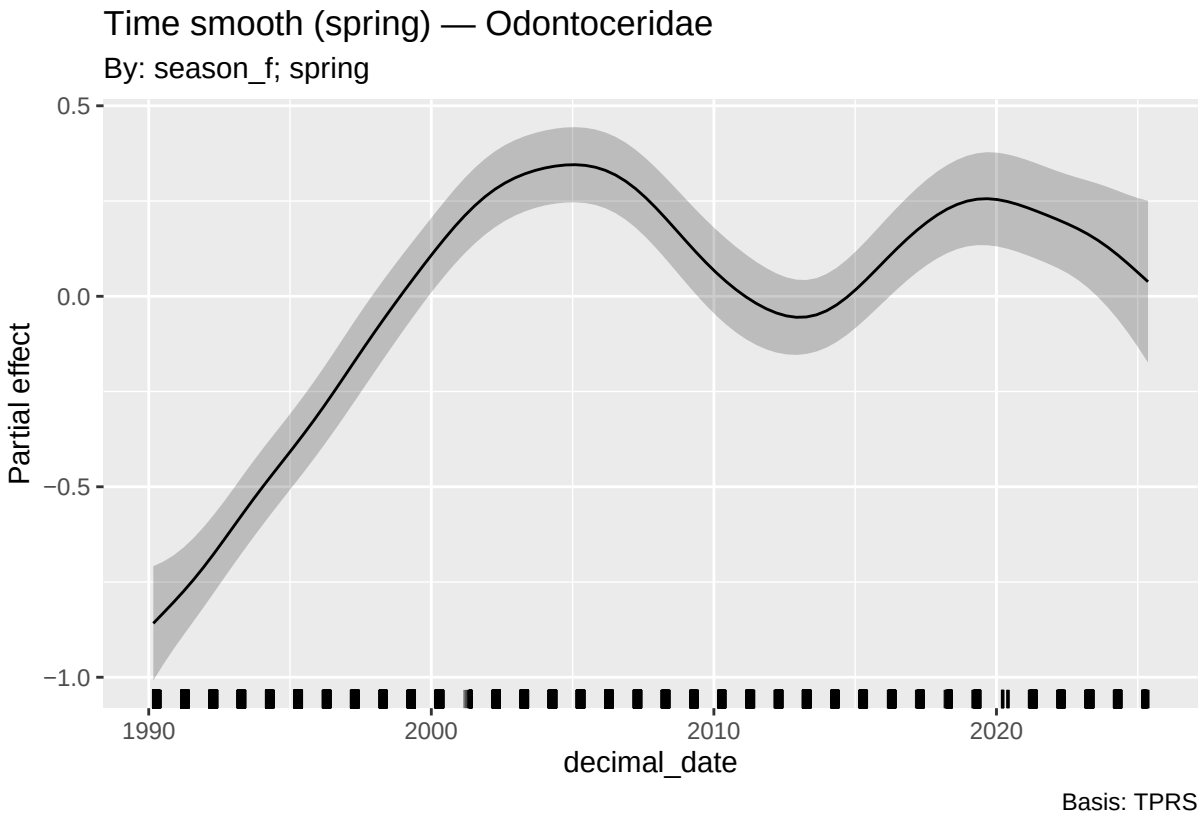

Time smooth (autumn) — Brachycentridae
By: season_f; autumn
Partial effect
1.0
0.5
0.0
−0.5
−1.0
1990
2000
2010
2020
decimal_date
Basis: TPRS

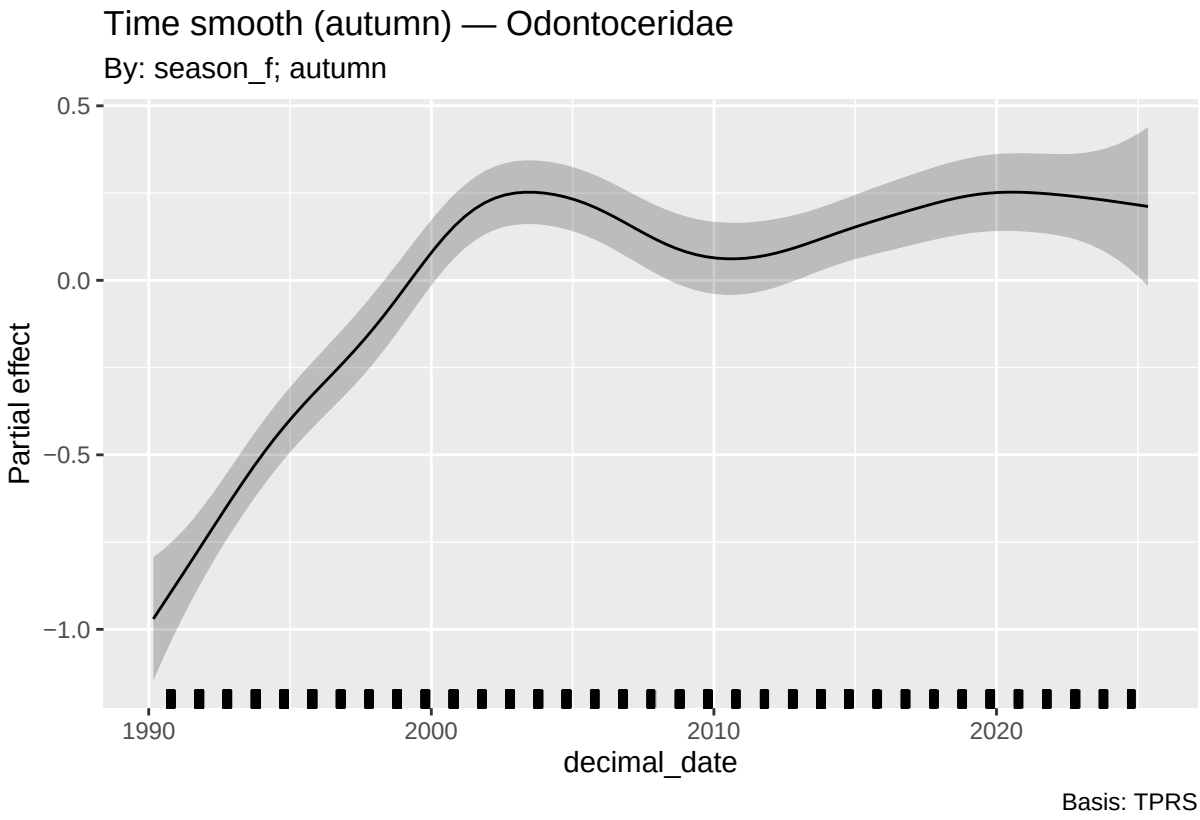

QQ plot — Odontoceridae
deviance residuals
theoretical quantiles
−2
−1
0
1
2

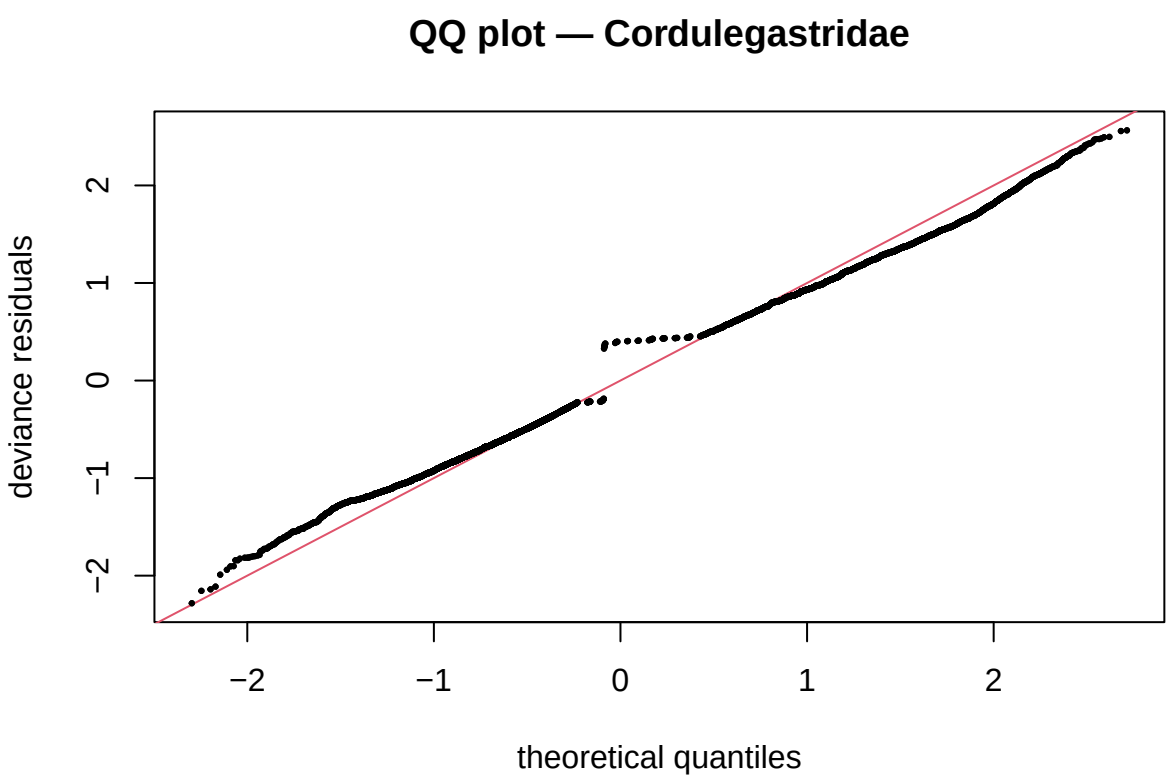

Residuals vs Linear Predictor — Odontoceridae
Deviance residuals
3
2
1
0
−1
−2
−4
−2
0
2
Linear predictor (link scale)

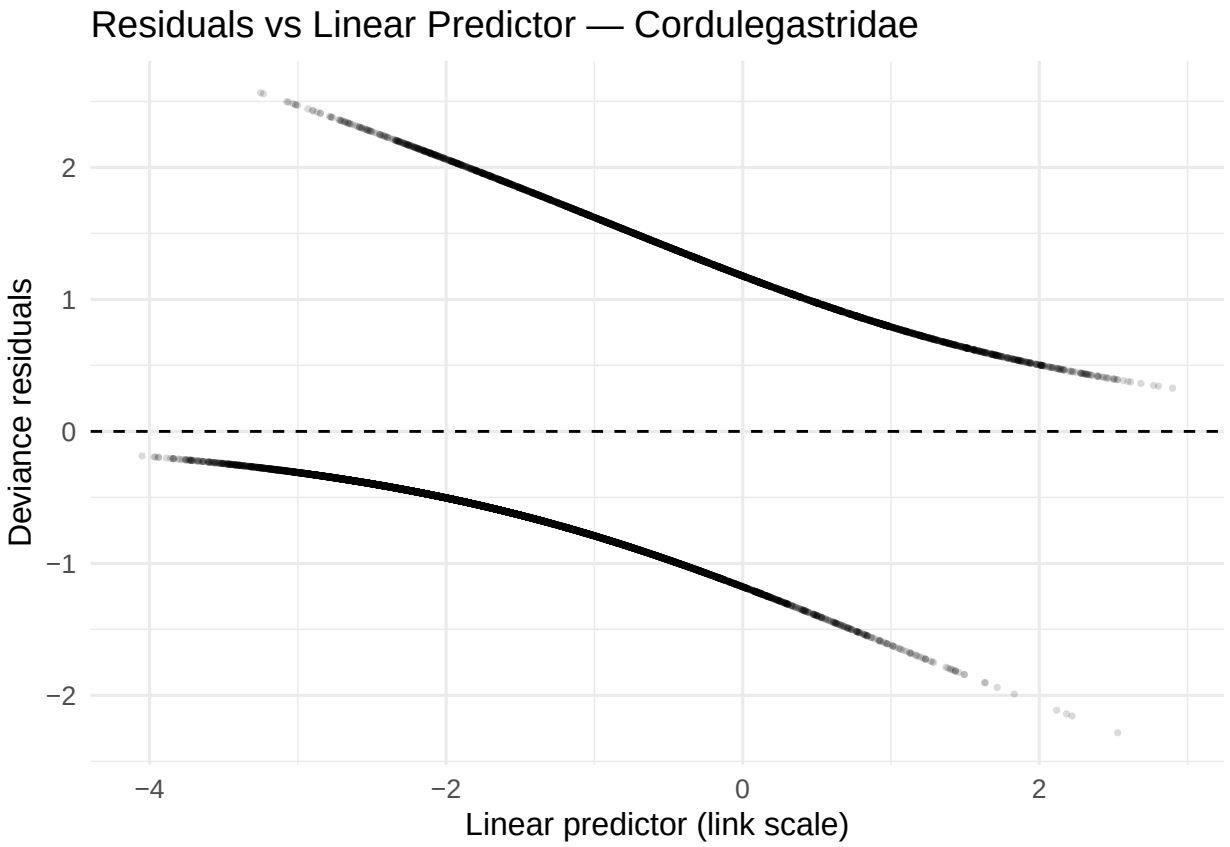

Histogram of Residuals — Odontoceridae
Frequency
4000
3000
2000
1000
0
−2
−1
0
1
2
3
Deviance residuals

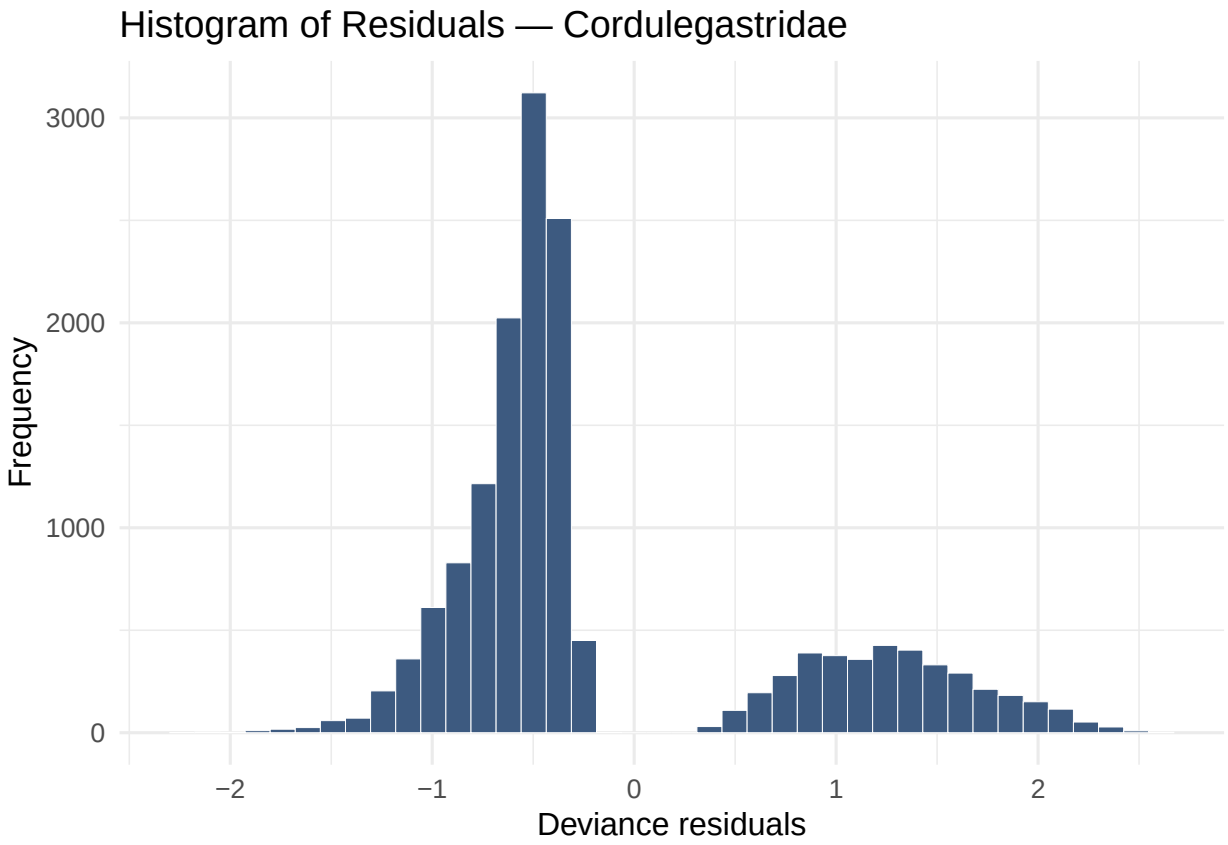

Response vs Fitted — Odontoceridae
Observed (0/1)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Fitted probability

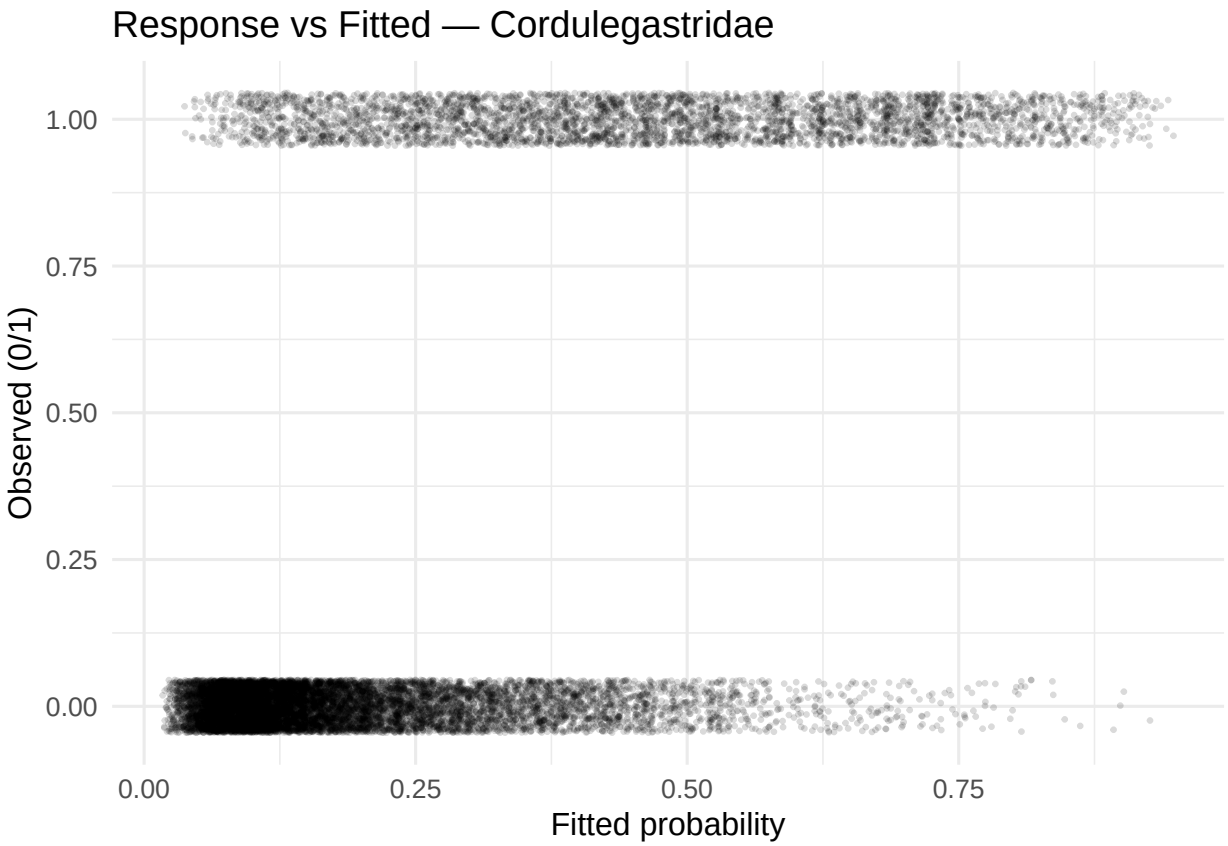

Time smooth (spring) — Odontoceridae
By: season_f; spring
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

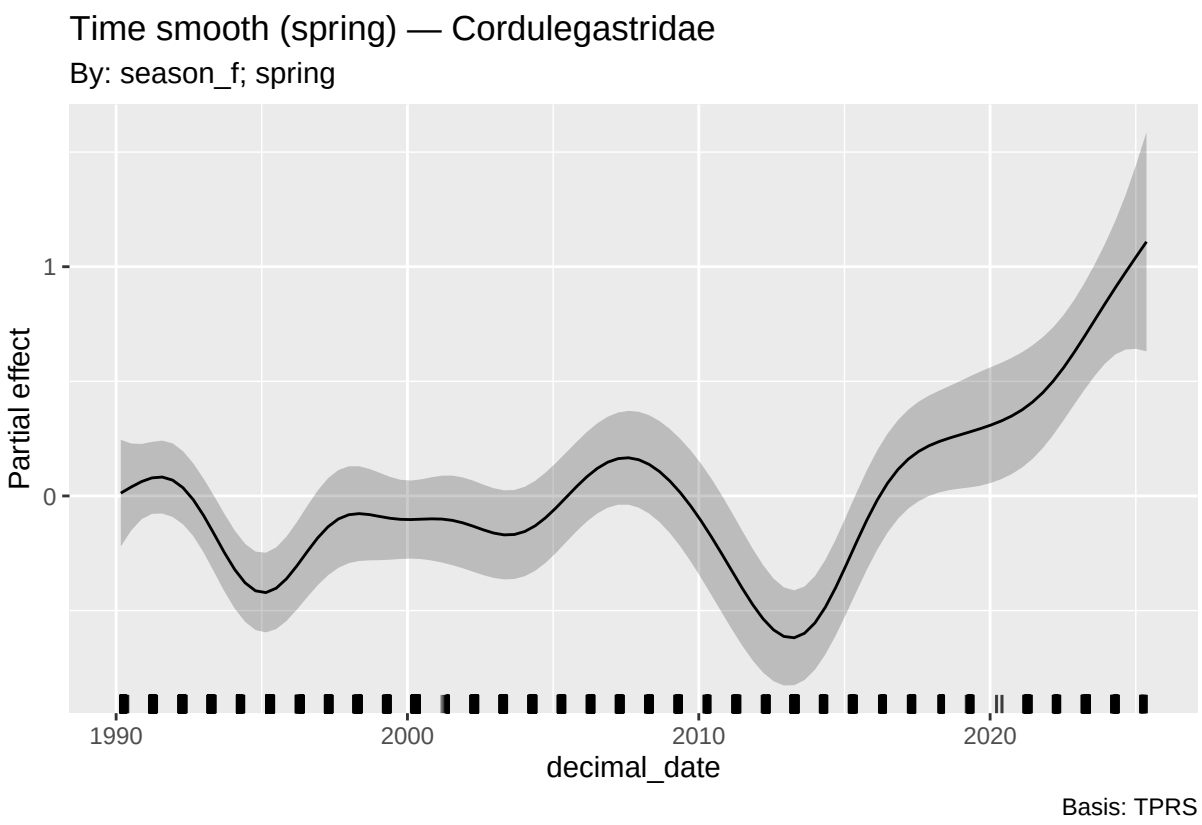

Time smooth (autumn) — Odontoceridae
By: season_f; autumn
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

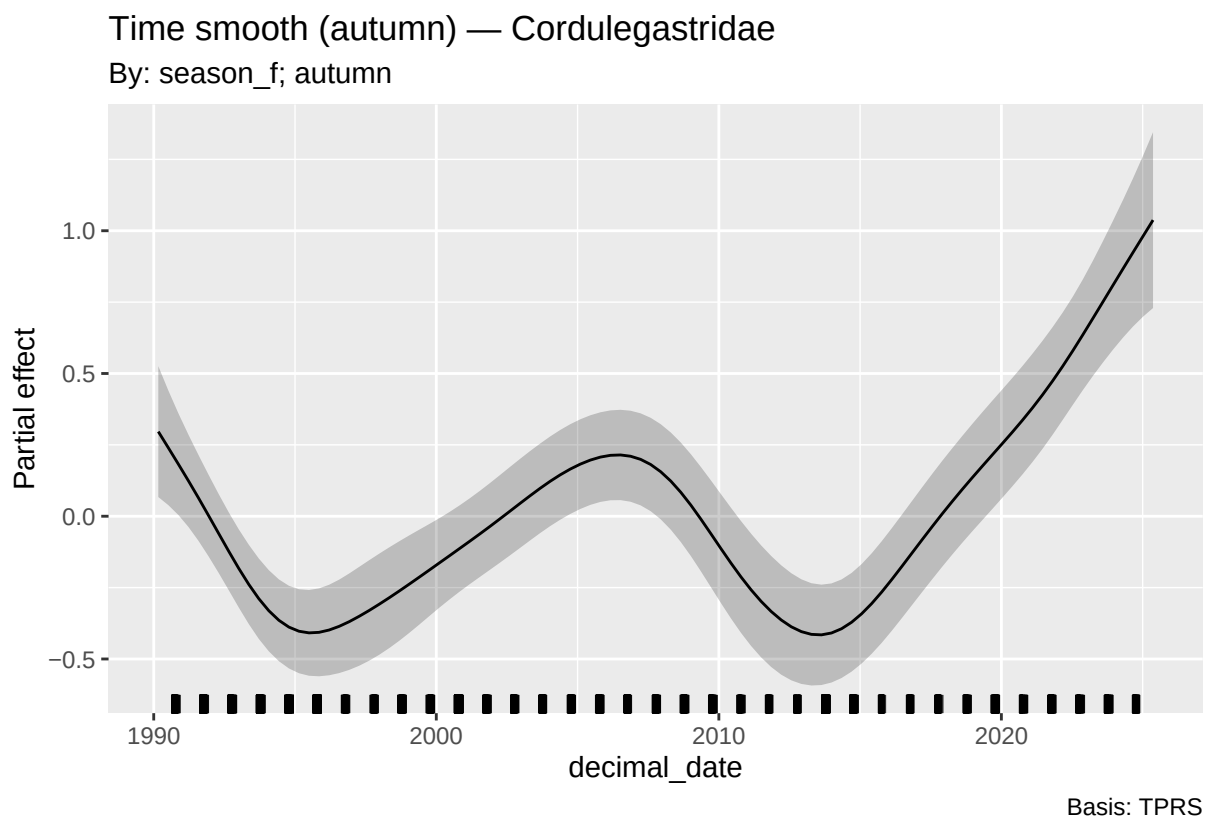

QQ plot — Cordulegastridae
deviance residuals
−2
−1
0
1
2
theoretical quantiles
−2
−1
0
1
2


Residuals vs Linear Predictor — Cordulegastridae
Deviance residuals
2
1
0
−1
−2
−4
−2
0
2
Linear predictor (link scale)


Histogram of Residuals — Cordulegastridae
Frequency
3000
2000
1000
0
−2
−1
0
1
2
Deviance residuals


Response vs Fitted — Cordulegastridae
Observed (0/1)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
Fitted probability


Time smooth (spring) — Cordulegastridae
By: season_f; spring
Partial effect
1
0
1990
2000
2010
2020
decimal_date
Basis: TPRS



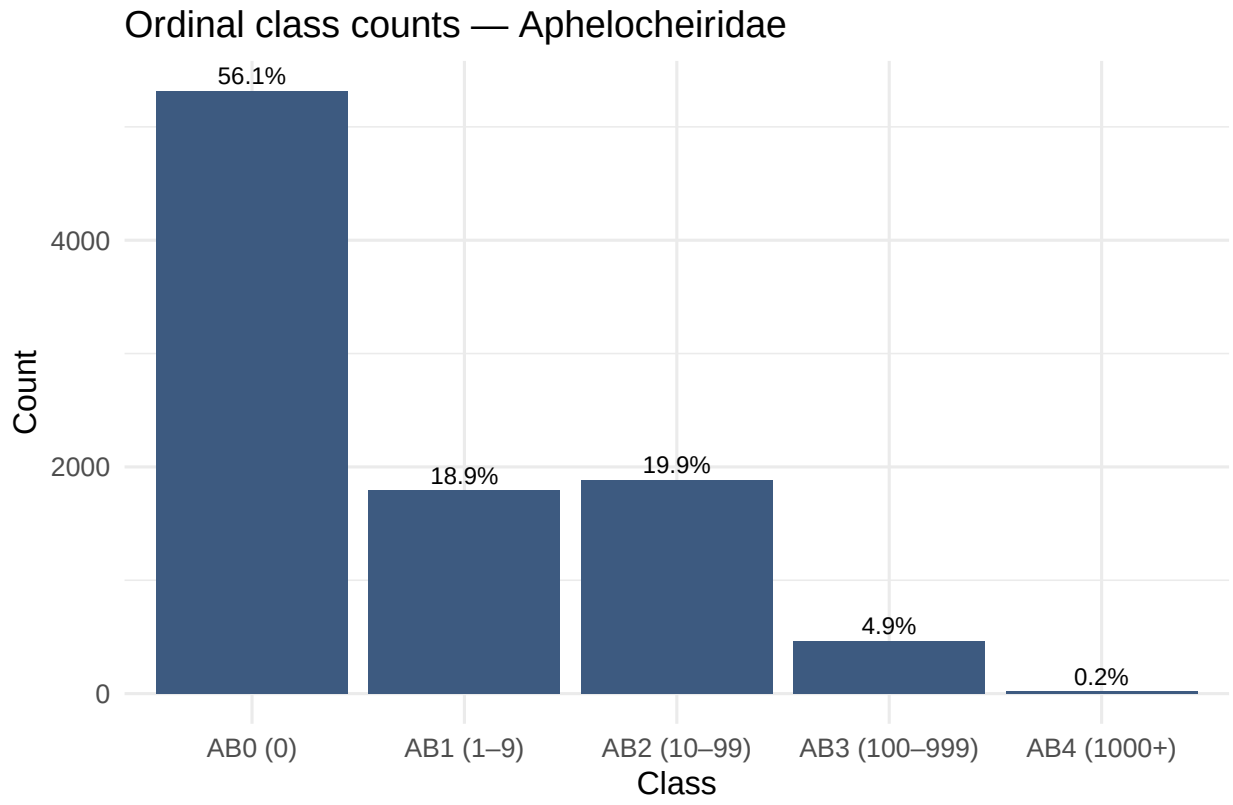
#4.2 ORDINAL(ORDERED-CATEGORICAL) GAMM MODEL

```
# ============================================================
# Model 2 : ORDINAL (ordered-categorical) MODEL - one family
# Pre-step: inspect class counts & sparsity; then fit ocat GAMM
# ============================================================

# ----- choose family -----
fam_name <- "Aphelocheiridae"          # set the target family (use any from your list)
pa_df    <- models_data[[fam_name]]$pa # start from PA table (has zeros + season_f + data_type)

# -------- optional: restrict to S3PO (EA-recommended) --------
USE_S3PO <- TRUE
if (USE_S3PO && "SAMPLE_METHOD" %in% names(pa_df)) {
  pa_df <- dplyr::filter(pa_df, SAMPLE_METHOD == "S3PO")  # keep S3PO-only if column exists
}

# -------- restrict to categorical samples for ordinal modelling --------
ord_base <- pa_df %>%
  filter(data_type == "bin")  # ordinal model uses the categorical (bin) subset only

# -------- build ordered classes using *intervals* (robust to odd values like 6/66/etc.) --------
# AB0 = 0, AB1 = 1-9, AB2 = 10-99, AB3 = 100-999, AB4 = 1000+
ord_df <- ord_base %>%
  mutate(
    ord_class = case_when(
```

```
      TOTAL_NUMBER == 0L                                ~ "AB0 (0)",
      TOTAL_NUMBER > 0L & TOTAL_NUMBER < 10L            ~ "AB1 (1-9)",
      TOTAL_NUMBER >= 10L & TOTAL_NUMBER < 100L         ~ "AB2 (10-99)",
      TOTAL_NUMBER >= 100L & TOTAL_NUMBER < 1000L       ~ "AB3 (100-999)",
      TOTAL_NUMBER >= 1000L                             ~ "AB4 (1000+)",
      TRUE ~ NA_character_
    ),
    ord_class = factor(
      ord_class,
      levels = c("AB0 (0)","AB1 (1-9)","AB2 (10-99)","AB3 (100-999)","AB4 (1000+)")
    )
  ) %>%
  filter(!is.na(ord_class)) %>%         # drop anything outside the intended bins
  mutate(SITE_ID.F = factor(SITE_ID))   # safety: ensure site is a factor for the RE

# --------- Plot & print class counts (sparsity check before fitting) ---------
class_counts <- ord_df %>% count(ord_class, name = "n") %>%
  mutate(prop = n / sum(n))     # proportions to show how dominant AB0 is, etc.

print(class_counts)             # numeric table in console (n and share per class)
```

```
## # A tibble: 5 x 3
##   ord_class         n    prop
##   <fct>         <int>   <dbl>
## 1 AB0 (0)        5316 0.561
## 2 AB1 (1-9)      1792 0.189
## 3 AB2 (10-99)    1886 0.199
## 4 AB3 (100-999)   464 0.0490
## 5 AB4 (1000+)      19 0.00200
```

```
ggplot(class_counts, aes(x = ord_class, y = n)) +
  geom_col(fill = "#3D5A80") +
  geom_text(aes(label = scales::percent(prop, accuracy = 0.1)),
            vjust = -0.4, size = 3.3) +
  labs(title = paste("Ordinal class counts -", fam_name),
       x = "Class", y = "Count",
       caption = "Proportions shown above bars. Consider merging AB4→AB3 if very sparse.") +
  theme_minimal(base_size = 12)
```



Proportions shown above bars. Consider merging AB4−>AB3 if very sparse.

```
# --------- Optional merge rule (AB4→AB3 if sparse); prints a recommendation ---------
min_n    <- 50     # minimum count threshold for highest class
min_prop <- 0.02   # or <2% of data -> merge down
n_ab4    <- class_counts$n[class_counts$ord_class == "AB4 (1000+)"]
p_ab4    <- class_counts$prop[class_counts$ord_class == "AB4 (1000+)"]
if (length(n_ab4) && (is.na(n_ab4) || n_ab4 < min_n || p_ab4 < min_prop)) {
  message(sprintf("AB4 is sparse for %s (n=%s, p=%.2f%%) -> merging AB4 into AB3.",
                  fam_name, ifelse(length(n_ab4)==0, 0, n_ab4), 100*ifelse(length(p_ab4)==0, 0, p_ab4)))
  ord_df <- ord_df %>%
    mutate(ord_class = fct_collapse(ord_class, `AB3 (100-999)` = c("AB3 (100-999)","AB4 (1000+)"))) %>%
    droplevels()  # keep only used levels after merge
}
```

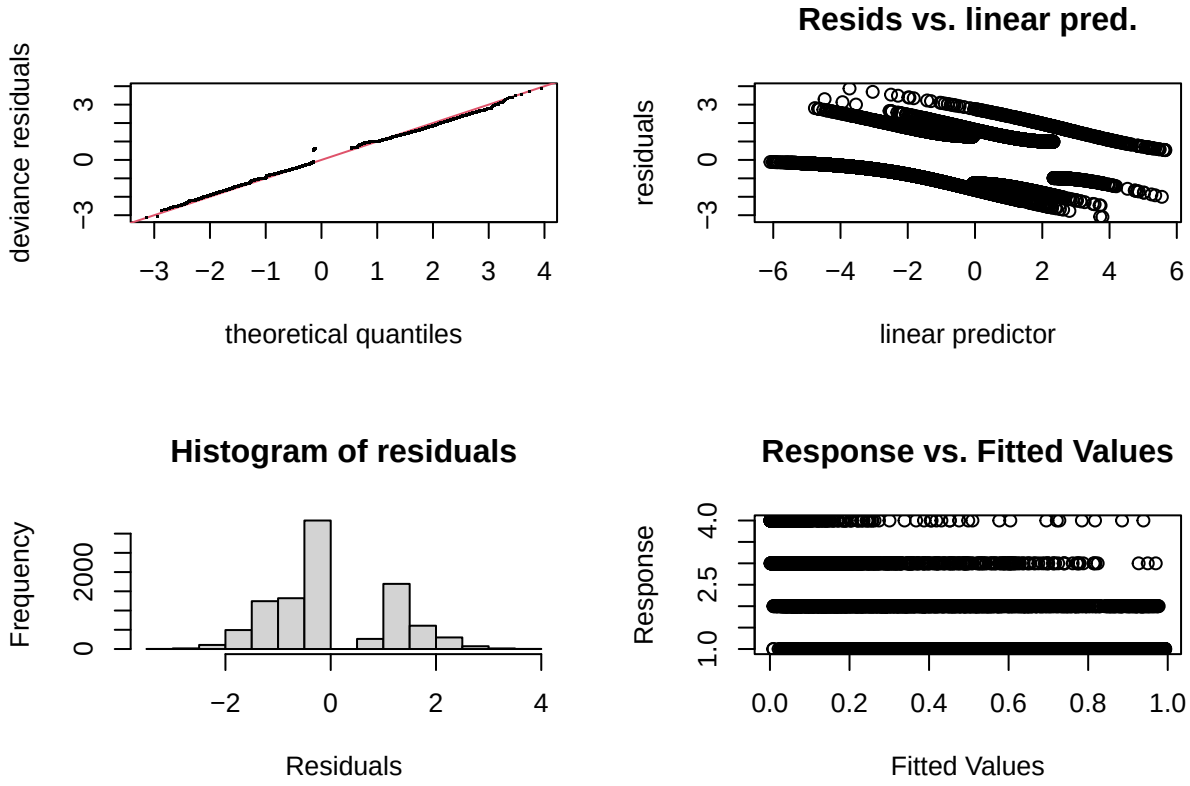
```
## AB4 is sparse for Aphelocheiridae (n=19, p=0.20%) -> merging AB4 into AB3.
```

```
# --------- Prepare response as integers 1..K expected by mgcv::ocat ---------
ord_df <- ord_df %>%
  mutate(
    y_ord = as.integer(factor(ord_class, levels = levels(ord_class))) # ordered categories -> 1..K
  )

K <- length(levels(ord_df$ord_class))  # number of ordinal categories after any merge

# --------- Choose k for the time smooth based on span of years ---------
k_time <- {
```

```r
  ny <- dplyr::n_distinct(lubridate::year(ord_df$SAMPLE_DATE))
  min(20, max(8, round(0.6 * ny)))  # pragmatic cap/floor to avoid under/overfitting
}

# --------- Fit ordered categorical GAMM (season-varying trend + site RE) ---------
# Notes:
# - season_f enters as parametric shift
# - s(decimal_date, by = season_f) allows different smooths by season
# - s(SITE_ID.F, bs = "re") is a random intercept for site
# - family = ocat(R = K) sets ordered categorical likelihood with K thresholds
ocat_model <- bam(
  y_ord ~
    season_f +
    s(decimal_date, by = season_f, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = ocat(R = K),     # ordered categorical family (proportional odds link by default)
  data     = ord_df,
  method   = "fREML",
  discrete = TRUE,            # speed-up for large data
  select   = TRUE,            # shrink unneeded wiggle
  gamma    = 1.2              # mild extra penalty against overfit
)

# --------- Minimal reporting (text only, no figures) ---------
summary(ocat_model)              # parametric terms, EDFs, deviance explained
```

```
##
## Family: Ordered Categorical(-1,0.9,3.77)
## Link function: identity
##
## Formula:
## y_ord ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.80600    0.09319  -8.649  < 2e-16 ***
## season_fautumn  0.32373    0.05216   6.206 5.66e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df      F p-value
## s(decimal_date):season_fspring   4.093     19  8.181  <2e-16 ***
## s(decimal_date):season_fautumn   3.893     19  9.905  <2e-16 ***
## s(SITE_ID.F)                   590.641    721 10.355  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained =   49%
## fREML = 6050.9  Scale est. = 1         n = 9477
```

```r
gam.check(ocat_model, rep = 0)    # basis checks & residual sanity (no re-smoothing)
```




```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 1.839592e-06 4.109021e-07 1.701461e-06 2.737975e-07 2.583552e-04
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.410077812 -0.091259022 -0.019026556 -0.006604064  -0.17319563
## [2,] -0.091259022  0.293650286 -0.006212100 -0.004082348   0.01921733
## [3,] -0.019026556 -0.006212100  1.179119980 -0.003936992  -0.03752700
## [4,] -0.006604064 -0.004082348 -0.003936992  0.109635035   0.04456324
## [5,] -0.173195633  0.019217330 -0.037527004  0.044563236 243.20095587
##
## Model rank =  762 / 762
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.09    0.96  <2e-16 ***
## s(decimal_date):season_fautumn  19.00   3.89    0.96  <2e-16 ***
## s(SITE_ID.F)                   722.00 590.64      NA      NA
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

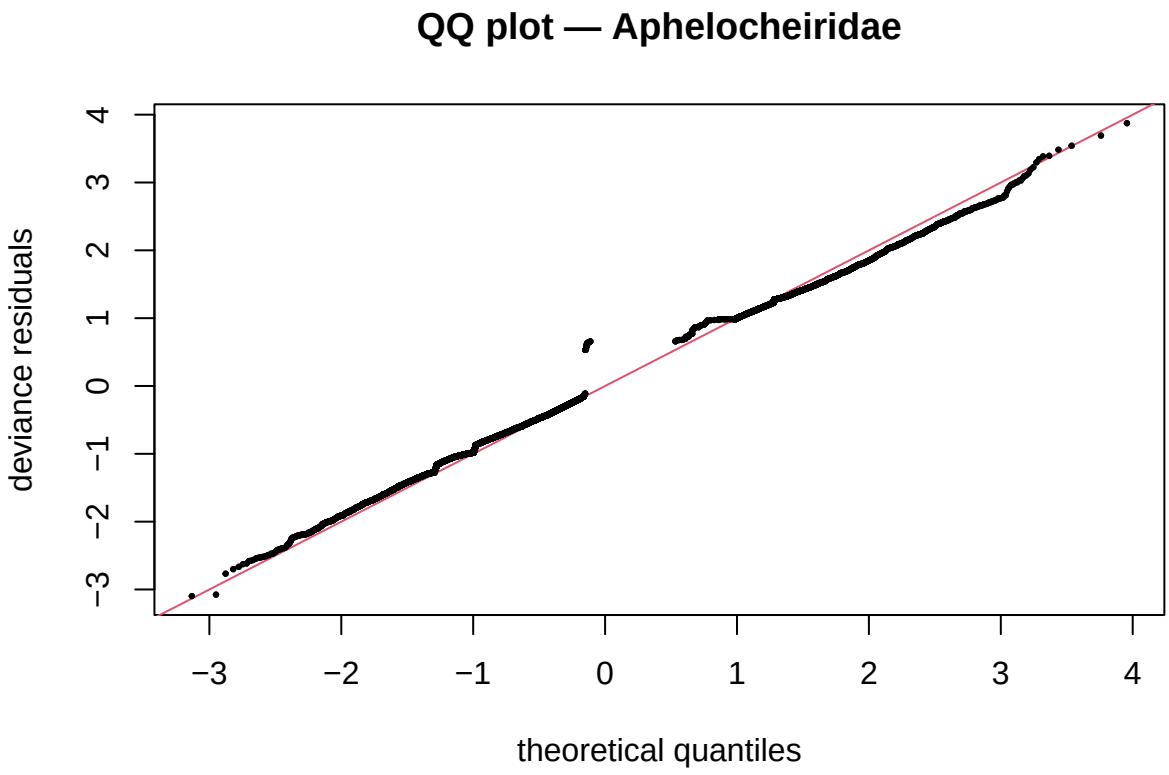
```
# Optional: store for later use (predictions/plots/report)
# saveRDS(ocat_model, paste0("ocat_model_", fam_name, ".rds"))
```

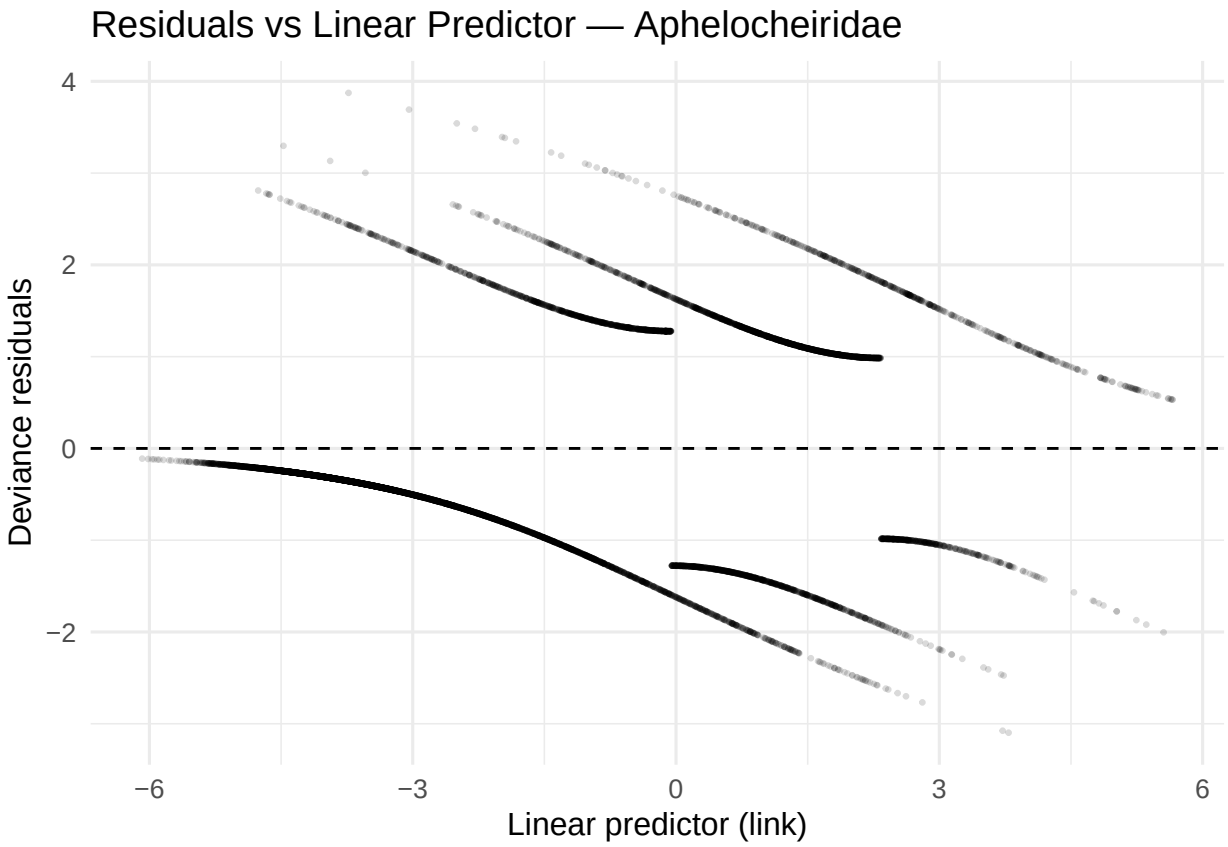
```
# ========================================================
# Ordinal (ocat) Model Diagnostics
# Requires: ocat_model, ord_df, fam_name
# Purpose: run lightweight checks + residual visuals + smooths + thresholds
# ========================================================

inspect_ocat <- function(model, data, fam = "Family") {
  # ---- 1) Basis/smooth adequacy (text only; suppress plot device) ----
  tmp <- tempfile(fileext = ".pdf")    # open a dummy device so gam.check doesn't draw to screen
  pdf(tmp)                             # divert any plots to a temp PDF
  gc_out <- try(gam.check(model, rep = 0), silent = TRUE)  # rep=0 avoids expensive re-smoothing
  dev.off(); unlink(tmp)               # close and delete the temp device/file
  if (!inherits(gc_out, "try-error")) print(gc_out)  # print the textual part if available

  # ---- 2) QQ plot of deviance residuals (single panel, base graphics) ----
  par(mfrow = c(1,1))                  # ensure a single plot frame
  mgcv::qq.gam(model, rep = 0, pch = 19, cex = 0.3,
               main = paste("QQ plot -", fam))  # quick normality check of residuals

  # ---- Prepare one data.frame with linear predictor & residuals (used below) ----
  dd <- data.frame(
    eta   = predict(model, type = "link"),          # linear predictor (latent scale)
    resid = residuals(model, type = "deviance")     # deviance residuals for ordered logit
  )

  # ---- 3) Residuals vs linear predictor (no smoother overlay) ----
  p_res_lin <- ggplot(dd, aes(eta, resid)) +
    geom_point(alpha = 0.15, size = 0.5) +          # light scatter; large n-safe
    geom_hline(yintercept = 0, linetype = 2) +      # zero-reference line
    labs(title = paste("Residuals vs Linear Predictor -", fam),
         x = "Linear predictor (link)", y = "Deviance residuals") +
    theme_minimal(base_size = 12)
  print(p_res_lin)

  # ---- 4) Histogram of deviance residuals (shape check) ----
  p_hist <- ggplot(dd, aes(resid)) +
    geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
    labs(title = paste("Histogram of Residuals -", fam),
         x = "Deviance residuals", y = "Frequency") +
    theme_minimal(base_size = 12)
  print(p_hist)

  # ---- 5) Partial effects: season-specific time smooths (spring/autumn) ----
  # Identify smooth indices that involve 'decimal_date' (the temporal smooth)
  sm_ids   <- which(grepl("decimal_date", smooths(model)))
  sm_names <- smooths(model)[sm_ids]
  for (j in seq_along(sm_ids)) {
    # Friendly facet label from smooth name
    lab <- if (grepl("spring", sm_names[j], ignore.case = TRUE)) "spring"
```

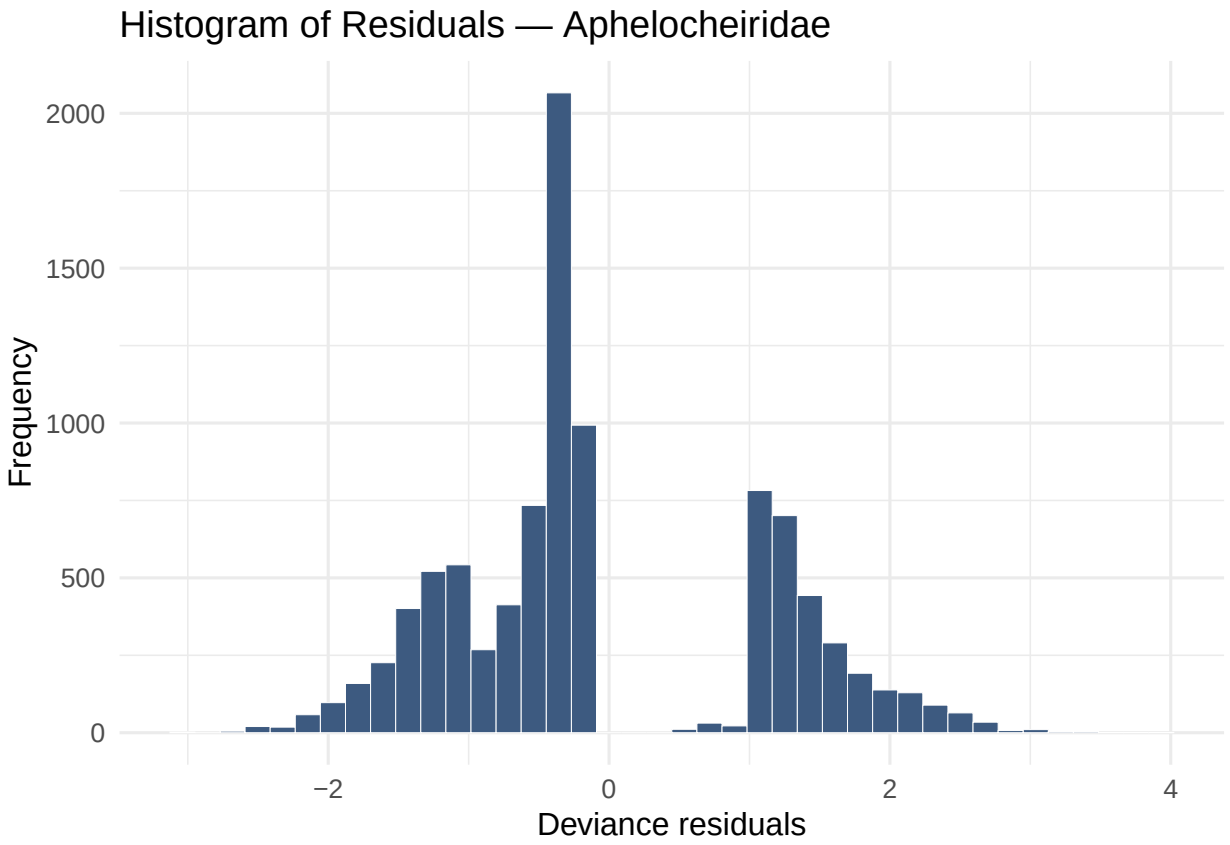
```
          else if (grepl("autumn", sm_names[j], ignore.case = TRUE)) "autumn"
          else sm_names[j]
    # Draw each selected smooth with gratia::draw()
    p_sm <- draw(model, select = sm_ids[j]) +
      ggtitle(paste("Time smooth (", lab, ") - ", fam, sep = ""))
    print(p_sm)
  }

  # ---- 6) Thresholds (cutpoints a1…aK-1) on the latent scale with 95% CI ----
  cf  <- coef(model)                  # full coefficient vector
  V   <- vcov(model)                  # covariance matrix of coefficients
  idx <- grep("^a[0-9]+$", names(cf)) # threshold parameters are named a1, a2, ...
  if (length(idx)) {
    # Assemble estimates and Wald CIs for the ordered cutpoints
    th <- data.frame(threshold = names(cf)[idx],
                     est = cf[idx],
                     se  = sqrt(diag(V)[idx]))
    th$lower <- th$est - 1.96 * th$se
    th$upper <- th$est + 1.96 * th$se
    th$threshold <- factor(th$threshold, levels = th$threshold)

    # Plot thresholds with error bars - increasing cutpoints indicates ordered categories
    p_th <- ggplot(th, aes(threshold, est)) +
      geom_point() +
      geom_errorbar(aes(ymin = lower, ymax = upper), width = 0.15) +
      labs(title = paste("Category thresholds (latent scale) -", fam),
           x = "Cutpoint (a1 < a2 < …)", y = "Estimate ± 95% CI") +
      theme_minimal(base_size = 12)
    print(p_th)
  }

  invisible(NULL)  # return nothing (side-effect is printed output/plots)
}

# ---- Run diagnostics on your fitted ordered categorical model ----
inspect_ocat(ocat_model, ord_df, fam_name)
```

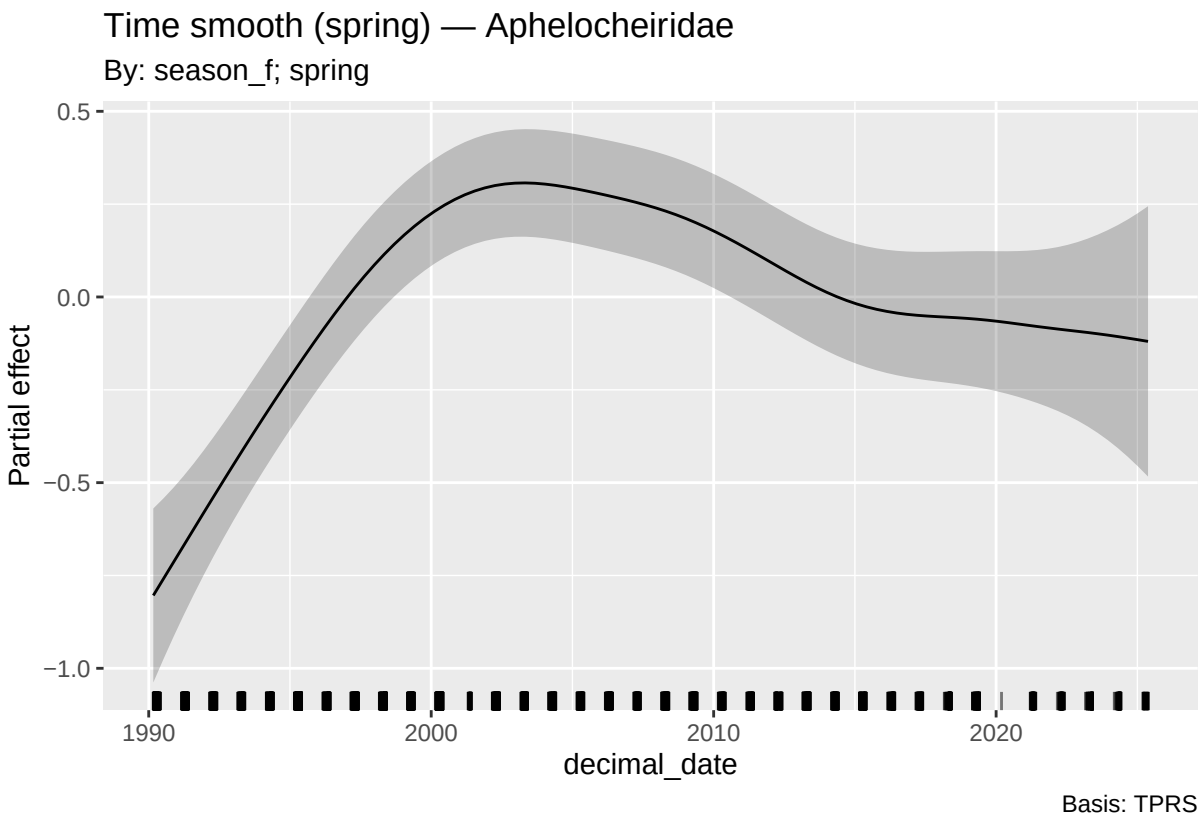
```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 1.839592e-06 4.109021e-07 1.701461e-06 2.737975e-07 2.583552e-04
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.410077812 -0.091259022 -0.019026556 -0.006604064  -0.17319563
## [2,] -0.091259022  0.293650286 -0.006212100 -0.004082348   0.01921733
## [3,] -0.019026556 -0.006212100  1.179119980 -0.003936992  -0.03752700
## [4,] -0.006604064 -0.004082348 -0.003936992  0.109635035   0.04456324
## [5,] -0.173195633  0.019217330 -0.037527004  0.044563236 243.20095587
##
## Model rank =  762 / 762
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
```

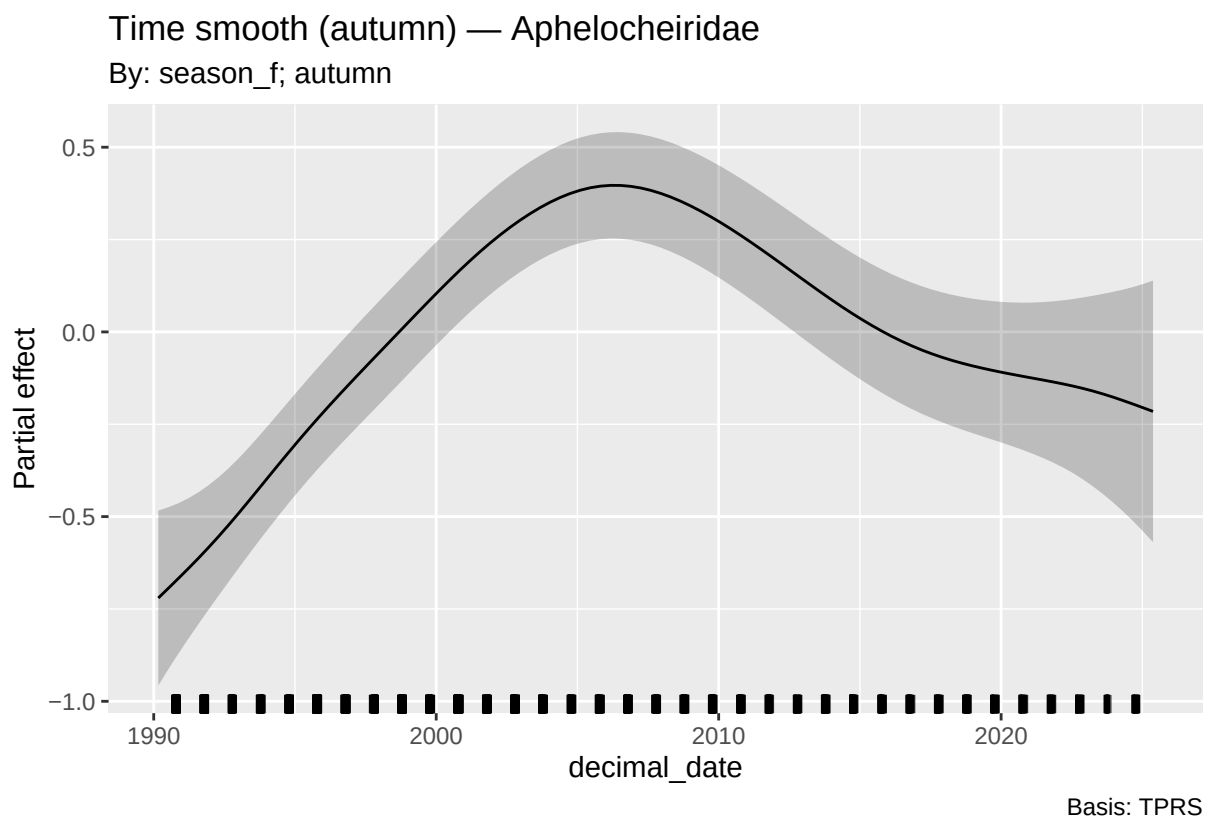
```
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.09    0.94  <2e-16 ***
## s(decimal_date):season_fautumn  19.00   3.89    0.94  <2e-16 ***
## s(SITE_ID.F)                   722.00 590.64      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
```




Residuals vs Linear Predictor — Aphelocheiridae
Deviance residuals
4
2
0
−2
−6
−3
0
3
6
Linear predictor (link)


Histogram of Residuals — Aphelocheiridae
Frequency
2000
1500
1000
500
0
−2
0
2
4
Deviance residuals


Time smooth (spring) — Aphelocheiridae
By: season_f; spring
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS




```r
# ================================================================
# PHASE 4 - Batch Ordered-Categorical (OCAT) GAMMs for 4 families
#  - Uses categorical samples only (data_type == "bin")
#  - Optional S3PO-only filter
#  - Top-down merge of sparse top class (AB4 -> AB3) when very rare
#  - Prints summary() and gam.check() output
#  - Returns compact results + a summary table
# ================================================================

# Families to run
families <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")
# ^ Vector of target macroinvertebrate families to process.

# Project options
USE_S3PO       <- TRUE   # EA-recommended
MIN_TOP_N      <- 50     # merge AB4 into AB3 if AB4 count < MIN_TOP_N ...
MIN_TOP_PROP   <- 0.02   # ... or if AB4 share < 2%
# ^ Controls for optional collapsing of a very sparse highest category.

# ---- Helper: build ordinal classes from a PA table ----
build_ordinal_df <- function(df_pa) {
  df_pa %>%
    mutate(
      # Map TOTAL_NUMBER into ordered bins using inclusive intervals:
      # AB0: exactly 0; AB1: 1-9; AB2: 10-99; AB3: 100-999; AB4: 1000+
      ord_class = case_when(
```

```
        TOTAL_NUMBER == 0L                          ~ "AB0 (0)",
        TOTAL_NUMBER > 0L   & TOTAL_NUMBER < 10L    ~ "AB1 (1-9)",
        TOTAL_NUMBER >= 10L & TOTAL_NUMBER < 100L   ~ "AB2 (10-99)",
        TOTAL_NUMBER >= 100L & TOTAL_NUMBER < 1000L~ "AB3 (100-999)",
        TOTAL_NUMBER >= 1000L                       ~ "AB4 (1000+)",
        TRUE ~ NA_character_
      ),
      # Fix the factor level order to reflect the natural ordinal scale
      ord_class = factor(
        ord_class,
        levels = c("AB0 (0)","AB1 (1-9)","AB2 (10-99)","AB3 (100-999)","AB4 (1000+)")
      ),
      # Ensure a site-level random-effect factor exists (create if missing)
      SITE_ID.F = if (!"SITE_ID.F" %in% names(.)) factor(SITE_ID) else SITE_ID.F
    ) %>%
    filter(!is.na(ord_class))  # drop any rows that failed classification
}

# ---- Helper: fit OCAT model for one family ----
fit_ocat_family <- function(fam_name,
                            md      = models_data,
                            use_s3po = TRUE,
                            min_top_n = MIN_TOP_N,
                            min_top_prop = MIN_TOP_PROP) {
  stopifnot(!is.null(md[[fam_name]]), "pa" %in% names(md[[fam_name]]))
  df <- md[[fam_name]]$pa
  # ^ Start from the Presence/Absence table for this family (already has zeros etc.)

  # Keep categorical samples only
  df <- dplyr::filter(df, data_type == "bin")
  # ^ The ordered-categorical model is defined on categorical observations.

  # Optional: S3PO-only
  if (use_s3po && "SAMPLE_METHOD" %in% names(df)) {
    df <- dplyr::filter(df, SAMPLE_METHOD == "S3PO")
  }
  # ^ Restrict to EA's recommended sampling method when available.

  # Season guard (should already exist if you added the preprocessing patch)
  if (!("season_f" %in% names(df))) {
    df <- df %>%
      mutate(m = month(SAMPLE_DATE),
             season_f = factor(if_else(m %in% 3:5, "spring", "autumn"),
                               levels = c("spring","autumn")))
  }
  # ^ Ensure a two-level seasonal factor (spring/autumn) is present.

  # Map to ordinal classes
  ord_df <- build_ordinal_df(df)
  # ^ Convert raw TOTAL_NUMBER into ordered categories (AB0…AB4).

  # Merge AB4 -> AB3 if very sparse
  class_counts <- ord_df %>% count(ord_class, name = "n") %>%
```

```
  mutate(prop = n / sum(n))
n_ab4 <- class_counts$n[class_counts$ord_class == "AB4 (1000+)"]
p_ab4 <- class_counts$prop[class_counts$ord_class == "AB4 (1000+)"]
if (length(n_ab4) && (is.na(n_ab4) || n_ab4 < min_top_n || p_ab4 < min_top_prop)) {
  ord_df <- ord_df %>%
    mutate(ord_class = fct_collapse(ord_class,
                      `AB3 (100-999)` = c("AB3 (100-999)","AB4 (1000+)"))) %>%
    droplevels()
}
# ^ Optional sparsity rule: collapse a rare top class to stabilize the fit.

# Response 1..K for ocat()
ord_df <- ord_df %>%
  mutate(y_ord = as.integer(factor(ord_class, levels = levels(ord_class))))
K <- nlevels(ord_df$ord_class)
# ^ mgcv::ocat expects integer categories 1..K; keep K for family().

# Sensible k from span of years
k_time <- {
  ny <- dplyr::n_distinct(lubridate::year(ord_df$SAMPLE_DATE))
  min(20, max(8, round(0.6 * ny)))
}
# ^ Time-smooth basis dimension chosen from temporal coverage (guardrails 8..20).

# Fit the ordered-categorical GAMM
m <- bam(
  y_ord ~
    season_f +
    s(decimal_date, by = season_f, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = ocat(R = K),
  data     = ord_df,
  method   = "fREML",
  discrete = TRUE,
  select   = TRUE,
  gamma    = 1.2
)
# ^ Model: season-specific temporal smooth + site random intercept on the latent scale.

# Header + summary
cat("\n============================\n")
cat(sprintf("OCAT Family: %s | Classes: %s\n", fam_name,
            paste(levels(ord_df$ord_class), collapse = " | ")))
cat("============================\n")
print(summary(m))
# ^ Print parameteric terms, EDFs, deviance explained, etc.

# gam.check (plots suppressed)
tmp <- tempfile(fileext = ".pdf")
pdf(tmp)
gc_out <- try(gam.check(m, rep = 0), silent = TRUE)
dev.off(); unlink(tmp)
if (!inherits(gc_out, "try-error")) print(gc_out)
```

```r
  # ^ Report k-index and residual checks without opening the 2x2 plot grid.

  # Metrics
  devex <- summary(m)$dev.expl
  aic   <- AIC(m)
  nobs_ <- nobs(m)
  # ^ Quick scalar diagnostics: deviance explained, AIC, sample size.

  list(
    family     = fam_name,          # family name
    model      = m,                 # fitted mgcv::bam object
    data       = ord_df,            # modelling data (ordinal-coded)
    classes    = levels(ord_df$ord_class),  # human-readable class labels
    K          = K,                 # number of ordinal levels
    k_time     = k_time,            # time-smooth basis size used
    dev_expl   = devex,             # deviance explained (fraction)
    AIC        = aic,               # AIC
    n          = nobs_              # number of observations
  )
}

# ---- Run for all families ----
ocat_results <- lapply(families, fit_ocat_family,
                       md = models_data, use_s3po = USE_S3PO)
```

```
##
## ============================
## OCAT Family: Aphelocheiridae | Classes: AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-
999)
## ============================
##
## Family: Ordered Categorical(-1,0.9,3.77)
## Link function: identity
##
## Formula:
## y_ord ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.80600    0.09319  -8.649  < 2e-16 ***
## season_fautumn  0.32373    0.05216   6.206 5.66e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df      F p-value
## s(decimal_date):season_fspring   4.093     19  8.181  <2e-16 ***
## s(decimal_date):season_fautumn   3.893     19  9.905  <2e-16 ***
## s(SITE_ID.F)                   590.641    721 10.355  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Deviance explained =   49%
## fREML = 6050.9  Scale est. = 1         n = 9477

##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 1.839592e-06 4.109021e-07 1.701461e-06 2.737975e-07 2.583552e-04
##
## $hess
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,]  1.410077812 -0.091259022 -0.019026556 -0.006604064  -0.17319563
## [2,] -0.091259022  0.293650286 -0.006212100 -0.004082348   0.01921733
## [3,] -0.019026556 -0.006212100  1.179119980 -0.003936992  -0.03752700
## [4,] -0.006604064 -0.004082348 -0.003936992  0.109635035   0.04456324
## [5,] -0.173195633  0.019217330 -0.037527004  0.044563236 243.20095587
##
## Model rank =  762 / 762
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  19.00   4.09    0.93  <2e-16 ***
## s(decimal_date):season_fautumn  19.00   3.89    0.93  <2e-16 ***
## s(SITE_ID.F)                   722.00 590.64      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## ============================
## OCAT Family: Brachycentridae | Classes: AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-
999)
## ============================
##
## Family: Ordered Categorical(-1,0.94,3.4)
## Link function: identity
##
## Formula:
## y_ord ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -2.50385    0.04098  -61.10   <2e-16 ***
## season_fautumn  0.86921    0.03114   27.91   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df      F p-value
## s(decimal_date):season_fspring   13.93     19 33.207  <2e-16 ***
## s(decimal_date):season_fautumn   14.32     19 26.923  <2e-16 ***
```

```
## s(SITE_ID.F)                   1578.68   2285  4.618  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained = 30.9%
## fREML =  16670  Scale est. = 1         n = 33477

##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  1.985823e-06 -7.192811e-06  2.180355e-06 -1.343909e-04  2.893132e-04
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  3.408275e+00 -1.222796e-06 -1.094901e-03 -9.642549e-06 -5.307630e-01
## [2,] -1.222796e-06  7.192707e-06  5.223836e-09 -2.346882e-11 -4.177524e-07
## [3,] -1.094901e-03  5.223836e-09  3.832777e+00 -4.387752e-04  4.813564e-02
## [4,] -9.642549e-06 -2.346882e-11 -4.387752e-04  1.345439e-04  1.253099e-04
## [5,] -5.307630e-01 -4.177524e-07  4.813564e-02  1.253099e-04  5.759264e+02
##
## Model rank =  2326 / 2326
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring   19.0   13.9    0.94  <2e-16 ***
## s(decimal_date):season_fautumn   19.0   14.3    0.94  <2e-16 ***
## s(SITE_ID.F)                   2286.0 1578.7      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## ============================
## OCAT Family: Odontoceridae | Classes: AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-
999)
## ============================
##
## Family: Ordered Categorical(-1,2.13,6.58)
## Link function: identity
##
## Formula:
## y_ord ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -1.45837    0.03550 -41.086   <2e-16 ***
## season_fautumn  0.27404    0.02997   9.143   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Approximate significance of smooth terms:
##                                    edf Ref.df      F p-value
## s(decimal_date):season_fspring   5.595     19 21.896  <2e-16 ***
## s(decimal_date):season_fautumn   5.589     19 22.239  <2e-16 ***
## s(SITE_ID.F)                  1682.691   2685  3.412  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained = 30.8%
## fREML =  14364  Scale est. = 1         n = 26419


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] 5.025057e-07 5.643168e-08 4.520608e-07 7.345728e-08 1.727339e-04
##
## $hess
##               [,1]          [,2]          [,3]          [,4]         [,5]
## [1,]  1.5618534063  0.1320972006 -0.0090169017  0.0007209585  -0.51543887
## [2,]  0.1320972006  0.1923868732 -0.0008130338 -0.0029257398   0.01607408
## [3,] -0.0090169017 -0.0008130338  1.3810298868 -0.1013921668  -0.39307415
## [4,]  0.0007209585 -0.0029257398 -0.1013921668  0.3685041334  -0.02704101
## [5,] -0.5154388743  0.0160740776 -0.3930741468 -0.0270410120 544.14915485
##
## Model rank =  2726 / 2726
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'     edf k-index p-value
## s(decimal_date):season_fspring   19.00    5.60    0.96   0.010 **
## s(decimal_date):season_fautumn   19.00    5.59    0.96   0.005 **
## s(SITE_ID.F)                   2686.00 1682.69      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## ============================
## OCAT Family: Cordulegastridae | Classes: AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-
999)
## ============================
##
## Family: Ordered Categorical(-1,3.17,8.51)
## Link function: identity
##
## Formula:
## y_ord ~ season_f + s(decimal_date, by = season_f, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -1.71518    0.04934 -34.761   <2e-16 ***
```

```
## season_fautumn  0.02332    0.04600   0.507    0.612
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F p-value
## s(decimal_date):season_fspring  11.284     19 8.768  <2e-16 ***
## s(decimal_date):season_fautumn   6.926     19 8.236  <2e-16 ***
## s(SITE_ID.F)                   880.177   1470 2.630  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Deviance explained = 30.8%
## fREML = 6453.7  Scale est. = 1         n = 14387

##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  1.198239e-07  4.847658e-09  8.748479e-08 -6.716564e-05  1.572276e-05
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.908895e+00 -1.763206e-01 -2.940403e-02 -4.618155e-07 -3.791585e-01
## [2,] -1.763206e-01  2.610001e-01  6.005971e-04 -2.677220e-07  4.387126e-02
## [3,] -2.940403e-02  6.005971e-04  2.301017e+00 -5.499023e-05 -4.269994e-01
## [4,] -4.618155e-07 -2.677220e-07 -5.499023e-05  6.715664e-05  3.964273e-06
## [5,] -3.791585e-01  4.387126e-02 -4.269994e-01  3.964273e-06  2.749002e+02
##
## Model rank =  1511 / 1511
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'     edf k-index p-value
## s(decimal_date):season_fspring   19.00   11.28    0.94  <2e-16 ***
## s(decimal_date):season_fautumn   19.00    6.93    0.94  <2e-16 ***
## s(SITE_ID.F)                   1471.00  880.18      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
```

```r
names(ocat_results) <- families
# ^ Fit OCAT models family-by-family and name the results list for convenience.

# ---- Summary table across families ----
ocat_summary <- do.call(rbind, lapply(ocat_results, function(x) {
  data.frame(
    family   = x$family,                        # family name
    n        = x$n,                             # sample size used
    K        = x$K,                             # number of ordinal classes
    k_time   = x$k_time,                        # basis size for time smooth
    dev_expl = x$dev_expl,                      # deviance explained
    AIC      = x$AIC,                           # AIC
```

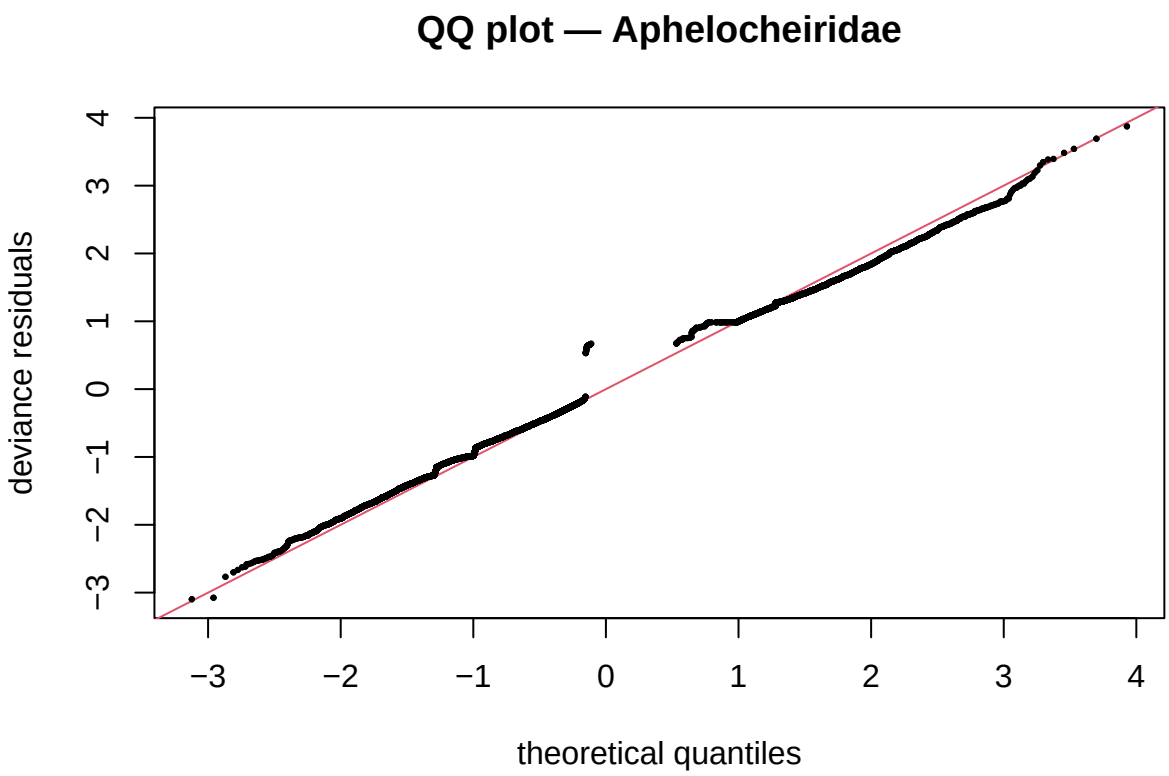
```
    classes  = paste(x$classes, collapse = " | "),  # label set used
    row.names = NULL
  )
}))
print(ocat_summary)
```

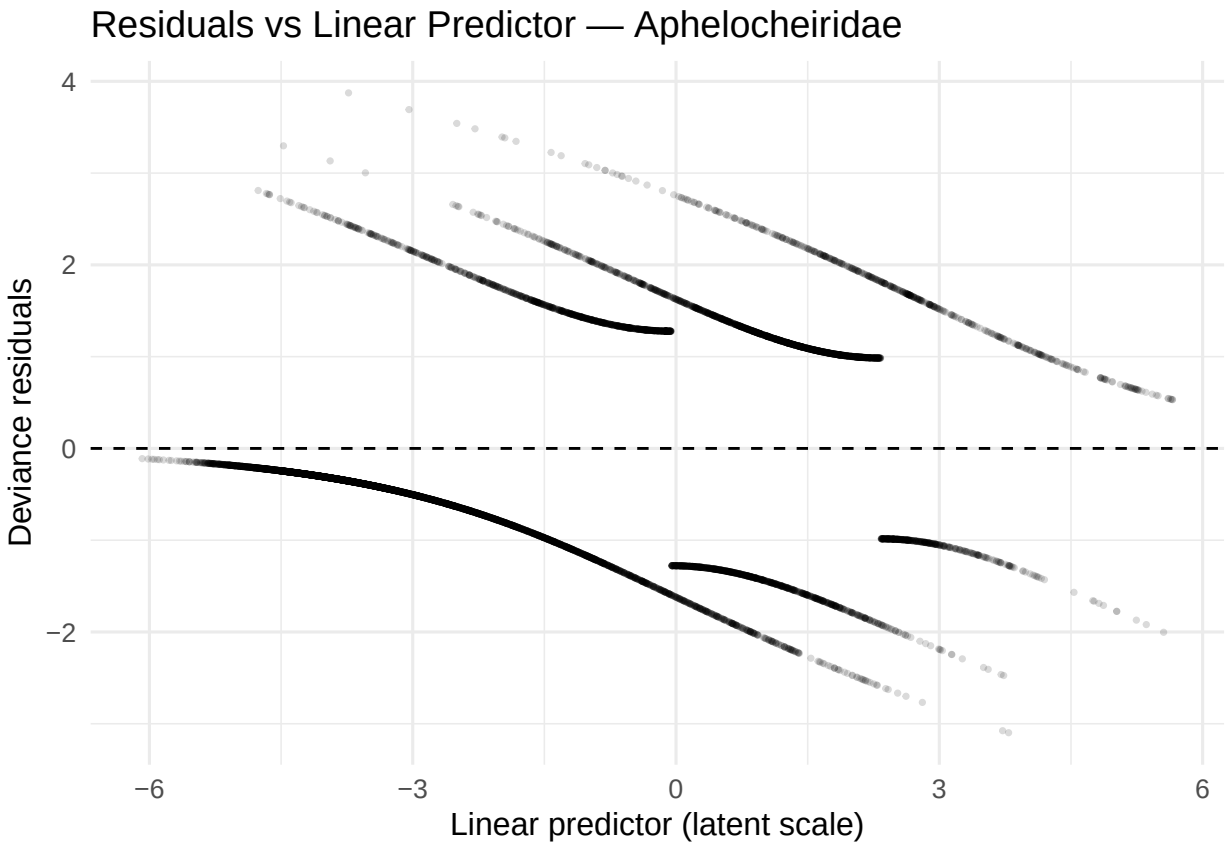
```
##                            family     n K k_time  dev_expl      AIC
## Aphelocheiridae   Aphelocheiridae  9477 4     20 0.4896212 13197.25
## Brachycentridae   Brachycentridae 33477 4     20 0.3087328 37341.18
## Odontoceridae       Odontoceridae 26419 4     20 0.3083054 32038.23
## Cordulegastridae Cordulegastridae 14387 4     20 0.3080516 14279.93
##                                                                classes
## Aphelocheiridae  AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-999)
## Brachycentridae  AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-999)
## Odontoceridae    AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-999)
## Cordulegastridae AB0 (0) | AB1 (1-9) | AB2 (10-99) | AB3 (100-999)
```

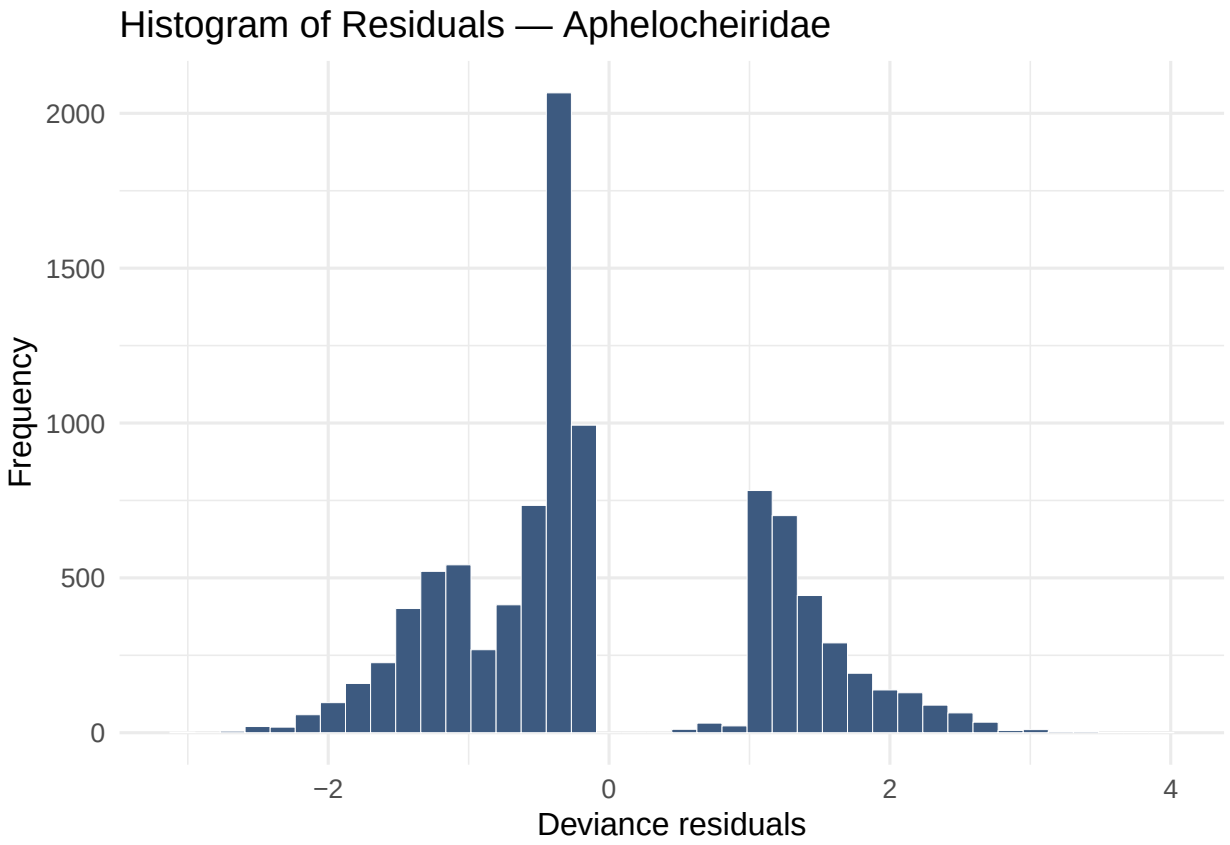
```
# ^ One tidy frame summarizing all fitted families.

# ---- Optional: save models ----
# invisible(lapply(ocat_results, function(x)
```

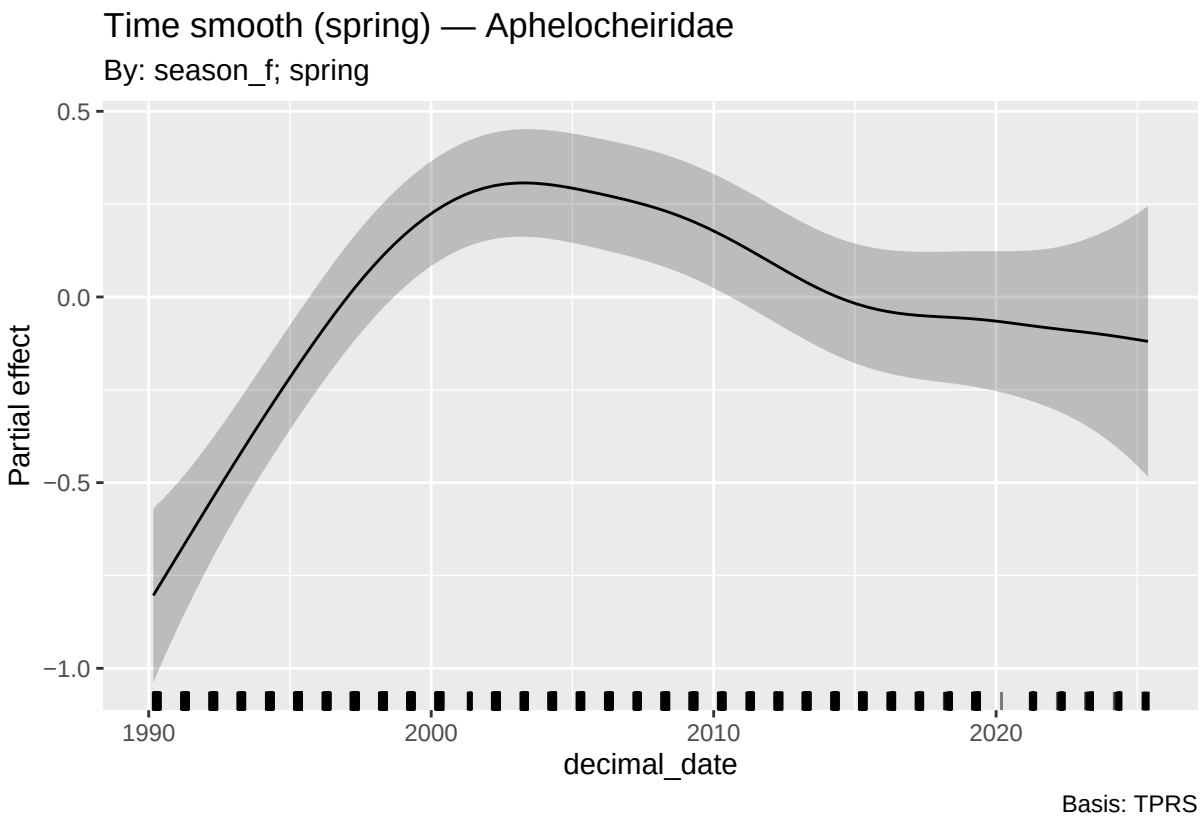
```
# ============================================================
# OCAT Model Diagnostics and Plots
# Expects: ocat_results (each item has $model, $family, $data)
# Purpose:
#   For each fitted ordered-categorical GAMM, produce a compact set
#   of diagnostic visuals:
#     1) QQ plot of residuals (base graphics)
#     2) Residuals vs linear predictor (no smoother)
#     3) Histogram of residuals
#     4) Season-specific partial-effect smooths over time
# ============================================================

plot_ocat_diagnostics_clean <- function(res) {
  m   <- res$model   # fitted mgcv::bam OCAT model
  fam <- res$family  # family name (for titles)
  df  <- res$data    # modelling data (not directly plotted here)

  # --- 1) QQ plot (single, base) ---
  #     Uses mgcv::qq.gam to compare residuals against theoretical quantiles.
  #     rep=0 avoids expensive re-smoothing; small points to de-clutter.
  par(mfrow = c(1,1))
  mgcv::qq.gam(m, rep = 0, pch = 19, cex = 0.3,
               main = paste("QQ plot -", fam))

  # Prepare diagnostics frame once
  # - eta: linear predictor on the latent (link) scale for OCAT
  # - resid: deviance residuals from the model
  eta   <- as.numeric(predict(m, type = "link"))
  resid <- as.numeric(residuals(m, type = "deviance"))
  dd    <- data.frame(eta = eta, resid = resid)
```

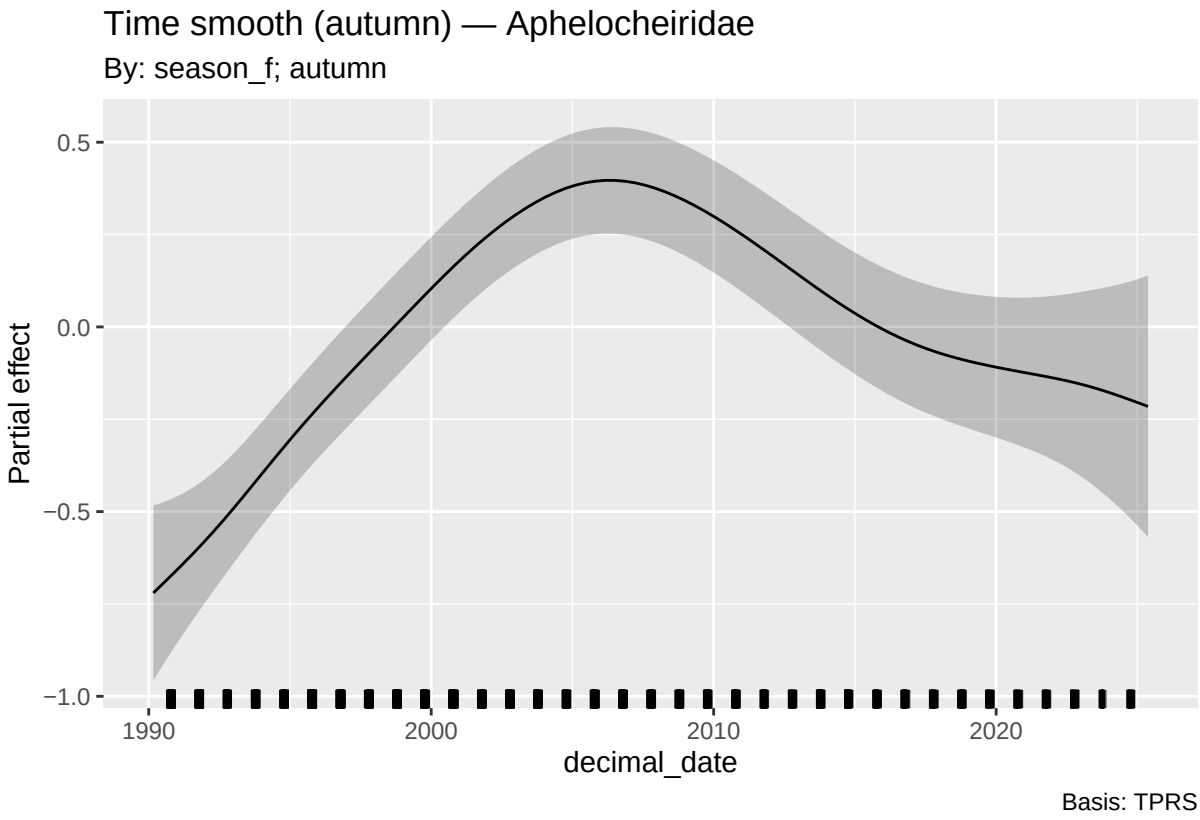
```r
  # --- 2) Residuals vs linear predictor (no smooth) ---
  #     Checks for structure in residuals as a function of the linear predictor.
  #     Only a horizontal reference line at 0 is shown (no smoother).
  p_res_lin <- ggplot(dd, aes(eta, resid)) +
    geom_point(alpha = 0.15, size = 0.6) +
    geom_hline(yintercept = 0, linetype = 2) +
    labs(title = paste("Residuals vs Linear Predictor -", fam),
         x = "Linear predictor (latent scale)", y = "Deviance residuals") +
    theme_minimal(base_size = 12)
  print(p_res_lin)

  # --- 3) Histogram of residuals ---
  #     Quick look at residual distribution; large bins for a smooth view.
  p_hist <- ggplot(dd, aes(resid)) +
    geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
    labs(title = paste("Histogram of Residuals -", fam),
         x = "Deviance residuals", y = "Frequency") +
    theme_minimal(base_size = 12)
  print(p_hist)

  # --- 4) Season-specific time smooths (spring/autumn), printed separately ---
  #     Identify the smooths that involve decimal_date (time), then draw them one by one.
  sm_ids   <- which(grepl("decimal_date", smooths(m)))
  sm_names <- smooths(m)[sm_ids]
  for (j in seq_along(sm_ids)) {
    nm  <- sm_names[j]
    # Give each panel a readable season label if present in the smooth name
    lab <- if (grepl("spring", nm, ignore.case = TRUE)) "spring"
           else if (grepl("autumn", nm, ignore.case = TRUE)) "autumn" else nm
    p_sm <- draw(m, select = sm_ids[j]) +
      ggtitle(paste("Time smooth (", lab, ") - ", fam, sep = ""))
    print(p_sm)
  }
}

# Run for all fitted OCAT models (prints plots for each family)
# - invisible() to avoid printing the lapply return value;
#   each plot is printed as a side effect within the function.
invisible(lapply(ocat_results, plot_ocat_diagnostics_clean))
```

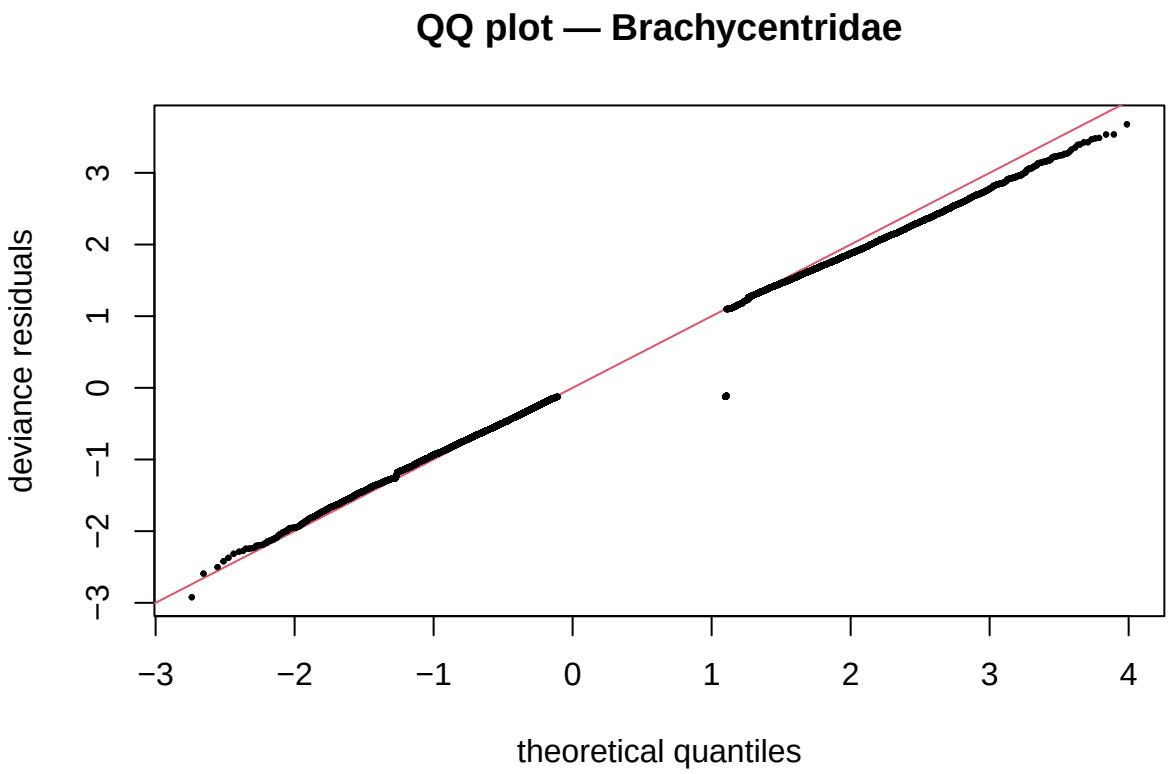

QQ plot — Aphelocheiridae
deviance residuals
theoretical quantiles
−3
−2
−1
0
1
2
3
4

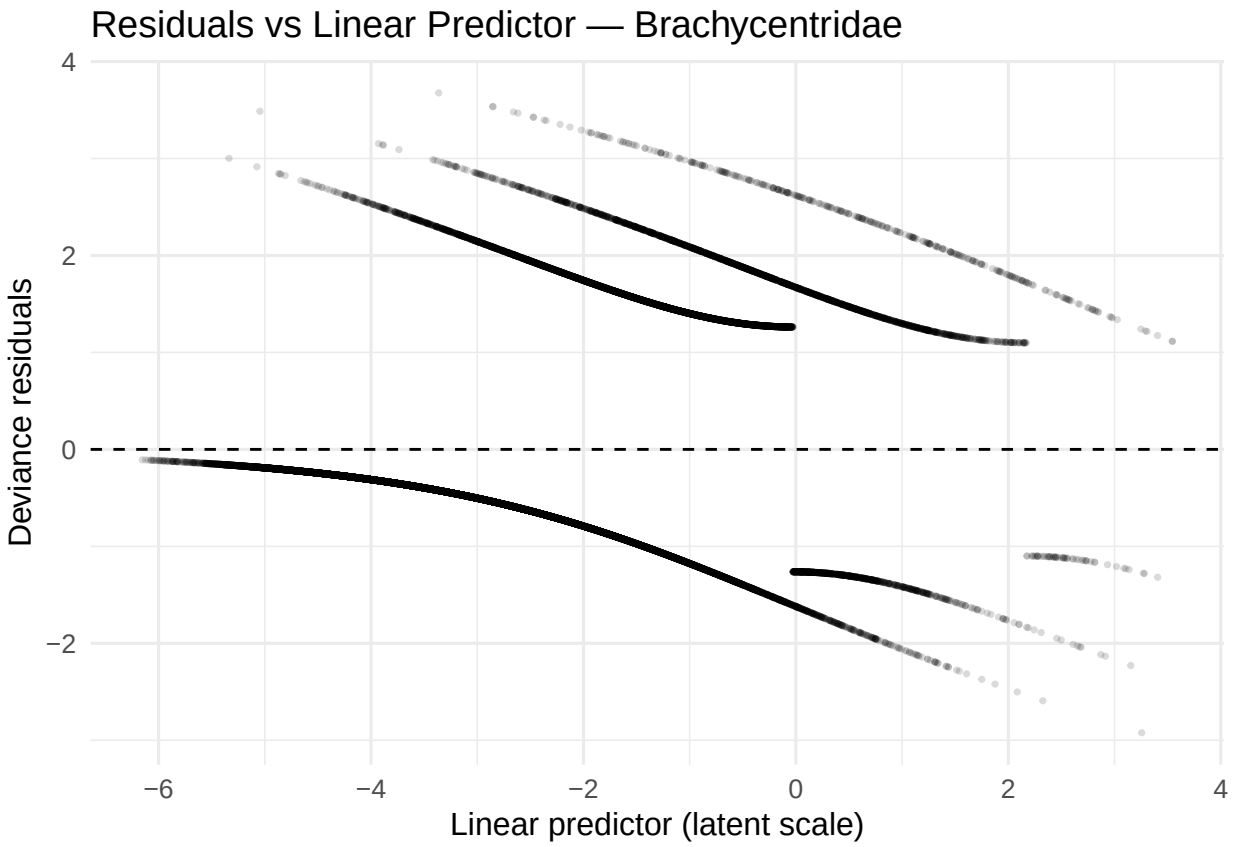

Residuals vs Linear Predictor — Aphelocheiridae
Deviance residuals
4
2
0
−2
−6
−3
0
3
6
Linear predictor (latent scale)

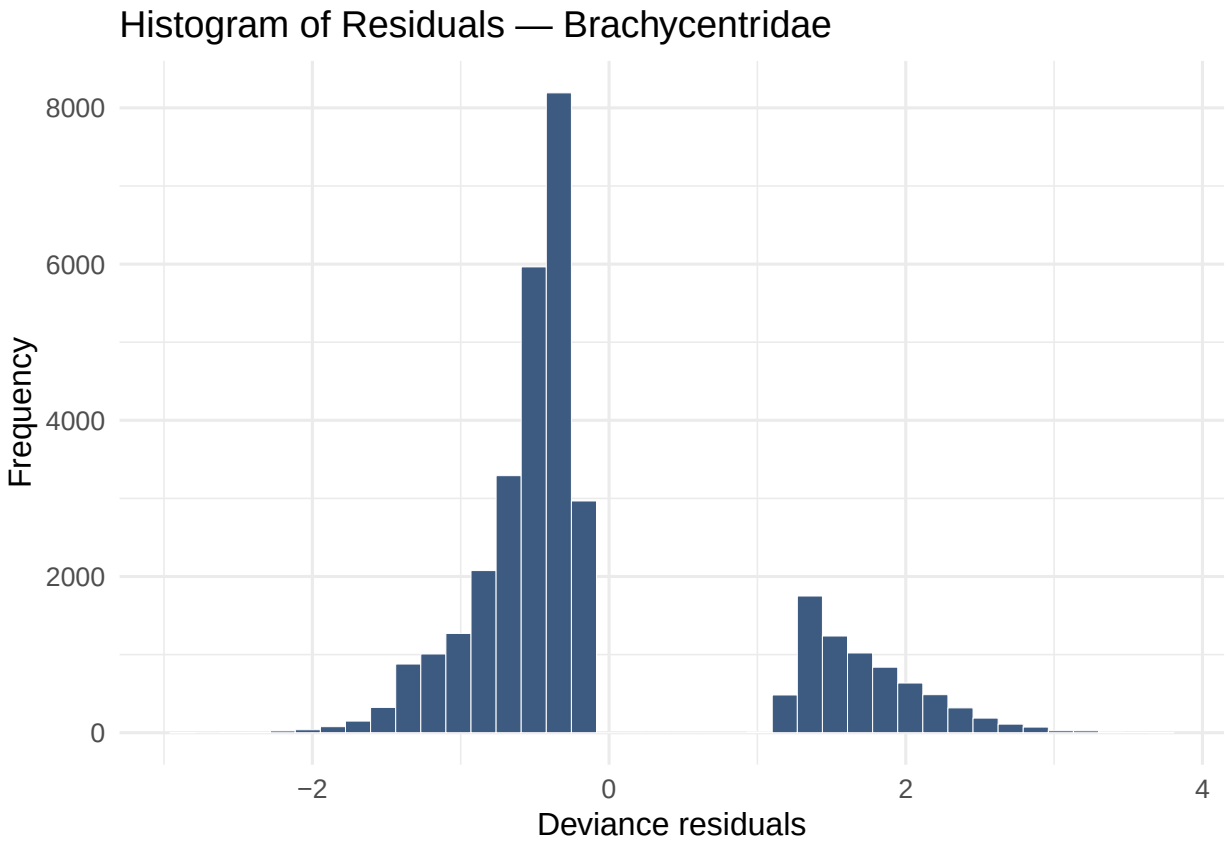

Histogram of Residuals — Aphelocheiridae
Frequency
2000
1500
1000
500
0
−2
0
2
4
Deviance residuals

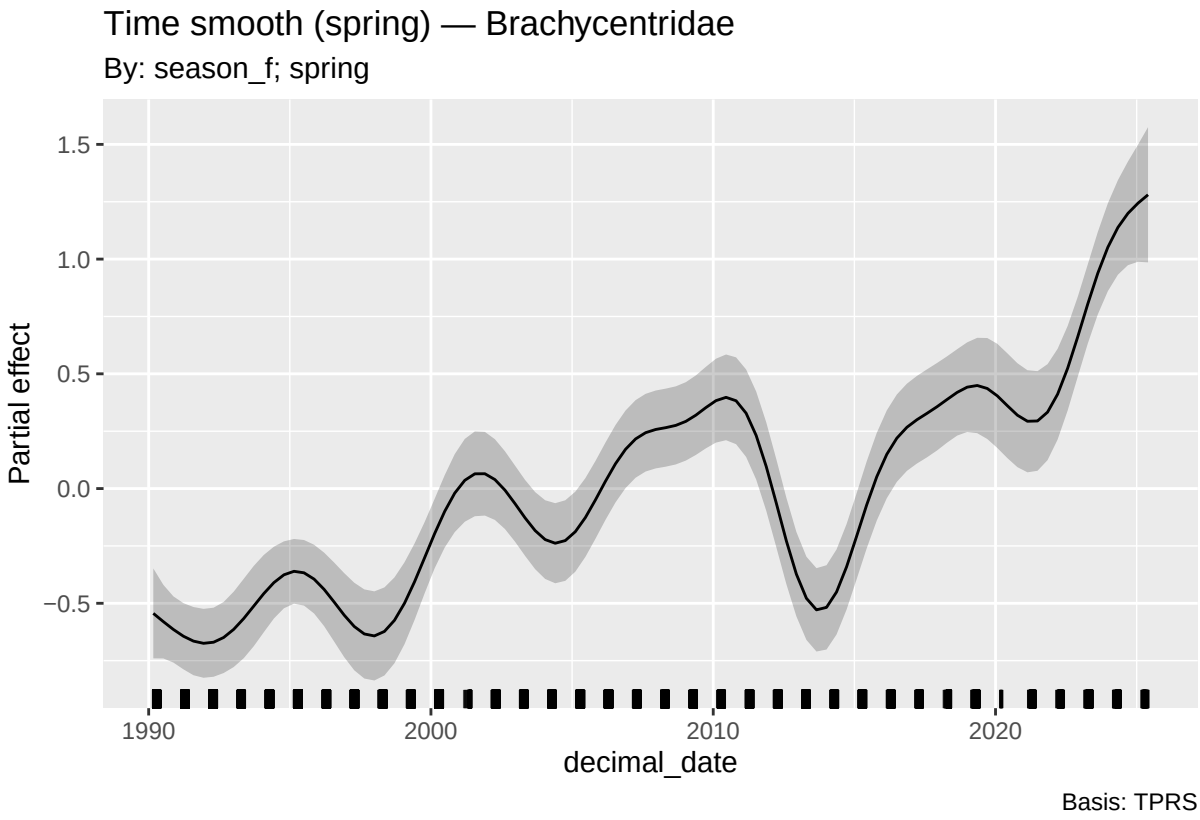

Time smooth (spring) — Aphelocheiridae
By: season_f; spring
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

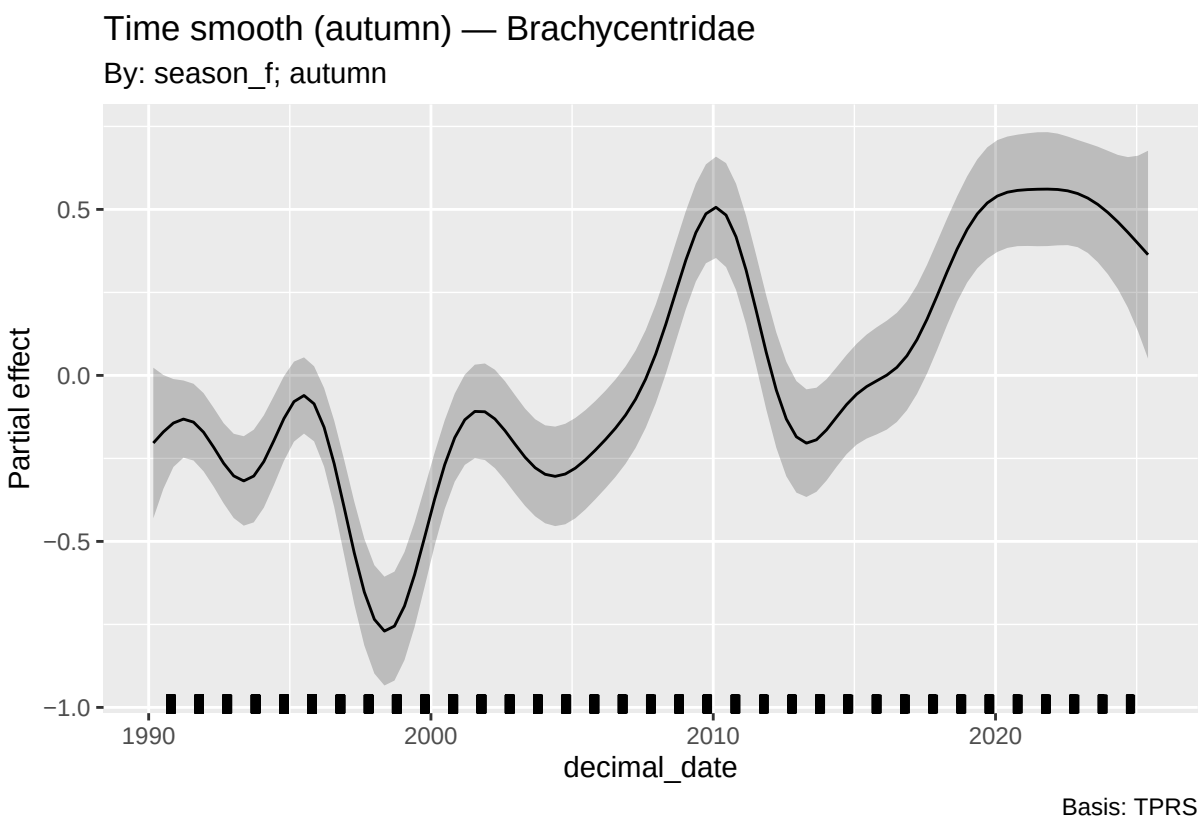

Time smooth (autumn) — Aphelocheiridae
By: season_f; autumn
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

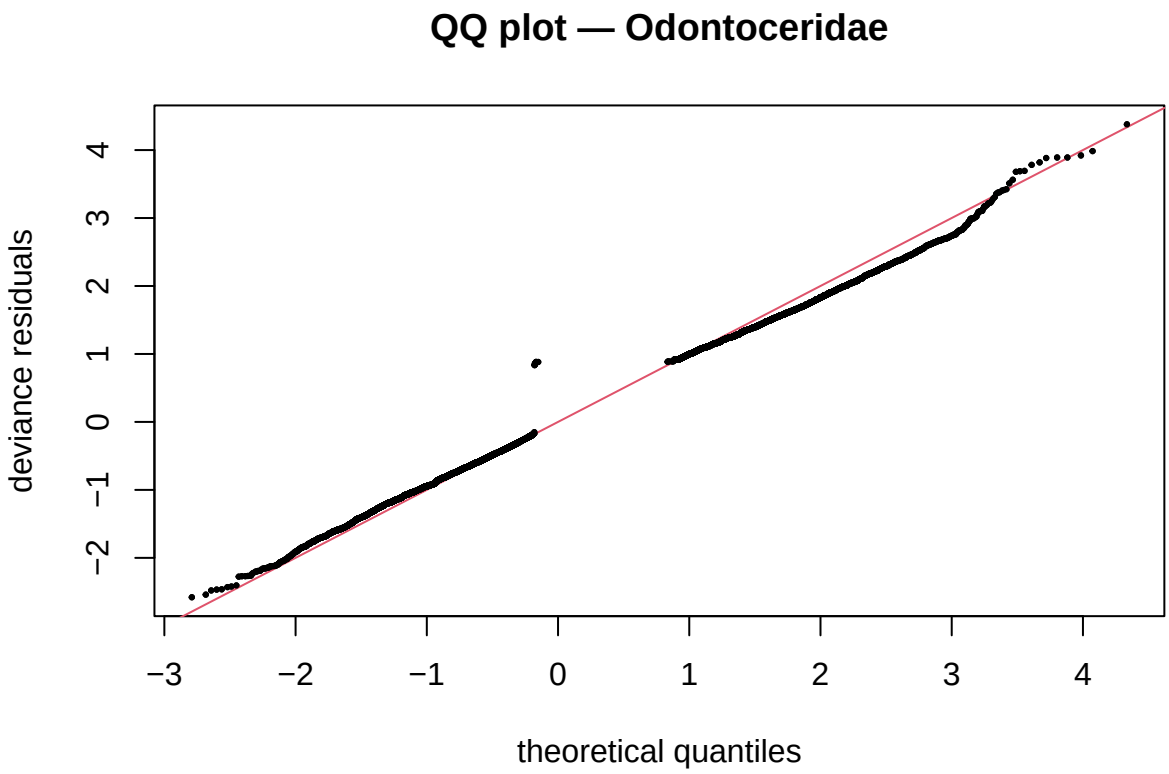

QQ plot — Brachycentridae
deviance residuals
theoretical quantiles
−3
−2
−1
0
1
2
3
4

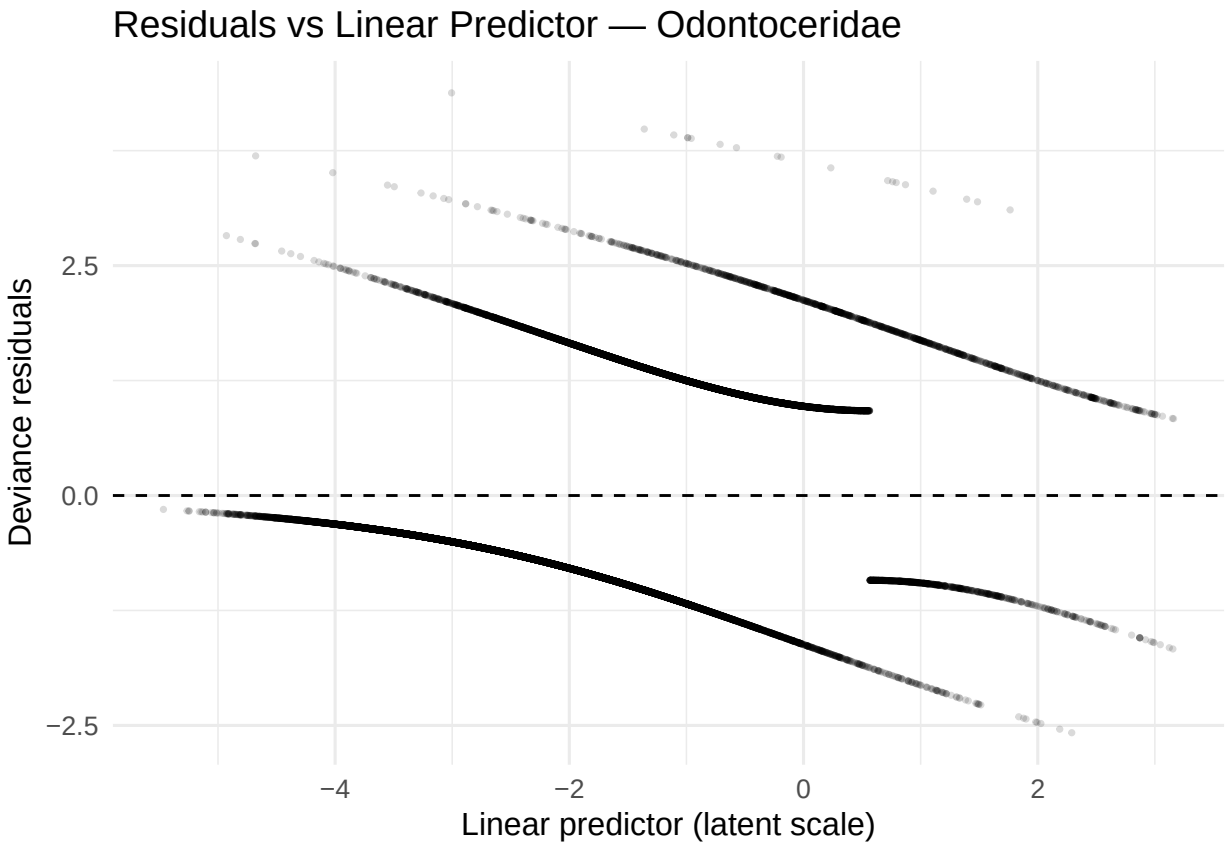

Residuals vs Linear Predictor — Brachycentridae
Deviance residuals
4
2
0
−2
−6
−4
−2
0
2
4
Linear predictor (latent scale)

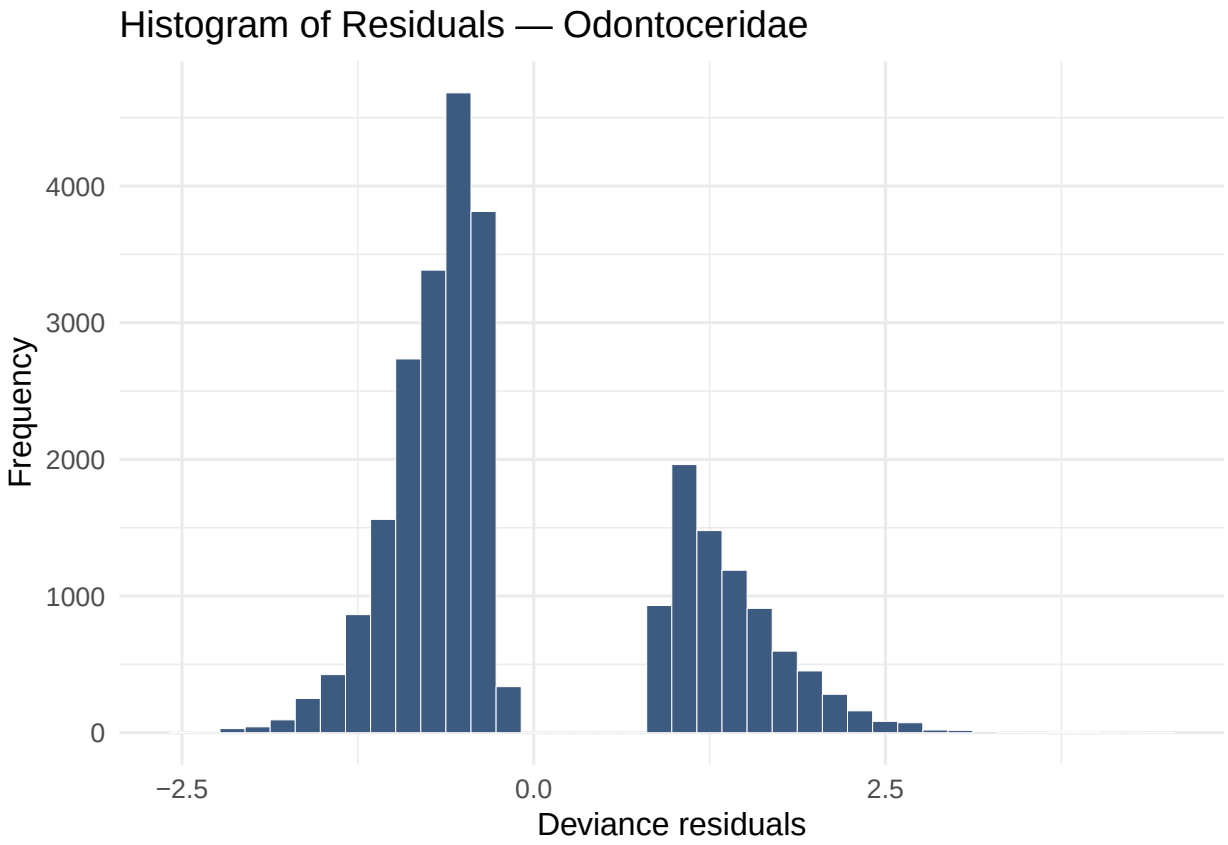

Histogram of Residuals — Brachycentridae
Frequency
8000
6000
4000
2000
0
−2
0
2
4
Deviance residuals

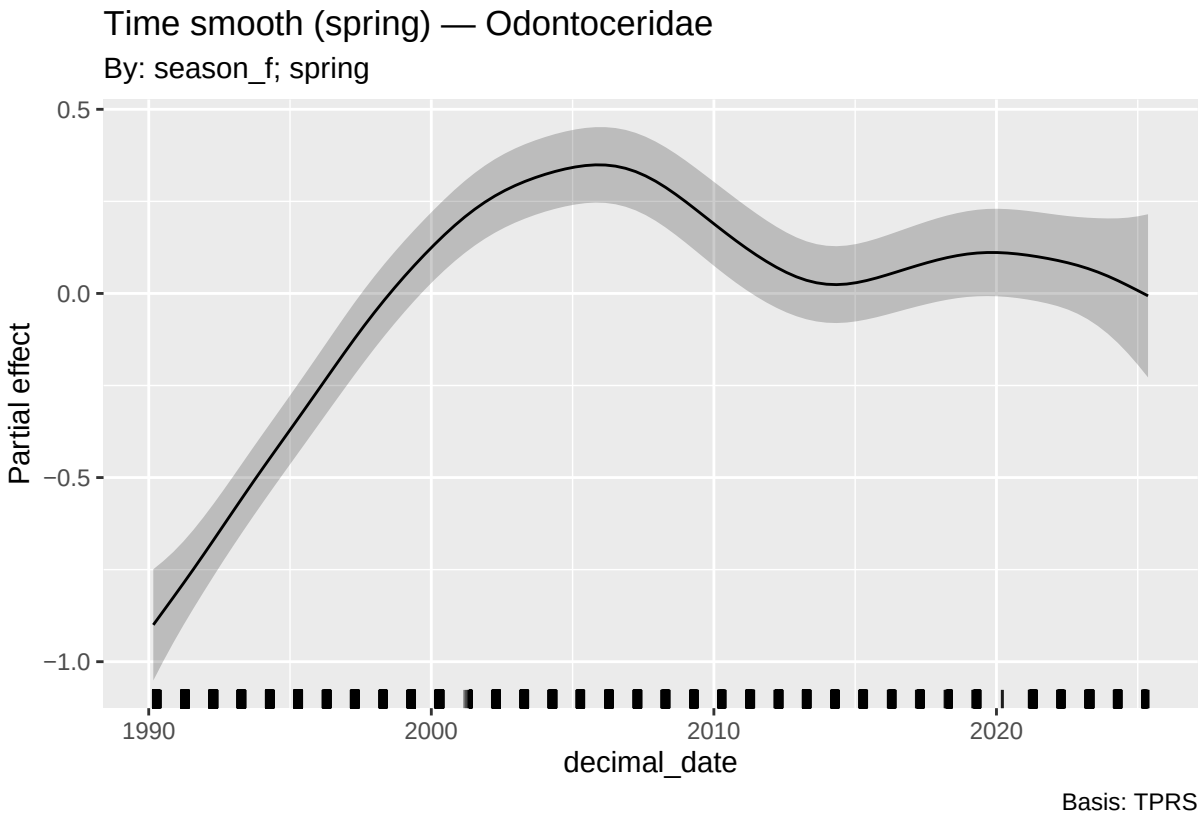

Time smooth (spring) — Brachycentridae
By: season_f; spring
Partial effect
1.5
1.0
0.5
0.0
−0.5
1990
2000
2010
2020
decimal_date
Basis: TPRS

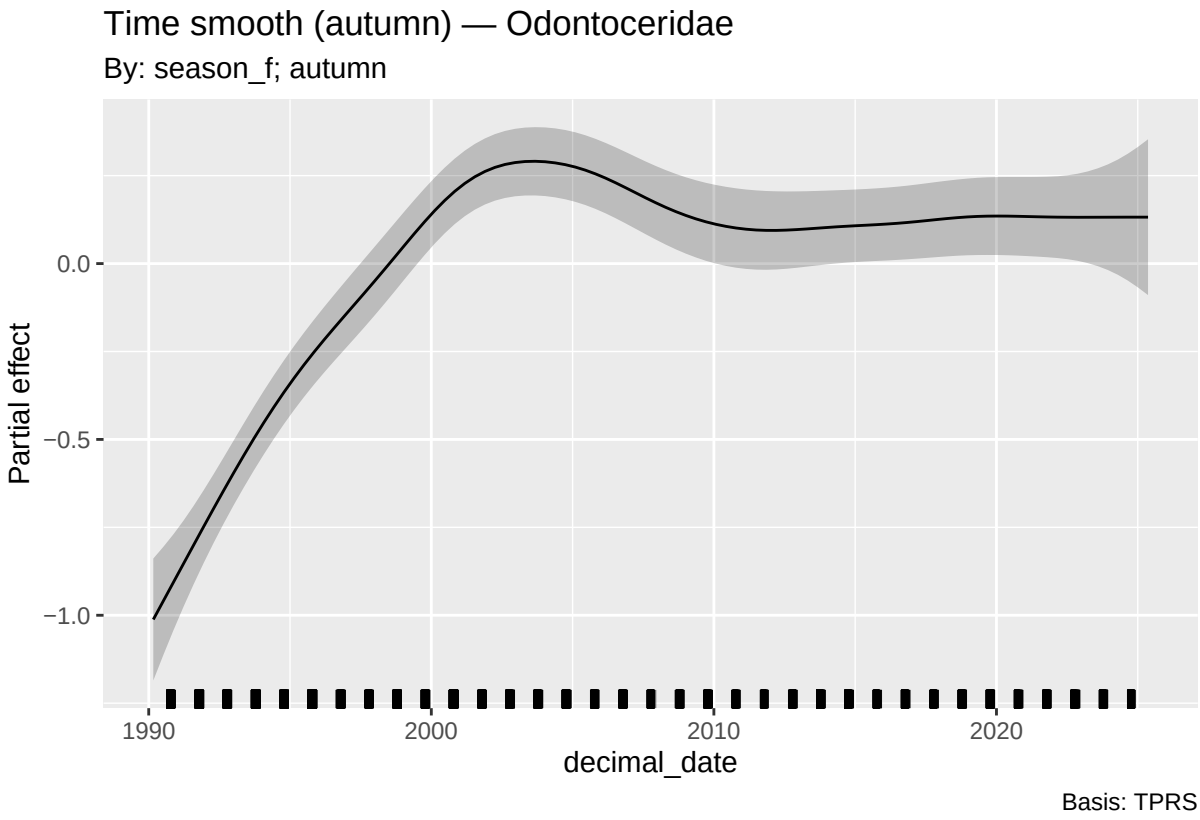

Time smooth (autumn) — Brachycentridae
By: season_f; autumn
Partial effect
0.5
0.0
−0.5
−1.0
1990
2000
2010
2020
decimal_date
Basis: TPRS

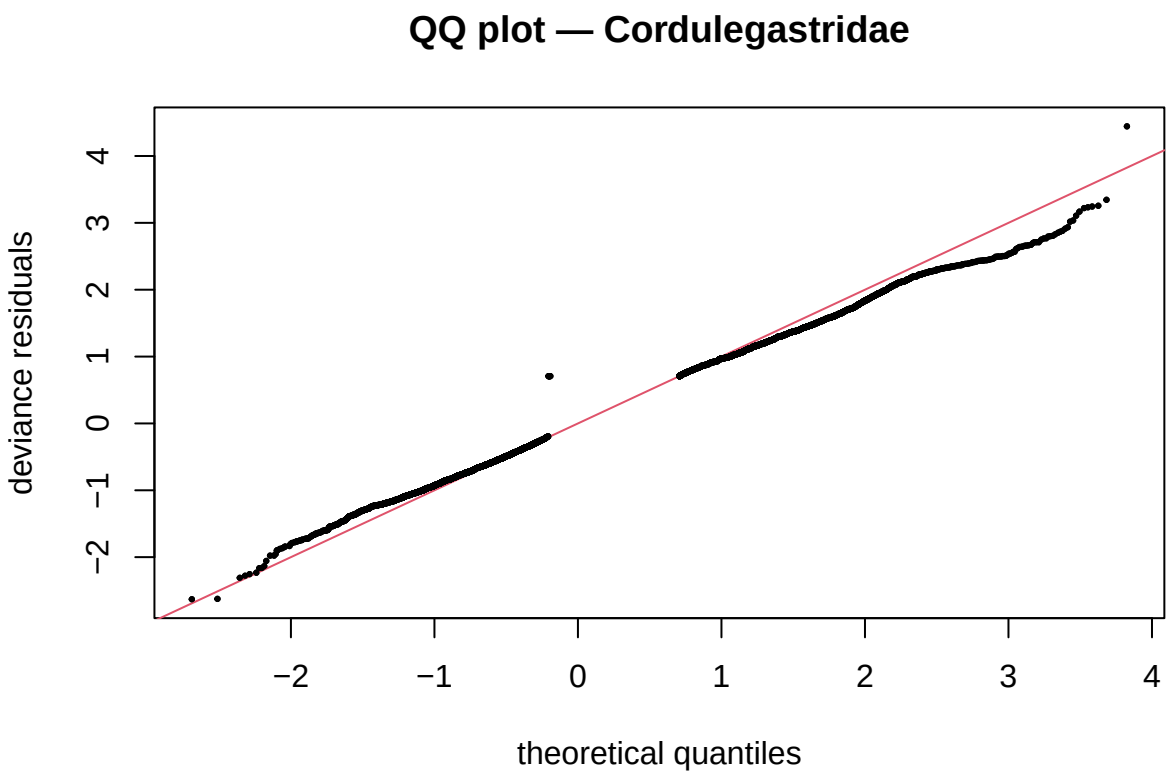

QQ plot — Odontoceridae
deviance residuals
theoretical quantiles
−3
−2
−1
0
1
2
3
4

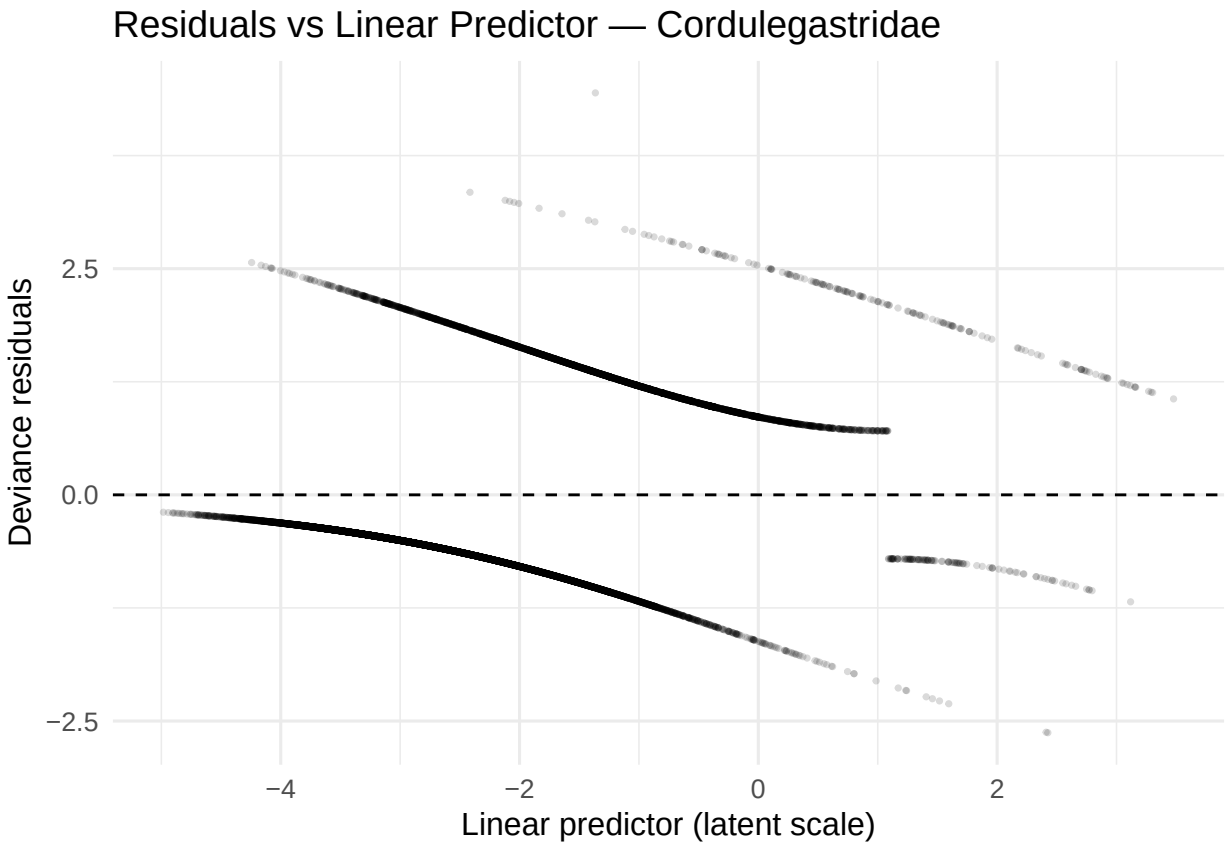

Residuals vs Linear Predictor — Odontoceridae
Deviance residuals
2.5
0.0
−2.5
−4
−2
0
2
Linear predictor (latent scale)

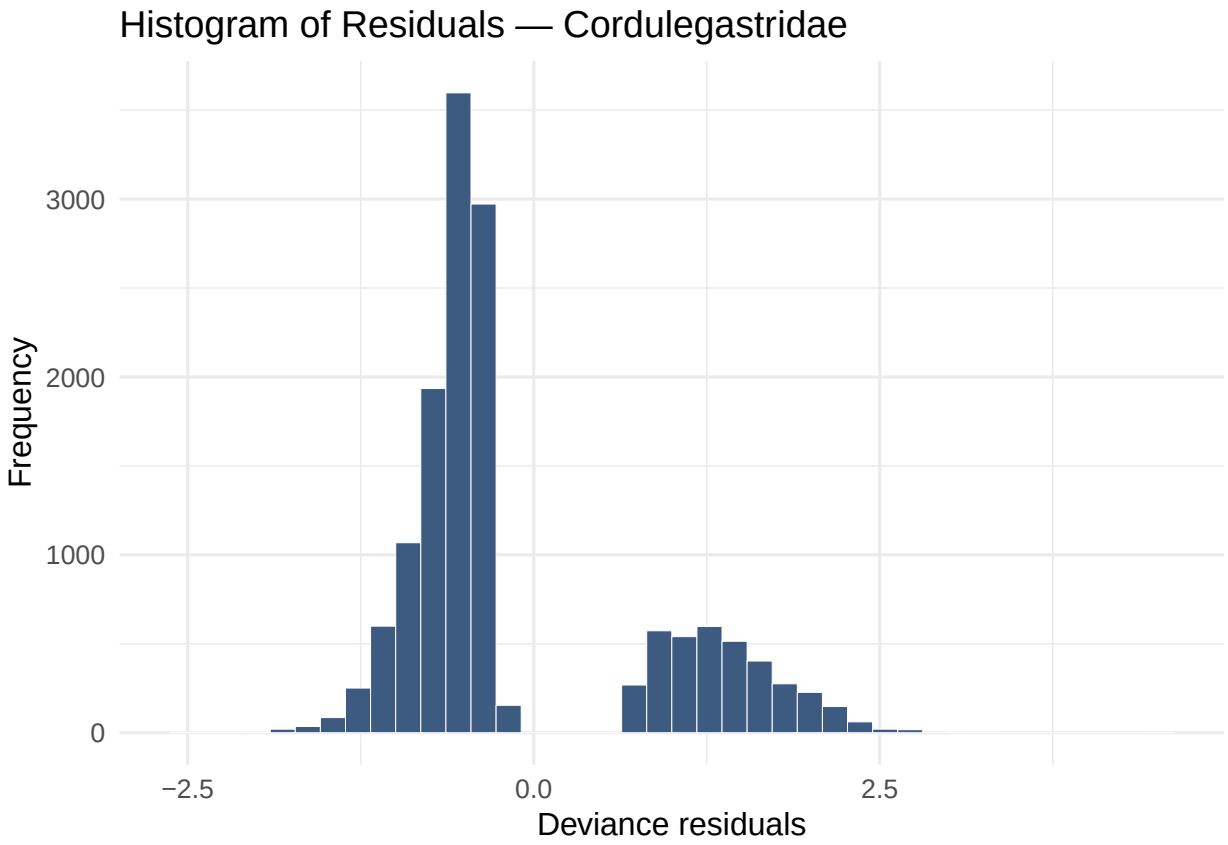

Histogram of Residuals — Odontoceridae
Frequency
4000
3000
2000
1000
0
−2.5
0.0
2.5
Deviance residuals

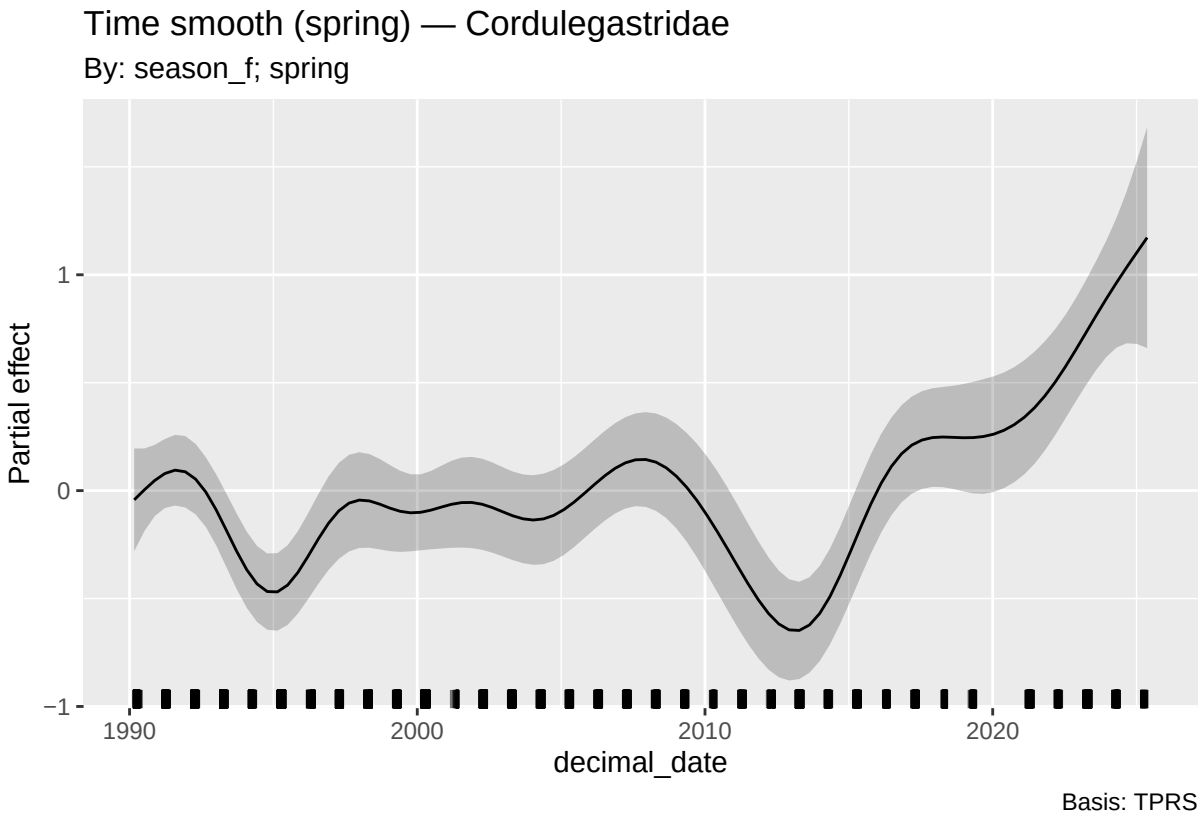

Time smooth (spring) — Odontoceridae
By: season_f; spring
0.5
0.0
−0.5
−1.0
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

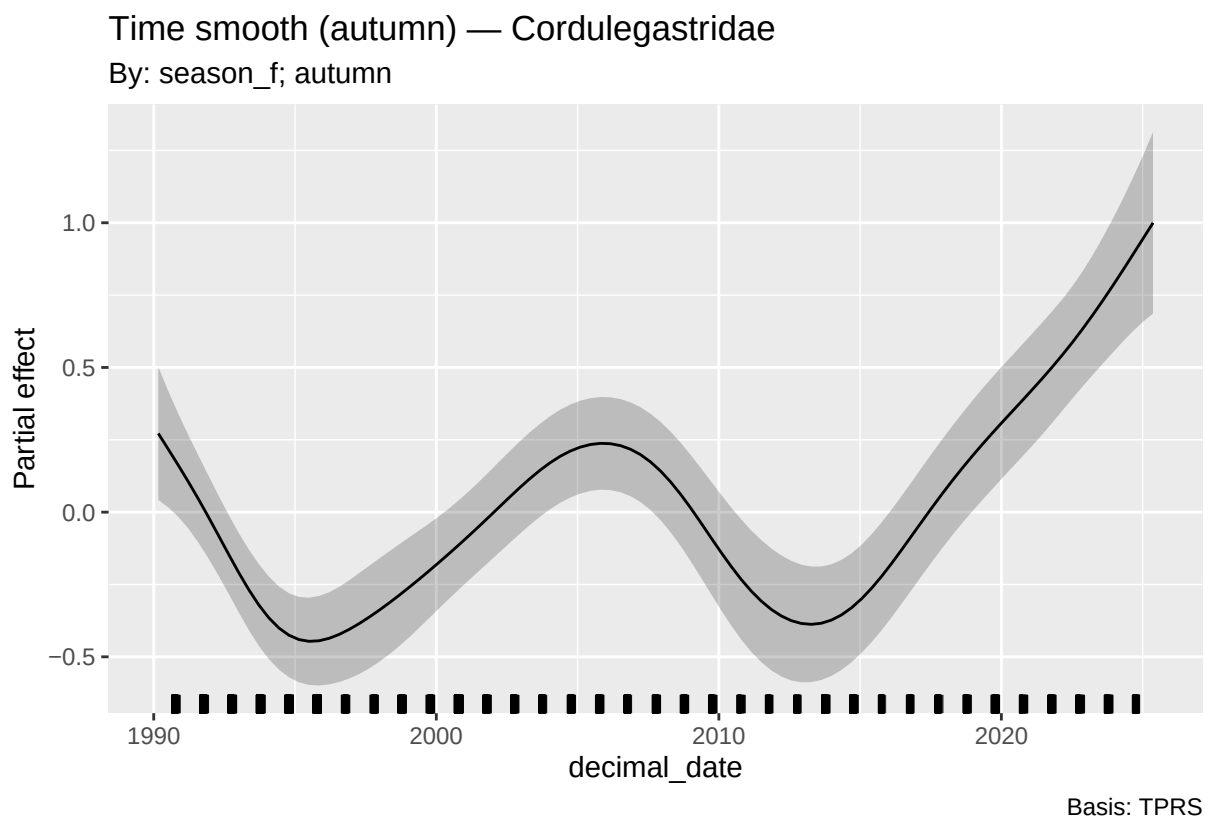

Time smooth (autumn) — Odontoceridae
By: season_f; autumn
Partial effect
0.0
−0.5
−1.0
1990
2000
2010
2020
decimal_date
Basis: TPRS


QQ plot — Cordulegastridae
deviance residuals
theoretical quantiles
−2
−1
0
1
2
3
4


Residuals vs Linear Predictor — Cordulegastridae
Deviance residuals
2.5
0.0
−2.5
−4
−2
0
2
Linear predictor (latent scale)


Histogram of Residuals — Cordulegastridae
Frequency
3000
2000
1000
0
−2.5
0.0
2.5
Deviance residuals


Time smooth (spring) — Cordulegastridae
By: season_f; spring
Partial effect
1
0
−1
1990
2000
2010
2020
decimal_date
Basis: TPRS



#4.3 CENSORED POISSON GAMM MODEL

```
# ================================================
# CENSORED POISSON (cpois) - one family
# Goal:
#   Fit an interval-censored Poisson GAMM to numeric (count) data
#   using 1-significant-figure censoring bounds and site random effects.
# Inputs expected:
#   - models_data (from Phase 3), containing per-family tables
#   - For the chosen family: models_data[[fam_name]]$pa with columns:
#       SITE_ID, SAMPLE_ID, SAMPLE_DATE, TOTAL_NUMBER, data_type,
#       season_f (spring/autumn; if absent it is built below),
#       and SAMPLE_METHOD (to optionally filter to S3PO)
# Notes:
#   - Code *assumes* numeric samples are flagged as data_type == "count".
#   - Bounds follow 1-s.f. rounding guidance (non-integer limits).
# ================================================

# ---- pick a family ----
# Choose one of your focus families present in models_data.
fam_name <- "Aphelocheiridae"

# ---- helper: half-up rounding & 1-sf interval bounds ----
# round_half_up(): base "half up" rounding (5 → up) at given digit place.
round_half_up <- function(x, digits = 0) {
  s <- sign(x); x <- abs(x) * 10^digits
```

```r
  s * floor(x + 0.5) / 10^digits
}

# bounds_1sf(): produce lower/upper censoring bounds implied by 1-s.f. rounding.
#   - For 0        → [-0.5, 0.5]
#   - For 1..9     → [x-0.5, x+0.5]
#   - For >= 10    → centre on the 1-s.f. rounded value with half-step width,
#                    minus 0.5 to align Poisson mass at integers (e.g. 20 → [14.5, 24.5]).
bounds_1sf <- function(v) {
  # vectorised container
  out <- matrix(NA_real_, nrow = length(v), ncol = 2,
                dimnames = list(NULL, c("lower","upper")))
  # exact zeros
  z <- v == 0L
  out[z,] <- cbind(-0.5, 0.5)
  # exact 1..9
  s <- v >= 1L & v <= 9L
  out[s,] <- cbind(v[s] - 0.5, v[s] + 0.5)
  # 10+
  b <- v >= 10L
  if (any(b)) {
    k     <- floor(log10(v[b]))                    # order of magnitude
    step  <- 10^k                                  # 1-s.f. step size (…, 1, 10, 100, …)
    s1sf  <- round_half_up(v[b], digits = -k)      # 1-s.f. rounded centre (5s round up)
    lower <- s1sf - step/2 - 0.5                   # lower non-integer bound
    upper <- s1sf + step/2 - 0.5                   # upper non-integer bound
    out[b,] <- cbind(lower, upper)
  }
  out
}

# ---- build modelling data from your Phase 3 objects ----
# Start from the presence/absence table but keep only numeric-count samples,
# optionally restrict to S3PO (EA recommendation), and ensure factors exist.
cp_df <- models_data[[fam_name]]$pa %>%
  filter(data_type == "count") %>%                 # only numeric-count samples
  filter(SAMPLE_METHOD == "S3PO") %>%              # EA-recommended (toggle here if needed)
  mutate(
    SITE_ID.F = if (!"SITE_ID.F" %in% names(.)) factor(SITE_ID) else SITE_ID.F,  # random-effect factor
    # safety: build season_f if absent (should already exist from preprocessing)
    season_f  = if (!"season_f" %in% names(.))
                  factor(if_else(month(SAMPLE_DATE) %in% 3:5, "spring", "autumn"),
                         levels = c("spring","autumn"))
                else season_f
  )

# Compute non-integer censoring bounds for each observation using the 1-s.f. rule.
B <- bounds_1sf(cp_df$TOTAL_NUMBER)
cp_df$lower <- B[, "lower"]
cp_df$upper <- B[, "upper"]

# Sensible k for the time smooth: scale with span of years, but capped.
k_time <- {
```

```r
  ny <- dplyr::n_distinct(lubridate::year(cp_df$SAMPLE_DATE))
  min(60, max(35, round(0.6 * ny)))
}

# ---- fit censored Poisson GAMM ----
# Model: season intercepts + season-varying smooth of decimal time + site RE.
# Family cpois() handles interval-censored integer counts via (lower, upper).
cpois_model <- bam(
  cbind(lower, upper) ~
    season_f +
    s(decimal_date, by = season_f, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = cpois(),         # interval-censored Poisson likelihood
  data     = cp_df,
  method   = "fREML",         # fast REML estimation
  discrete = TRUE,            # large-data speed-up
  select   = TRUE,            # shrink unnecessary wiggle
  gamma    = 1.2              # mild extra penalty (guard against overfit)
)

# ---- minimal reporting ----
# Print model summary (parametric + smooth terms, deviance explained, etc.).
print(summary(cpois_model))
```

```
##
## Family: cpois
## Link function: log
##
## Formula:
## cbind(lower, upper) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.6207     0.2065   3.006  0.00273 **
## season_fautumn   0.3996     0.2374   1.683  0.09281 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df        F p-value
## s(decimal_date):season_fspring  30.62     34 10884.67  <2e-16 ***
## s(decimal_date):season_fautumn  29.84     34  9338.99  <2e-16 ***
## s(SITE_ID.F)                   269.04    313    53.26  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.698   Deviance explained = 85.8%
## fREML = 4664.9  Scale est. = 1         n = 1089
```

```r
# Run gam.check with plots diverted to a temp PDF (so only text prints here),
# then delete the temp file. If gam.check succeeded, echo its (text) output.
tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
```

```r
gc_out <- try(gam.check(cpois_model, rep = 0), silent = TRUE)
```

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  1.297473e-07 -3.229102e-06 -6.752067e-08 -3.635216e-06  2.696238e-05
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.116194e+01 -2.332011e-06  2.348726e-01  1.715381e-08 -2.792819e-01
## [2,] -2.332011e-06  3.229081e-06  8.622013e-09 -1.059584e-15 -6.232250e-10
## [3,]  2.348726e-01  8.622013e-09  1.048827e+01 -2.557398e-06 -2.602784e-01
## [4,]  1.715381e-08 -1.059584e-15 -2.557398e-06  3.635190e-06  5.130067e-08
## [5,] -2.792819e-01 -6.232245e-10 -2.602784e-01  5.130067e-08  1.134983e+02
##
## Model rank =  384 / 384
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'   edf k-index p-value
## s(decimal_date):season_fspring  34.0  30.6    0.96    0.17
## s(decimal_date):season_fautumn  34.0  29.8    0.96    0.17
## s(SITE_ID.F)                   314.0 269.0      NA      NA
```

```r
dev.off(); unlink(tmp)
```

```
## pdf
##   2
```

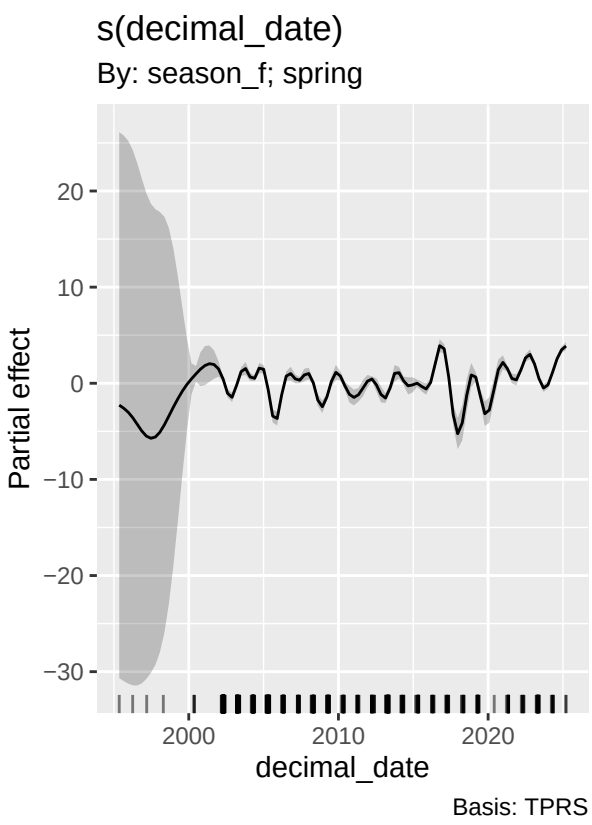
```r
if (!inherits(gc_out, "try-error")) print(gc_out)
```

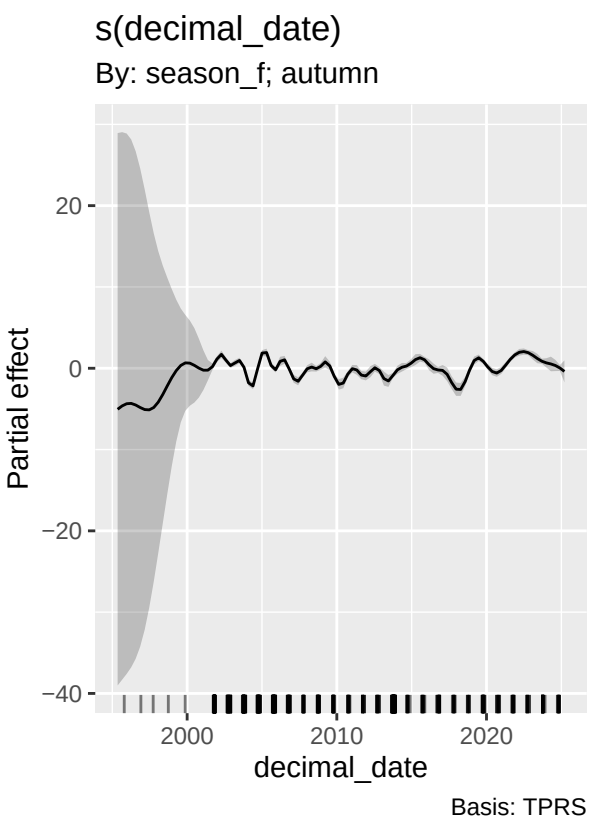
```
## $mfrow
## [1] 2 2
```

```r
# Optional quick metric line:
cat(sprintf("Deviance explained: %.1f%% | AIC: %.1f | N: %d\n",
            100*summary(cpois_model)$dev.expl, AIC(cpois_model), nobs(cpois_model)))
```

```
## Deviance explained: 85.8% | AIC: 9738.0 | N: 1089
```

```r
# Plot season-specific time smooths (spring/autumn) for quick visual inspection.
gratia::draw(cpois_model, select = which(grepl("decimal_date", smooths(cpois_model))))
```





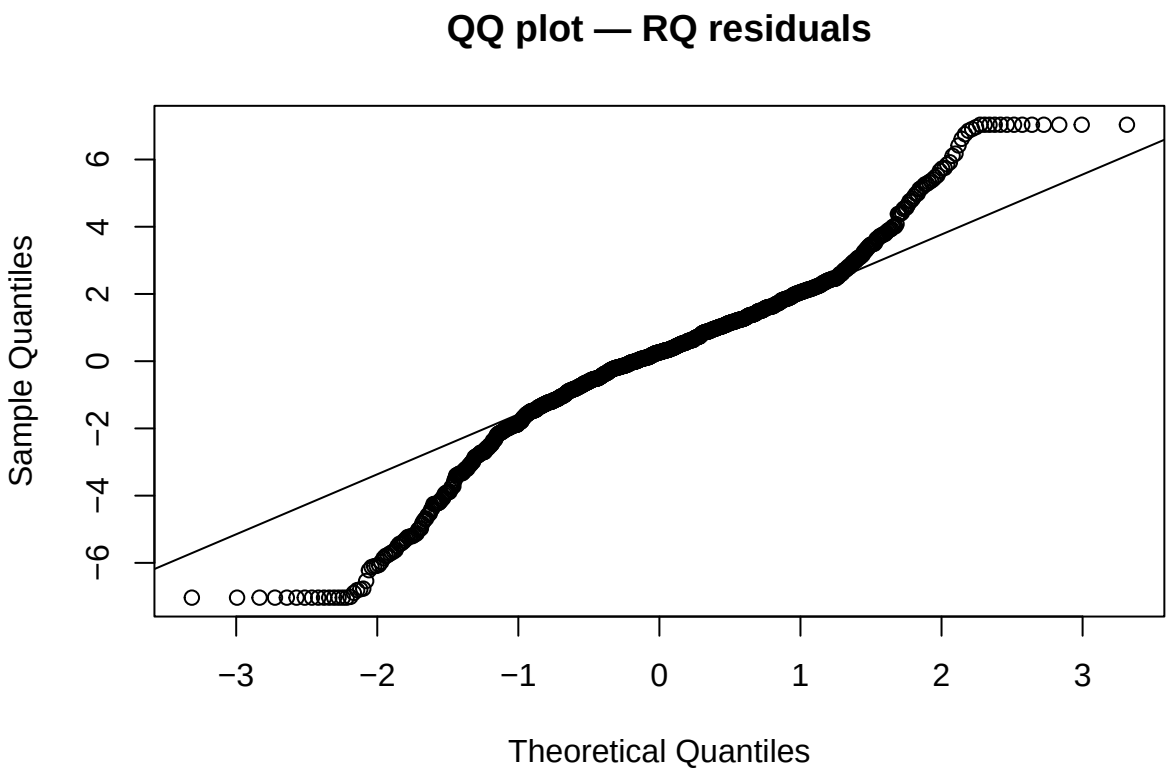

```r
# ============================================================
# CPOIS Model diagnostics - robust & compact
# Requires: cpois_model (bam fit), cp_df with lower/upper
# Purpose:
#   - Check basis adequacy (k-index), concurvity, and dispersion.
#   - Use randomized quantile residuals (RQR) tailored for *censored Poisson*.
#   - Provide quick visual checks: QQ, residual-vs-fitted, histogram, ACF.
#   - Report simple overdispersion proxies and headline fit metrics.
# Notes:
#   - This code assumes you already fitted `cpois_model` and built `cp_df`
#     containing numeric bounds `lower`/`upper` (non-integers OK).
# ============================================================

# --- Randomised quantile residuals (robust) ---
# For each observation i with integer bounds [li, ui], compute:
#   U_i ~ Uniform(F(li-1; _i), F(ui; _i)), where F is Poisson CDF with mean _i.
# Then map U_i through the standard normal inverse CDF to get N(0,1) under the model.
# * This properly accounts for censoring and discreteness.
cpois_rqr <- function(fit, lower, upper, eps = 1e-12, seed = 123) {
  n  <- length(lower)
  mu <- as.numeric(pmax(fitted(fit, type = "response"), eps))  # fitted  (guard tiny)

  # integer bounds; enforce ui >= li and li >= 0
  li <- pmax(0L, ceiling(lower))   # move up to nearest integer
  ui <- floor(upper)               # move down to nearest integer
  ui <- pmax(ui, li)               # ensure non-empty interval
```

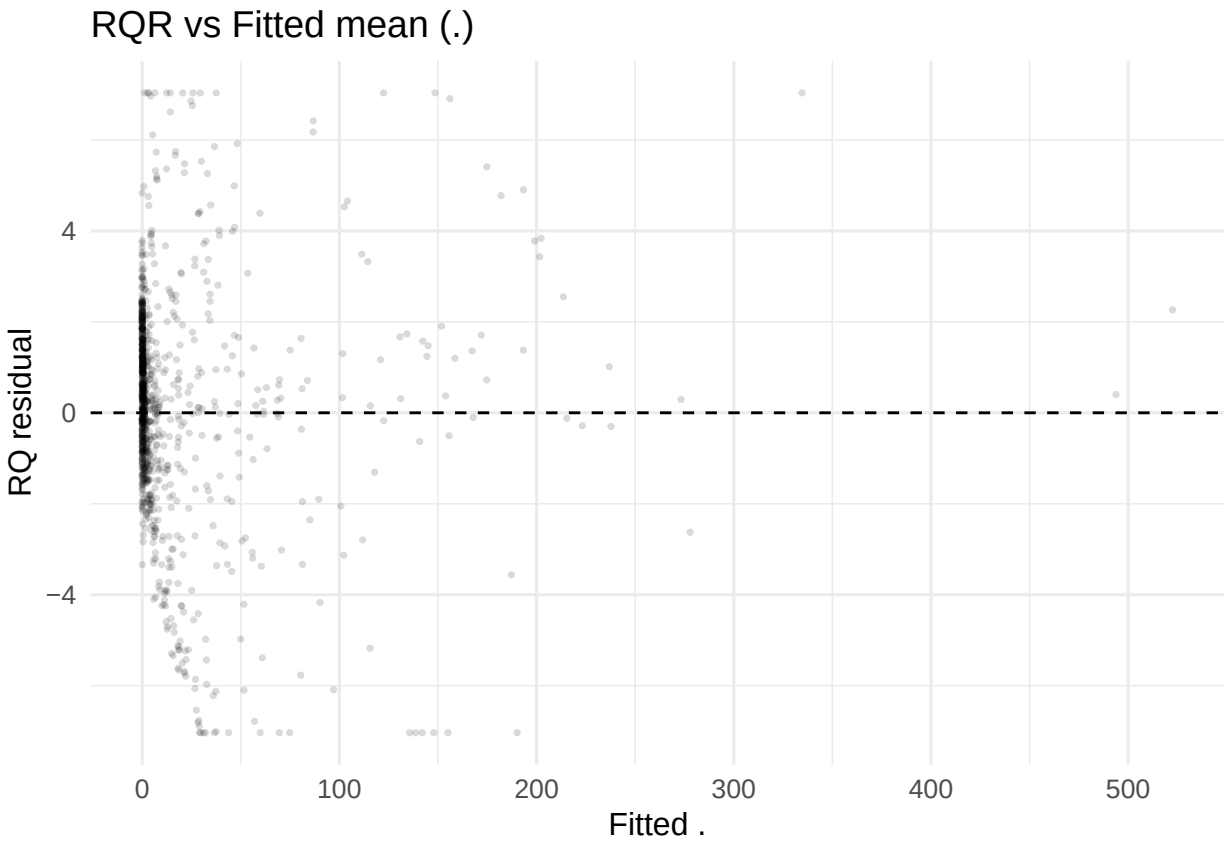
```
  Flo <- ppois(li - 1L, mu)        # F(lo-1) : mass strictly below li
  Fup <- ppois(ui,      mu)        # F(up)   : mass up to ui

  w <- pmax(Fup - Flo, 0)          # interval probability (non-negative)
  set.seed(seed)
  u <- Flo + w * runif(n)          # randomized PIT within the interval
  u <- pmin(pmax(u, eps), 1 - eps) # clamp away from 0/1 for stable qnorm

  qnorm(u)                         # RQR ~ N(0,1) if model well-specified
}

# --- Truncated Poisson conditional mean E[Y | li <= Y <= ui] ---
# Deterministic imputation used only for a Pearson-like dispersion proxy.
# If the interval probability is extremely small, fall back to a clamped round().
tpois_mean <- function(li, ui, lambda, eps = 1e-12) {
  li <- as.integer(pmax(0L, li))
  ui <- as.integer(pmax(li, ui))           # ensure ui >= li
  # support
  xs <- li:ui
  # handle single-point interval quickly
  if (length(xs) == 1L) return(xs)
  # pmf on support
  pr <- dpois(xs, lambda)
  s  <- sum(pr)
  if (!is.finite(s) || s < eps) {
    # fallback: nearest integer to lambda within [li, ui]
    return(max(li, min(ui, round(lambda))))
  }
  sum(xs * pr) / s
}

diagnose_cpois <- function(fit, data) {
  # 1) Basis/smooth adequacy (suppress plots)
  #    - gam.check text includes k-index (should be ~1) and EDF vs k checks.
  tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
  gc_out <- try(gam.check(fit, rep = 0), silent = TRUE)
  dev.off(); unlink(tmp)
  if (!inherits(gc_out, "try-error")) print(gc_out)

  # 2) Concurvity (safe)
  #    - High concurvity suggests redundant smooth structure (instability).
  con <- try(concurvity(fit, full = TRUE), silent = TRUE)
  if (!inherits(con, "try-error")) {
    cat("\nConcurvity (rounded):\n")
    if (is.list(con) && !is.null(con$estimate)) print(round(con$estimate, 3)) else print(con)
  }

  # 3) Randomised quantile residuals
  #    - Correct for censoring; should be ~N(0,1) if the model is well specified.
  rqr <- cpois_rqr(fit, data$lower, data$upper)

  # QQ plot for RQR
  par(mfrow = c(1,1))
```

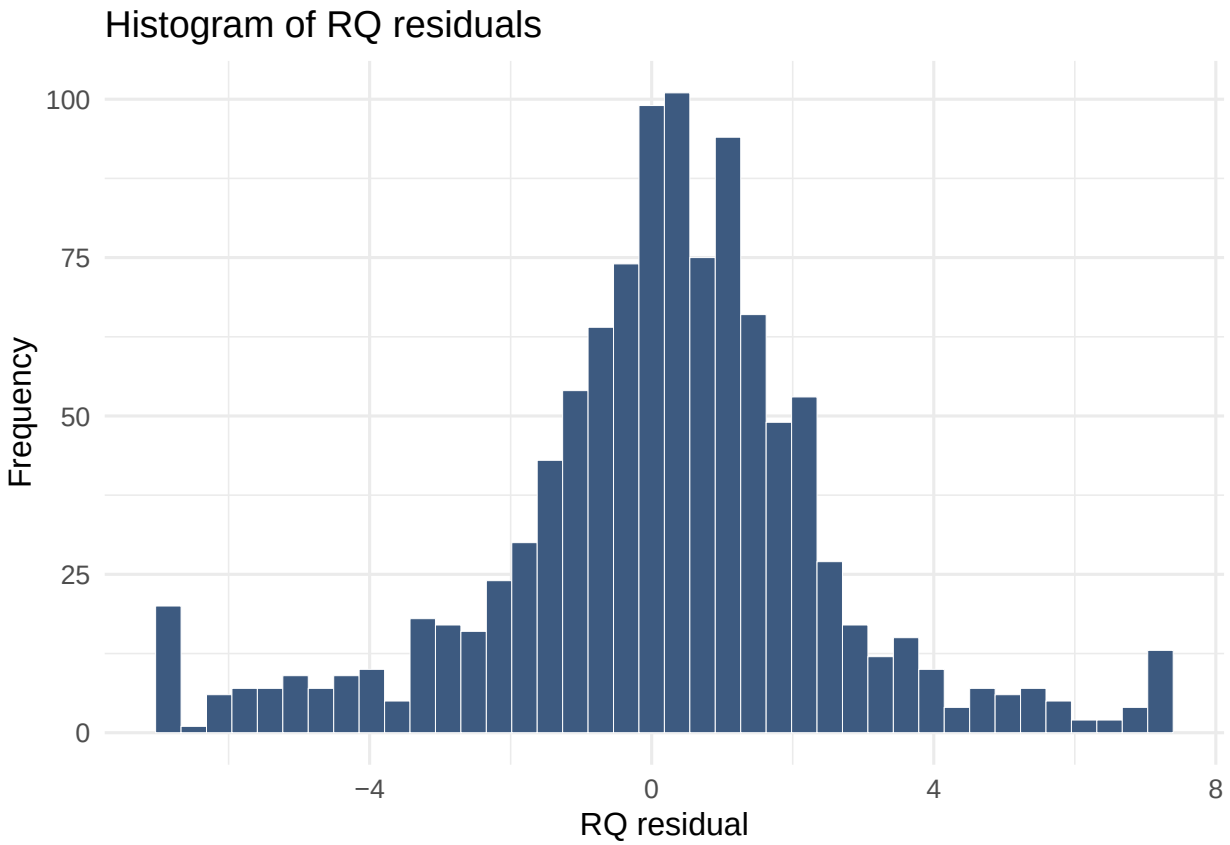
```r
  qqnorm(rqr, main = "QQ plot - RQ residuals"); qqline(rqr)

  # Residuals vs fitted (no smoother) + histogram
  mu <- fitted(fit, type = "response")
  dd <- tibble(mu = mu, rqr = rqr)

  print(
    ggplot(dd, aes(mu, rqr)) +
      geom_point(alpha = 0.15, size = 0.6) +
      geom_hline(yintercept = 0, linetype = 2) +
      labs(title = "RQR vs Fitted mean ( )", x = "Fitted  ", y = "RQ residual") +
      theme_minimal(base_size = 12)
  )
  print(
    ggplot(dd, aes(rqr)) +
      geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
      labs(title = "Histogram of RQ residuals", x = "RQ residual", y = "Frequency") +
      theme_minimal(base_size = 12)
  )
  # Autocorrelation: substantial spikes may indicate temporal dependence not captured.
  acf(rqr[is.finite(rqr)], na.action = na.pass, main = "ACF of RQ residuals")

  # 4) Overdispersion proxies (no sampling)
  #    - SD of RQR: should be ~1 under correct variance.
  #    - Pearson-like: use truncated-Poisson E[Y|interval] as "observed" y.
  sd_rqr <- sd(rqr, na.rm = TRUE)

  li <- pmax(0L, ceiling(data$lower))
  ui <- pmax(li, floor(data$upper))
  y_bar <- mapply(tpois_mean, li, ui, mu)   # deterministic imputation on integer support
  pearson <- sum((y_bar - mu)^2 / pmax(mu, 1e-8))   # guard by tiny
  disp    <- pearson / fit$df.residual               # dispersion   1 is ideal

  cat(sprintf("\nOverdispersion - SD(RQR): %.2f ( 1 ideal) | Pearson proxy: %.2f\n",
              sd_rqr, disp))
}

# ---- Run on your model/data ----
# Ensure `cpois_model` and `cp_df` exist in the workspace before calling.
diagnose_cpois(cpois_model, cp_df)
```

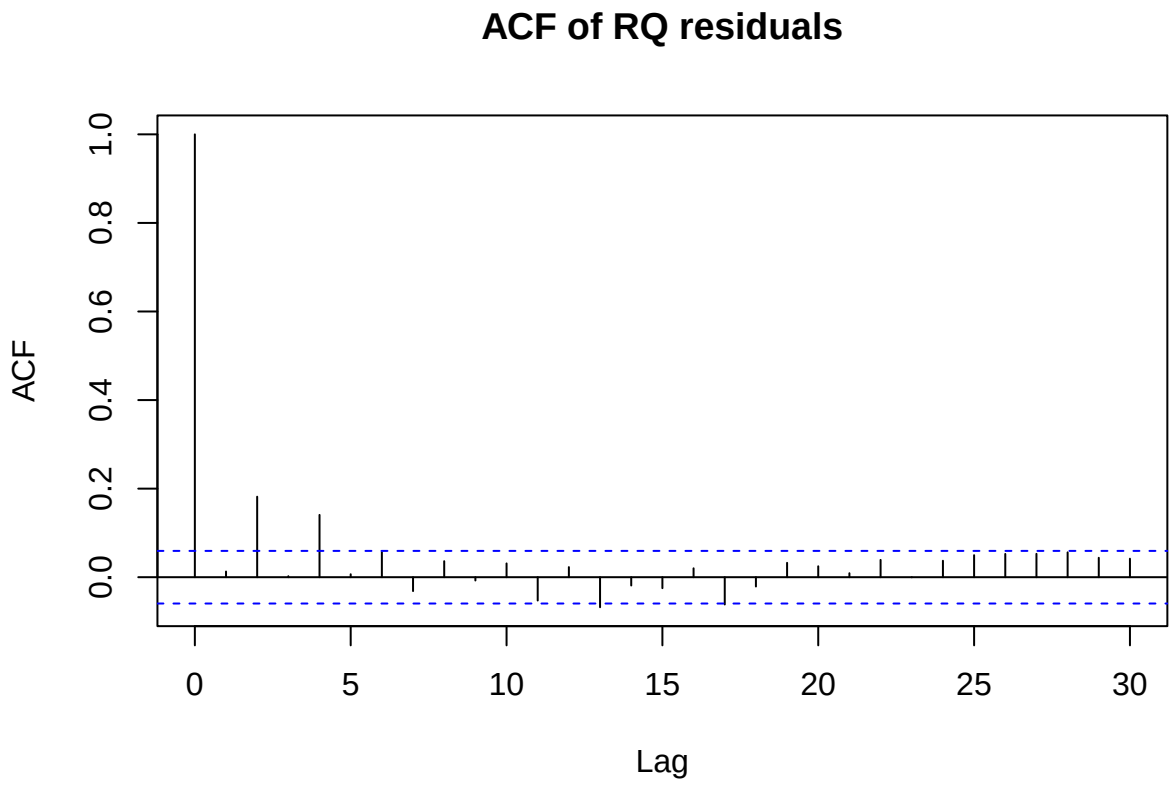
```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  1.297473e-07 -3.229102e-06 -6.752067e-08 -3.635216e-06  2.696238e-05
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.116194e+01 -2.332011e-06  2.348726e-01  1.715381e-08 -2.792819e-01
## [2,] -2.332011e-06  3.229081e-06  8.622013e-09 -1.059584e-15 -6.232250e-10
## [3,]  2.348726e-01  8.622013e-09  1.048827e+01 -2.557398e-06 -2.602784e-01
## [4,]  1.715381e-08 -1.059584e-15 -2.557398e-06  3.635190e-06  5.130067e-08
## [5,] -2.792819e-01 -6.232245e-10 -2.602784e-01  5.130067e-08  1.134983e+02
```

```
##
## Model rank =  384 / 384
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'   edf k-index p-value
## s(decimal_date):season_fspring  34.0  30.6    0.96    0.14
## s(decimal_date):season_fautumn  34.0  29.8    0.96    0.16
## s(SITE_ID.F)                   314.0 269.0      NA      NA
## $mfrow
## [1] 2 2
##
##
## Concurvity (rounded):
##          para s(decimal_date):season_fspring s(decimal_date):season_fautumn
## worst       1                       0.8190903                       0.7514057
## observed    1                       0.2854928                       0.2937161
## estimate    1                       0.5009502                       0.5588610
##          s(SITE_ID.F)
## worst      1.00000000
## observed   0.10790258
## estimate   0.06683649
```




RQR vs Fitted mean (.)
RQ residual
4
0
−4
0
100
200
300
400
500
Fitted .


Histogram of RQ residuals
Frequency
100
75
50
25
0
−4
0
4
8
RQ residual




```
##
## Overdispersion - SD(RQR): 2.45 ( 1 ideal) | Pearson proxy: 47.84
```

#4.4 CENSORED NORMAL GAMM MODEL

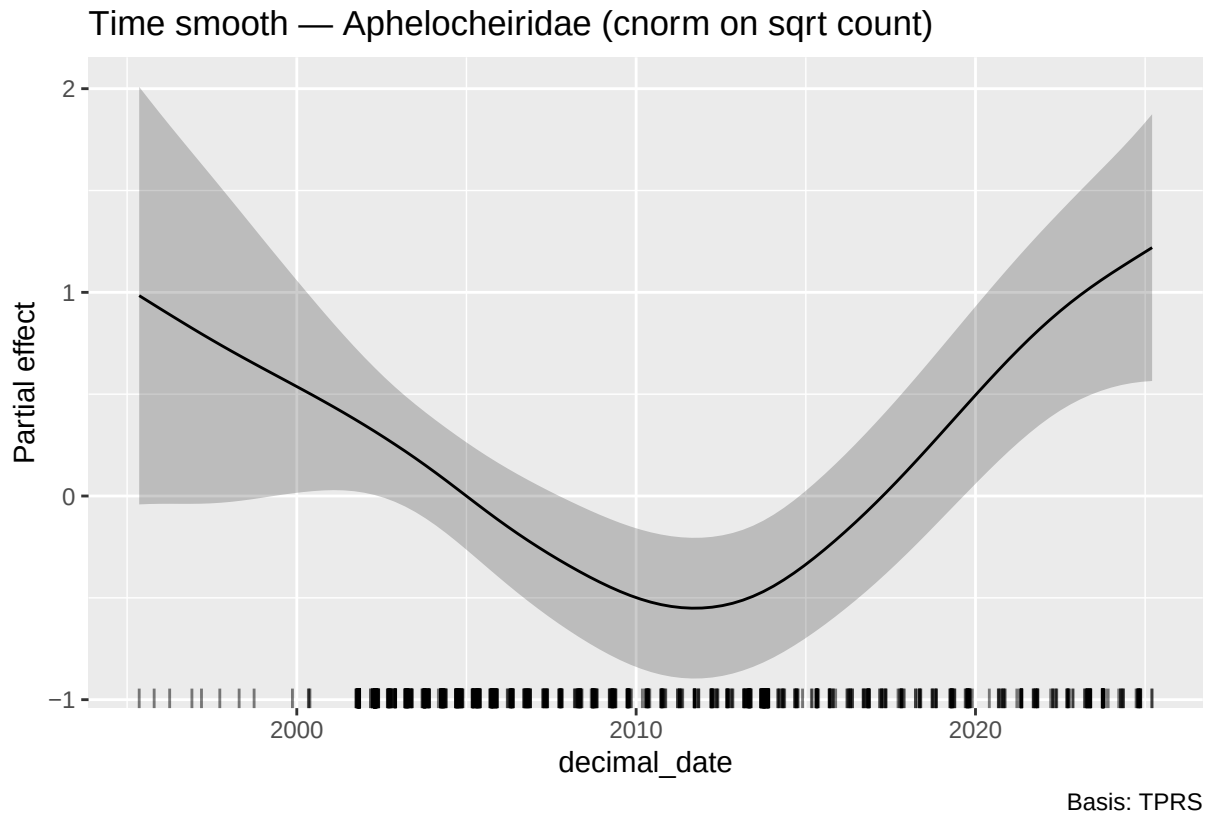
```
# ================================================
# CENSORED NORMAL (cnorm) - one family + partial effects
# Goal:
#   Fit an interval-censored Normal GAMM to sqrt-transformed
#   count intervals for a single family, with:
#     • season main effect (spring vs autumn)
#     • a single smooth temporal trend
#     • site-level random intercepts
#   Then print a text summary and the partial time smooth.
# Assumes:
#   - `models_data[[fam_name]]$cnorm` exists from Phase 3 and
#     already contains count-scale bounds `lower`/`upper`,
#     `season_f`, `decimal_date`, `SITE_ID.F`, `SAMPLE_DATE`.
# ================================================

# ---- choose family ----
# Change the string to any of your four focus families as needed.
fam_name <- "Aphelocheiridae"   # change as needed

# ---- get modelling data built earlier (same as cpois) ----
```

```r
# Pull the cnorm-ready table for the chosen family.
cn_df <- models_data[[fam_name]]$cnorm

# ---- build sqrt-intervals (avoid sqrt of negatives) ----
# Convert the count-scale interval bounds to sqrt(count) scale,
# clamping at 0 first to prevent sqrt of negative values.
to_sqrt_bounds <- function(l, u) {
  l2 <- pmax(as.numeric(l), 0)
  u2 <- pmax(as.numeric(u), 0)
  cbind(lower_t = sqrt(l2), upper_t = sqrt(u2))
}
SB <- to_sqrt_bounds(cn_df$lower, cn_df$upper)
cn_df$lower_t <- SB[, "lower_t"]
cn_df$upper_t <- SB[, "upper_t"]

# ---- sensible k from span of years ----
# Choose basis dimension k as a function of the number of sampled years:
# not too small (>=10) and capped at 30 to avoid overfitting.
k_time <- {
  ny <- dplyr::n_distinct(lubridate::year(cn_df$SAMPLE_DATE))
  min(30, max(10, round(0.6 * ny)))
}

# ---- fit cnorm: season main effect + single time smooth ----
# Interval response: cbind(lower_t, upper_t) on sqrt scale.
#   • season_f as a parametric factor (spring/autumn intercept shift)
#   • s(decimal_date) for a single smooth trend (same shape across seasons)
#   • s(SITE_ID.F, "re") as site random intercepts
# Family: cnorm() - censored Normal with identity link on sqrt scale.
# Fitting: bam(..., method="fREML", discrete=TRUE, select=TRUE, gamma=1.2)
cnorm_model <- bam(
  cbind(lower_t, upper_t) ~
    season_f +
    s(decimal_date, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = cnorm(),
  data     = cn_df,
  method   = "fREML",
  discrete = TRUE,
  select   = TRUE,
  gamma    = 1.2
)

# ---- summary ----
# Prints parametric terms, smooth EDFs, deviance explained, etc.
summary(cnorm_model)
```

```
##
## Family: cnorm(2.133)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, k = k_time) +
```

```
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.5186     0.1987  12.678  < 2e-16 ***
## season_fautumn   0.3972     0.1395   2.846  0.00453 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                      edf Ref.df      F  p-value
## s(decimal_date)   3.085     18 12.493 0.000949 ***
## s(SITE_ID.F)    228.337    313  6.317  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.597   Deviance explained = 68.5%
## fREML = 2806.1  Scale est. = 1         n = 1089
```

```r
# ---- partial-effect plots ----
# (1) time smooth: shows the marginal temporal trend on sqrt scale.
draw(cnorm_model, select = 1) +
  ggplot2::ggtitle(paste("Time smooth -", fam_name, "(cnorm on sqrt count)"))
```



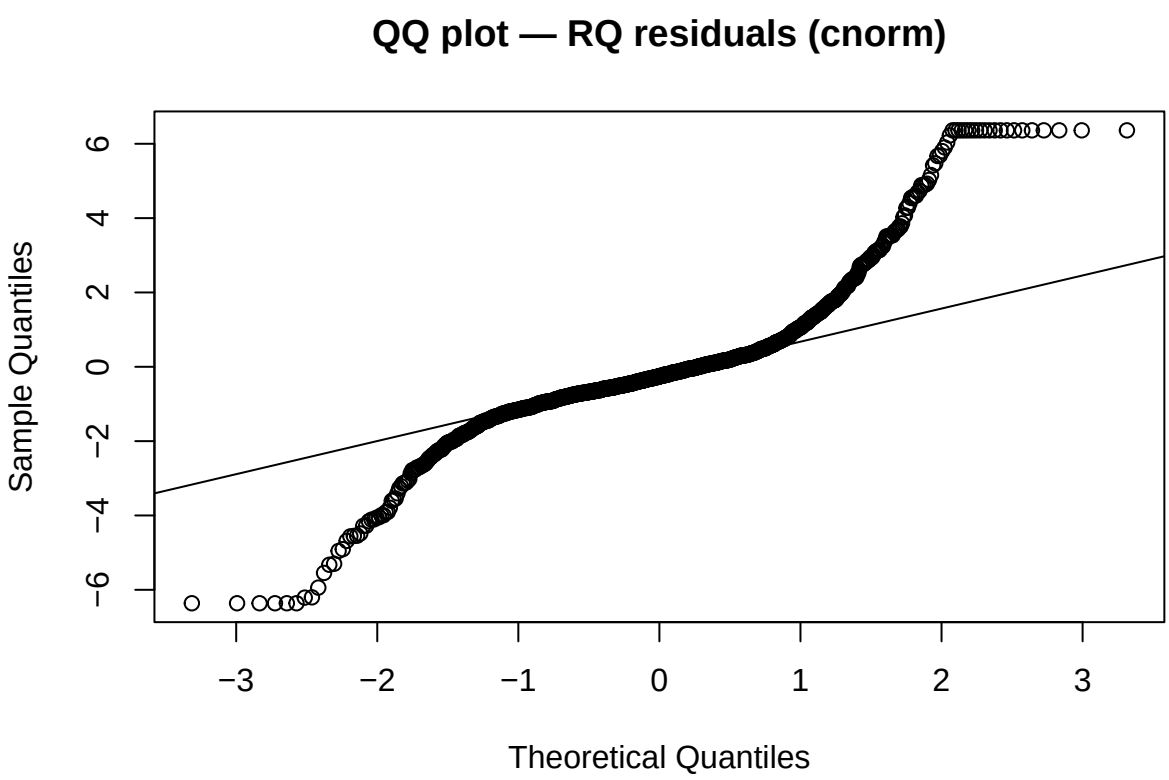
```r
# (2) optional: show site RE distribution (not a smooth, but useful)
# If you want to plot site random effects, uncomment the next line
# and ensure the object name matches (`cnorm_model2` here).
# gratia::draw(cnorm_model, select = 2)  # uncomment if you want the RE plot
```

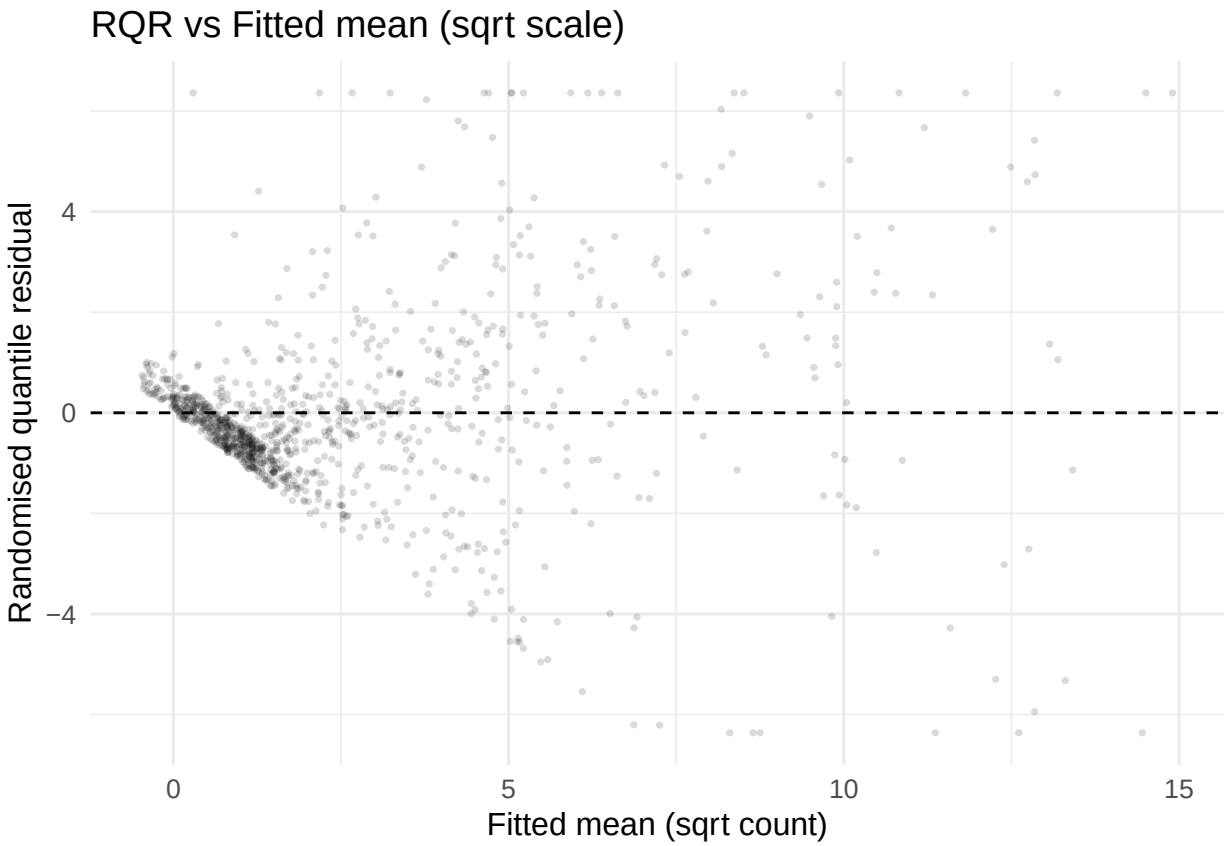
```r
# ============================================================
# CNORM Model Diagnostics
# Requires: cnorm_model (bam fit), cn_df with lower_t/upper_t
# Purpose:
#   - Run lightweight but meaningful checks for an interval-censored
#     Normal GAMM fitted on sqrt(count) scale (mgcv::cnorm()).
#   - Provide: basis adequacy, concurvity, randomized-quantile residuals,
#     simple residual panels (QQ / vs-fitted / hist / ACF), dispersion cue,
#     and headline fit metrics. Optionally shows time smooth partial effects.
# Notes:
#   - All plots are cheap and reproducible.
#   - Expects cn_df to already have lower_t/upper_t (sqrt bounds). If not,
#     they are created from count-scale lower/upper via ensure_sqrt_bounds().
# ============================================================

# --- Ensure sqrt-bounds exist (build from count-scale bounds if needed)
# Input:  df with either {lower_t, upper_t} OR {lower, upper} on count scale
# Output: df guaranteed to include lower_t/upper_t on sqrt(count) scale
ensure_sqrt_bounds <- function(df) {
  if (!all(c("lower_t","upper_t") %in% names(df))) {
    stopifnot(all(c("lower","upper") %in% names(df)))
    df <- df %>%
      mutate(
        lower_t = sqrt(pmax(as.numeric(lower), 0)),  # clamp at 0 before sqrt
        upper_t = sqrt(pmax(as.numeric(upper), 0))
      )
  }
  df
}

# --- Randomised quantile residuals for censored Normal ---
# Constructs PIT values from the Normal CDF between [lt, ut] and randomises
# uniformly within the censoring interval; then maps to N(0,1) via qnorm.
# Arguments:
#   fit : mgcv::bam fit with family=cnorm()
#   lt,ut : sqrt-scale lower/upper bounds
#   eps : numerical guard against exact 0/1 PIT values
#   seed : RNG seed for reproducibility of the randomisation step
cnorm_rqr <- function(fit, lt, ut, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))    # on sqrt scale
  sd <- sqrt(summary(fit)$scale)                      # sigma on sqrt scale

  Flo <- pnorm(lt, mean = mu, sd = sd)                # F(lower)
  Fup <- pnorm(ut, mean = mu, sd = sd)                # F(upper)

  w <- pmax(Fup - Flo, 0)                             # interval mass
  set.seed(seed)
  u <- Flo + w * runif(length(mu))                    # randomised PIT in [Flo,Fup]
```

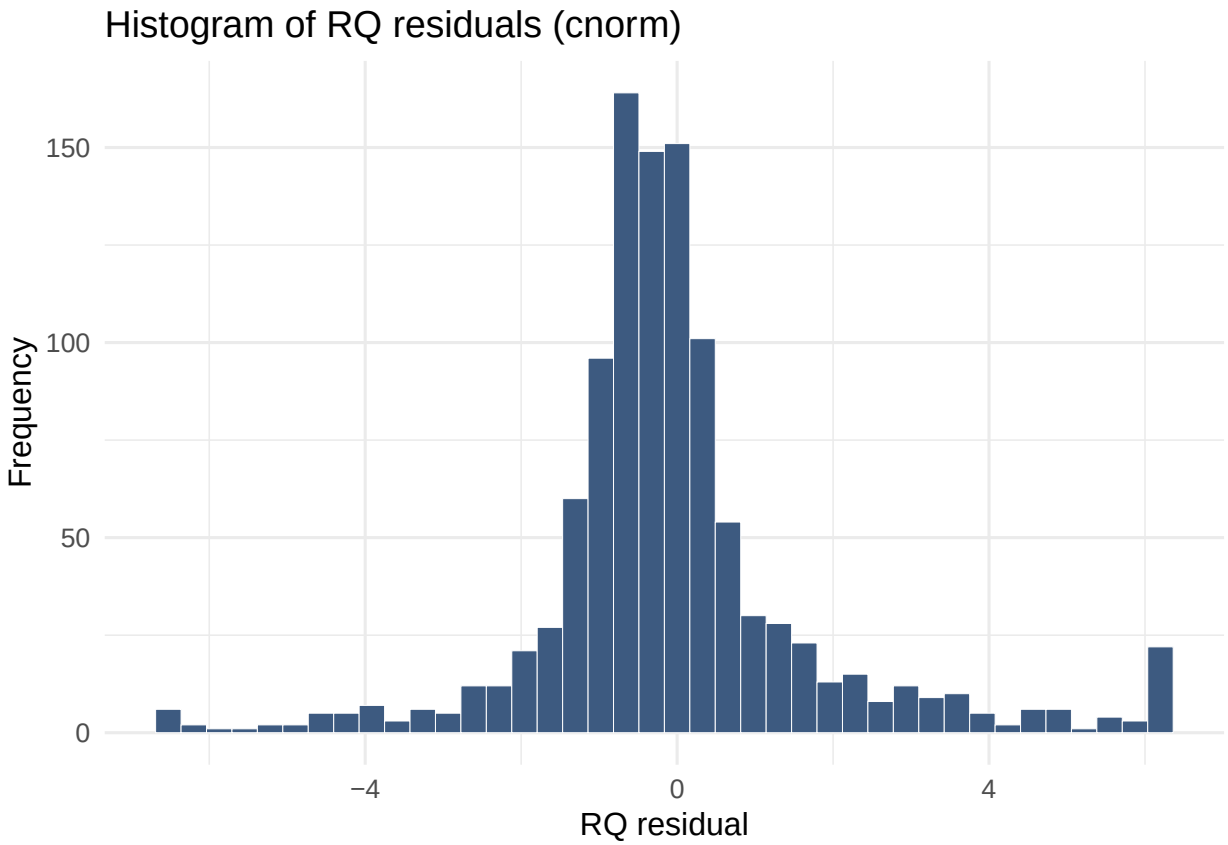
```r
  u <- pmin(pmax(u, eps), 1 - eps)                    # clamp away from 0/1
  qnorm(u)                                            # -> approx N(0,1) if model is correct
}

diagnose_cnorm <- function(fit, data) {
  # Standardise inputs: guarantee sqrt-scale bounds
  data <- ensure_sqrt_bounds(data)

  # 1) Basis/smooth adequacy (suppress 2×2 plot)
  # Uses gam.check() with rep=0 (no expensive re-smoothing); plots are diverted.
  tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
  gc_out <- try(gam.check(fit, rep = 0), silent = TRUE)
  dev.off(); unlink(tmp)
  if (!inherits(gc_out, "try-error")) print(gc_out)

  # 2) Concurvity (safe)
  # Detects non-linear dependence among smooth terms; prints rounded estimates.
  con <- try(concurvity(fit, full = TRUE), silent = TRUE)
  if (!inherits(con, "try-error")) {
    cat("\nConcurvity (rounded):\n")
    if (is.list(con) && !is.null(con$estimate)) print(round(con$estimate, 3)) else print(con)
  }

  # 3) Randomised quantile residuals (Normal PIT on sqrt scale)
  # RQRs should be ~N(0,1) with minimal structure if model is appropriate.
  rqr <- cnorm_rqr(fit, data$lower_t, data$upper_t)

  # QQ plot (RQR) - quick Gaussianity check
  par(mfrow = c(1,1))
  qqnorm(rqr, main = "QQ plot - RQ residuals (cnorm)"); qqline(rqr)

  # Residuals vs fitted & histogram - check for mean-variance issues/outliers
  mu <- fitted(fit, type = "response")
  dd <- tibble(mu = mu, rqr = rqr)

  print(
    ggplot(dd, aes(mu, rqr)) +
      geom_point(alpha = 0.15, size = 0.6) +
      geom_hline(yintercept = 0, linetype = 2) +
      labs(title = "RQR vs Fitted mean (sqrt scale)",
           x = "Fitted mean (sqrt count)", y = "Randomised quantile residual") +
      theme_minimal(base_size = 12)
  )

  print(
    ggplot(dd, aes(rqr)) +
      geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
      labs(title = "Histogram of RQ residuals (cnorm)",
           x = "RQ residual", y = "Frequency") +
      theme_minimal(base_size = 12)
  )

  # ACF of RQR - temporal autocorrelation check (should be small if independent)
```

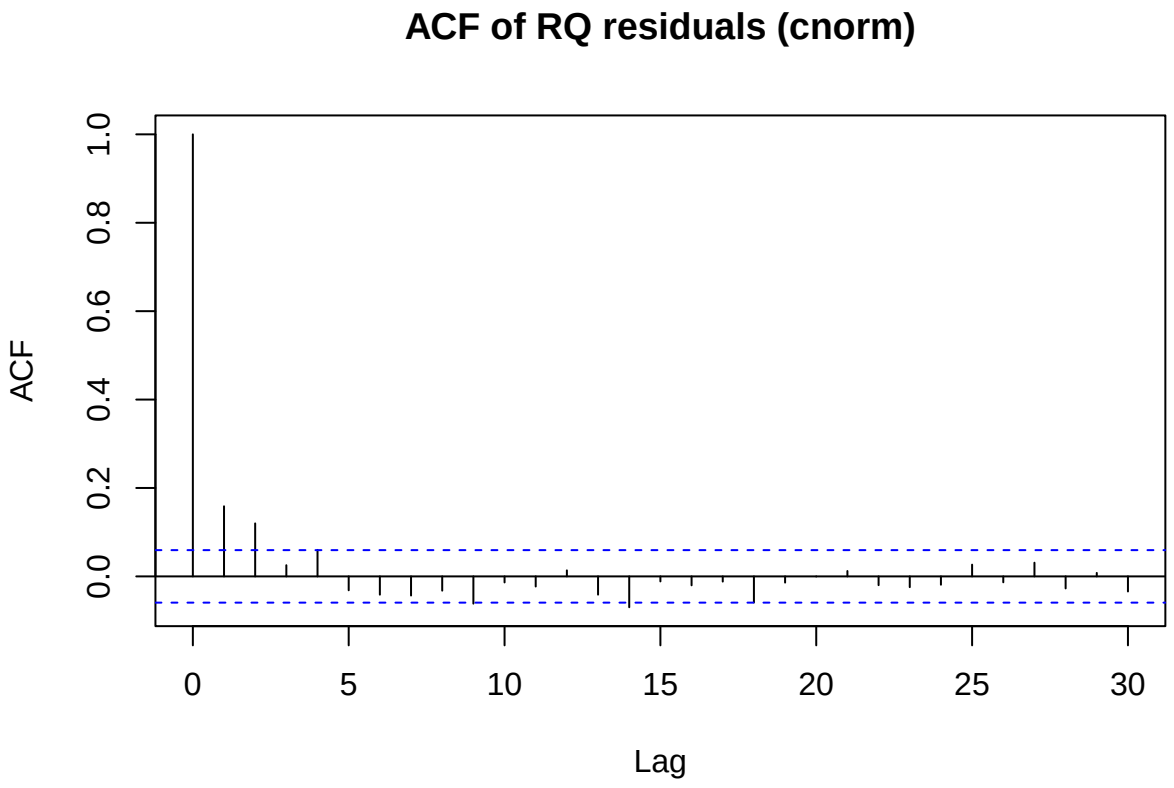
```r
  acf(rqr[is.finite(rqr)], na.action = na.pass, main = "ACF of RQ residuals (cnorm)")

  # 4) Dispersion cue: SD(RQR) should be ~1 under correct specification
  sd_rqr <- sd(rqr, na.rm = TRUE)
  cat(sprintf("\nResidual spread - SD(RQR): %.2f ( 1 ideal)\n", sd_rqr))

  # 5) Time smooth (partial effect) - printed by season structure
  # Draw all smooths containing 'decimal_date' (single or by-season).
  sm_ids <- which(grepl("decimal_date", smooths(fit)))
  if (length(sm_ids)) print(draw(fit, select = sm_ids))

  # 6) Headline metrics - quick readout of fit quality
  cat(sprintf("Deviance explained: %.1f%% | AIC: %.1f | N: %d | sigma(sqrt-scale): %.3f\n",
              100 * summary(fit)$dev.expl, AIC(fit), nobs(fit), sqrt(summary(fit)$scale)))
}

# ---- Run on your cnorm model/data ----
# Expects:
#   - cnorm_model : mgcv::bam fit with family = cnorm()
#   - cn_df       : data.frame used to fit (has lower_t/upper_t or lower/upper)
diagnose_cnorm(cnorm_model, cn_df)
```

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -7.946976e-13 -7.210606e-06  7.227641e-11
##
## $hess
##               [,1]          [,2]         [,3]
## [1,]  1.017651e+00 -4.071266e-06 2.261197e-01
## [2,] -4.071266e-06  7.210508e-06 1.137395e-06
## [3,]  2.261197e-01  1.137395e-06 9.124777e+01
##
## Model rank =  334 / 334
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                     k'    edf k-index p-value
## s(decimal_date)  18.00   3.09    0.95    0.06 .
## s(SITE_ID.F)    314.00 228.34      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## Concurvity (rounded):
##          para s(decimal_date) s(SITE_ID.F)
## worst       1       0.8157129   1.00000000
## observed    1       0.4324731   0.04408350
## estimate    1       0.6351632   0.04113726
```

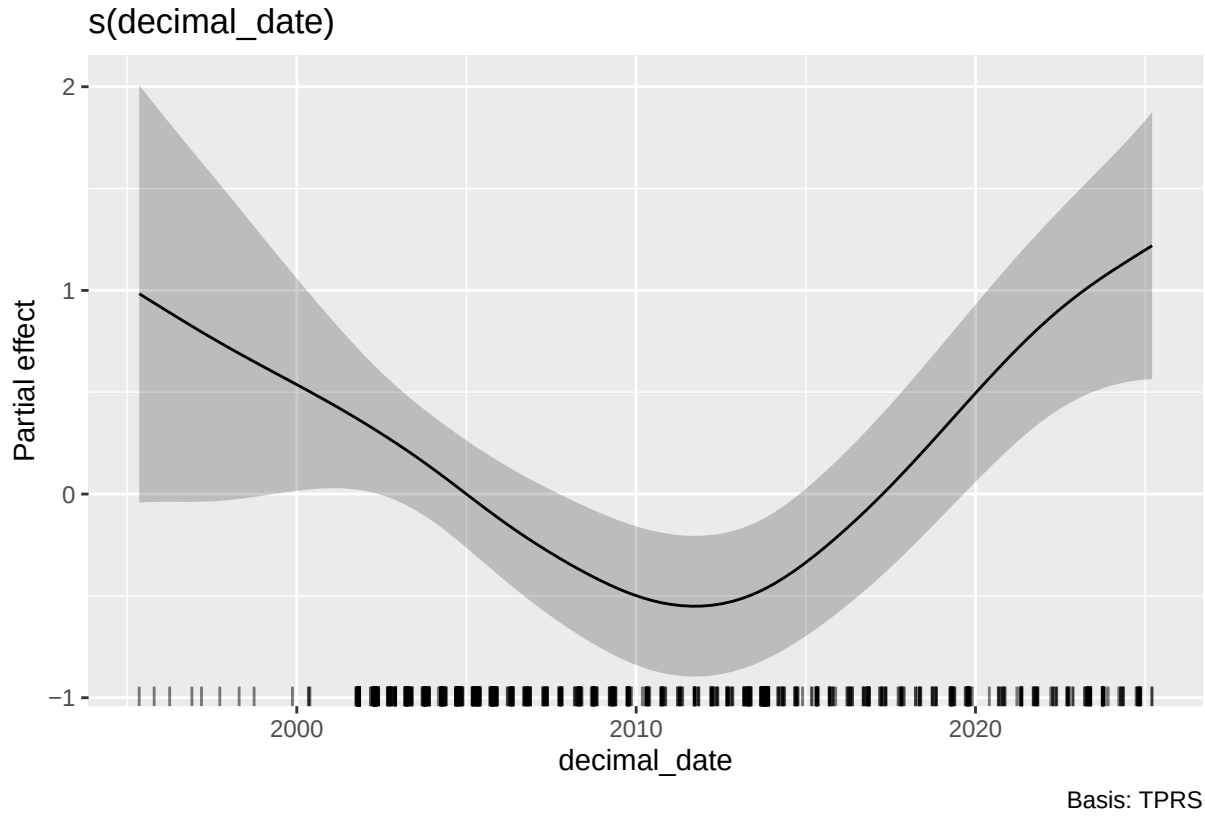

QQ plot — RQ residuals (cnorm)
Sample Quantiles
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2
3
Theoretical Quantiles


RQR vs Fitted mean (sqrt scale)
Randomised quantile residual
4
0
−4
0
5
10
15
Fitted mean (sqrt count)


Histogram of RQ residuals (cnorm)
Frequency
150
100
50
0
−4
0
4
RQ residual




```
##
## Residual spread - SD(RQR): 1.85 ( 1 ideal)
```




```
## Deviance explained: 68.5% | AIC: 6349.5 | N: 1089 | sigma(sqrt-scale): 1.000
```

#4.4.1 CNORM MODEL(INTERVAL WEIGHTING + POWER TUNING with REPORTING_AREA COVARIATE)

```r
# ========================================================
# Make cnorm tighter: interval weighting + power tuning
# ========================================================

# --- helper: RQ residuals for cnorm (same as before)
#     Given a fitted censored-normal model on the transformed scale, compute
#     randomized quantile residuals by sampling uniformly within each case's
#     CDF interval [F(lower), F(upper)]. Ideal dispersion: SD  1.
cnorm_rqr <- function(fit, lt, ut, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))  # fitted mean on transformed (e.g., sqrt) scale
  sd <- sqrt(summary(fit)$scale)                    # model scale ( ) on the same scale as mu
  Flo <- pnorm(lt, mean = mu, sd = sd)              # CDF at lower bound
  Fup <- pnorm(ut, mean = mu, sd = sd)              # CDF at upper bound
  w   <- pmax(Fup - Flo, 0)                         # interval probability (non-negative)
  set.seed(seed)
  u <- Flo + w * runif(length(mu))                  # uniform draw within [Flo, Fup]
  u <- pmin(pmax(u, eps), 1 - eps)                  # clamp away from 0/1 to avoid ±Inf
  qnorm(u)                                          # convert PIT to Normal residuals
}
```

```r
# --- power-transform the bounds *after* you've computed correct 1-s.f. count-scale bounds
#     Input: df with count-scale 'lower'/'upper' (non-integer, 1-s.f. rounding).
#     Output: matching bounds on transformed scale: sqrt if p=0.5, or x^p otherwise.
transform_bounds_p <- function(df, p = 0.5) {
  stopifnot(all(c("lower","upper") %in% names(df)))
  lt <- pmax(as.numeric(df$lower), 0)               # guard against negative lower
  ut <- pmax(as.numeric(df$upper), 0)               # guard against negative upper
  if (p == 0.5) {
    data.frame(lower_t = sqrt(lt), upper_t = sqrt(ut))
  } else {
    data.frame(lower_t = lt^p,     upper_t = ut^p)
  }
}

# --- one fit with given p and weighting exponent alpha_w
#     - Transforms bounds with power p
#     - Weights each row by interval width^{-alpha_w} so narrower intervals get more weight
#     - Optionally adds an EA reporting-area random effect if 'sites_clean' exists
fit_cnorm_power <- function(df, p = 0.5, alpha_w = 1, add_area_re = TRUE) {
  B  <- transform_bounds_p(df, p)                    # transform bounds to 'lower_t'/'upper_t'
  df$lower_t <- B$lower_t; df$upper_t <- B$upper_t

  # weights: narrower intervals get higher weight
  width_t <- pmax(df$upper_t - df$lower_t, 1e-6)    # avoid division by ~0
  wts     <- (1 / width_t)^alpha_w                   # alpha_w=1 gives strong emphasis

  # add EA region RE if you have sites_clean
  # (joins REPORTING_AREA by SITE_ID; only if not already present and object exists)
  if (add_area_re && !"REPORTING_AREA" %in% names(df) && exists("sites_clean")) {
    df <- dplyr::left_join(
      df,
      dplyr::select(sites_clean, SITE_ID, REPORTING_AREA),  # qualify select to avoid conflicts
      by = "SITE_ID"
    )
  }
  if ("REPORTING_AREA" %in% names(df)) df$REPORTING_AREA <- factor(df$REPORTING_AREA)

  # sensible k for the time smooth: scales with number of unique years (cap 35)
  k_time <- {
    ny <- dplyr::n_distinct(lubridate::year(df$SAMPLE_DATE))
    min(35, max(12, round(0.7 * ny)))
  }

  # Model formula:
  # - season_f fixed effect (spring/autumn intercept shift)
  # - single time smooth (no by=season to keep parsimony here)
  # - site random intercept; optional reporting-area RE if available
  fml <- if ("REPORTING_AREA" %in% names(df)) {
    cbind(lower_t, upper_t) ~ season_f + s(decimal_date, k = k_time) +
      s(SITE_ID.F, bs = "re") + s(REPORTING_AREA, bs = "re")
  } else {
    cbind(lower_t, upper_t) ~ season_f + s(decimal_date, k = k_time) +
      s(SITE_ID.F, bs = "re")
```

```
  }

  # Fit censored normal GAM (bam for speed):
  # - weights emphasize precise intervals
  # - select=TRUE + gamma=1.2 provide gentle extra regularization
  m <- bam(
    fml,
    family   = cnorm(),
    data     = df,
    weights  = wts,
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  # dispersion cue via RQ residuals (should be ~1 if well calibrated)
  rqr <- cnorm_rqr(m, df$lower_t, df$upper_t)
  sd_rqr <- sd(rqr, na.rm = TRUE)

  list(model = m, data = df, p = p, alpha_w = alpha_w, sd_rqr = sd_rqr)  # return fit + diagnostics
}

# ---- run a small grid over p and pick what tightens RQ spread the most
cn_df <- models_data[["Aphelocheiridae"]]$cnorm  # or your chosen family
# IMPORTANT: ensure cn_df$lower/upper are the *correct 1-s.f.* count-scale bounds

grid_p   <- c(0.35, 0.4, 0.45, 0.5, 0.6)   # ✓ is 0.5; lower p compresses high counts more
alpha_w  <- 1                               # try 0.5 and 1 to vary interval weighting strength

# Fit across the grid of p, holding alpha_w fixed; collect SD of RQ residuals
fits <- lapply(grid_p, function(pp) fit_cnorm_power(cn_df, p = pp, alpha_w = alpha_w))
sd_tab <- data.frame(
  p = sapply(fits, `[[`, "p"),
  SD_RQR = sapply(fits, `[[`, "sd_rqr")
)
print(sd_tab)  # inspect which p gives SD_RQR closest to 1 and/or smallest
```

```
##      p    SD_RQR
## 1 0.35 0.9478238
## 2 0.40 1.2135600
## 3 0.45 1.5329182
## 4 0.50 1.8565437
## 5 0.60 2.4111371
```

```
# choose the best (smallest SD_RQR)
best_idx <- which.min(sd_tab$SD_RQR)
best_fit <- fits[[best_idx]]
cat(sprintf("\nChosen p = %.2f | SD(RQR) = %.2f (alpha_w = %.2f)\n",
            best_fit$p, best_fit$sd_rqr, best_fit$alpha_w))
```

```
##
## Chosen p = 0.35 | SD(RQR) = 0.95 (alpha_w = 1.00)
```

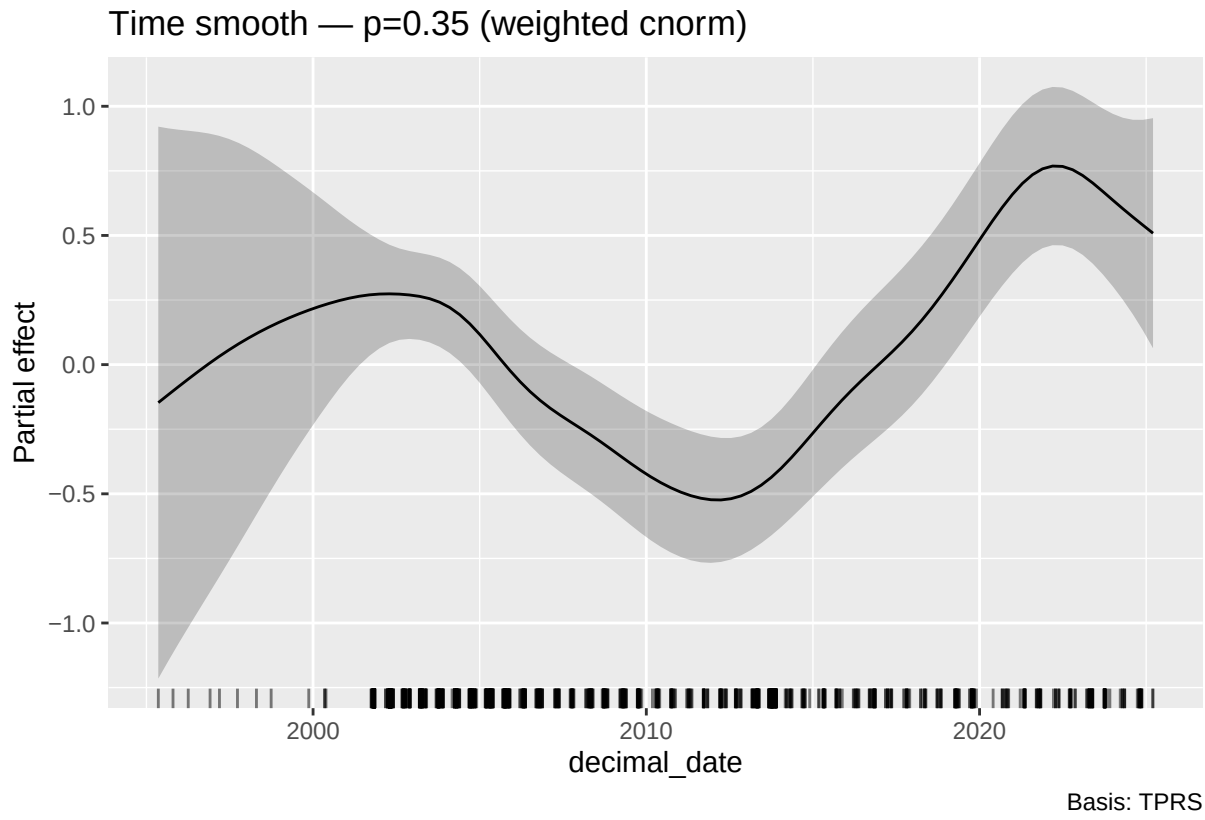
```r
# inspect k-index and smooth quickly (plots suppressed by sending to a temp PDF)
tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
print(try(gam.check(best_fit$model, rep = 0), silent = TRUE))
```

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -1.122658e-12 -1.054746e-05 -2.324470e-10 -1.136397e-05
##
## $hess
##               [,1]          [,2]         [,3]         [,4]
## [1,]  3.608887e-01 -8.662743e-06 3.031816e-01 9.668003e-08
## [2,] -8.662743e-06  1.054724e-05 1.551059e-07 2.985228e-12
## [3,]  3.031816e-01  1.551059e-07 1.003305e+02 7.143075e-05
## [4,]  9.668003e-08  2.985228e-12 7.143075e-05 1.136599e-05
##
## Model rank =  351 / 351
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                         k'      edf k-index p-value
## s(decimal_date)   2.10e+01 5.78e+00    0.94    0.12
## s(SITE_ID.F)      3.14e+02 2.18e+02      NA      NA
## s(REPORTING_AREA) 1.40e+01 3.23e-04      NA      NA
## $mfrow
## [1] 2 2
```

```r
dev.off(); unlink(tmp)
```

```
## pdf
##   2
```

```r
# optional: draw the time smooth (partial effect) for the chosen p
draw(best_fit$model, select = 1) + ggtitle(sprintf("Time smooth - p=%.2f (weighted cnorm)", best_fit$p)
```



#4.4.2 CNORM MODEL(P=O.35 WITH NO REPORTING_AREA COVARIATE)

```
# ============================================================
# CENSORED-NORMAL with p = 0.35 (no REPORTING_AREA term)
# Keeps everything else the same: 1-s.f. intervals, weights,
# season main effect, single time smooth, site RE.
# ------------------------------------------------------------
# Purpose of this script:
#   • Fit a censored Normal GAM on power-transformed bounds (p = 0.35).
#   • Use interval-width weights (narrower intervals = higher weight).
#   • Include fixed season effect, one smooth trend over decimal_date,
#     and a random intercept for SITE_ID.F.
# Inputs expected (already prepared upstream):
#   • models_data[[fam_name]]$cnorm with columns:
#       lower/upper (1-s.f. count-scale bounds), SAMPLE_DATE, decimal_date,
#       season_f (or SAMPLE_DATE to derive it), SITE_ID or SITE_ID.F, TOTAL_NUMBER (optional).
# Notes:
#   • This version intentionally omits a REPORTING_AREA random effect.
#   • Only prints model summary and text-only gam.check output (no plots saved).
# ============================================================

# --- helper: RQ residuals for cnorm ---
# Computes randomized quantile residuals for the censored Normal fit:
#   1) Get fitted mean (mu) and residual scale (sd) on the transformed scale.
#   2) Compute Normal CDF at lower/upper bounds.
#   3) Draw a uniform random value within [F(lower), F(upper)] to respect censoring.
```

```r
#   4) Map back to z-scale via qnorm for approximately N(0,1) residuals.
cnorm_rqr <- function(fit, lt, ut, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- sqrt(summary(fit)$scale)
  Flo <- pnorm(lt, mean = mu, sd = sd)
  Fup <- pnorm(ut, mean = mu, sd = sd)
  w   <- pmax(Fup - Flo, 0)
  set.seed(seed)
  u <- Flo + w * runif(length(mu))
  u <- pmin(pmax(u, eps), 1 - eps)
  qnorm(u)
}

# --- power-transform bounds AFTER you've built correct 1-s.f. count-scale bounds
# Transforms raw count-scale bounds to the p-power scale used by the cnorm family.
# Here p = 0.35 (from your tuning). Assumes non-negative bounds.
transform_bounds_p <- function(df, p = 0.35) {
  stopifnot(all(c("lower","upper") %in% names(df)))
  lt <- pmax(as.numeric(df$lower), 0)
  ut <- pmax(as.numeric(df$upper), 0)
  data.frame(lower_t = lt^p, upper_t = ut^p)
}

# --- fit cnorm with p=0.35; NO REPORTING_AREA term ---
# Wraps the entire fitting pipeline:
#   • Builds transformed bounds lower_t/upper_t with p = 0.35.
#   • Computes precision weights = 1 / (interval width)^alpha_w.
#   • Chooses k for the time smooth based on year span.
#   • Fits bam() with family = cnorm().
# Returns a small list with model, data used, SD of RQ residuals, and k_time.
fit_cnorm_p035 <- function(df, alpha_w = 1) {
  B  <- transform_bounds_p(df, p = 0.35)
  df$lower_t <- B$lower_t
  df$upper_t <- B$upper_t

  # precision weights: narrower transformed intervals -> higher weight
  width_t <- pmax(df$upper_t - df$lower_t, 1e-6)
  wts     <- (1 / width_t)^alpha_w

  # sensible k for time smooth
  k_time <- {
    ny <- dplyr::n_distinct(lubridate::year(df$SAMPLE_DATE))
    min(35, max(12, round(0.7 * ny)))
  }

  m <- bam(
    cbind(lower_t, upper_t) ~
      season_f +
      s(decimal_date, k = k_time) +
      s(SITE_ID.F, bs = "re"),
    family   = cnorm(),
    data     = df,
    weights  = wts,
```

```
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  # quick dispersion cue
  rqr <- cnorm_rqr(m, df$lower_t, df$upper_t)
  sd_rqr <- sd(rqr, na.rm = TRUE)

  list(model = m, data = df, sd_rqr = sd_rqr, k_time = k_time)
}

# ----------------- RUN -----------------
# Choose the family to fit:
#   • Ensure models_data[[fam_name]]$cnorm exists and has correct lower/upper.
fam_name <- "Aphelocheiridae"  # change as needed
cn_df    <- models_data[[fam_name]]$cnorm   # assumes lower/upper are correct 1-s.f. bounds

# Fit the weighted cnorm with p = 0.35 (no area RE). Keep alpha_w = 1 unless you have reason to tweak.
res_p035 <- fit_cnorm_p035(cn_df, alpha_w = 1)

# summaries (no extra plots)
#   • summary(): parametric terms, smooth EDFs, deviance explained, scale, etc.
print(summary(res_p035$model))
```

```
##
## Family: cnorm(1.892)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, k = k_time) +
##     s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.75222    0.10801  16.223  < 2e-16 ***
## season_fautumn  0.34209    0.07365   4.645 3.93e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df     F p-value
## s(decimal_date)   5.782     21 41.53 9.2e-06 ***
## s(SITE_ID.F)    217.724    313 10.51 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.724   Deviance explained = 78.3%
## fREML = 2599.4  Scale est. = 1         n = 1089
```

```r
#   • gam.check(): text output only (plots diverted to and discarded from a temp PDF)
tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
print(try(gam.check(res_p035$model, rep = 0), silent = TRUE))
```

```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -9.765522e-13 -1.048974e-05 -4.044978e-10
##
## $hess
##               [,1]          [,2]         [,3]
## [1,]  3.608932e-01 -8.615329e-06 3.031800e-01
## [2,] -8.615329e-06  1.048952e-05 1.542610e-07
## [3,]  3.031800e-01  1.542610e-07 1.003306e+02
##
## Model rank =  337 / 337
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                     k'    edf k-index p-value
## s(decimal_date)  21.00   5.78    0.94    0.12
## s(SITE_ID.F)    314.00 217.72      NA      NA
## $mfrow
## [1] 2 2
```

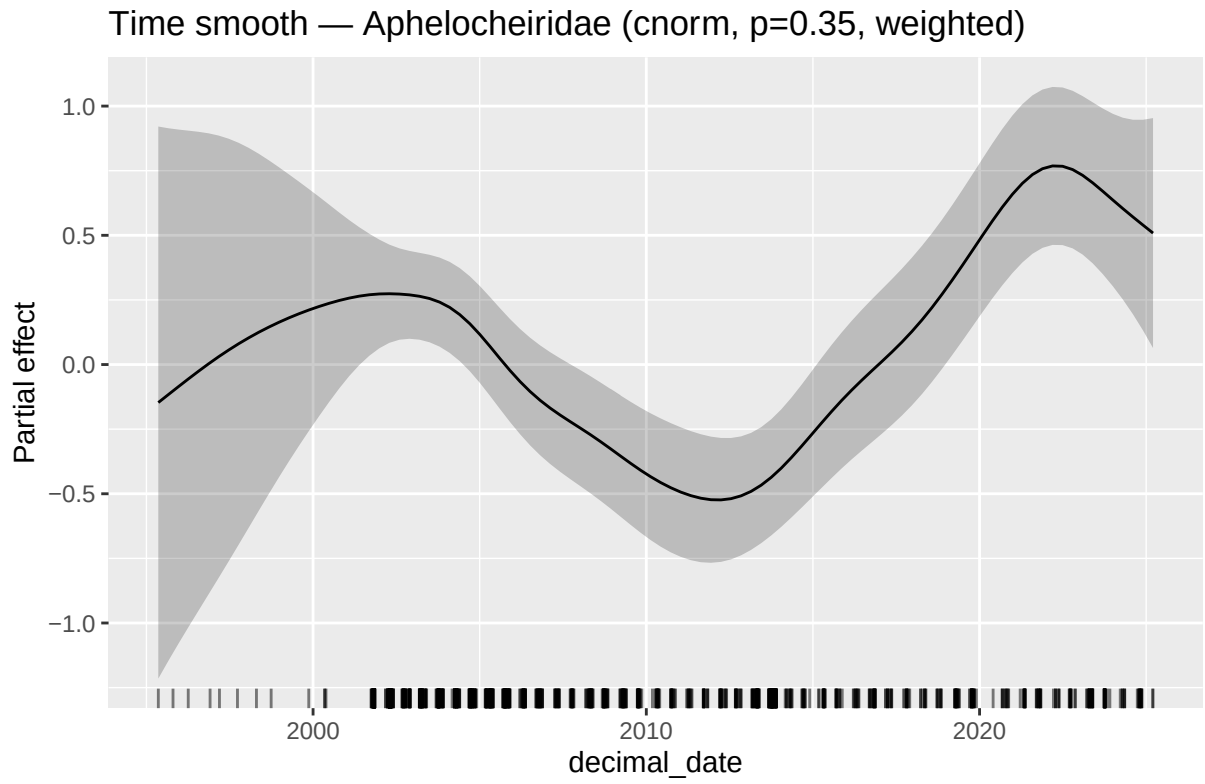
```r
dev.off(); unlink(tmp)
```

```
## pdf
##   2
```

```r
#   • Compact headline metrics for your log
cat(sprintf("SD(RQ residuals): %.2f | Dev.expl: %.1f%% | AIC: %.1f | N: %d\n",
            res_p035$sd_rqr, 100 * summary(res_p035$model)$dev.expl,
            AIC(res_p035$model), nobs(res_p035$model)))
```

```
## SD(RQ residuals): 0.95 | Dev.expl: 78.3% | AIC: 5838.8 | N: 1089
```

```r
# optional: partial effect of time
#   • Visualises the single time smooth on the transformed scale (no REs shown).
draw(res_p035$model, select = 1) +
  ggtitle(paste0("Time smooth - ", fam_name, " (cnorm, p=0.35, weighted)"))
```



Basis: TPRS

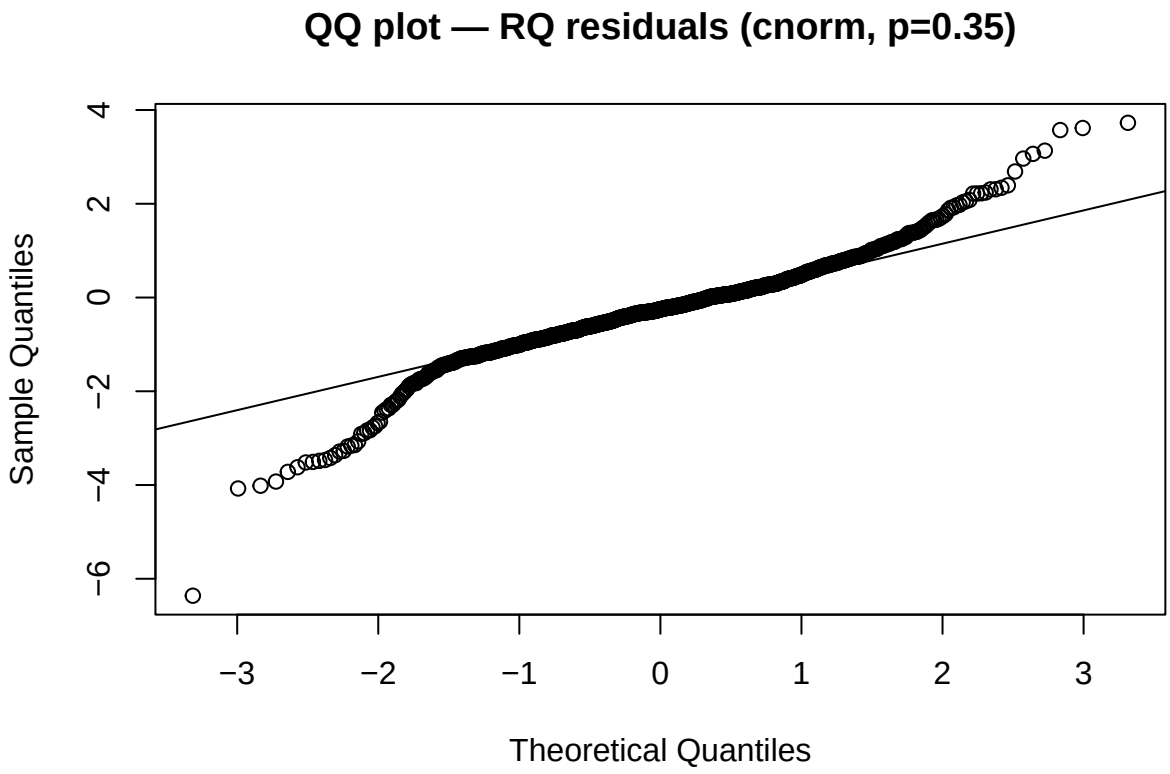
```
# ============================================================
# Model Diagnostics for the p = 0.35 cnorm model (no REPORTING_AREA)
# Works with objects returned by `res_p035 <- fit_cnorm_p035(...)`
# ------------------------------------------------------------
# What this script does:
#   • Accepts a fitted censored-Normal GAM (on a p-power scale) and the data used.
#   • Rebuilds transformed bounds if missing, using the same power p (default 0.35).
#   • Runs text-only k-adequacy and concurvity checks.
#   • Computes randomized quantile residuals (RQR) for censored Normal and
#     visualizes them (QQ, residuals vs fitted, histogram, ACF).
#   • Prints headline fit metrics (DevExpl, AIC, N, sigma on p-scale).
# Assumptions:
#   • `data` has raw count-scale bounds `lower`/`upper`; or already has `lower_t`/`upper_t`.
#   • The model `fit` is a `bam()` fit with `family = cnorm()`.
#   • The same p used at fit time (0.35 here) is passed to diagnostics for consistency.
# ============================================================


# --- helper: build transformed bounds if needed (p-scale) ---
# Takes raw count-scale bounds (lower/upper) and applies the p-power transform,
# returning a data.frame with columns `lower_t` and `upper_t` (both on the p-scale).
transform_bounds_p <- function(df, p = 0.35) {
  stopifnot(all(c("lower","upper") %in% names(df)))
  lt <- pmax(as.numeric(df$lower), 0)
  ut <- pmax(as.numeric(df$upper), 0)
  data.frame(lower_t = lt^p, upper_t = ut^p)
```

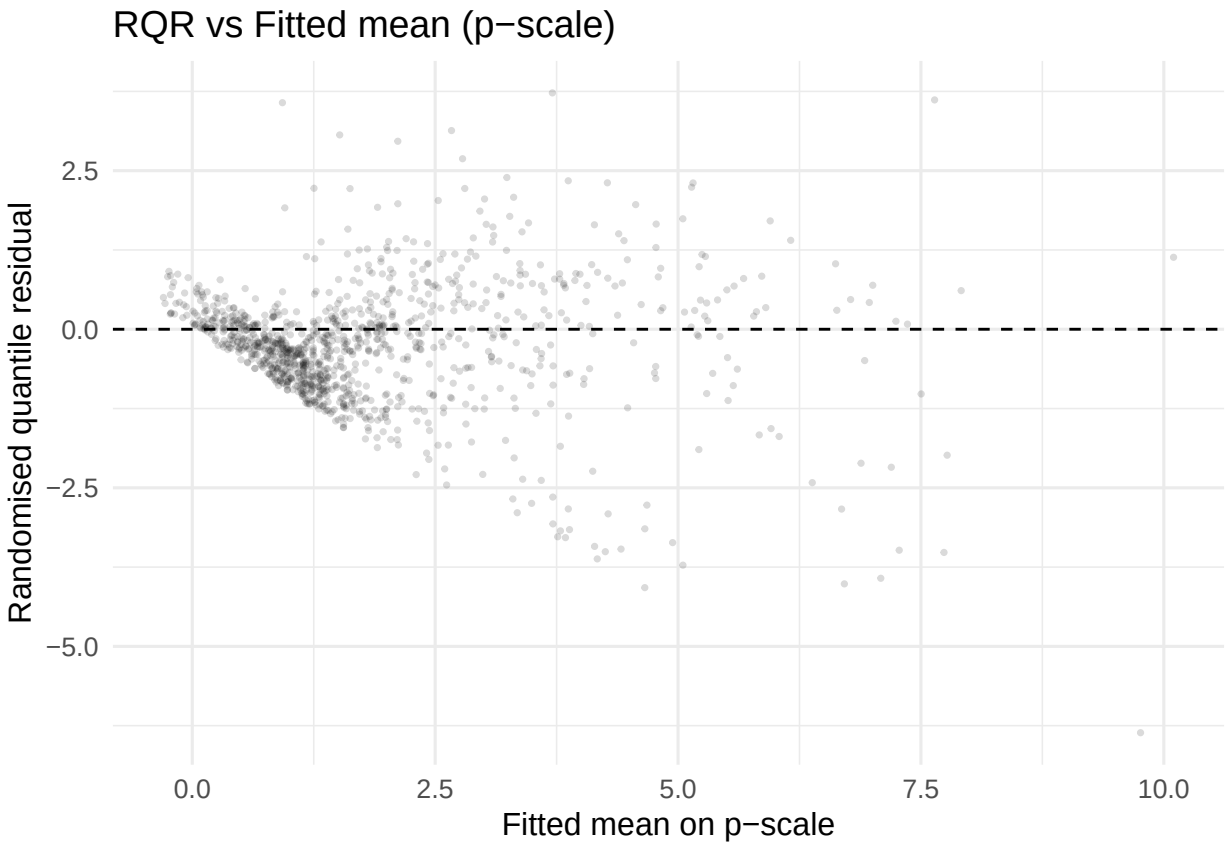
```
}

# --- Randomised quantile residuals for censored Normal (generic p-scale) ---
# For each observation, draws a uniform value between F(lower_t) and F(upper_t)
# under N(mu, sd^2) on the p-scale, then converts to a standard normal score.
# This produces residuals that should be ~N(0,1) if the model/distribution is correct.
cnorm_rqr <- function(fit, lt, ut, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- sqrt(summary(fit)$scale)
  Flo <- pnorm(lt, mean = mu, sd = sd)
  Fup <- pnorm(ut, mean = mu, sd = sd)
  w   <- pmax(Fup - Flo, 0)
  set.seed(seed)
  u <- Flo + w * runif(length(mu))
  u <- pmin(pmax(u, eps), 1 - eps)
  qnorm(u)
}

diagnose_cnorm_p <- function(fit, data, p = 0.35) {
  # Ensure transformed bounds exist
  if (!all(c("lower_t","upper_t") %in% names(data))) {
    B <- transform_bounds_p(data, p = p)
    data$lower_t <- B$lower_t; data$upper_t <- B$upper_t
  }

  # 1) k-adequacy (no 2x2 plot)
  # Diverts any plots into a temp device to avoid clutter; prints text output if available.
  tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
  print(try(gam.check(fit, rep = 0), silent = TRUE))
  dev.off(); unlink(tmp)

  # 2) Concurvity
  # Safe wrapper: if concurvity computation fails, we just skip it.
  con <- try(concurvity(fit, full = TRUE), silent = TRUE)
  if (!inherits(con, "try-error")) {
    cat("\nConcurvity (rounded):\n")
    if (is.list(con) && !is.null(con$estimate)) print(round(con$estimate, 3)) else print(con)
  }

  # 3) Randomised quantile residuals
  # Build RQR on the p-scale bounds then visualize basic diagnostics.
  rqr <- cnorm_rqr(fit, data$lower_t, data$upper_t)

  par(mfrow = c(1,1))
  qqnorm(rqr, main = "QQ plot - RQ residuals (cnorm, p=0.35)"); qqline(rqr)

  mu <- fitted(fit, type = "response")
  dd <- tibble(mu = mu, rqr = rqr)

  print(
    ggplot(dd, aes(mu, rqr)) +
      geom_point(alpha = 0.15, size = 0.6) +
      geom_hline(yintercept = 0, linetype = 2) +
```

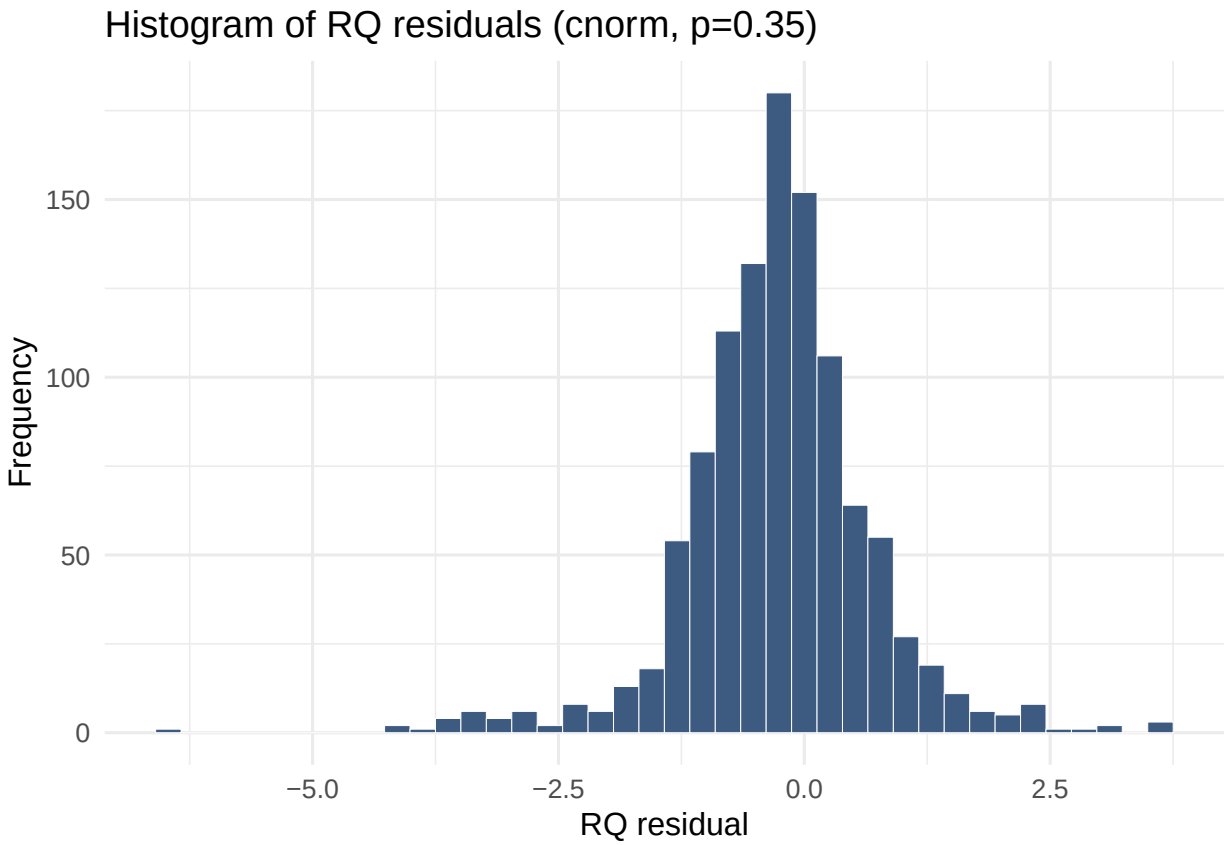
```
      labs(title = "RQR vs Fitted mean (p-scale)",
           x = "Fitted mean on p-scale", y = "Randomised quantile residual") +
      theme_minimal(base_size = 12)
  )

  print(
    ggplot(dd, aes(rqr)) +
      geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
      labs(title = "Histogram of RQ residuals (cnorm, p=0.35)",
           x = "RQ residual", y = "Frequency") +
      theme_minimal(base_size = 12)
  )

  acf(rqr[is.finite(rqr)], na.action = na.pass,
      main = "ACF of RQ residuals (cnorm, p=0.35)")

  # 4) Dispersion cue
  # SD of RQR should be close to 1. Values <<1 suggest underdispersion;
  # values >>1 suggest overdispersion or model misspecification.
  sd_rqr <- sd(rqr, na.rm = TRUE)
  cat(sprintf("\nResidual spread - SD(RQR): %.2f ( 1 ideal)\n", sd_rqr))

  # 5) Time smooth
  # Prints the partial effect(s) for the smooth(s) involving decimal_date (if present).
  sm_ids <- which(grepl("decimal_date", smooths(fit)))
  if (length(sm_ids)) print(draw(fit, select = sm_ids))

  # 6) Headline metrics
  # Quick, text-only summary to add to logs.
  cat(sprintf("Deviance explained: %.1f%% | AIC: %.1f | N: %d | sigma(p-scale): %.3f\n",
              100 * summary(fit)$dev.expl, AIC(fit), nobs(fit), sqrt(summary(fit)$scale)))
}

# ---- Run on your fitted model/data (from previous step) ----
# `res_p035` is expected to be created by fit_cnorm_p035(); we reuse the same p (=0.35).
diagnose_cnorm_p(res_p035$model, res_p035$data, p = 0.35)
```

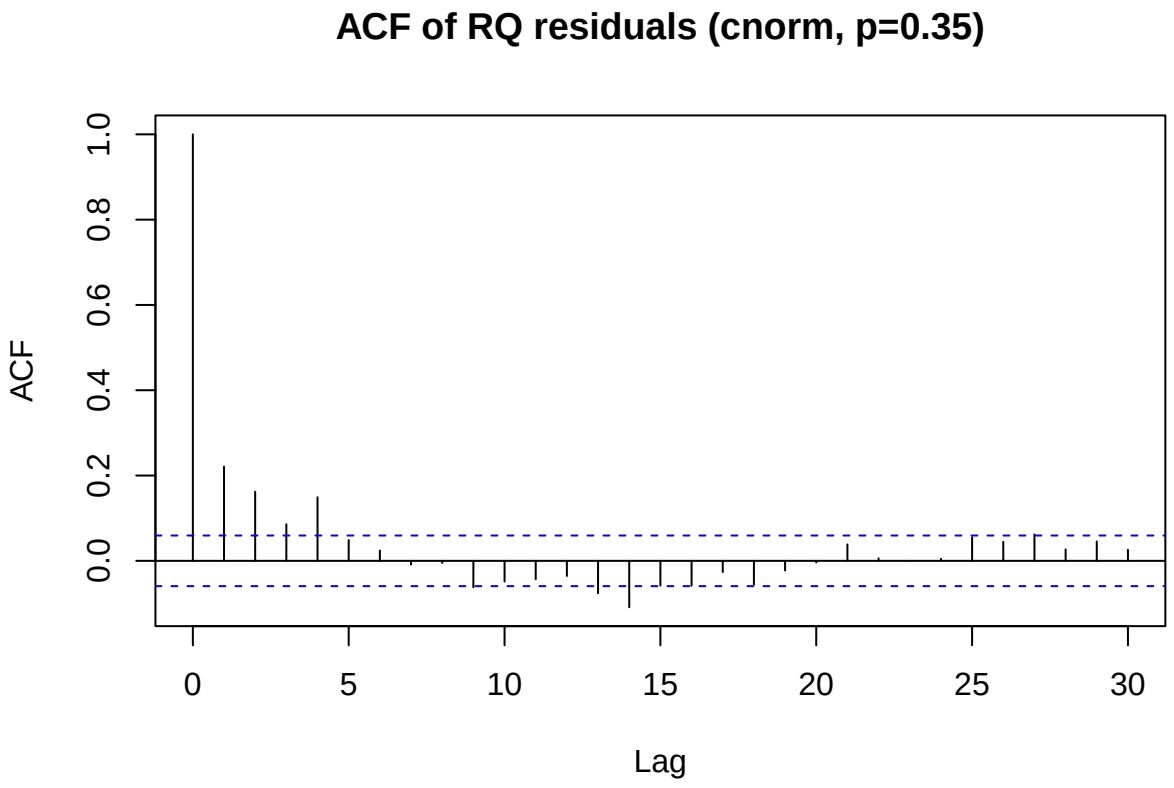
```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -9.765522e-13 -1.048974e-05 -4.044978e-10
##
## $hess
##               [,1]          [,2]         [,3]
## [1,]  3.608932e-01 -8.615329e-06 3.031800e-01
## [2,] -8.615329e-06  1.048952e-05 1.542610e-07
## [3,]  3.031800e-01  1.542610e-07 1.003306e+02
##
## Model rank =  337 / 337
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
```

```
##                       k'    edf k-index p-value
## s(decimal_date)  21.00   5.78    0.94   0.075 .
## s(SITE_ID.F)    314.00 217.72      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## Concurvity (rounded):
##          para s(decimal_date) s(SITE_ID.F)
## worst       1       0.8271427   1.00000000
## observed    1       0.4414159   0.04077406
## estimate    1       0.6378030   0.04271298
```

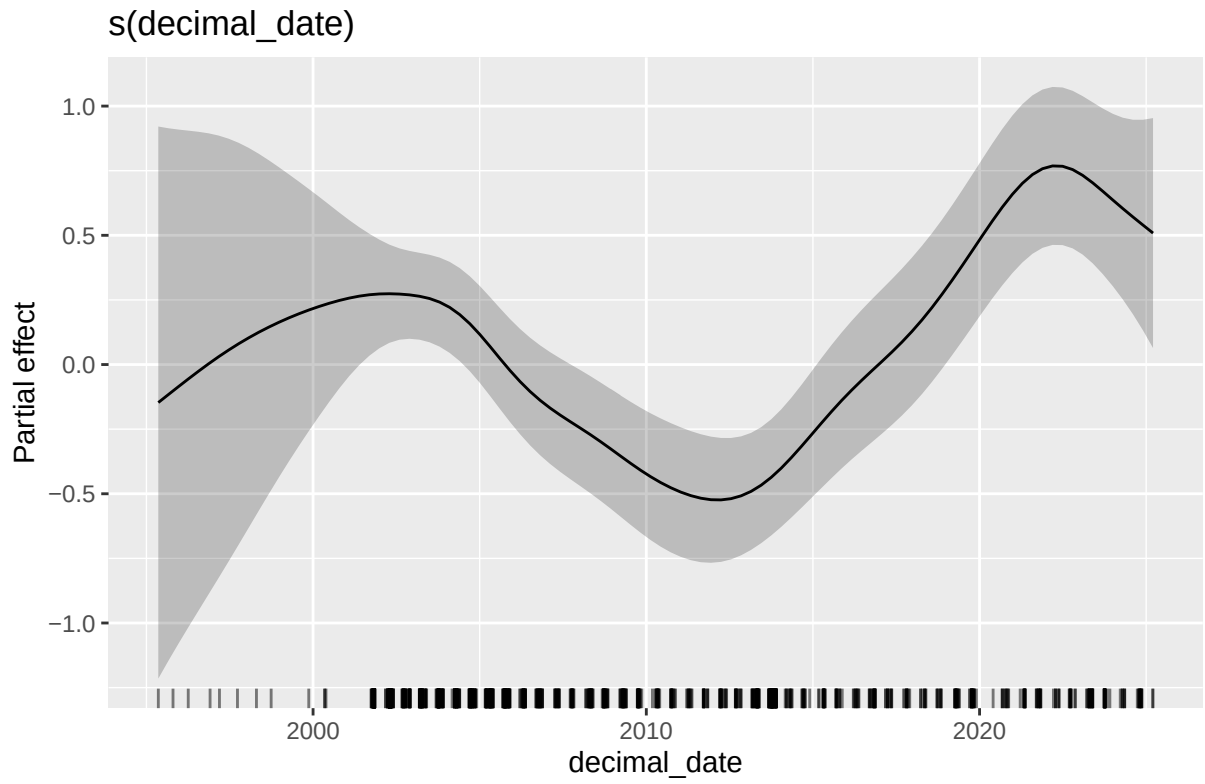



RQR vs Fitted mean (p−scale)
Randomised quantile residual
2.5
0.0
−2.5
−5.0
0.0
2.5
5.0
7.5
10.0
Fitted mean on p−scale


Histogram of RQ residuals (cnorm, p=0.35)
Frequency
150
100
50
0
−5.0
−2.5
0.0
2.5
RQ residual




```
##
## Residual spread - SD(RQR): 0.95 ( 1 ideal)
```



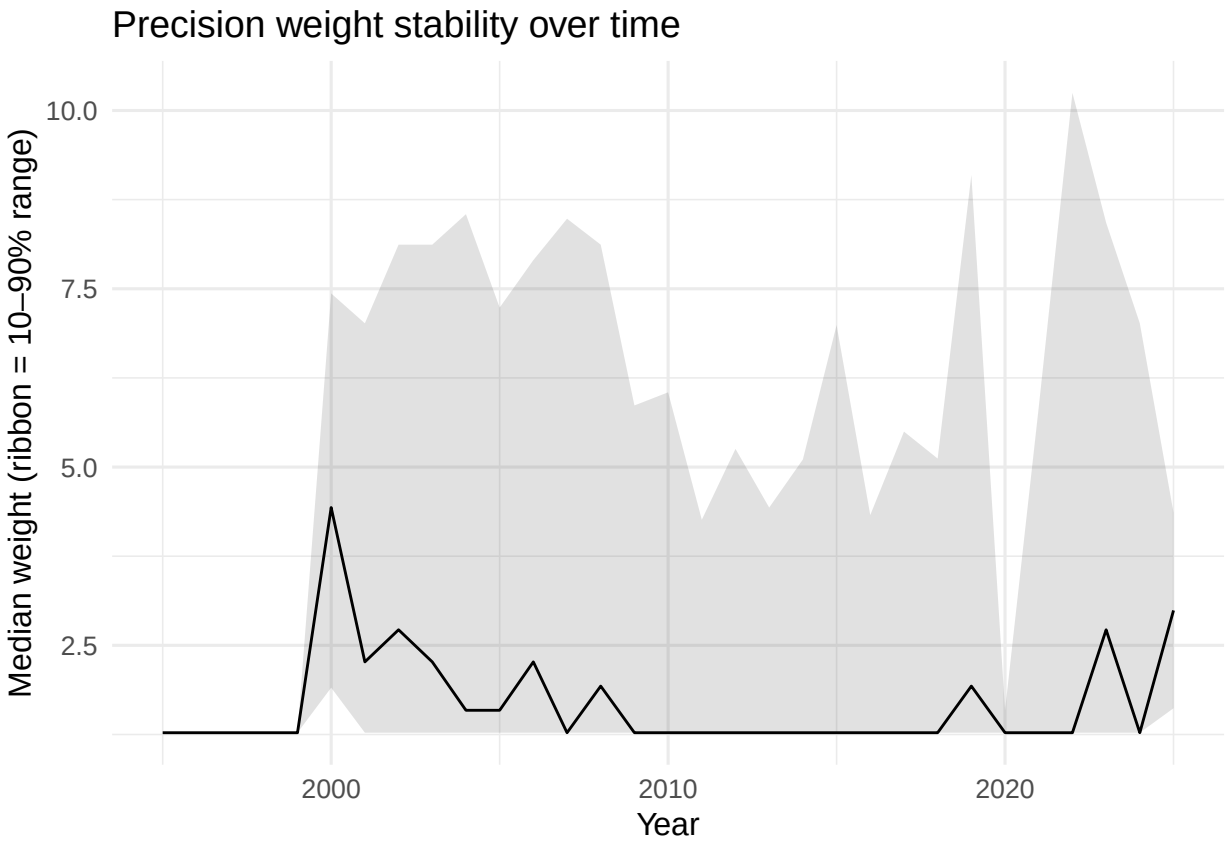

```
## Deviance explained: 78.3% | AIC: 5838.8 | N: 1089 | sigma(p-scale): 1.000
```

```
# ================================
# Minor checks for p = 0.35 cnorm
# --------------------------------
# Purpose of this script:
#   • Recreate the per-row precision weights used in the fitted cnorm model.
#   • Check whether those weights are stable over time (summary + quick plot).
#   • Run a leave-one-site-out (for top 10 sites) influence check on the
#     population-level time curve (with site RE excluded in prediction).
#   • Compare the baseline single time smooth vs a season-varying time smooth.
# Notes:
#   • Assumes `res_p035` exists and was returned by `fit_cnorm_p035(...)`.
#   • Assumes `models_data` preprocessing created season_f, decimal_date, SITE_ID.F.
#   • All modelling settings are kept exactly as in your fit; no code is changed.
# ================================

# Extract fitted model and data from the p=0.35 cnorm result object
m  <- res_p035$model
df <- res_p035$data

# Recreate the analysis weights used in the fit
# - width_t: width of the transformed (p-scale) interval = upper_t - lower_t
# - wt: precision weight = 1 / width_t (same exponent as fit: alpha_w = 1)
# - year: helper for time summaries
```

```r
df <- df %>%
  mutate(width_t = pmax(upper_t - lower_t, 1e-6),
         wt      = 1 / width_t,
         year    = year(SAMPLE_DATE))

# ------------------------------------------------
# 1) Weight stability through time (quick summary)
# ------------------------------------------------
# For each year: number of rows, median interval width, and 10th/median/90th
# percentiles of the precision weights. Large temporal drift could indicate
# a change in interval construction or data quality through time.
w_summary <- df %>%
  group_by(year) %>%
  summarise(n = n(),
            med_width = median(width_t),
            p10_wt    = quantile(wt, 0.10),
            med_wt    = median(wt),
            p90_wt    = quantile(wt, 0.90),
            .groups = "drop")
print(w_summary, n = 10)
```

```
## # A tibble: 31 x 6
##     year     n med_width p10_wt med_wt p90_wt
##    <dbl> <int>     <dbl>  <dbl>  <dbl>  <dbl>
##  1  1995     2     0.785   1.27   1.27   1.27
##  2  1996     2     0.785   1.27   1.27   1.27
##  3  1997     2     0.785   1.27   1.27   1.27
##  4  1998     2     0.785   1.27   1.27   1.27
##  5  1999     1     0.785   1.27   1.27   1.27
##  6  2000     3     0.226   1.91   4.43   7.43
##  7  2001    59     0.441   1.27   2.27   7.02
##  8  2002   117     0.368   1.27   2.72   8.12
##  9  2003    87     0.441   1.27   2.27   8.12
## 10  2004    80     0.629   1.27   1.59   8.55
## # i 21 more rows
```

```r
# Optional tiny diagnostic plot (comment out if not needed)
# Median weight per year with an uncertainty ribbon (10-90% quantiles).
ggplot(w_summary, aes(year, med_wt)) +
  geom_ribbon(aes(ymin = p10_wt, ymax = p90_wt), alpha = 0.15) +
  geom_line() +
  labs(title = "Precision weight stability over time",
       x = "Year", y = "Median weight (ribbon = 10-90% range)") +
  theme_minimal(base_size = 12)
```




```
# Spearman correlation between weight and time (should be small)
# If |rho| is large, weights systematically shift over years, which you may want to note.
cor_w_time <- with(df, cor(wt, year, method = "spearman"))
cat(sprintf("\nSpearman(weight, year) = %.3f (|rho| small is good)\n", cor_w_time))
```

```
##
## Spearman(weight, year) = -0.120 (|rho| small is good)
```

```
# ------------------------------------------------
# 2) Leave-one-site-out (top 10 sites) influence
#     on population time curve (site RE excluded)
# ------------------------------------------------
# Build a population-level prediction grid across time:
#  - Fixed season_f = "spring" for a single representative trend;
#  - SITE_ID.F is set to any valid level (random effect is excluded in predict()).
tgrid <- data.frame(
  decimal_date = seq(min(df$decimal_date), max(df$decimal_date), length.out = 200),
  season_f     = factor("spring", levels = levels(df$season_f)),
  SITE_ID.F    = df$SITE_ID.F[1]   # placeholder (RE excluded anyway)
)
# Baseline population curve from the full model (exclude site RE)
pred0 <- as.numeric(predict(m, newdata = tgrid, type = "response",
                            exclude = "s(SITE_ID.F)"))

# Identify the 10 sites with the largest number of observations
```

```r
top_sites <- df %>% count(SITE_ID.F, sort = TRUE) %>% slice_head(n = 10) %>% pull(SITE_ID.F)

# For each top site:
#   • Refit the model dropping that site's data.
#   • Predict the population curve on the same tgrid (exclude site RE).
#   • Compute max/median absolute differences relative to the baseline curve.
loso_tab <- map_df(top_sites, function(sid) {
  d2 <- df %>% filter(SITE_ID.F != sid)
  ny <- n_distinct(year(d2$SAMPLE_DATE))
  k_time <- min(35, max(12, round(0.7 * ny)))
  m2 <- bam(
    cbind(lower_t, upper_t) ~ season_f + s(decimal_date, k = k_time) + s(SITE_ID.F, bs = "re"),
    family   = cnorm(),
    data     = d2,
    weights  = (1 / pmax(d2$upper_t - d2$lower_t, 1e-6)),
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )
  p2 <- as.numeric(predict(m2, newdata = tgrid, type = "response",
                           exclude = "s(SITE_ID.F)"))
  tibble(site = as.character(sid),
         max_abs_diff = max(abs(p2 - pred0)),
         med_abs_diff = median(abs(p2 - pred0)))
})
print(loso_tab)
```

```
## # A tibble: 10 x 3
##    site  max_abs_diff med_abs_diff
##    <chr>        <dbl>        <dbl>
##  1 52504       0.842       0.0853
##  2 50094       0.0721      0.00931
##  3 52988       0.0687      0.00661
##  4 51896       0.992       0.108
##  5 46924       0.0641      0.00828
##  6 64118       0.144       0.0646
##  7 1327        0.0793      0.00618
##  8 51324       0.384       0.0663
##  9 48468       0.0234      0.00671
## 10 51634       0.178       0.0376
```

```r
# ------------------------------------------------
# 3) Season-varying trend vs single trend (model test)
# ------------------------------------------------
# Compare:
#   Base model  : single smooth over time (same as in res_p035 fit)
#   Alternate   : season-varying smooth s(decimal_date, by = season_f)
# Report AIC and deviance explained; ΔAIC indicates which is preferred.
ny <- n_distinct(df$year)
k_time <- min(35, max(12, round(0.7 * ny)))

m_alt <- bam(
```

```
  cbind(lower_t, upper_t) ~ season_f +
    s(decimal_date, by = season_f, k = k_time) +
    s(SITE_ID.F, bs = "re"),
  family   = cnorm(),
  data     = df,
  weights  = (1 / pmax(df$upper_t - df$lower_t, 1e-6)),
  method   = "fREML",
  discrete = TRUE,
  select   = TRUE,
  gamma    = 1.2
)

# Gather headline metrics for both models and print a concise comparison
base_AIC <- AIC(m); alt_AIC <- AIC(m_alt)
base_dev <- summary(m)$dev.expl; alt_dev <- summary(m_alt)$dev.expl

cat(sprintf("\nModel comparison (cnorm p=0.35)\n  Base (single time smooth):   AIC = %.1f, DevExpl = %.
            base_AIC, 100*base_dev, alt_AIC, 100*alt_dev, alt_AIC - base_AIC,
            ifelse(alt_AIC < base_AIC, "season-varying", "single-smooth")))
```

```
##
## Model comparison (cnorm p=0.35)
##   Base (single time smooth):   AIC = 5838.8, DevExpl = 78.3%
##   Alt  (season-varying time): AIC = 5810.2, DevExpl = 79.2%
##   ΔAIC (alt - base) = -28.6  -> prefer season-varying
```

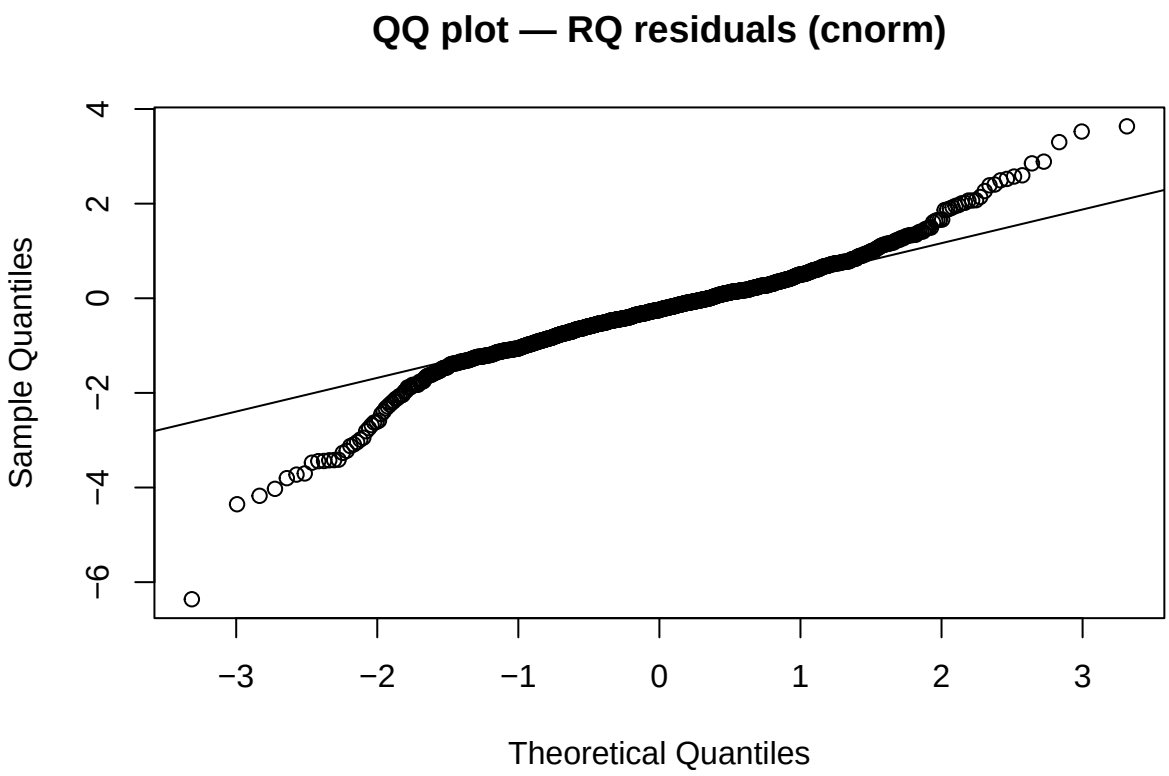
#4.4.3 CNORM MODEL(P=O.35 WITH SEASON VARYING TREND)

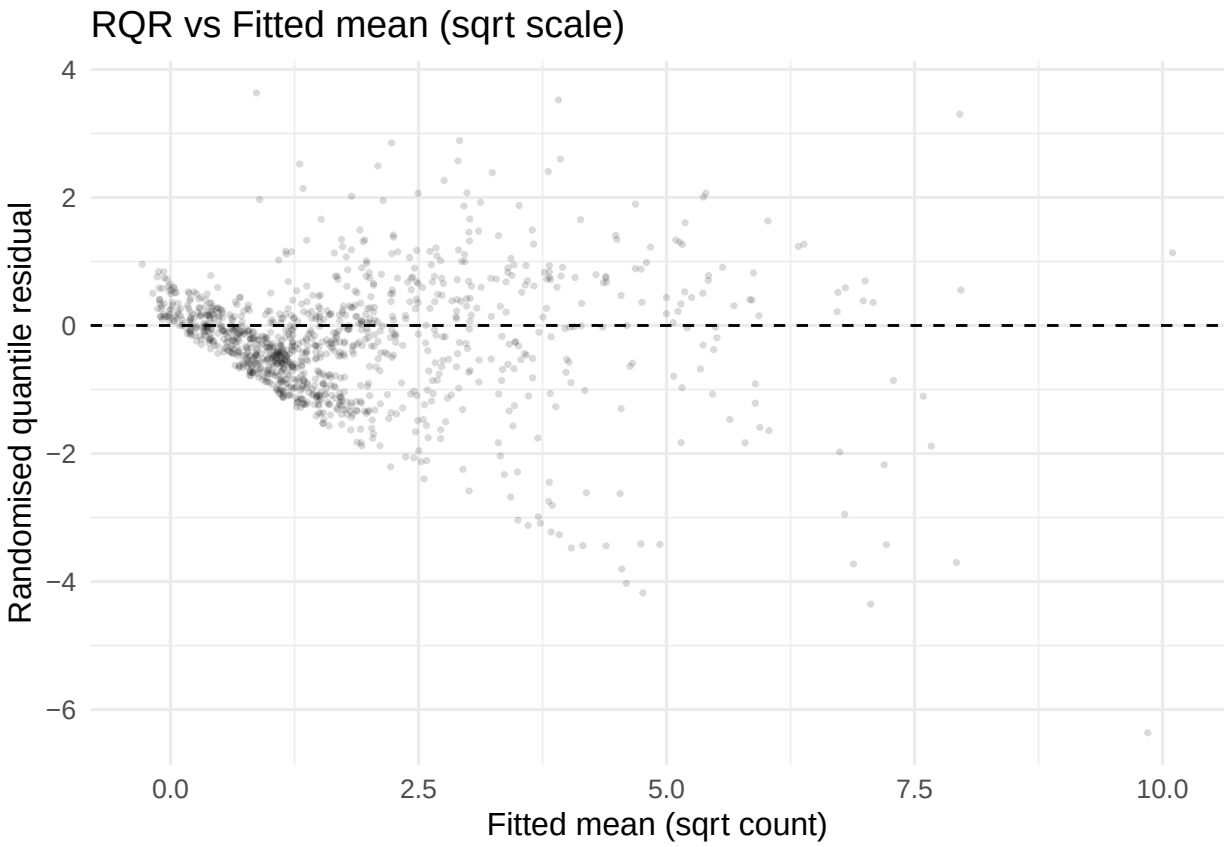
```
# ================================================
# Cnorm (p = 0.35) with season-varying trend
# ------------------------------------------------
# What this script does:
#   • Fits a censored-normal GAMM on a power-transformed scale (p = 0.35).
#   • Uses precision weights based on interval width (narrower = higher weight).
#   • Includes a season main effect and a season-varying smooth trend over time.
#   • Adds a site random intercept (s(SITE_ID.F, bs="re")).
#   • Runs diagnostics (via your existing diagnose_cnorm()).
#   • Produces population-level trend curves (random effects excluded) and a plot.
#
# Expected columns in `cn_df`:
#   lower, upper          -> raw 1-s.f. bounds on the count scale
#   lower_t, upper_t      -> transformed bounds (if already present)
#   season_f              -> factor with levels c("spring","autumn")
#   decimal_date          -> numeric time (e.g., 1995.5)
#   SITE_ID.F             -> site factor used for the random effect
#   SAMPLE_DATE           -> Date, used to size the smooth basis (k)
# ================================================

# cn_df: your family-specific censored-normal table with
# columns lower, upper, lower_t, upper_t, season_f, decimal_date, SITE_ID.F, SAMPLE_DATE

fit_cnorm_seasonvary <- function(df, p = 0.35, alpha_w = 1) {
  # Ensure transformed bounds exist on the p-scale:
```

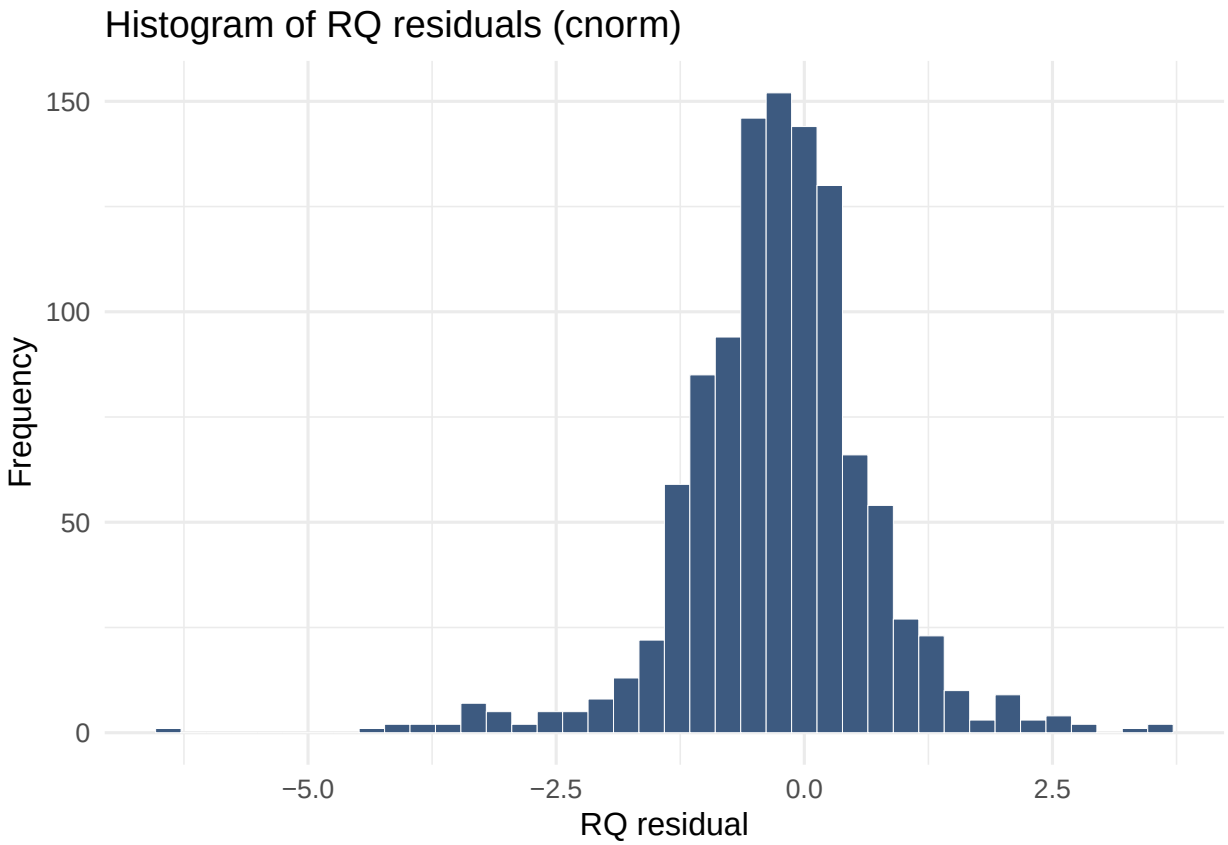
```r
  # If lower_t/upper_t were not precomputed, create them from raw bounds.
  if (!all(c("lower_t","upper_t") %in% names(df))) {
    df <- df %>% mutate(
      lower_t = pmax(as.numeric(lower), 0)^p,  # guard against negatives before transform
      upper_t = pmax(as.numeric(upper), 0)^p
    )
  }
  # Precision weights: intervals with smaller width on the p-scale carry more weight.
  # Raising by alpha_w lets you increase/decrease emphasis on narrow intervals.
  wts <- (1 / pmax(df$upper_t - df$lower_t, 1e-6))^alpha_w

  # Choose the size of the time smooth (k) from the span of years in the data.
  k_time <- {
    ny <- n_distinct(lubridate::year(df$SAMPLE_DATE))
    min(35, max(12, round(0.7 * ny)))
  }

  # Fit the cnorm BAM:
  #   • season_f main effect
  #   • season-varying smooth of decimal_date
  #   • site random intercept
  #   • interval-width weights
  m <- bam(
    cbind(lower_t, upper_t) ~ season_f +
      s(decimal_date, by = season_f, k = k_time) +
      s(SITE_ID.F, bs = "re"),
    family   = cnorm(),
    data     = df,
    weights  = wts,
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  # Return the fitted model, data (with transformed bounds), and meta info
  list(model = m, data = df, p = p, k_time = k_time)
}

# ---- Fit the final model for the current family
# `cn_df` must already be defined in your environment as described above.
final <- fit_cnorm_seasonvary(cn_df, p = 0.35)

# ---- Run your existing cnorm diagnostics on the final model
# Uses your previously defined diagnose_cnorm(fit, data) helper.
diagnose_cnorm(final$model, final$data)
```

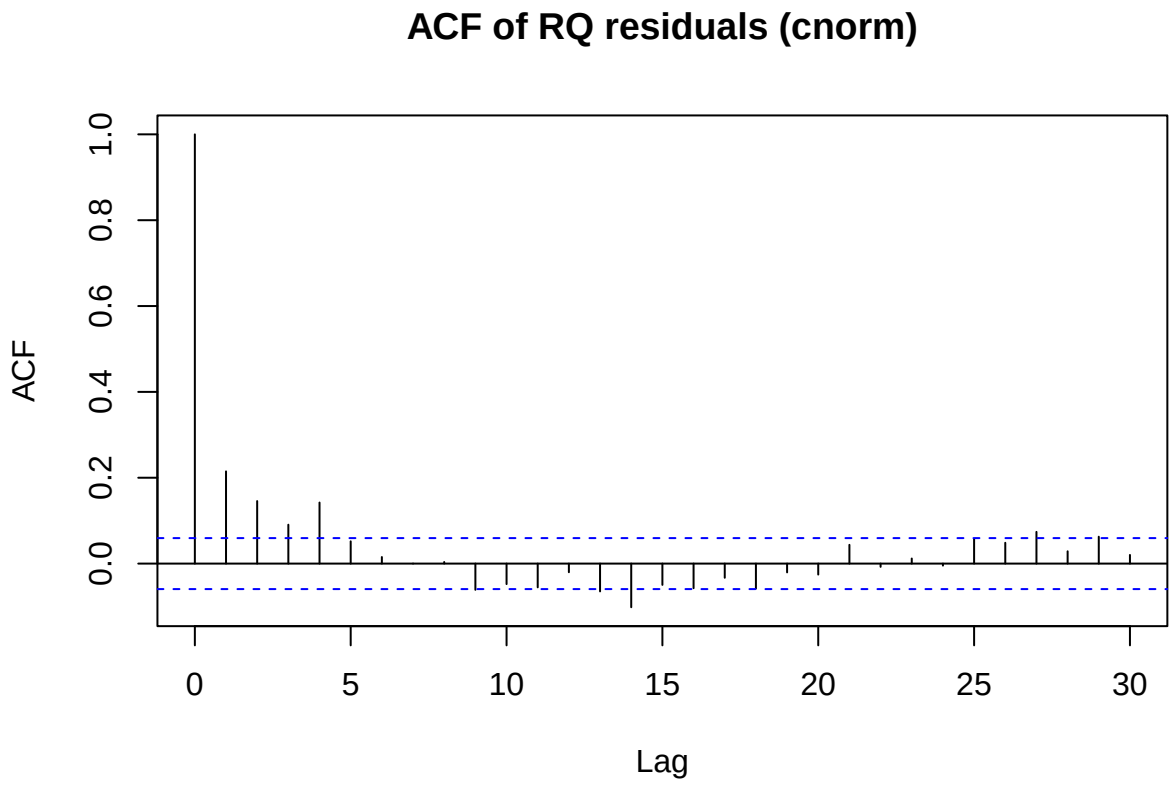
```
##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  2.302381e-12 -6.270876e-06  5.355716e-12 -8.664558e-06  8.392220e-10
##
## $hess
```

```
##               [,1]          [,2]          [,3]         [,4]          [,5]
## [1,]  1.011673e+00 -4.641615e-06  8.298455e-03 8.026674e-08 -6.032370e-02
## [2,] -4.641615e-06  6.270798e-06 -6.133146e-09 6.128533e-13  2.689780e-07
## [3,]  8.298455e-03 -6.133146e-09  9.013057e-01 3.783793e-06  6.712258e-01
## [4,]  8.026674e-08  6.128533e-13  3.783793e-06 8.664700e-06  2.708971e-06
## [5,] -6.032370e-02  2.689780e-07  6.712258e-01 2.708971e-06  1.011438e+02
##
## Model rank =  358 / 358
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  21.00   3.14    0.94    0.10 .
## s(decimal_date):season_fautumn  21.00  10.27    0.94    0.08 .
## s(SITE_ID.F)                   314.00 219.09      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## $mfrow
## [1] 2 2
##
##
## Concurvity (rounded):
##          para s(decimal_date):season_fspring s(decimal_date):season_fautumn
## worst       1                       0.7468341                       0.7273840
## observed    1                       0.2738995                       0.3541170
## estimate    1                       0.4986010                       0.5438437
##          s(SITE_ID.F)
## worst      1.00000000
## observed   0.04708978
## estimate   0.04944612
```

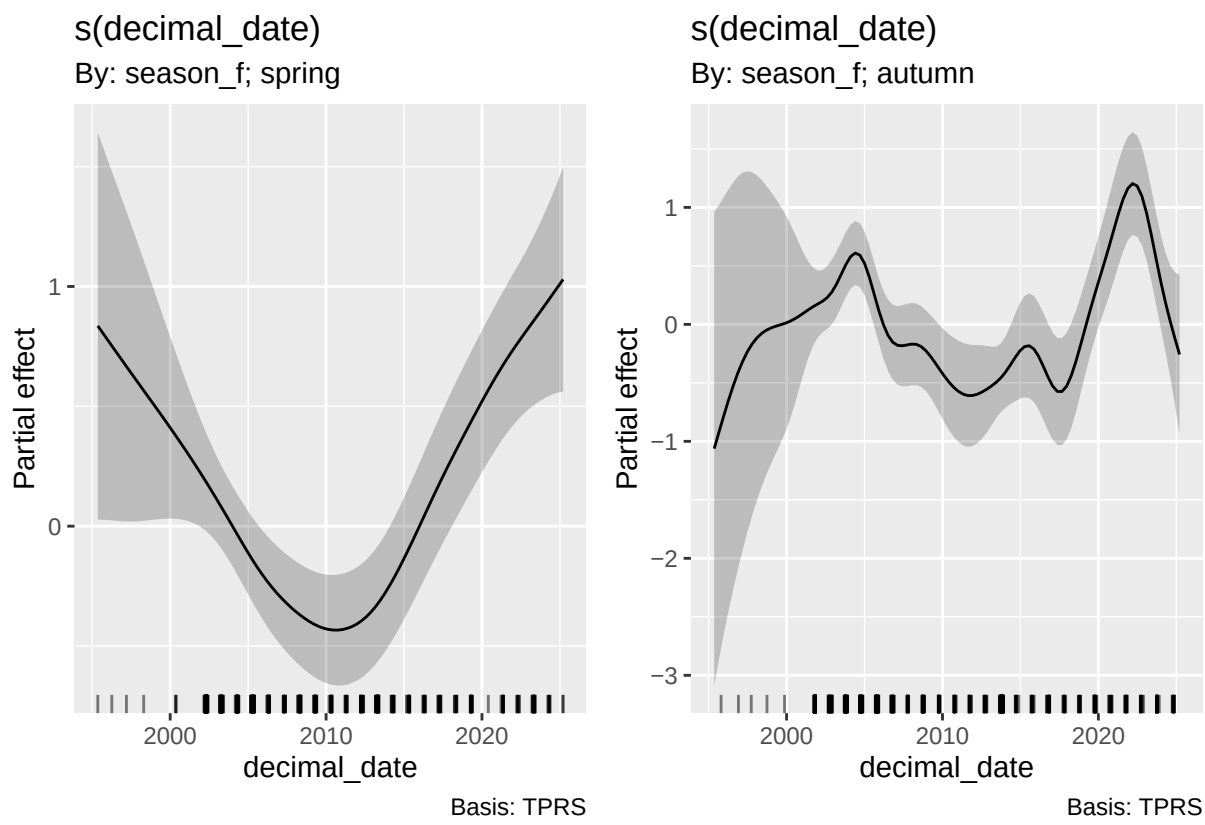

QQ plot — RQ residuals (cnorm)
Sample Quantiles
Theoretical Quantiles
4
2
0
−2
−4
−6
−3
−2
−1
0
1
2
3


RQR vs Fitted mean (sqrt scale)
Randomised quantile residual
4
2
0
−2
−4
−6
0.0
2.5
5.0
7.5
10.0
Fitted mean (sqrt count)


Histogram of RQ residuals (cnorm)
Frequency
150
100
50
0
−5.0
−2.5
0.0
2.5
RQ residual



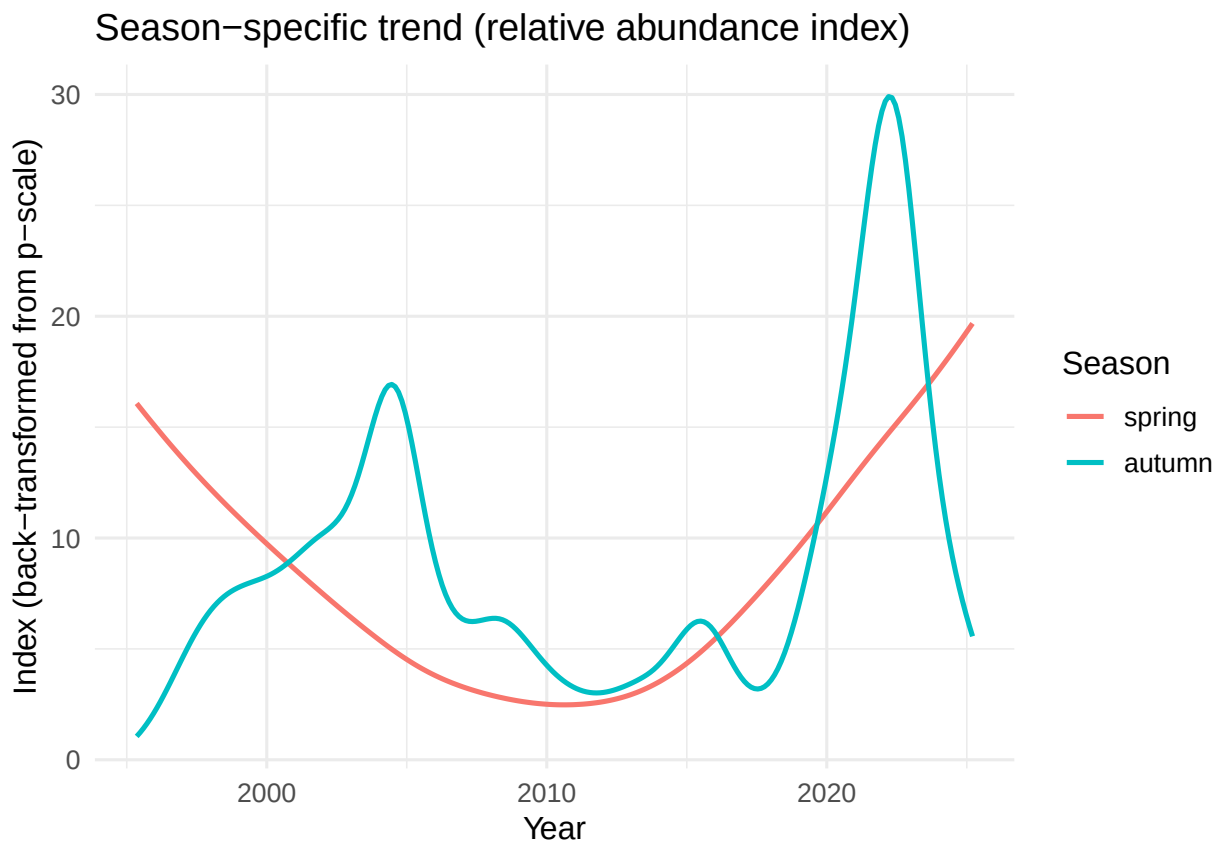

```
##
## Residual spread - SD(RQR): 0.95 ( 1 ideal)
```




```
## Deviance explained: 79.2% | AIC: 5810.2 | N: 1089 | sigma(sqrt-scale): 1.000
```

```r
# ---- Season-specific trend curves (population-level; RE excluded)
# Builds a time grid for each season, makes predictions on the p-scale with
# the site random effect excluded, then back-transforms to a relative index.
make_trend_curves <- function(fit, df, p = 0.35) {
  tgrid <- expand.grid(
    decimal_date = seq(min(df$decimal_date), max(df$decimal_date), length.out = 250),
    season_f     = levels(df$season_f)
  )
  # Dummy site factor value (any valid level is fine since we exclude the RE)
  tgrid$SITE_ID.F <- df$SITE_ID.F[1]

  # Predict on the p-scale; exclude s(SITE_ID.F) to remove random-effect influence.
  pred_p <- as.numeric(predict(fit, newdata = tgrid, type = "response",
                               exclude = "s(SITE_ID.F)"))
  # Back-transform to an index on the original count scale.
  # Note: (E[X^p])^(1/p) is a comparable index, not an unbiased mean count.
  tgrid$index_count <- pmax(pred_p, 0)^(1/p)
  tgrid
}

# Build the trend dataframe and plot the season-specific indices through time
trend_df <- make_trend_curves(final$model, final$data, p = final$p)
```

```r
ggplot(trend_df, aes(decimal_date, index_count, colour = season_f)) +
  geom_line(linewidth = 0.9) +
  labs(title = "Season-specific trend (relative abundance index)",
       x = "Year", y = "Index (back-transformed from p-scale)",
       colour = "Season") +
  theme_minimal(base_size = 12)
```




```r
# ---- Quick headline metrics (concise model summary)
sum_final <- summary(final$model)
cat(sprintf("\nFINAL (season-varying) - Dev.expl: %.1f%% | AIC: %.1f | k_time: %d\n",
            100*sum_final$dev.expl, AIC(final$model), final$k_time))
```

```
##
## FINAL (season-varying) - Dev.expl: 79.2% | AIC: 5810.2 | k_time: 22
```

```r
# ============================================================
# Batch cnorm(p = 0.35) for 4 families
#  - Prints full summary() and gam.check() text for each family
#  - No diagnostic plots are shown
#  - Returns/prints a compact summary table at the end
# Requirements:
#   models_data[["<family>"]]$cnorm  (with lower/upper bounds, season_f,
#   decimal_date, SITE_ID.F, SAMPLE_DATE)
# ------------------------------------------------
```

```r
# What this script provides:
#   • A helper to power-transform interval bounds and build precision weights.
#   • A helper to compute randomised-quantile residuals for cnorm fits.
#   • A family fitter that:
#       - guards for missing season/site factors,
#       - sizes the time-smooth basis from the year span,
#       - fits a season-varying time smooth + site RE with weights,
#       - prints textual diagnostics (summary + gam.check text),
#       - returns a compact list for tabulation.
#   • A loop over your four families and a tidy summary table.
# Notes:
#   • No figures are produced here (gam.check plots are diverted to a temp PDF).
#   • The weights favour narrower transformed intervals (1/width on p-scale).
#   • The "index of fit tightness" we print is SD of RQ residuals (ideal   1).
# ================================================================

FAMILIES <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")

# --- helpers ------------------------------------------------------

# Ensure p-power transformed bounds & precision weights
# -----------------------------------------------------
# Expects raw count-scale interval columns `lower` and `upper`.
# Produces:
#   • lower_t, upper_t  -> bounds on the p-scale (p=0.35),
#   • width_t           -> interval width on p-scale (clamped at 1e-6),
#   • wt                -> precision weight = 1/width_t,
#   • year              -> convenience field for choosing k.
ensure_bounds_p <- function(df, p = 0.35) {
  stopifnot(all(c("lower","upper") %in% names(df)))
  df %>%
    mutate(
      lower_t = pmax(as.numeric(lower), 0)^p,
      upper_t = pmax(as.numeric(upper), 0)^p,
      width_t = pmax(upper_t - lower_t, 1e-6),
      wt      = 1 / width_t,                 # narrower intervals => larger weight
      year    = year(SAMPLE_DATE)
    )
}

# Randomised-quantile residuals for censored Normal (on p-scale)
# --------------------------------------------------------------
# Given a fitted cnorm model and transformed bounds (lt/ut),
# draw a uniform from the conditional CDF slice and map via qnorm.
# Use as a quick dispersion/shape diagnostic (SD   1 is ideal).
cnorm_rqr <- function(fit, lt, ut, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- sqrt(summary(fit)$scale)
  Flo <- pnorm(lt, mean = mu, sd = sd)
  Fup <- pnorm(ut, mean = mu, sd = sd)
  set.seed(seed)
  u <- pmin(pmax(Flo + pmax(Fup - Flo, 0) * runif(length(mu)), eps), 1 - eps)
  qnorm(u)
```

```r
}

# Fit one family and PRINT summary + gam.check (no plots)
# ------------------------------------------------------
# Inputs:
#   fam     -> family name key in models_data
#   md      -> list-like container with models_data[[fam]]$cnorm
#   p       -> power for transformation (fixed to 0.35 as per design)
# Side effects:
#   • Prints model summary and textual gam.check output.
# Returns:
#   • Compact list (model, data, diagnostics) for tabulation.
fit_cnorm_family_print <- function(fam, md = models_data, p = 0.35) {
  stopifnot(!is.null(md[[fam]]), "cnorm" %in% names(md[[fam]]))
  df <- md[[fam]]$cnorm %>% ensure_bounds_p(p)

  # guard rails: season factor & site factor
  # ------------------------------------------------
  # If season_f was not created upstream, derive spring/autumn from months.
  # Ensure site IDs are a proper factor for the random intercept term.
  if (!("season_f" %in% names(df))) {
    df <- df %>%
      mutate(m = month(SAMPLE_DATE),
             season_f = factor(if_else(m %in% 3:5, "spring", "autumn"),
                               levels = c("spring","autumn")))
  }
  if (!is.factor(df$SITE_ID.F)) df$SITE_ID.F <- factor(df$SITE_ID)

  # choose k from span of years
  # ------------------------------------------------
  # Basis dimension increases with the number of distinct sampling years,
  # but is clamped to [12, 35] to avoid under/over-fitting extremes.
  k_time <- {
    ny <- n_distinct(df$year)
    min(35, max(12, round(0.7 * ny)))
  }

  # season-varying time smooth + site RE; precision weights
  # ------------------------------------------------------
  # The cnorm family takes cbind(lower_t, upper_t) as the response.
  # We include:
  #   • season_f (main effect),
  #   • s(decimal_date, by = season_f, k = k_time) for different trends by season,
  #   • s(SITE_ID.F, bs="re") for a site random intercept,
  #   • df$wt to emphasize tighter intervals on the p-scale.
  m <- bam(
    cbind(lower_t, upper_t) ~ season_f +
      s(decimal_date, by = season_f, k = k_time) +
      s(SITE_ID.F, bs = "re"),
    family   = cnorm(),
    data     = df,
    weights  = df$wt,
    method   = "fREML",
```

```
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  # ---- PRINTED OUTPUTS (no plots) ----
  cat("\n======================================================\n")
  cat(sprintf("Family: %s  |  cnorm(p = %.2f)\n", fam, p))
  cat("======================================================\n")
  print(summary(m))

  # gam.check text without plotting the 2x2 panel
  # ------------------------------------------------
  # We divert any plots into a temporary PDF (and delete it),
  # keeping only the textual adequacy checks in the console.
  tmp <- tempfile(fileext = ".pdf"); pdf(tmp)
  gc_out <- try(gam.check(m, rep = 0), silent = TRUE)
  dev.off(); unlink(tmp)
  if (!inherits(gc_out, "try-error")) {
    cat("\n--- gam.check (text) ---\n")
    print(gc_out)
  }

  # simple numeric diagnostics
  # ------------------------------------------------
  # SD of RQ residuals (ideal  1), deviance explained, AIC, N, and k used.
  rqr_sd <- sd(cnorm_rqr(m, df$lower_t, df$upper_t), na.rm = TRUE)
  devex  <- summary(m)$dev.expl
  aic    <- AIC(m)

  cat(sprintf("\nSD(RQ residuals): %.2f | Dev.expl: %.1f%% | AIC: %.1f | N: %d | k_time: %d\n",
              rqr_sd, 100*devex, aic, nobs(m), k_time))

  # return compact object for summary table
  list(
    family   = fam,
    model    = m,
    data     = df,
    rqr_sd   = rqr_sd,
    dev_expl = devex,
    AIC      = aic,
    n        = nobs(m),
    k_time   = k_time
  )
}

# --- RUN for all families ---------------------------------------
# Map the fitter across the four families and name the output list
# for convenient downstream access (cn_results[["Aphelocheiridae"]] etc.).
cn_results <- map(FAMILIES, fit_cnorm_family_print) %>% set_names(FAMILIES)
```

```
##
## ======================================================
```

```
## Family: Aphelocheiridae  |  cnorm(p = 0.35)
## ==================================================
##
## Family: cnorm(1.851)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.8076     0.1089  16.606  < 2e-16 ***
## season_fautumn   0.2734     0.0773   3.537 0.000427 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df      F p-value
## s(decimal_date):season_fspring   3.144     21  8.632 1.7e-06 ***
## s(decimal_date):season_fautumn  10.268     21 20.883 < 2e-16 ***
## s(SITE_ID.F)                   219.092    313 10.718 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.732   Deviance explained = 79.2%
## fREML = 2601.6  Scale est. = 1         n = 1089


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  2.302381e-12 -6.270876e-06  5.355716e-12 -8.664558e-06  8.392220e-10
##
## $hess
##               [,1]          [,2]          [,3]         [,4]          [,5]
## [1,]  1.011673e+00 -4.641615e-06  8.298455e-03 8.026674e-08 -6.032370e-02
## [2,] -4.641615e-06  6.270798e-06 -6.133146e-09 6.128533e-13  2.689780e-07
## [3,]  8.298455e-03 -6.133146e-09  9.013057e-01 3.783793e-06  6.712258e-01
## [4,]  8.026674e-08  6.128533e-13  3.783793e-06 8.664700e-06  2.708971e-06
## [5,] -6.032370e-02  2.689780e-07  6.712258e-01 2.708971e-06  1.011438e+02
##
## Model rank =  358 / 358
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
## s(decimal_date):season_fspring  21.00   3.14    0.94    0.10 .
## s(decimal_date):season_fautumn  21.00  10.27    0.94    0.08 .
## s(SITE_ID.F)                   314.00 219.09      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
```

```
## $mfrow
## [1] 2 2
##
##
## SD(RQ residuals): 0.95 | Dev.expl: 79.2% | AIC: 5810.2 | N: 1089 | k_time: 22
##
## ====================================================
## Family: Brachycentridae  |  cnorm(p = 0.35)
## ====================================================
##
## Family: cnorm(1.697)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.87336    0.04566   19.13   <2e-16 ***
## season_fautumn  0.62374    0.03780   16.50   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df      F p-value
## s(decimal_date):season_fspring  17.81     24 15.254  <2e-16 ***
## s(decimal_date):season_fautumn   7.80     24 16.104  <2e-16 ***
## s(SITE_ID.F)                   613.08   1109  7.158  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.597   Deviance explained = 63.3%
## fREML =  11948  Scale est. = 1         n = 5917


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -5.718093e-12 -4.229885e-05  2.115197e-12 -4.937858e-05  1.182116e-09
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  3.455324e+00 -1.910345e-05  9.880315e-03 -7.952269e-09  1.627180e-01
## [2,] -1.910345e-05  4.229531e-05  1.033282e-07  8.726602e-13 -1.193494e-06
## [3,]  9.880315e-03  1.033282e-07  6.269889e-01 -3.892090e-05 -1.069222e-01
## [4,] -7.952269e-09  8.726602e-13 -3.892090e-05  4.937370e-05  2.263908e-06
## [5,]  1.627180e-01 -1.193494e-06 -1.069222e-01  2.263908e-06  2.795255e+02
##
## Model rank =  1160 / 1160
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
```

```
## s(decimal_date):season_fspring   24.0   17.8    0.92   0.005 **
## s(decimal_date):season_fautumn   24.0    7.8    0.92  <2e-16 ***
## s(SITE_ID.F)                   1110.0  613.1      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQ residuals): 0.83 | Dev.expl: 63.3% | AIC: 27685.9 | N: 5917 | k_time: 25
##
## ====================================================
## Family: Odontoceridae  |  cnorm(p = 0.35)
## ====================================================
##
## Family: cnorm(0.612)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.01445    0.01829  55.480   <2e-16 ***
## season_fautumn  0.01771    0.01485   1.193    0.233
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                   edf Ref.df      F p-value
## s(decimal_date):season_fspring   8.370     23 22.104  <2e-16 ***
## s(decimal_date):season_fautumn   5.921     23 13.347  <2e-16 ***
## s(SITE_ID.F)                   920.031   1297  6.798  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.607   Deviance explained = 62.2%
## fREML = 8582.5  Scale est. = 1         n = 6674


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -5.514895e-10 -4.170785e-05 -1.563771e-10 -1.636469e-11 -1.832580e-07
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  7.735952e-01 -2.205786e-05 -4.697875e-02 -1.759320e-04 -3.791870e-01
## [2,] -2.205786e-05  4.170529e-05 -2.911592e-07 -1.471006e-07  5.294832e-06
## [3,] -4.697875e-02 -2.911592e-07  1.387287e+00  5.745210e-02 -6.432059e-02
## [4,] -1.759320e-04 -1.471006e-07  5.745210e-02  7.725794e-02 -4.211768e-03
## [5,] -3.791870e-01  5.294832e-06 -6.432059e-02 -4.211768e-03  3.549236e+02
```

```
##
## Model rank =  1346 / 1346
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'     edf k-index p-value
## s(decimal_date):season_fspring   23.00    8.37     0.9   0.005 **
## s(decimal_date):season_fautumn   23.00    5.92     0.9  <2e-16 ***
## s(SITE_ID.F)                   1298.00  920.03      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQ residuals): 0.44 | Dev.expl: 62.2% | AIC: 18901.2 | N: 6674 | k_time: 24
##
## ======================================================
## Family: Cordulegastridae  |  cnorm(p = 0.35)
## ======================================================
##
## Family: cnorm(0.278)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.703959   0.025180  27.957   <2e-16 ***
## season_fautumn -0.009081   0.021556  -0.421    0.674
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F  p-value
## s(decimal_date):season_fspring   1.941     21 6.368 0.000359 ***
## s(decimal_date):season_fautumn   2.704     21 8.779 0.000409 ***
## s(SITE_ID.F)                   251.197    378 6.022  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.783   Deviance explained = 82.6%
## fREML =  828.5  Scale est. = 1         n = 1064


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -2.835515e-09 -4.769194e-06 -1.834308e-07 -2.244357e-06 -6.384391e-06
##
```

```
## $hess
##               [,1]         [,2]          [,3]          [,4]          [,5]
## [1,]  5.405099e-01 7.005052e-06 -3.587495e-02  6.276987e-07  8.942884e-01
## [2,]  7.005052e-06 4.770583e-06  1.552500e-06  6.272036e-11  2.384613e-05
## [3,] -3.587495e-02 1.552500e-06  6.167712e-01 -5.667918e-06  9.865801e-02
## [4,]  6.276987e-07 6.272036e-11 -5.667918e-06  2.244645e-06 -8.273712e-06
## [5,]  8.942884e-01 2.384613e-05  9.865801e-02 -8.273712e-06  1.126834e+02
##
## Model rank =  423 / 423
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(decimal_date):season_fspring  21.00   1.94    0.85   0.005 **
## s(decimal_date):season_fautumn  21.00   2.70    0.85   0.025 *
## s(SITE_ID.F)                   379.00 251.20      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQ residuals): 0.29 | Dev.expl: 82.6% | AIC: 1575.8 | N: 1064 | k_time: 22
```

```
# --- SUMMARY TABLE ------------------------------------------------
# Bind the compact diagnostics from each family into a single table and
# print a clean, rounded view suitable for a log or appendix.
cn_summary <- bind_rows(lapply(cn_results, function(x) {
  data.frame(
    family   = x$family,
    n        = x$n,
    k_time   = x$k_time,
    rqr_sd   = round(x$rqr_sd, 2),
    dev_expl = round(100 * x$dev_expl, 1),
    AIC      = round(x$AIC, 1),
    stringsAsFactors = FALSE
  )
}))
cat("\n==================== Summary table (cnorm, p=0.35) ====================\n")
```

```
##
## ==================== Summary table (cnorm, p=0.35) ====================
```

```
print(cn_summary, row.names = FALSE)
```

```
##            family    n k_time rqr_sd dev_expl     AIC
##   Aphelocheiridae 1089     22   0.95     79.2  5810.2
##   Brachycentridae 5917     25   0.83     63.3 27685.9
##     Odontoceridae 6674     24   0.44     62.2 18901.2
##  Cordulegastridae 1064     22   0.29     82.6  1575.8
```

```r
# ============================================================
# Quick check: cnorm with p = 0.35 and LIGHTER interval weighting
# ------------------------------------------------------------
# What this script does
#   • Keeps your chosen power p = 0.35 (transforming count-scale bounds).
#   • Uses precision weights raised to alpha_w = 0.5 (so wide intervals are
#     still down-weighted, but less aggressively than alpha_w = 1).
#   • Fits the same cnorm GAMM form as before (season-varying smooth + site RE).
#   • Reports SD of randomised-quantile residuals ( 1 ideal), AIC, deviance
#     explained and k used for the time smooth, for each family.
#
# Inputs expected (built earlier in your pipeline)
#   • models_data[["<family>"]]$cnorm with: lower/upper (count-scale bounds),
#     season_f, decimal_date, SITE_ID.F, SAMPLE_DATE (and optionally lower_t/upper_t).
# ============================================================

# Families to evaluate (same set you've been using)
families <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")

# ------------------------------------------------------------
# Helper: extract sigma from a fitted cnorm model
#   • mgcv prints the family as "cnorm(<sigma>)" in the family string.
#   • We parse that out so we can compute RQ residuals with the correct SD.
# ------------------------------------------------------------
get_sigma <- function(fit) {
  fam_str <- fit$family$family
  as.numeric(sub(".*cnorm\\(([^)]+)\\).*", "\\1", fam_str))
}

# ------------------------------------------------------------
# Helper: SD of randomised-quantile residuals on the p-scale
#   • Computes PIT for the censored Normal on transformed scale using
#     the model's fitted mean (mu) and sigma from get_sigma().
#   • Returns the SD of qnorm(U); ideal value is ~1 if dispersion is OK.
# ------------------------------------------------------------
rqr_sd_corrected <- function(fit, lt, ut) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- get_sigma(fit)
  u  <- pmin(pmax(pnorm(lt, mu, sd) + (pnorm(ut, mu, sd) - pnorm(lt, mu, sd)) * runif(length(mu)), 1e-1
  sd(qnorm(u), na.rm = TRUE)
}

# ------------------------------------------------------------
# Core fitter for one dataset with alpha_w = 0.5
#   Arguments:
#     df       : family-specific cnorm table (from models_data[[fam]]$cnorm)
#     p        : power for transforming raw bounds (default 0.35)
#     alpha_w  : exponent for precision weights (default 0.5 here)
#
#   Steps:
#     1) Ensure transformed bounds exist (lower_t/upper_t on p-scale).
#     2) Build precision weights = (1/width)^alpha_w on p-scale.
#     3) Choose k for time smooth from span of years.
```

```r
#     4) Fit cnorm with season main effect, season-varying time smooth,
#        and site random intercept; use weights.
#     5) Return compact metrics for quick comparison.
# ------------------------------------------------------------------
fit_cnorm_alpha <- function(df, p = 0.35, alpha_w = 0.5) {
  if (!all(c("lower_t","upper_t") %in% names(df))) {
    df <- df %>% mutate(lower_t = pmax(as.numeric(lower),0)^p,
                        upper_t = pmax(as.numeric(upper),0)^p)
  }
  df <- df %>% mutate(wt = (1 / pmax(upper_t - lower_t, 1e-6))^alpha_w,
                      year = year(SAMPLE_DATE))
  ny <- n_distinct(df$year)
  k_time <- min(35, max(12, round(0.7 * ny)))
  m <- bam(
    cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f, k = k_time) + s(SITE_ID.F, bs =
    family   = cnorm(), data = df, weights  = df$wt,
    method   = "fREML", discrete = TRUE, select = TRUE, gamma = 1.2
  )
  data.frame(
    SD_RQR = round(rqr_sd_corrected(m, df$lower_t, df$upper_t), 2),
    AIC    = round(AIC(m), 1),
    DevExpl = round(100*summary(m)$dev.expl, 1),
    k_time = k_time
  )
}

# ------------------------------------------------------------------
# Run for all four families and print a tidy comparison table
#   • Each row: family, SD(RQR), AIC, % deviance explained, k_time.
# ------------------------------------------------------------------
alpha05_results <- map_dfr(families, function(fam) {
  df <- models_data[[fam]]$cnorm
  out <- fit_cnorm_alpha(df, p = 0.35, alpha_w = 0.5)
  cbind(family = fam, out)
})
print(alpha05_results, row.names = FALSE)
```

```
##            family SD_RQR     AIC DevExpl k_time
##   Aphelocheiridae   0.72  5470.5    75.2     22
##   Brachycentridae   0.70 24478.5    56.2     25
##     Odontoceridae   0.83 17832.3    58.9     24
##  Cordulegastridae   0.89  1496.0    77.7     22
```

```r
# ================================================================
# Tune p (common grid) and weight exponent  for all families
# - Uses models_data[[fam]]$cnorm (with lower/upper, season_f, decimal_date, SITE_ID.F, SAMPLE_DATE)
# - Tries p  {0.35, 0.40, 0.45} and  {0, 0.5}
# - Reports SD(RQR) (using fitted sigma), AIC, DevExpl, and suggests best
# ================================================================

FAMILIES <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")
P_GRID   <- c(0.35, 0.40, 0.45)
ALPHAS   <- c(0.0, 0.5)
```

```r
# --- helpers ---
# Extract  from a fitted cnorm() family string, e.g. "cnorm(0.89)" -> 0.89
get_sigma <- function(fit) {
  fam_str <- fit$family$family
  as.numeric(sub(".*cnorm\\(([^)]+)\\).*", "\\1", fam_str))
}

# Compute SD of randomised-quantile residuals for censored Normal on the transformed scale.
# Uses fitted  and  from the model; clamps PIT away from {0,1}; SD  1 indicates good dispersion.
rqr_sd_corrected <- function(fit, lt, ut) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- get_sigma(fit)
  u  <- pmin(pmax(pnorm(lt, mu, sd) + (pnorm(ut, mu, sd) - pnorm(lt, mu, sd)) * runif(length(mu)), 1e-1
  sd(qnorm(u), na.rm = TRUE)
}

# Prepare bounds and weights for a given power p and weight exponent alpha_w:
# - (Re)build transformed bounds lower_t/upper_t = lower^p / upper^p (lower/upper on count-scale)
# - width_t = interval width on transformed scale
# - wt_raw = 1/width_t; wt = wt_raw^alpha_w, lightly capped at 95th percentile to avoid extremes
# - year column used to set k for time smooth
prep_bounds <- function(df, p, alpha_w) {
  if (!all(c("lower_t","upper_t") %in% names(df))) {
    df <- df %>% mutate(lower_t = pmax(as.numeric(lower),0)^p,
                        upper_t = pmax(as.numeric(upper),0)^p)
  } else {
    # If lower_t/upper_t exist but for a different p, recompute for current p
    df <- df %>% mutate(lower_t = pmax(as.numeric(lower),0)^p,
                        upper_t = pmax(as.numeric(upper),0)^p)
  }
  df %>%
    mutate(
      width_t = pmax(upper_t - lower_t, 1e-6),
      wt_raw  = 1 / width_t,
      # weight exponent and gentle capping to avoid a few huge weights dominating
      wt      = (wt_raw^alpha_w) %>% pmin(quantile(., 0.95, na.rm = TRUE)),
      year    = year(SAMPLE_DATE)
    )
}

# Fit one cnorm model at a specific (p, alpha_w) setting:
# - Ensures season_f (spring/autumn) and SITE_ID.F exist
# - Chooses k_time from span of years (slightly generous)
# - Fits: season main effect + season-varying time smooth + site RE, with precision weights
# - Returns a one-row data.frame of metrics (SD_RQR, AIC, DevExpl, k_time, n)
fit_once <- function(df, p, alpha_w) {
  d <- prep_bounds(df, p, alpha_w)
  if (!("season_f" %in% names(d))) {
    d <- d %>% mutate(m = month(SAMPLE_DATE),
                      season_f = factor(if_else(m %in% 3:5, "spring", "autumn"),
                                        levels = c("spring","autumn")))
  }
  if (!is.factor(d$SITE_ID.F)) d$SITE_ID.F <- factor(d$SITE_ID)
```

```
  ny <- n_distinct(d$year)
  k_time <- min(40, max(14, round(0.8 * ny)))   # slightly larger k to appease low k-index families

  m <- bam(
    cbind(lower_t, upper_t) ~ season_f +
      s(decimal_date, by = season_f, k = k_time) +
      s(SITE_ID.F, bs = "re"),
    family   = cnorm(),
    data     = d,
    weights  = d$wt,
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  data.frame(
    p        = p,
    alpha_w  = alpha_w,
    SD_RQR   = round(rqr_sd_corrected(m, d$lower_t, d$upper_t), 2),
    AIC      = round(AIC(m), 1),
    DevExpl  = round(100*summary(m)$dev.expl, 1),
    k_time   = k_time,
    n        = nobs(m),
    stringsAsFactors = FALSE
  )
}

# Grid-search tuner for one family:
# - Evaluates all p× combinations
# - Picks the setting with SD_RQR closest to 1, breaking ties by lowest AIC
# - Returns the full table and the chosen 'best' row
tune_family <- function(fam) {
  base <- models_data[[fam]]$cnorm
  grid <- expand.grid(p = P_GRID, alpha_w = ALPHAS, KEEP.OUT.ATTRS = FALSE)
  res  <- purrr::pmap_dfr(grid, ~ fit_once(base, ..1, ..2))
  # pick SD_RQR closest to 1 (tie-break by lowest AIC)
  res <- res %>% mutate(dist = abs(SD_RQR - 1))
  best <- res %>% filter(dist == min(dist)) %>% slice_min(AIC, n = 1)
  list(table = res, best = best)
}

# ---- run
# Iterate families, store the tuning table + best choice per family; attach family name
tuned <- lapply(FAMILIES, function(f) { out <- tune_family(f); out$family <- f; out })
names(tuned) <- FAMILIES

# ---- print per-family tables and choices
# For each family: print the full results table, then a concise "Recommended" line
for (f in FAMILIES) {
  cat("\n================ ", f, " ================\n", sep = "")
  print(tuned[[f]]$table, row.names = FALSE)
  cat("→ Recommended: ",
```

```
      with(tuned[[f]]$best, sprintf("p=%.2f,  =%.1f (SD_RQR=%0.2f, AIC=%0.1f, Dev=%.1f%%, k=%d, n=%d)",
                                    p, alpha_w, SD_RQR, AIC, DevExpl, k_time, n)),
      "\n", sep = "")
}
```

```
##
## ================ Aphelocheiridae ================
##     p alpha_w SD_RQR    AIC DevExpl k_time    n dist
##  0.35     0.0   0.99 5164.8    72.7     25 1089 0.01
##  0.40     0.0   1.00 5538.8    71.5     25 1089 0.00
##  0.45     0.0   0.99 5936.0    70.0     25 1089 0.01
##  0.35     0.5   0.73 5438.7    74.4     25 1089 0.27
##  0.40     0.5   0.75 5777.5    73.4     25 1089 0.25
##  0.45     0.5   0.78 6128.3    72.2     25 1089 0.22
## → Recommended: p=0.40,  =0.0 (SD_RQR=1.00, AIC=5538.8, Dev=71.5%, k=25, n=1089)
##
## ================ Brachycentridae ================
##     p alpha_w SD_RQR     AIC DevExpl k_time    n dist
##  0.35     0.0   0.97 21869.1    49.8     29 5917 0.03
##  0.40     0.0   0.95 24450.0    47.3     29 5917 0.05
##  0.45     0.0   0.92 27198.0    44.8     29 5917 0.08
##  0.35     0.5   0.73 23749.1    54.4     29 5917 0.27
##  0.40     0.5   0.74 26163.8    50.9     29 5917 0.26
##  0.45     0.5   0.75 28678.5    47.3     29 5917 0.25
## → Recommended: p=0.35,  =0.0 (SD_RQR=0.97, AIC=21869.1, Dev=49.8%, k=29, n=5917)
##
## ================ Odontoceridae ================
##     p alpha_w SD_RQR     AIC DevExpl k_time    n dist
##  0.35     0.0   1.00 16864.9    55.5     28 6674 0.00
##  0.40     0.0   1.00 17987.6    56.2     28 6674 0.00
##  0.45     0.0   1.00 19234.1    57.0     28 6674 0.00
##  0.35     0.5   0.83 17714.5    58.4     28 6674 0.17
##  0.40     0.5   0.84 18692.3    58.0     28 6674 0.16
##  0.45     0.5   0.86 19731.9    57.3     28 6674 0.14
## → Recommended: p=0.35,  =0.0 (SD_RQR=1.00, AIC=16864.9, Dev=55.5%, k=28, n=6674)
##
## ================ Cordulegastridae ================
##     p alpha_w SD_RQR    AIC DevExpl k_time    n dist
##  0.35     0.0   1.00 1428.9    71.5     25 1064 0.00
##  0.40     0.0   1.00 1552.7    72.1     25 1064 0.00
##  0.45     0.0   0.96 1702.5    72.3     25 1064 0.04
##  0.35     0.5   0.92 1490.2    75.8     25 1064 0.08
##  0.40     0.5   0.91 1604.3    75.1     25 1064 0.09
##  0.45     0.5   0.90 1735.0    74.0     25 1064 0.10
## → Recommended: p=0.35,  =0.0 (SD_RQR=1.00, AIC=1428.9, Dev=71.5%, k=25, n=1064)
```

```
# ---- compact summary of chosen settings
# Bind all 'best' rows into one data.frame for quick comparison across families
chosen <- do.call(rbind, lapply(tuned, function(x) cbind(family = x$family, x$best)))
cat("\n========= Chosen settings per family =========\n")
```

```
##
## ========= Chosen settings per family =========
```

```
print(chosen, row.names = FALSE)
```

```
##            family    p alpha_w SD_RQR     AIC DevExpl k_time    n dist
##   Aphelocheiridae 0.40       0   1.00  5538.8    71.5     25 1089 0.00
##   Brachycentridae 0.35       0   0.97 21869.1    49.8     29 5917 0.03
##     Odontoceridae 0.35       0   1.00 16864.9    55.5     28 6674 0.00
##  Cordulegastridae 0.35       0   1.00  1428.9    71.5     25 1064 0.00
```

#4.4.4 FINAL CNORM MODEL FOR ALL FAMILIES WITH NO INTERVAL WEIGHTING

```
# ================================
# Final cnorm fits per family (no plots)
# p per family; alpha = 0 (no interval weighting)
# Prints summary() and textual gam.check()
# ================================

# Family-specific p (from your tuning)
final_p <- tibble::tribble(
  ~family,             ~p,
  "Aphelocheiridae",   0.35,
  "Brachycentridae",   0.35,
  "Odontoceridae",     0.40,
  "Cordulegastridae",  0.40
)

# Helper: build p-transformed bounds, no weights, guard season/site factors
prep_cnorm_df <- function(df, p) {
  stopifnot(all(c("lower","upper") %in% names(df)))
  df %>%
    mutate(
      lower_t = pmax(as.numeric(lower), 0)^p,        # transform count-scale lower bound to p-scale
      upper_t = pmax(as.numeric(upper), 0)^p         # transform count-scale upper bound to p-scale
    ) %>%
    { if (!("season_f" %in% names(.)))               # if season factor missing, build spring/autumn fr
        mutate(., m = month(SAMPLE_DATE),
                  season_f = factor(if_else(m %in% 3:5, "spring", "autumn"),
                                    levels = c("spring","autumn")))
      else . } %>%
    { if (!is.factor(.$SITE_ID.F)) mutate(., SITE_ID.F = factor(SITE_ID)) else . } %>%  # ensure site R
    mutate(year = year(SAMPLE_DATE))                  # convenience: calendar year for k selection
}

# Correct-sigma RQ residual SD
get_sigma <- function(fit) {
  fam_str <- fit$family$family
  as.numeric(sub(".*cnorm\\(([^)]+)\\).*", "\\1", fam_str))  # parse sigma from cnorm(<sigma>)
}
sd_rqr_corrected <- function(fit, lt, ut) {
  mu <- as.numeric(fitted(fit, type = "response"))           # fitted mean on p-scale
  sd <- get_sigma(fit)                                       # model sigma on p-scale
  u  <- pmin(pmax(pnorm(lt, mu, sd) + (pnorm(ut, mu, sd) - pnorm(lt, mu, sd)) * runif(length(mu)), 1e-1
  sd(qnorm(u), na.rm = TRUE)                                 # SD of PIT→Normal = dispersion cue (~1 id
}
```

```r
fit_cnorm_final <- function(fam, p, md = models_data) {
  stopifnot(!is.null(md[[fam]]), "cnorm" %in% names(md[[fam]]))
  df <- prep_cnorm_df(md[[fam]]$cnorm, p)                    # prepare df (bounds→p-scale, season/site

  # k chosen from span of years; a touch generous helps low k-index cases
  ny <- dplyr::n_distinct(df$year)
  k_time <- min(40, max(14, round(0.8 * ny)))

  m <- bam(
    cbind(lower_t, upper_t) ~ season_f +
      s(decimal_date, by = season_f, k = k_time) +          # season-varying smooth over time
      s(SITE_ID.F, bs = "re"),                              # site random intercept
    family   = cnorm(),
    data     = df,
    method   = "fREML",
    discrete = TRUE,
    select   = TRUE,
    gamma    = 1.2
  )

  # ---- print full outputs (no plots) ----
  cat("\n======================================================\n")
  cat(sprintf("Family: %s  |  cnorm(p = %.2f)\n", fam, p))
  cat("======================================================\n")
  print(summary(m))                                          # detailed param/smooth summary

  tmp <- tempfile(fileext = ".pdf"); pdf(tmp)                # suppress 2×2 gam.check panel to text-onl
  gc_out <- try(gam.check(m, rep = 0), silent = TRUE)
  dev.off(); unlink(tmp)
  if (!inherits(gc_out, "try-error")) {
    cat("\n--- gam.check (text) ---\n")
    print(gc_out)
  }

  # compact metrics
  rqr_sd <- sd_rqr_corrected(m, df$lower_t, df$upper_t)      # dispersion  1 good
  aic    <- AIC(m); devx <- summary(m)$dev.expl               # parsimony + fit
  cat(sprintf("\nSD(RQR): %.2f | Dev.expl: %.1f%% | AIC: %.1f | N: %d | k_time: %d\n",
              rqr_sd, 100*devx, aic, nobs(m), k_time))

  list(family = fam, p = p, model = m, data = df,
       rqr_sd = rqr_sd, AIC = aic, dev_expl = devx, k_time = k_time, n = nobs(m))
}

# Run all four and build a summary table
final_results <- pmap(final_p, ~ fit_cnorm_final(..1, ..2))  # iterate families with their chosen p
```

```
##
## ======================================================
## Family: Aphelocheiridae  |  cnorm(p = 0.35)
## ======================================================
##
## Family: cnorm(0.899)
```

```
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     1.61696    0.09048  17.870  < 2e-16 ***
## season_fautumn  0.16146    0.06110   2.643  0.00838 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F  p-value
## s(decimal_date):season_fspring   2.660     24 2.892 8.52e-05 ***
## s(decimal_date):season_fautumn   2.863     24 2.758 0.000301 ***
## s(SITE_ID.F)                   238.596    313 7.771  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.64   Deviance explained = 72.7%
## fREML = 2334.7  Scale est. = 1         n = 1089


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  1.587397e-12 -5.654184e-06 -1.270672e-11 -7.306129e-06 -3.562803e-09
##
## $hess
##               [,1]          [,2]          [,3]          [,4]         [,5]
## [1,]  1.077475e+00 -4.191924e-06 -1.377777e-02  2.106415e-07 4.961386e-02
## [2,] -4.191924e-06  5.654120e-06 -7.164068e-08 -1.102991e-12 1.649710e-07
## [3,] -1.377777e-02 -7.164068e-08  7.559661e-01 -4.213359e-06 2.689430e-01
## [4,]  2.106415e-07 -1.102991e-12 -4.213359e-06  7.306036e-06 1.068318e-06
## [5,]  4.961386e-02  1.649710e-07  2.689430e-01  1.068318e-06 9.791677e+01
##
## Model rank =  364 / 364
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
## s(decimal_date):season_fspring  24.00   2.66    0.94   0.055 .
## s(decimal_date):season_fautumn  24.00   2.86    0.94   0.040 *
## s(SITE_ID.F)                   314.00 238.60      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
```

```
## SD(RQR): 1.00 | Dev.expl: 72.7% | AIC: 5164.8 | N: 1089 | k_time: 25
##
## ====================================================
## Family: Brachycentridae  |  cnorm(p = 0.35)
## ====================================================
##
## Family: cnorm(0.778)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.67483    0.02787   24.21   <2e-16 ***
## season_fautumn  0.43863    0.02404   18.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F  p-value
## s(decimal_date):season_fspring   3.664     28 1.357 0.000283 ***
## s(decimal_date):season_fautumn   3.685     28 2.951  < 2e-16 ***
## s(SITE_ID.F)                   633.408   1109 3.926  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.429   Deviance explained = 49.8%
## fREML = 9485.8  Scale est. = 1         n = 5917


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1]  9.345635e-11  3.630152e-12 -9.938939e-11 -2.067389e-05 -4.444195e-08
##
## $hess
##               [,1]          [,2]          [,3]          [,4]         [,5]
## [1,]  7.765528e-01 -1.344091e-01 -2.126369e-02  1.050617e-07 3.643877e-01
## [2,] -1.344091e-01  2.386411e-01 -1.751788e-03  4.056507e-08 4.468279e-02
## [3,] -2.126369e-02 -1.751788e-03  7.086170e-01 -1.737645e-05 7.036703e-01
## [4,]  1.050617e-07  4.056507e-08 -1.737645e-05  2.067304e-05 3.126465e-07
## [5,]  3.643877e-01  4.468279e-02  7.036703e-01  3.126465e-07 2.082895e+02
##
## Model rank =  1168 / 1168
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'     edf k-index p-value
## s(decimal_date):season_fspring   28.00    3.66    0.87  <2e-16 ***
## s(decimal_date):season_fautumn   28.00    3.69    0.87  <2e-16 ***
## s(SITE_ID.F)                   1110.00  633.41      NA      NA
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQR): 0.97 | Dev.expl: 49.8% | AIC: 21869.1 | N: 5917 | k_time: 29
##
## ====================================================
## Family: Odontoceridae  |  cnorm(p = 0.40)
## ====================================================
##
## Family: cnorm(0.434)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.88818    0.01895  46.870   <2e-16 ***
## season_fautumn  0.02379    0.01517   1.568    0.117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F  p-value
## s(decimal_date):season_fspring   5.768     27 5.840  < 2e-16 ***
## s(decimal_date):season_fautumn   4.460     27 3.068 2.31e-06 ***
## s(SITE_ID.F)                   921.919   1297 5.019  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.452   Deviance explained = 56.2%
## fREML = 8160.3  Scale est. = 1         n = 6674


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -5.593384e-10 -5.485404e-05 -4.384564e-10 -5.256190e-11 -5.769536e-07
##
## $hess
##               [,1]          [,2]          [,3]          [,4]         [,5]
## [1,]  1.703654e+00 -1.607663e-05 -5.267704e-02 -9.981934e-04 2.426703e-01
## [2,] -1.607663e-05  5.485285e-05 -1.555095e-06 -5.739391e-07 1.100173e-05
## [3,] -5.267704e-02 -1.555095e-06  1.235667e+00  7.349408e-02 5.670982e-02
## [4,] -9.981934e-04 -5.739391e-07  7.349408e-02  7.322349e-02 2.530766e-02
## [5,]  2.426703e-01  1.100173e-05  5.670982e-02  2.530766e-02 3.058375e+02
##
## Model rank =  1354 / 1354
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
```

```
## indicate that k is too low, especially if edf is close to k'.
##
##                                     k'     edf k-index p-value
## s(decimal_date):season_fspring   27.00    5.77    0.95  <2e-16 ***
## s(decimal_date):season_fautumn   27.00    4.46    0.95  <2e-16 ***
## s(SITE_ID.F)                   1298.00  921.92      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQR): 1.00 | Dev.expl: 56.2% | AIC: 17987.6 | N: 6674 | k_time: 28
##
## ====================================================
## Family: Cordulegastridae  |  cnorm(p = 0.40)
## ====================================================
##
## Family: cnorm(0.232)
## Link function: identity
##
## Formula:
## cbind(lower_t, upper_t) ~ season_f + s(decimal_date, by = season_f,
##     k = k_time) + s(SITE_ID.F, bs = "re")
##
## Parametric coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.64274    0.02525  25.457   <2e-16 ***
## season_fautumn  0.01234    0.02267   0.544    0.587
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                                    edf Ref.df     F  p-value
## s(decimal_date):season_fspring   1.784     24 1.443  0.00162 **
## s(decimal_date):season_fautumn   2.606     24 3.197 7.35e-05 ***
## s(SITE_ID.F)                   235.137    378 3.045  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.542   Deviance explained = 72.1%
## fREML = 785.26  Scale est. = 1         n = 1064


##
## Method: fREML   Optimizer: perf chol
## $grad
## [1] -2.577610e-08 -8.121900e-06 -2.009362e-07 -6.453387e-06 -1.358961e-05
##
## $hess
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  4.904871e-01  4.073930e-05 -5.527804e-02  3.113428e-08  9.453900e-01
## [2,]  4.073930e-05  8.144176e-06 -7.600897e-07  1.279090e-11  9.718129e-05
```

```
## [3,] -5.527804e-02 -7.600897e-07  7.524897e-01 -5.588897e-06  4.030936e-01
## [4,]  3.113428e-08  1.279090e-11 -5.588897e-06  6.453304e-06 -7.576925e-07
## [5,]  9.453900e-01  9.718129e-05  4.030936e-01 -7.576925e-07  8.704327e+01
##
## Model rank =  429 / 429
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                   k'    edf k-index p-value
## s(decimal_date):season_fspring  24.00   1.78    0.85  <2e-16 ***
## s(decimal_date):season_fautumn  24.00   2.61    0.85  <2e-16 ***
## s(SITE_ID.F)                   379.00 235.14      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## --- gam.check (text) ---
## $mfrow
## [1] 2 2
##
##
## SD(RQR): 0.98 | Dev.expl: 72.1% | AIC: 1552.7 | N: 1064 | k_time: 25
```

```r
names(final_results) <- final_p$family

final_summary <- dplyr::bind_rows(lapply(final_results, function(x) {
  data.frame(
    family   = x$family,
    p        = x$p,
    n        = x$n,
    k_time   = x$k_time,
    SD_RQR   = round(x$rqr_sd, 2),
    DevExpl  = round(100*x$dev_expl, 1),
    AIC      = round(x$AIC, 1),
    stringsAsFactors = FALSE
  )
}))
cat("\n==================== Final cnorm summary ====================\n")
```

```
##
## ==================== Final cnorm summary ====================
```

```r
print(final_summary, row.names = FALSE)                    # compact per-family comparison
```

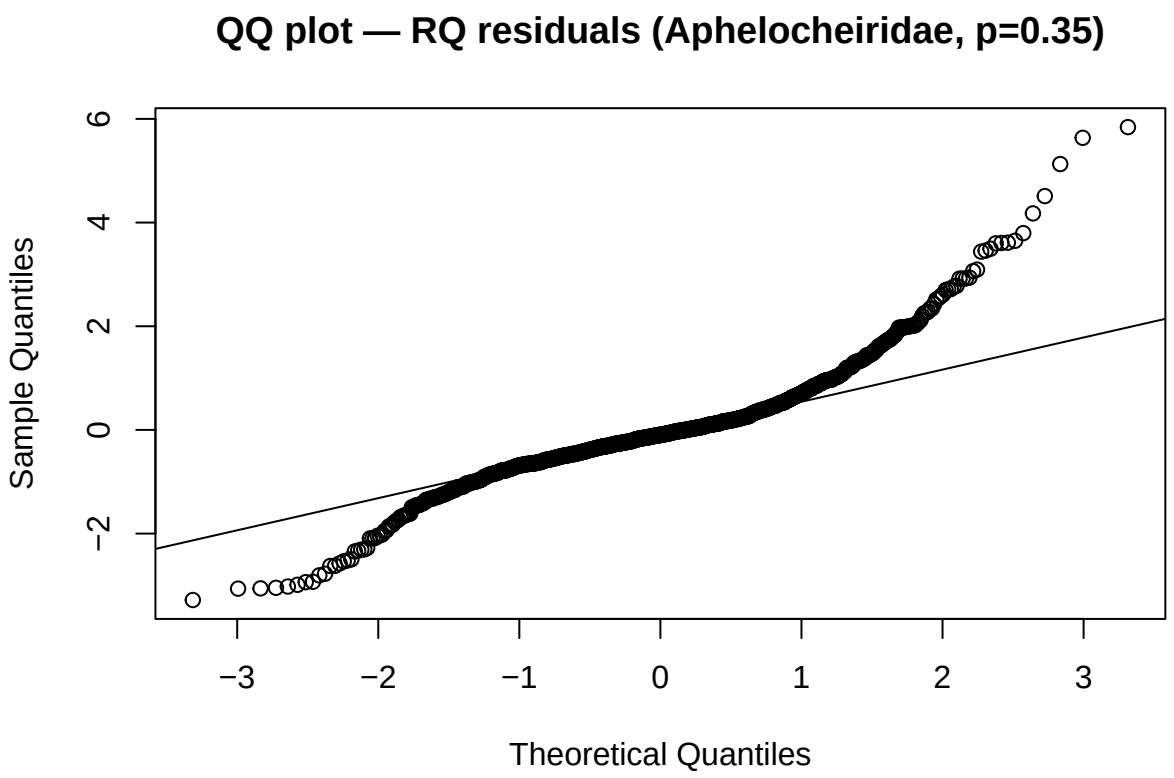
```
##            family    p    n k_time SD_RQR DevExpl     AIC
##   Aphelocheiridae 0.35 1089     25   1.00    72.7  5164.8
##   Brachycentridae 0.35 5917     29   0.97    49.8 21869.1
##     Odontoceridae 0.40 6674     28   1.00    56.2 17987.6
##  Cordulegastridae 0.40 1064     25   0.98    72.1  1552.7
```

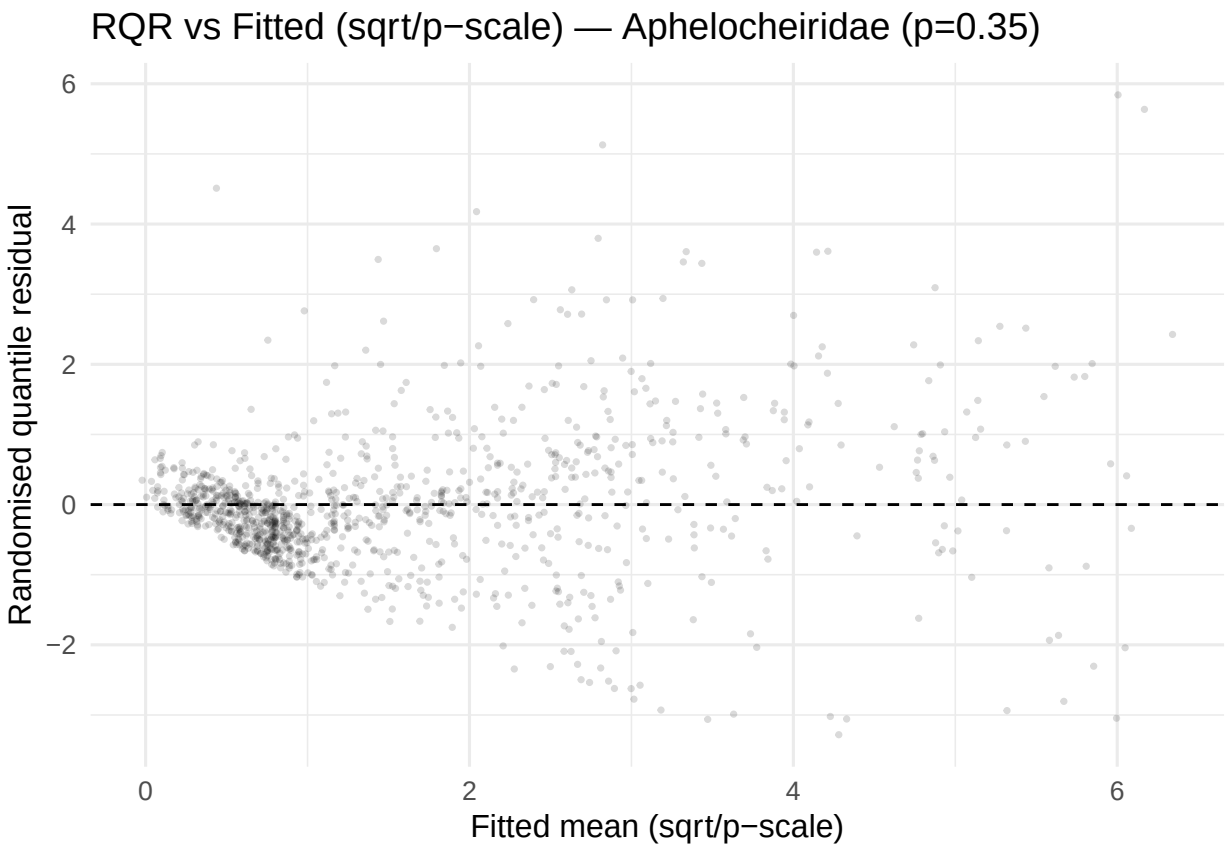
```r
# ============================================================
# CNORM - per-family Model Diagnostics + partial-effect plots
```

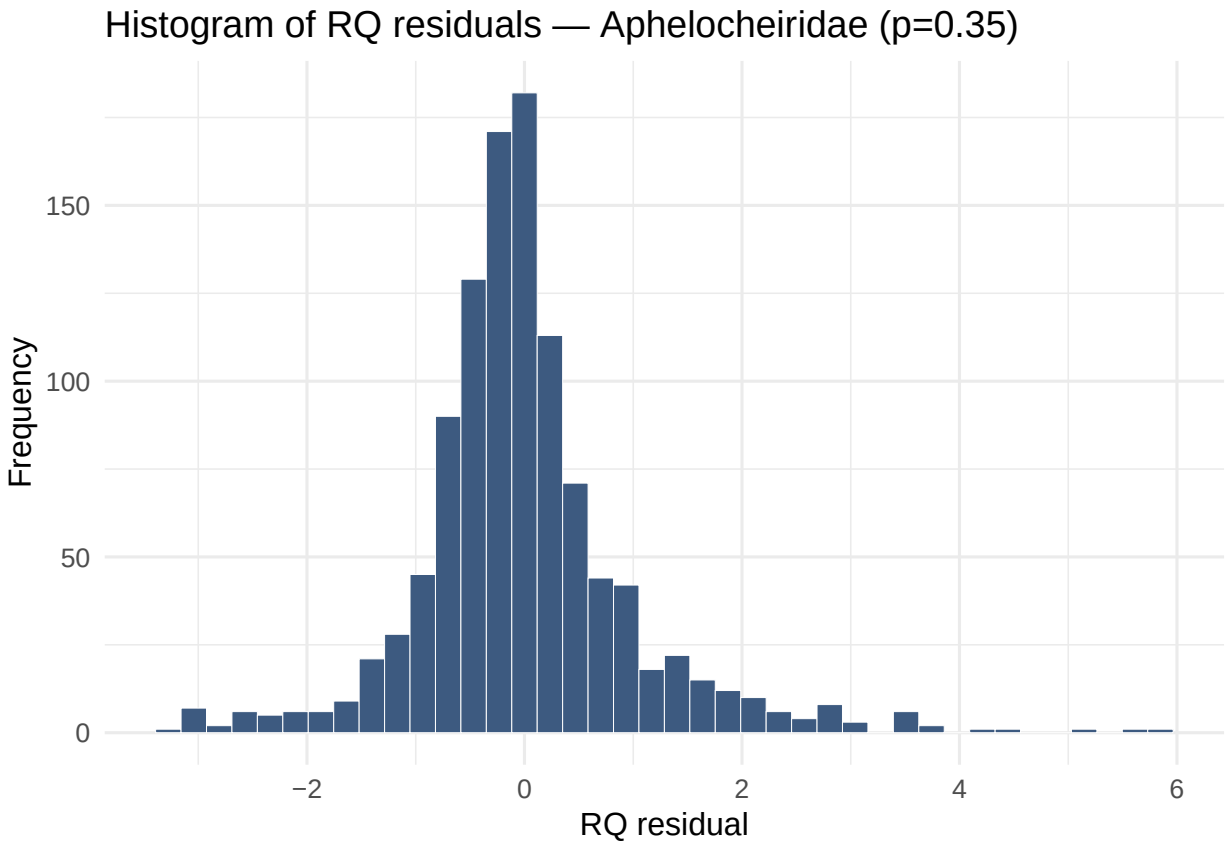
```r
# Assumes: final_results (named list) with elements containing:
#   $family, $p, $model (bam fit), $data (with lower_t, upper_t)
# ------------------------------------------------------------------
# What this script does:
# - Defines helpers to extract the cnorm sigma and to compute
#   randomised-quantile residuals (RQR) on the p-scale.
# - For each fitted family model in `final_results`, it:
#     * draws a QQ plot of RQRs,
#     * prints Residuals vs Fitted and Residual histogram panels,
#     * shows residual ACF,
#     * prints season-specific time smooth partial effects,
#     * and prints a compact console footer with SD(RQR), DevExpl, AIC, N.
# Notes:
# - No model refitting happens here; we only diagnose fitted objects.
# - Random effects are not excluded in partial-effect plots (these are
#   standard gratia::draw() outputs for the smooth terms).
# ==================================================================

# ---- helpers -----------------------------------------------------
# Purpose: parse the fitted Normal sigma from mgcv's cnorm family string,
# e.g. "cnorm(0.612)". Falls back to 1.0 if parsing fails.
get_cnorm_sigma <- function(fit) {
  fam_str <- fit$family$family
  m <- regexpr("cnorm\\(([^)]+)\\)", fam_str)
  if (m > 0) as.numeric(sub("cnorm\\(([^)]+)\\)", "\\1", regmatches(fam_str, m))) else 1.0
}

# Purpose: compute RQRs using the *fitted* sigma on the p-scale.
# Inputs are transformed bounds (lower_t/upper_t) on the same p-scale as the model.
cnorm_rqr <- function(fit, lower_t, upper_t, eps = 1e-10, seed = 123) {
  mu <- as.numeric(fitted(fit, type = "response"))
  sd <- get_cnorm_sigma(fit)
  Flo <- pnorm(lower_t, mean = mu, sd = sd)
  Fup <- pnorm(upper_t, mean = mu, sd = sd)
  set.seed(seed)
  u <- pmin(pmax(Flo + pmax(Fup - Flo, 0) * runif(length(mu)), eps), 1 - eps)
  qnorm(u)
}

# Make a single family's full diagnostic + partial-effect plots
# Expects `res` to be one element of final_results with $model, $data, $family, $p.
plot_cnorm_family <- function(res) {
  fit <- res$model
  df  <- res$data
  fam <- res$family
  pwr <- res$p

  # --- RQ residuals (correct sigma) ---
  rqr <- cnorm_rqr(fit, df$lower_t, df$upper_t)
  mu  <- fitted(fit, type = "response")

  # 1) QQ plot (base graphics) - visual check for ~Normal RQR
  par(mfrow = c(1,1))
```

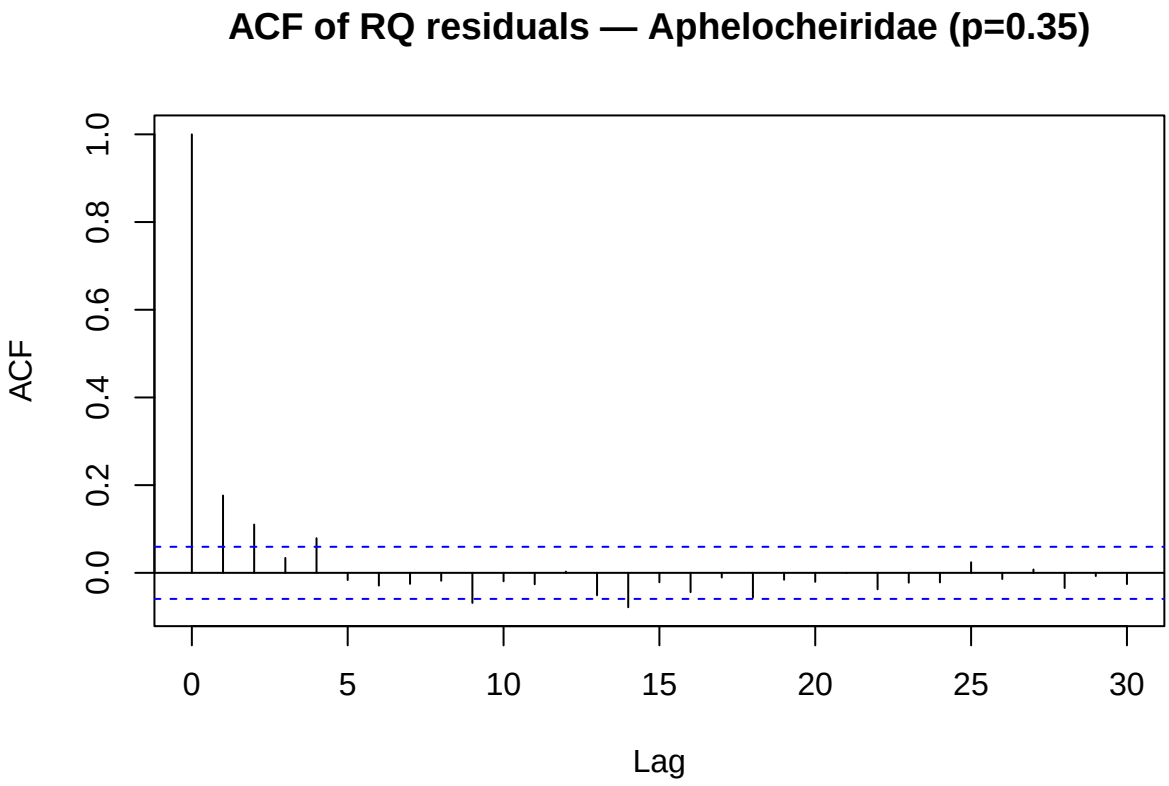
```r
  qqnorm(rqr, main = paste0("QQ plot - RQ residuals (", fam, ", p=", pwr, ")"))
  qqline(rqr)

  # Prep a small tibble for ggplot panels
  dd <- data.frame(mu = mu, rqr = rqr)

  # 2) Residuals vs Fitted (no smooth line) - check mean/variance structure
  print(
    ggplot(dd, aes(mu, rqr)) +
      geom_point(alpha = 0.15, size = 0.6) +
      geom_hline(yintercept = 0, linetype = 2) +
      labs(
        title = paste0("RQR vs Fitted (sqrt/p-scale) - ", fam, " (p=", pwr, ")"),
        x = "Fitted mean (sqrt/p-scale)", y = "Randomised quantile residual"
      ) +
      theme_minimal(base_size = 12)
  )

  # 3) Histogram of residuals - check approximate symmetry/scale
  print(
    ggplot(dd, aes(rqr)) +
      geom_histogram(bins = 40, fill = "#3D5A80", colour = "white", linewidth = 0.15) +
      labs(
        title = paste0("Histogram of RQ residuals - ", fam, " (p=", pwr, ")"),
        x = "RQ residual", y = "Frequency"
      ) +
      theme_minimal(base_size = 12)
  )

  # 4) ACF of residuals - check serial dependence
  acf(rqr[is.finite(rqr)], na.action = na.pass,
      main = paste0("ACF of RQ residuals - ", fam, " (p=", pwr, ")"))

  # 5) Partial effects - season-specific time smooths (printed separately)
  sm_ids   <- which(grepl("decimal_date", smooths(fit)))
  sm_names <- smooths(fit)[sm_ids]
  for (j in seq_along(sm_ids)) {
    season_lab <- if (grepl("spring", sm_names[j], ignore.case = TRUE)) "spring"
      else if (grepl("autumn", sm_names[j], ignore.case = TRUE)) "autumn"
      else sm_names[j]
    p_sm <- draw(fit, select = sm_ids[j]) +
      ggtitle(paste0("Time smooth (", season_lab, ") - ", fam, " (p=", pwr, ")"))
    print(p_sm)
  }

  # Console footer: quick calibration/fit cues for the family
  cat(sprintf("\n[%s] SD(RQR)=%.2f | Dev.expl=%.1f%% | AIC=%.1f | N=%d\n",
              fam, sd(rqr, na.rm = TRUE), 100*summary(fit)$dev.expl,
              AIC(fit), nobs(fit)))
  invisible(NULL)
}

# ---- RUN for all families ------------------------------------
```

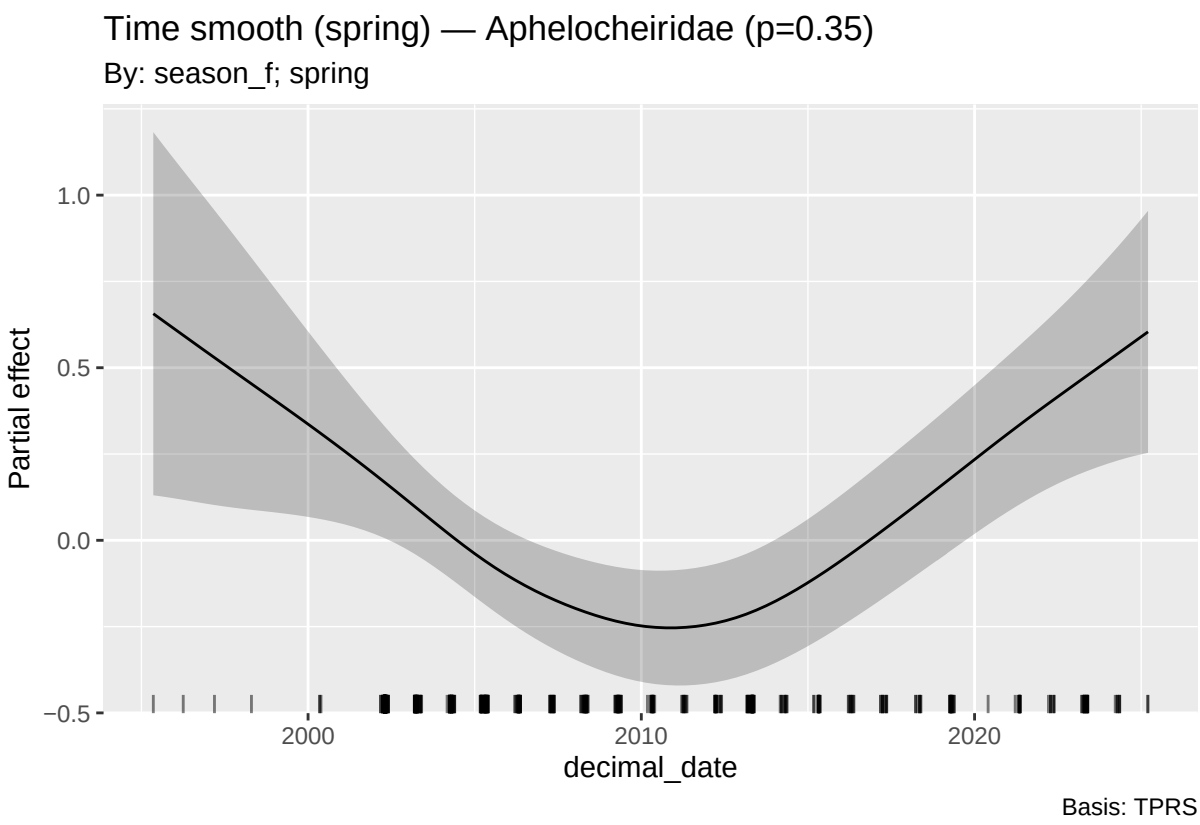
```r
# Guard: ensure final_results exists and is a list; then loop and plot for each family.
stopifnot(exists("final_results"), is.list(final_results))
invisible(lapply(final_results, plot_cnorm_family))
```

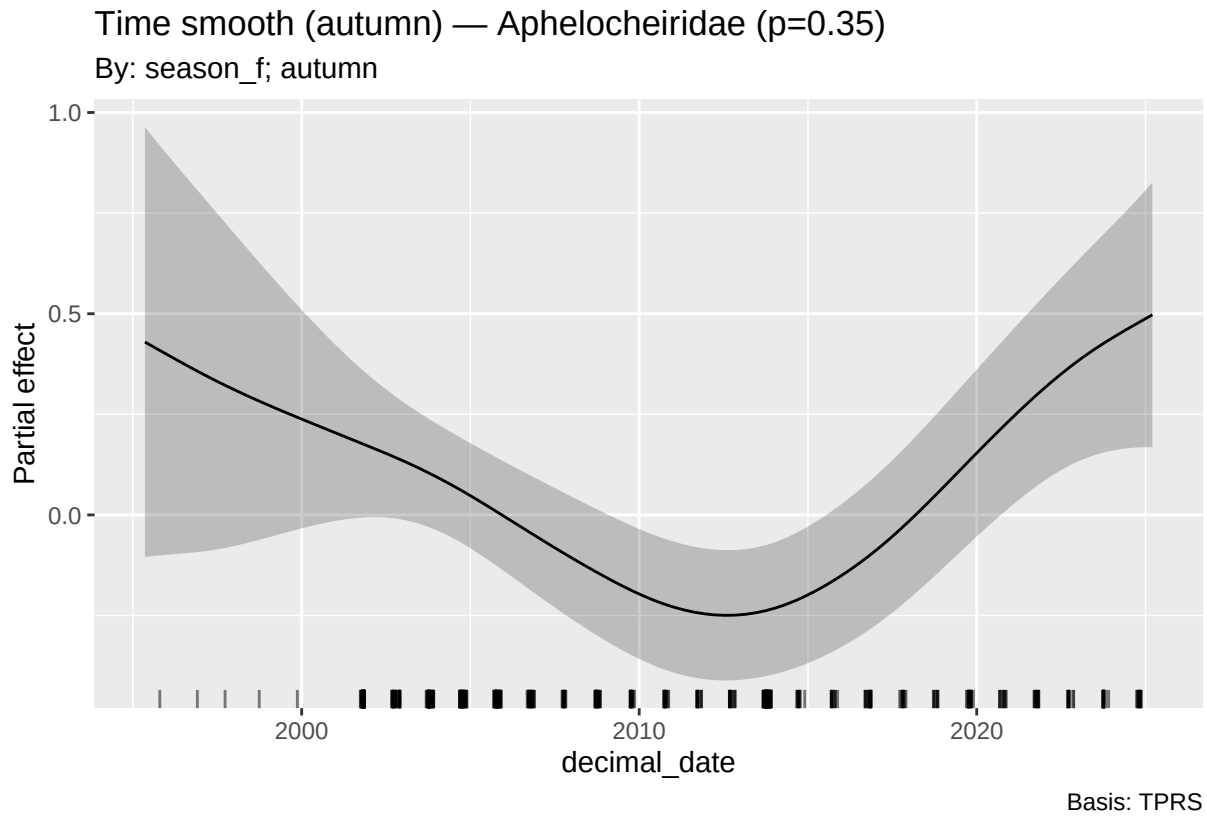



RQR vs Fitted (sqrt/p−scale) — Aphelocheiridae (p=0.35)
Randomised quantile residual
6
4
2
0
−2
0
2
4
6
Fitted mean (sqrt/p−scale)

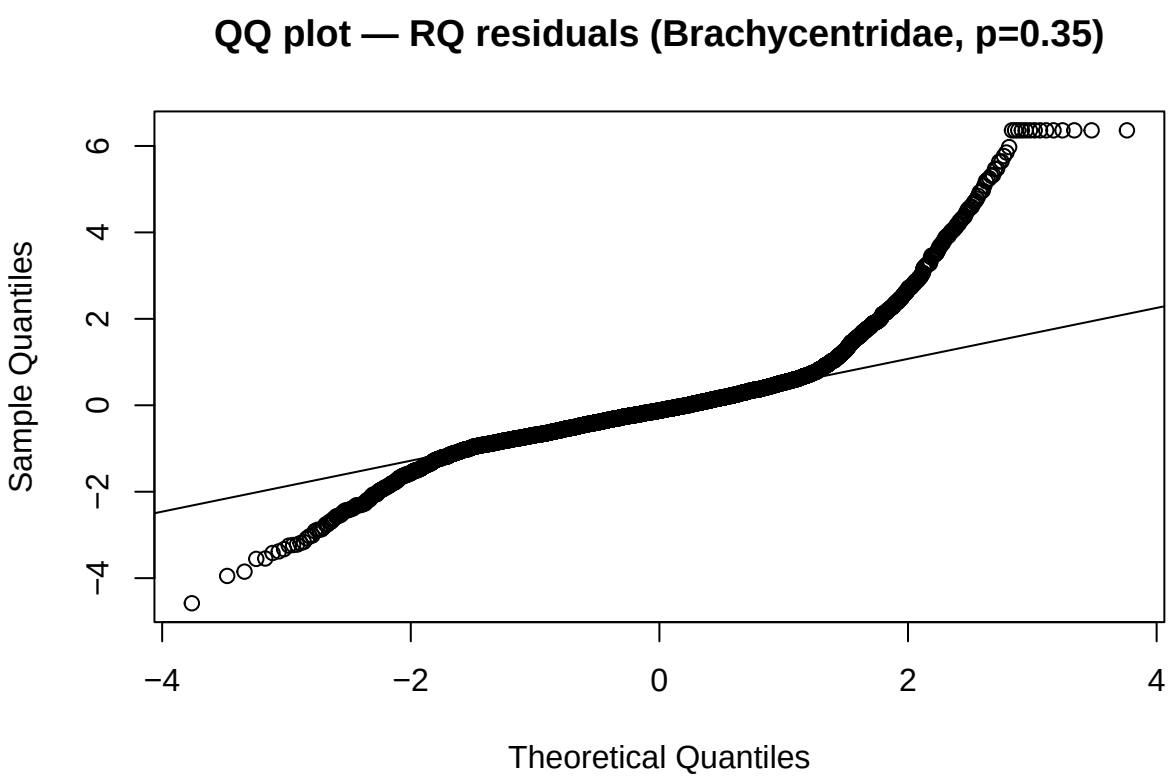

Histogram of RQ residuals — Aphelocheiridae (p=0.35)
Frequency
150
100
50
0
−2
0
2
4
6
RQ residual

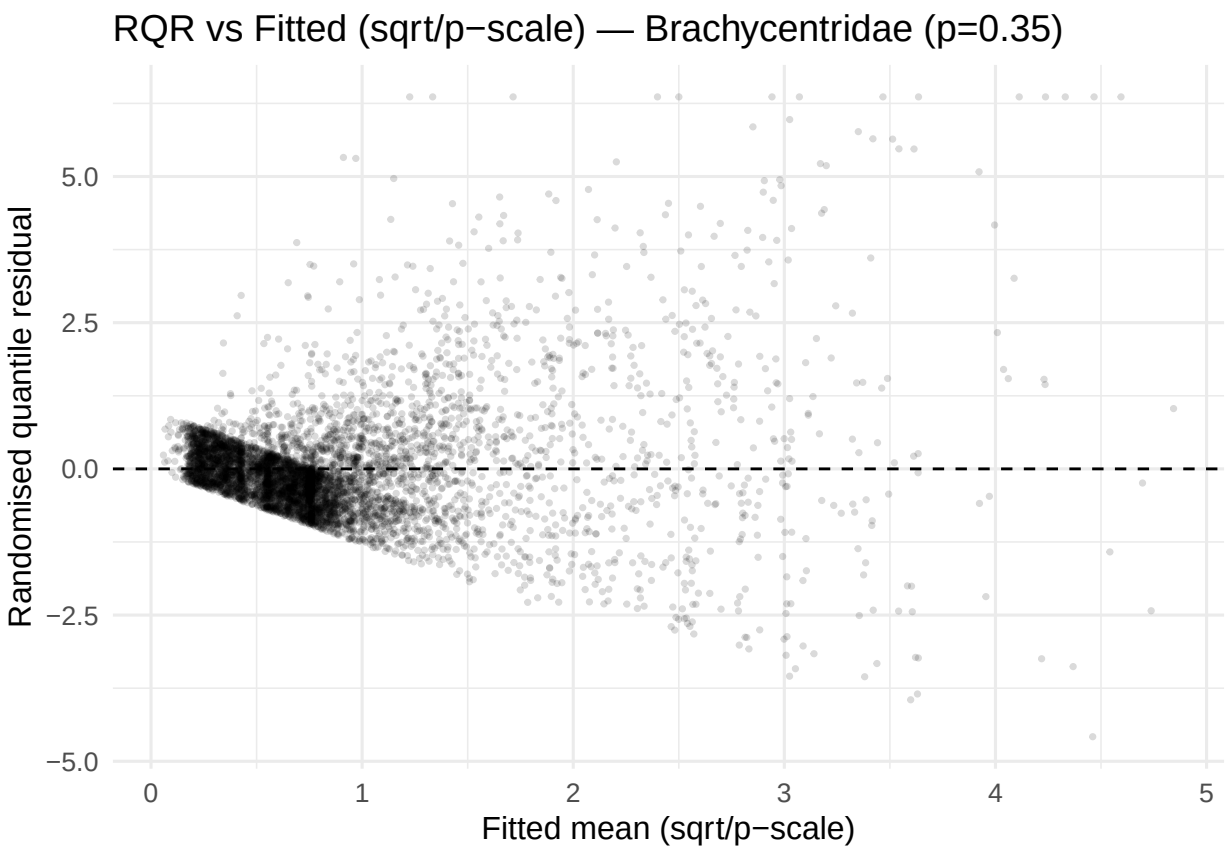

ACF of RQ residuals — Aphelocheiridae (p=0.35)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag

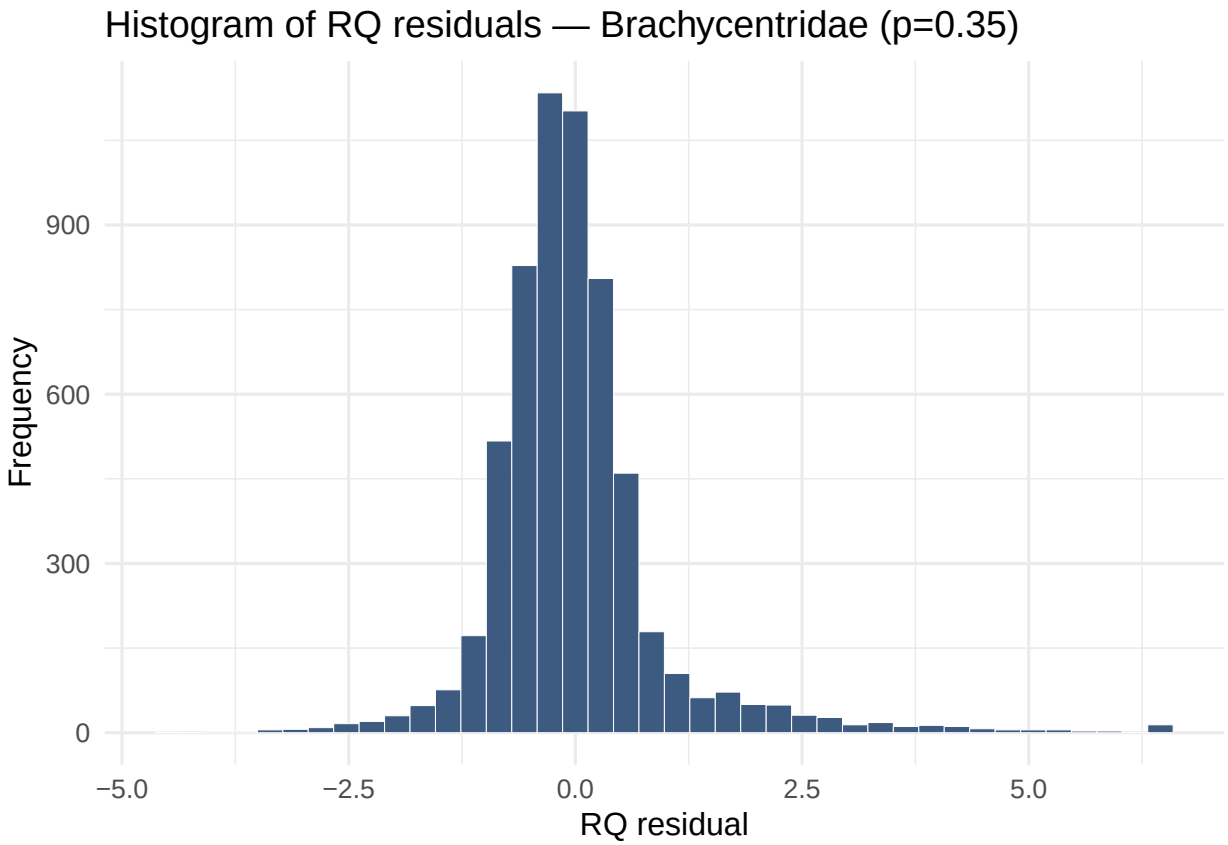

Time smooth (spring) — Aphelocheiridae (p=0.35)
By: season_f; spring
1.0
0.5
0.0
−0.5
Partial effect
2000
2010
2020
decimal_date
Basis: TPRS

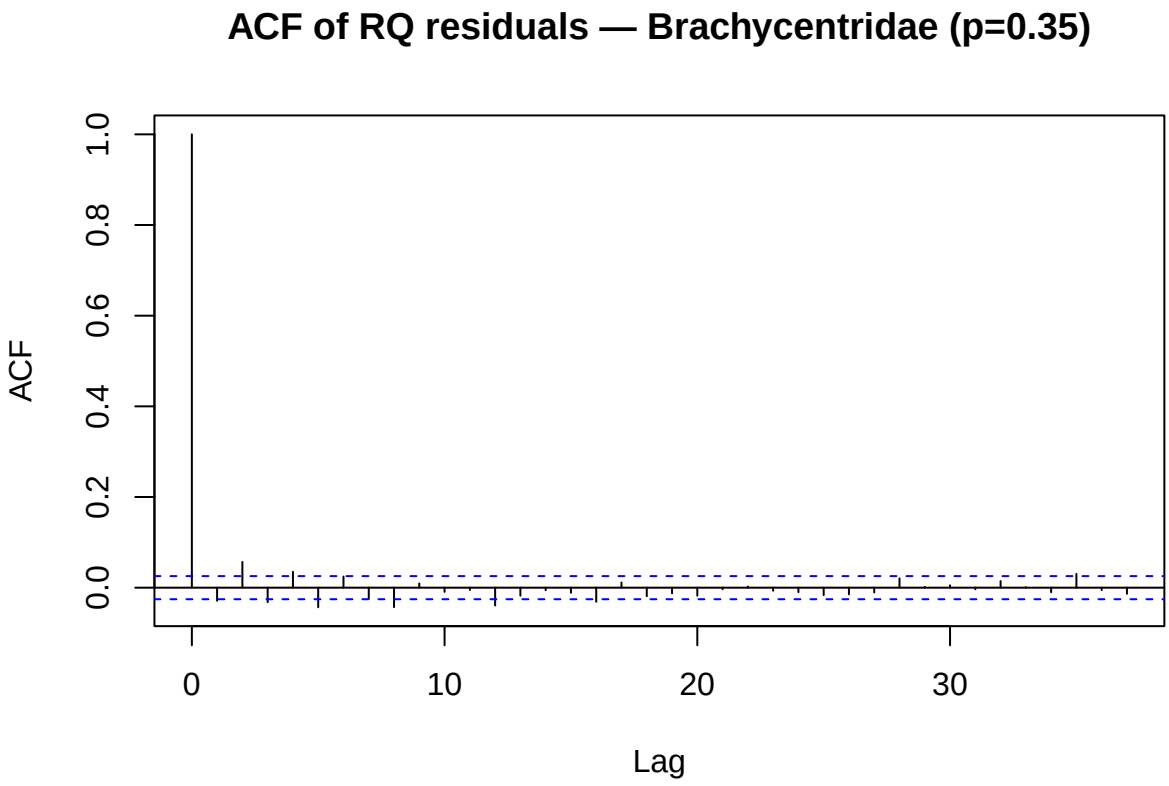


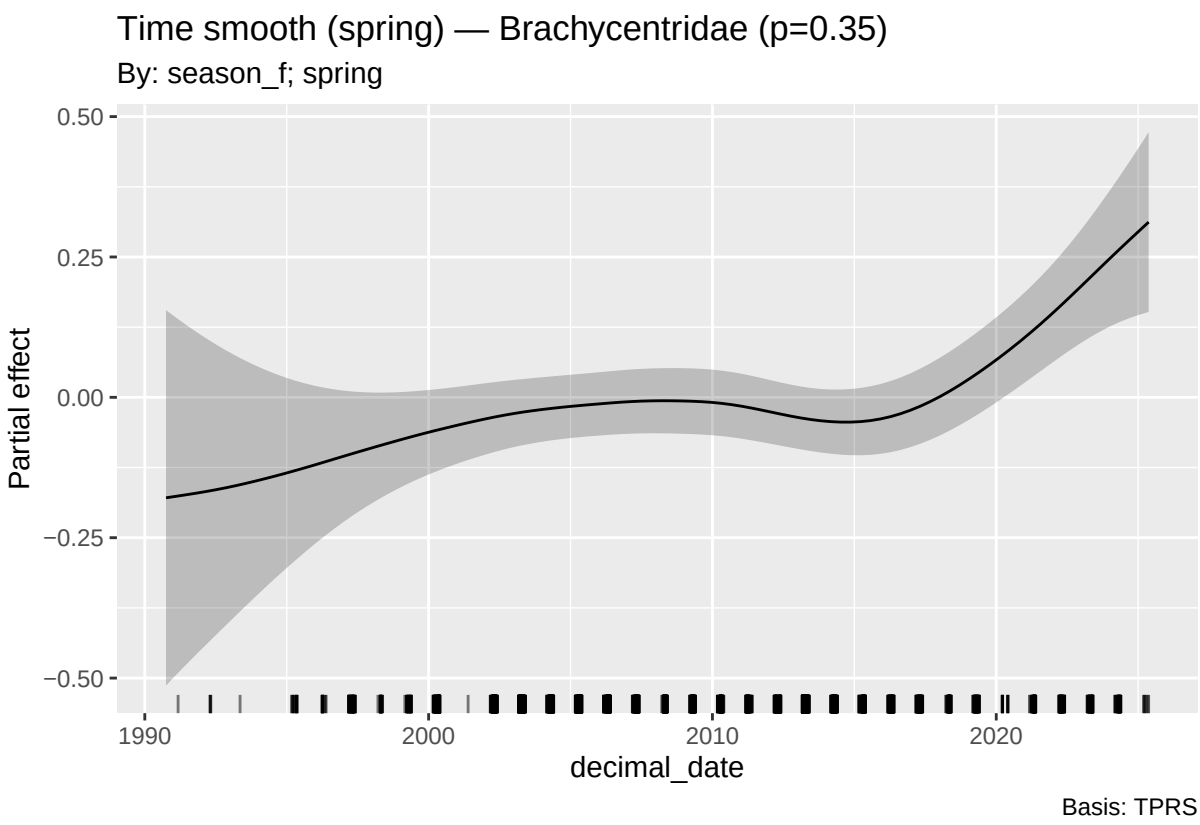

```
##
## [Aphelocheiridae] SD(RQR)=1.00 | Dev.expl=72.7% | AIC=5164.8 | N=1089
```

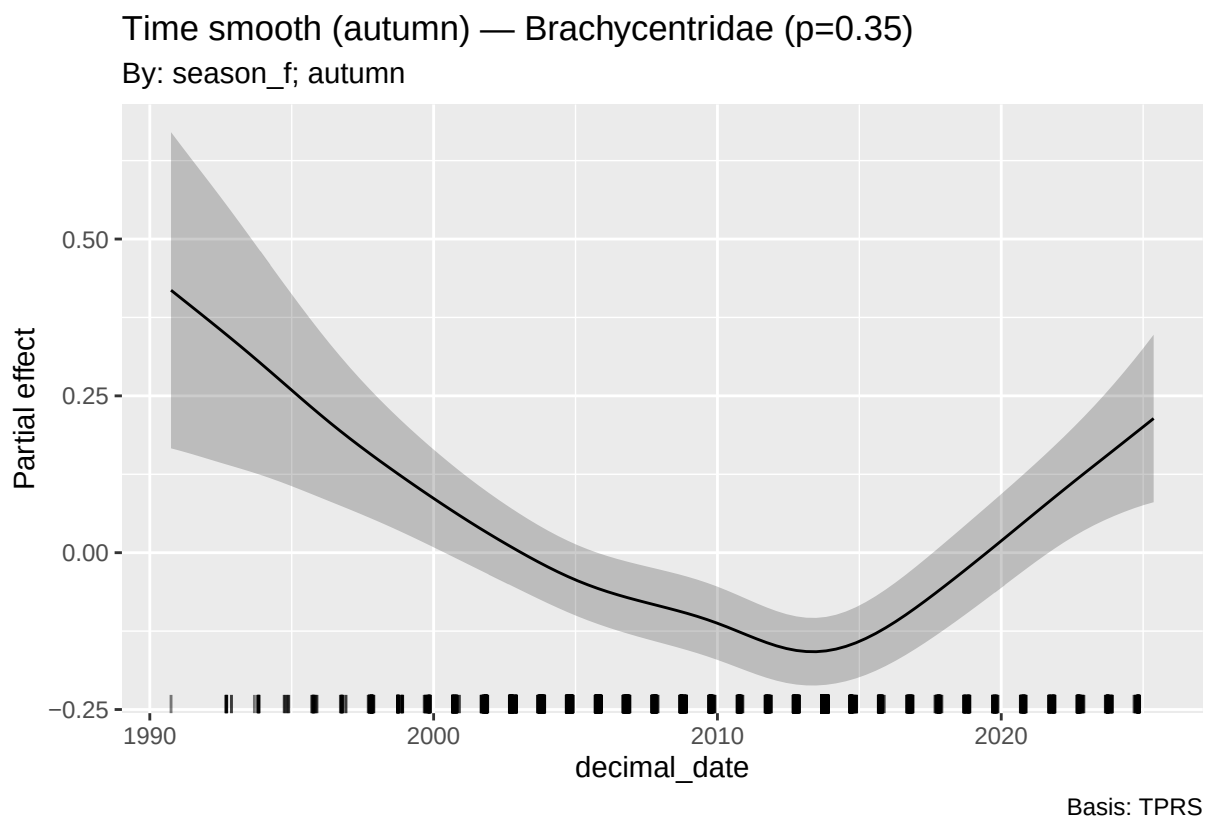

QQ plot — RQ residuals (Brachycentridae, p=0.35)
Sample Quantiles
Theoretical Quantiles
−4
−2
0
2
4
6


RQR vs Fitted (sqrt/p−scale) — Brachycentridae (p=0.35)
Randomised quantile residual
5.0
2.5
0.0
−2.5
−5.0
0
1
2
3
4
5
Fitted mean (sqrt/p−scale)

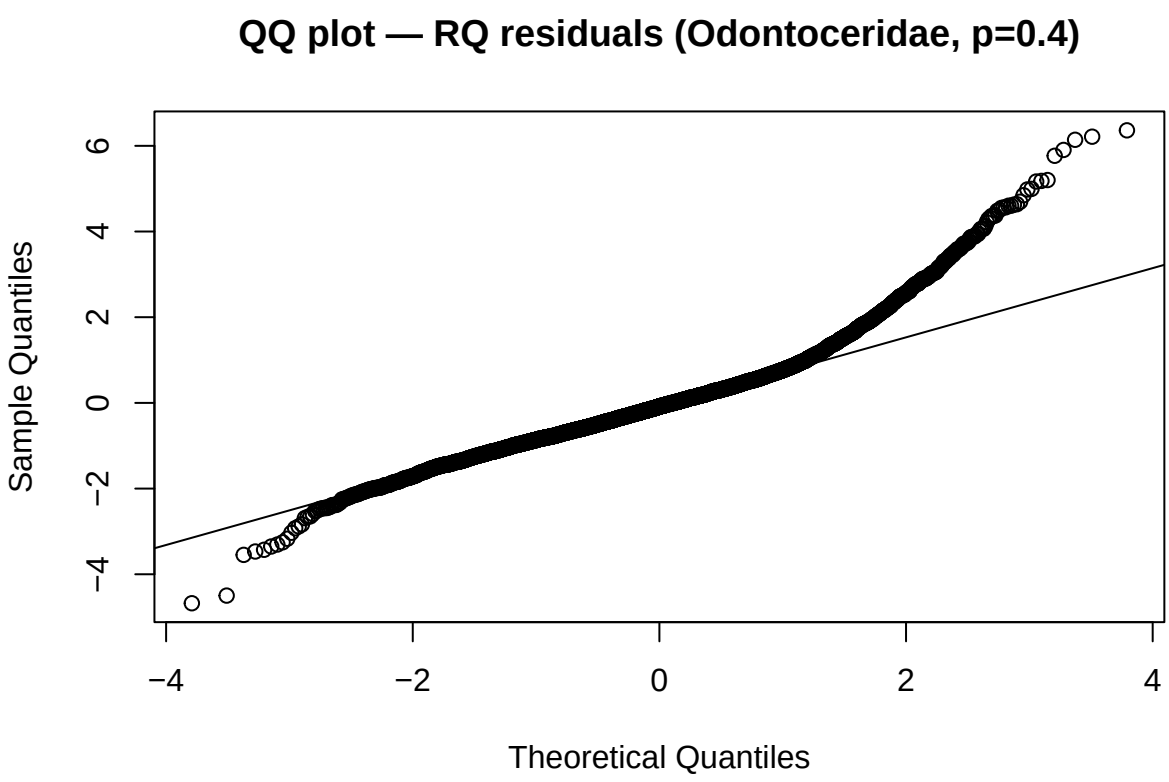

Histogram of RQ residuals — Brachycentridae (p=0.35)
Frequency
900
600
300
0
−5.0
−2.5
0.0
2.5
5.0
RQ residual

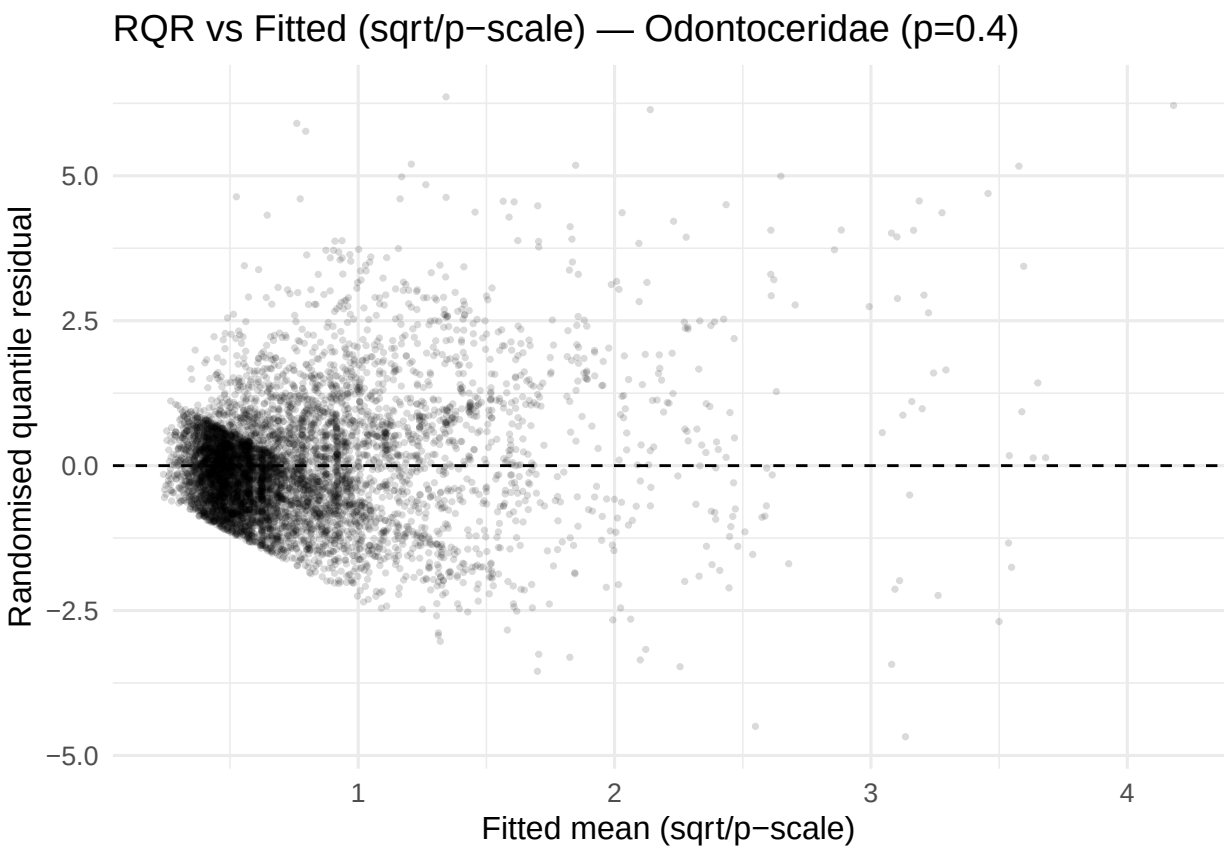

ACF of RQ residuals — Brachycentridae (p=0.35)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
Lag

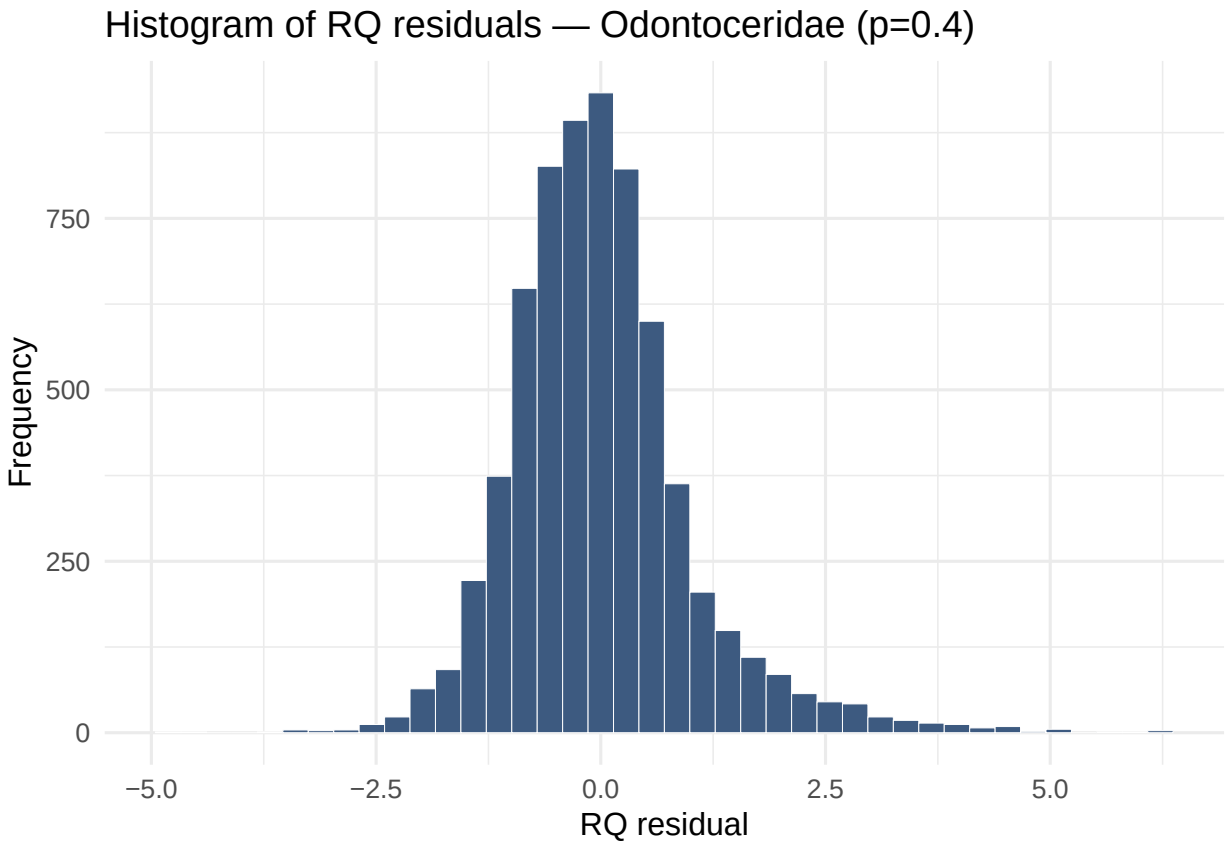

Time smooth (spring) — Brachycentridae (p=0.35)
By: season_f; spring
0.50
0.25
0.00
−0.25
−0.50
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

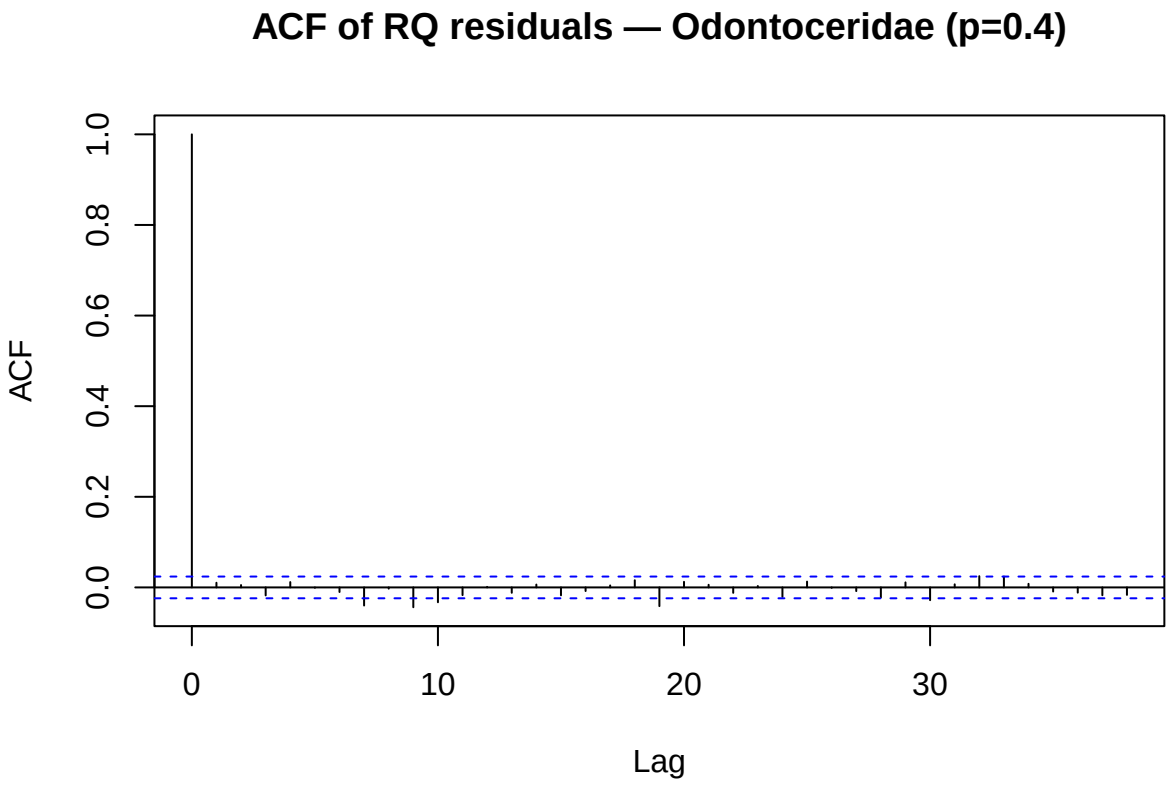


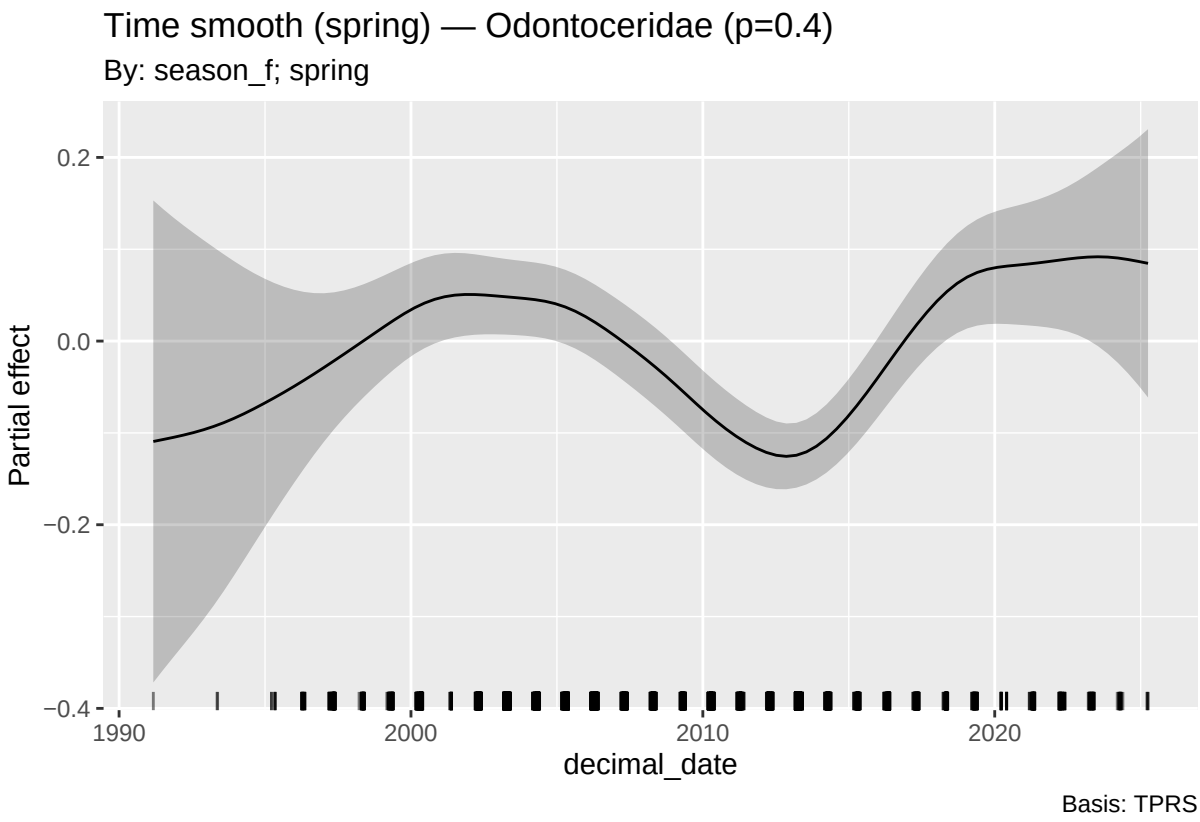

```
##
## [Brachycentridae] SD(RQR)=0.97 | Dev.expl=49.8% | AIC=21869.1 | N=5917
```

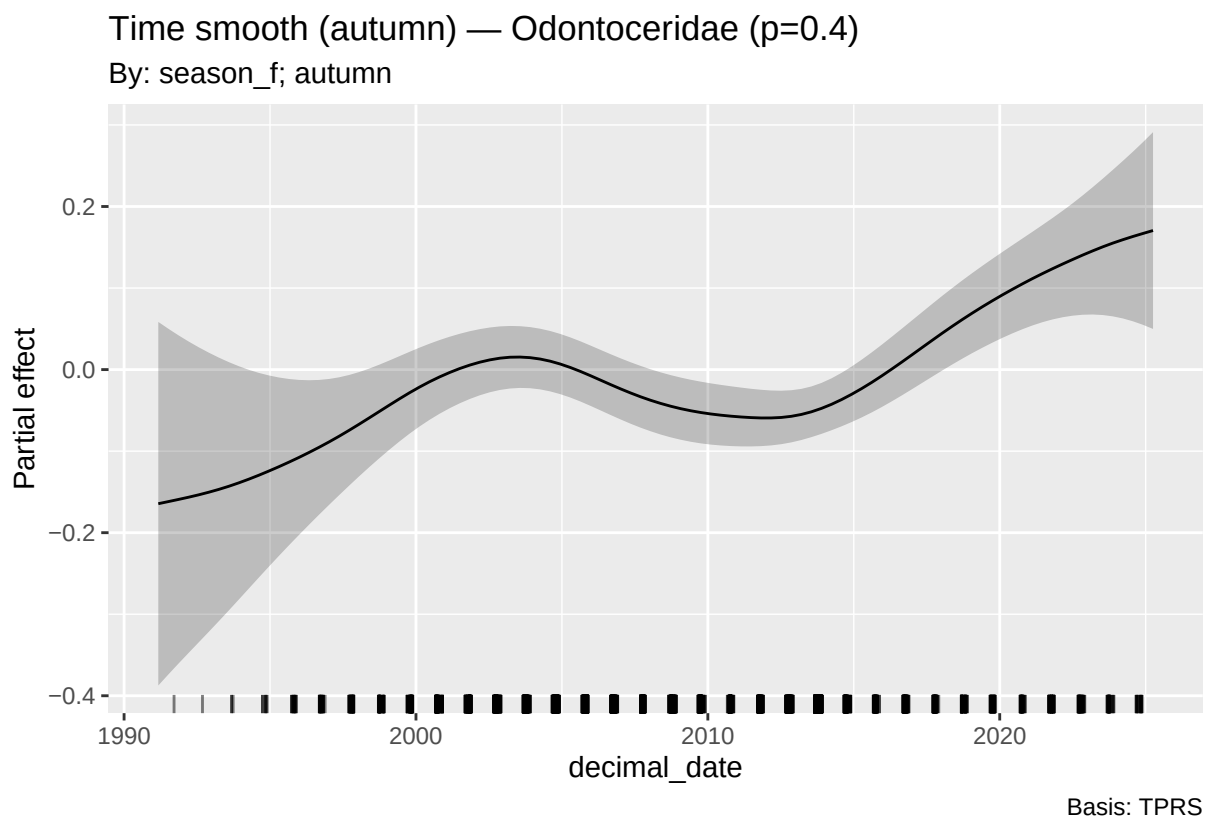

QQ plot — RQ residuals (Odontoceridae, p=0.4)
Sample Quantiles
Theoretical Quantiles
−4
−2
0
2
4
6


RQR vs Fitted (sqrt/p−scale) — Odontoceridae (p=0.4)
Randomised quantile residual
5.0
2.5
0.0
−2.5
−5.0
1
2
3
4
Fitted mean (sqrt/p−scale)

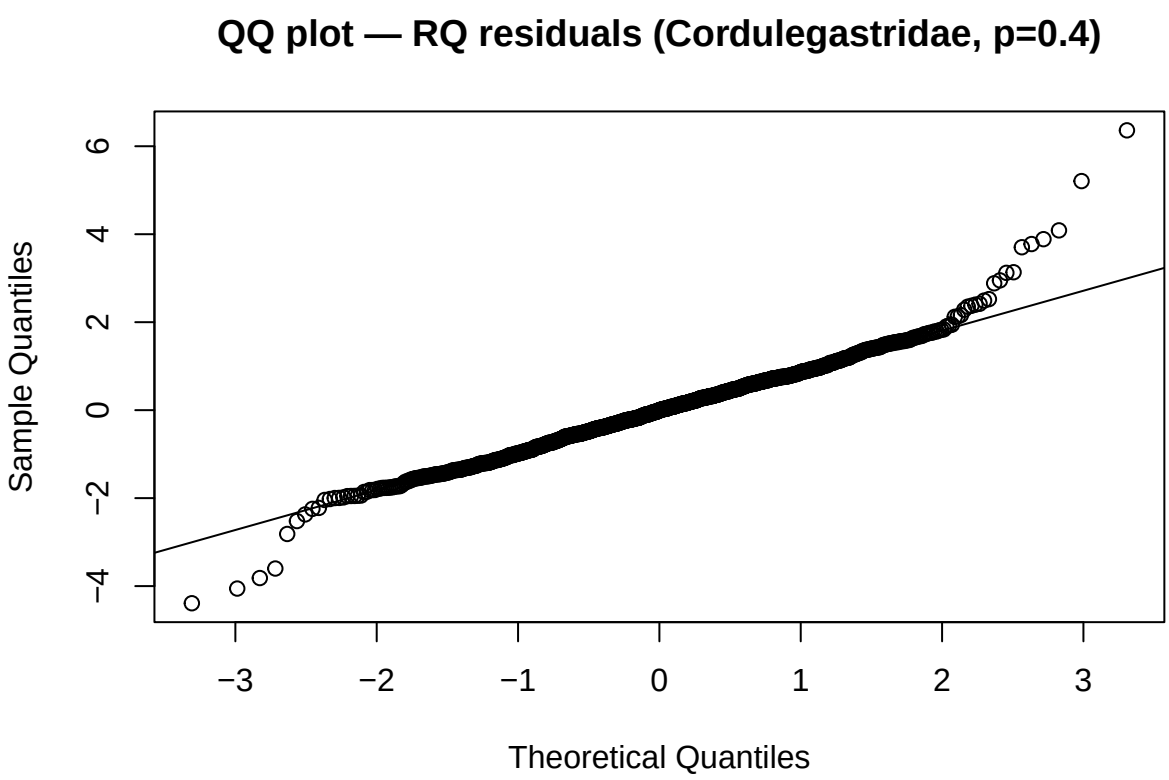

Histogram of RQ residuals — Odontoceridae (p=0.4)
Frequency
750
500
250
0
−5.0
−2.5
0.0
2.5
5.0
RQ residual

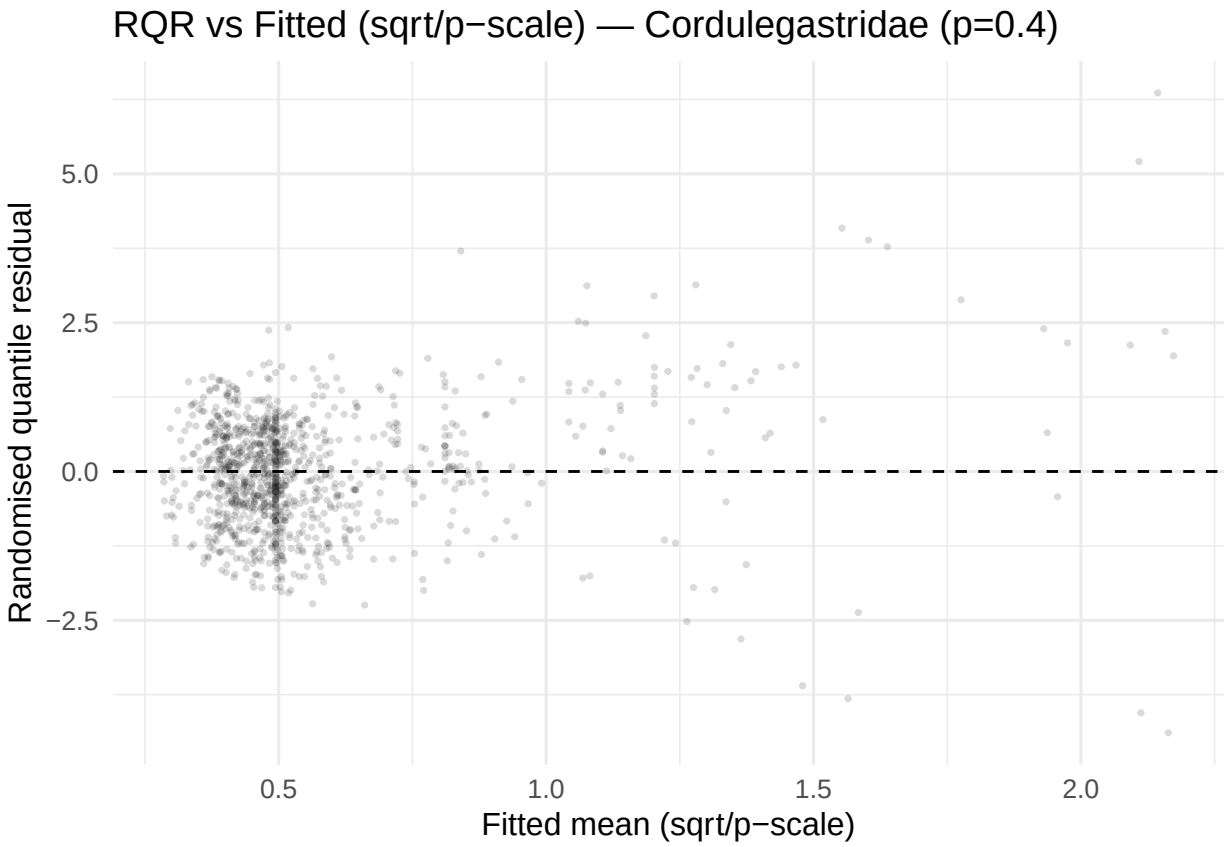

ACF of RQ residuals — Odontoceridae (p=0.4)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
Lag

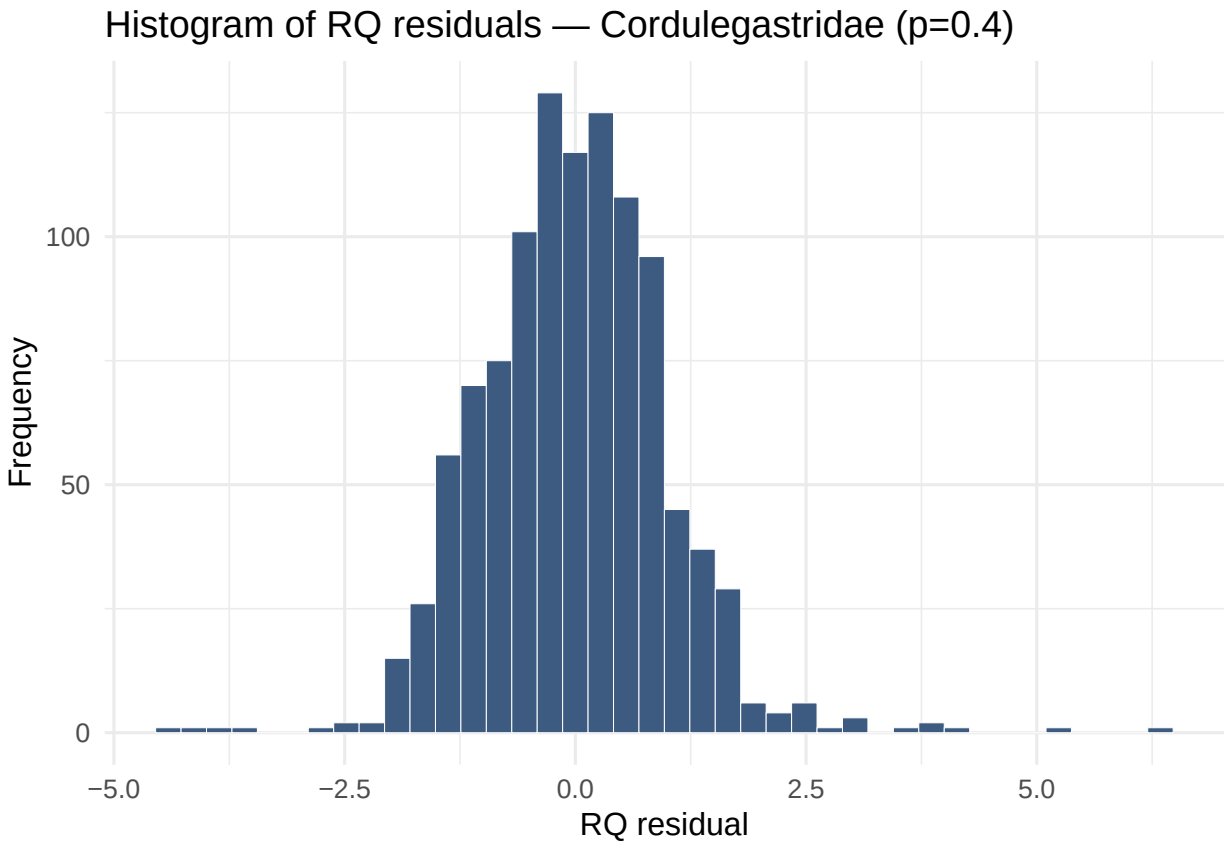

Time smooth (spring) — Odontoceridae (p=0.4)
By: season_f; spring
0.2
0.0
−0.2
−0.4
Partial effect
1990
2000
2010
2020
decimal_date
Basis: TPRS

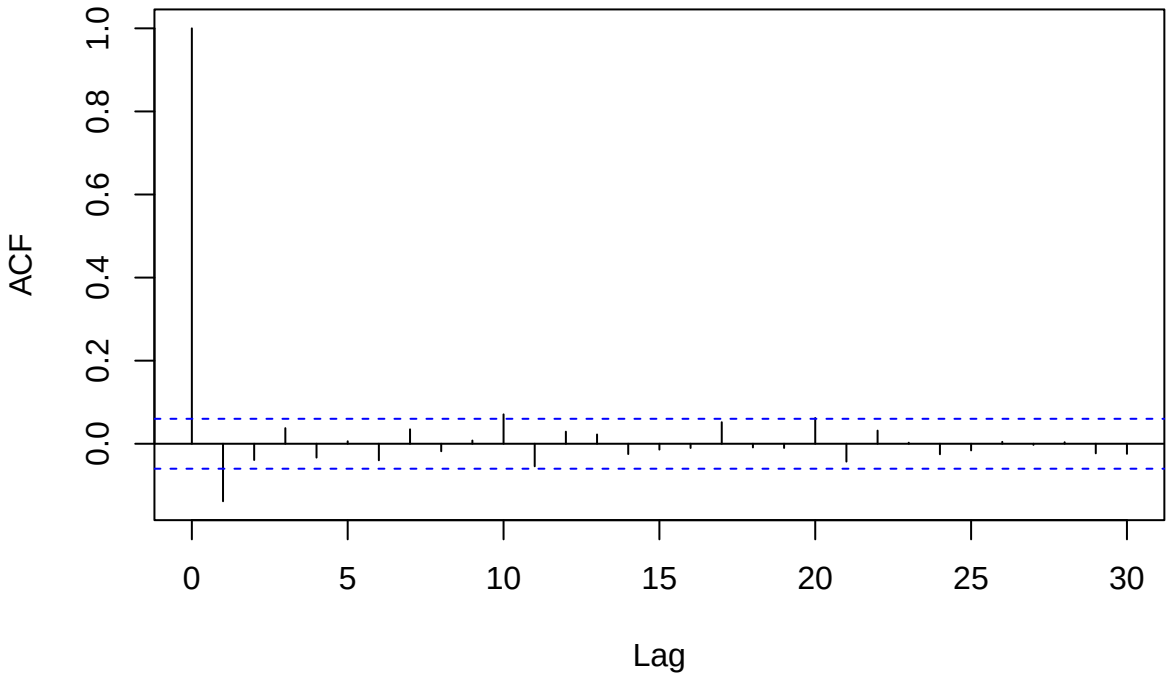


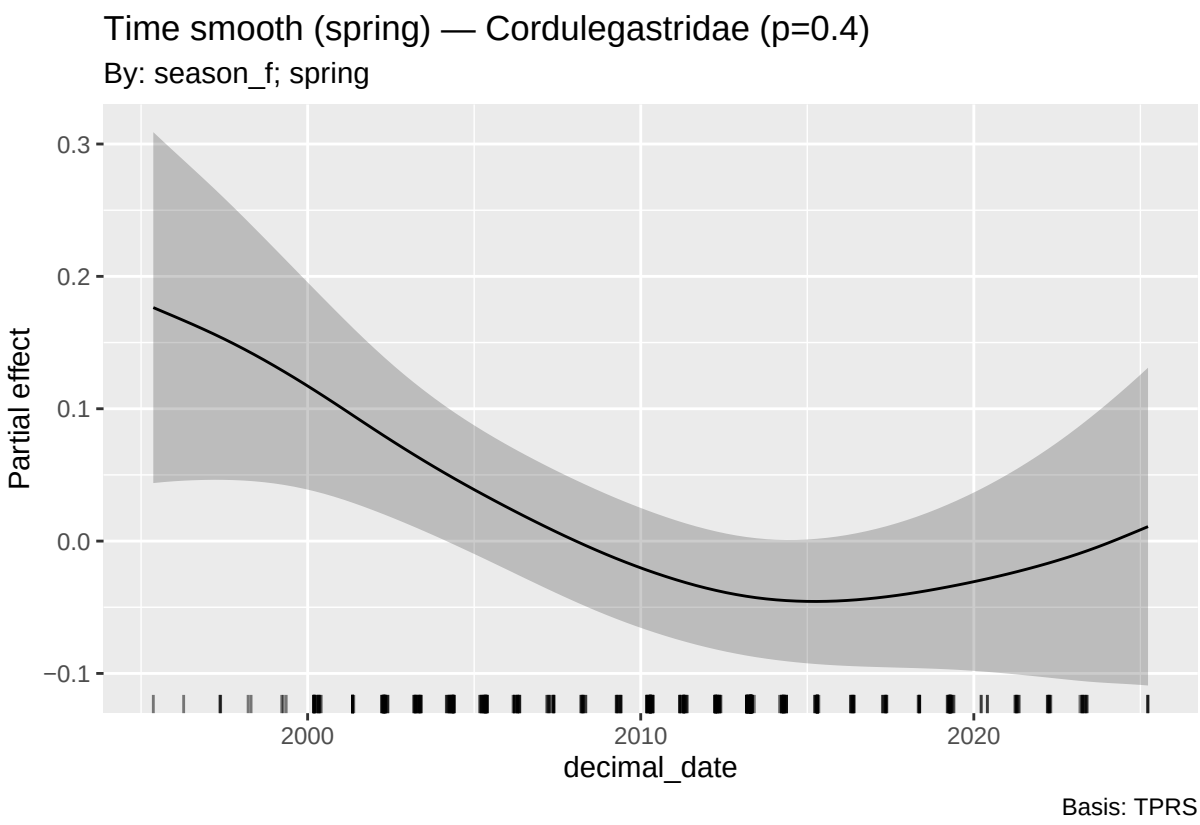

```
##
## [Odontoceridae] SD(RQR)=1.00 | Dev.expl=56.2% | AIC=17987.6 | N=6674
```

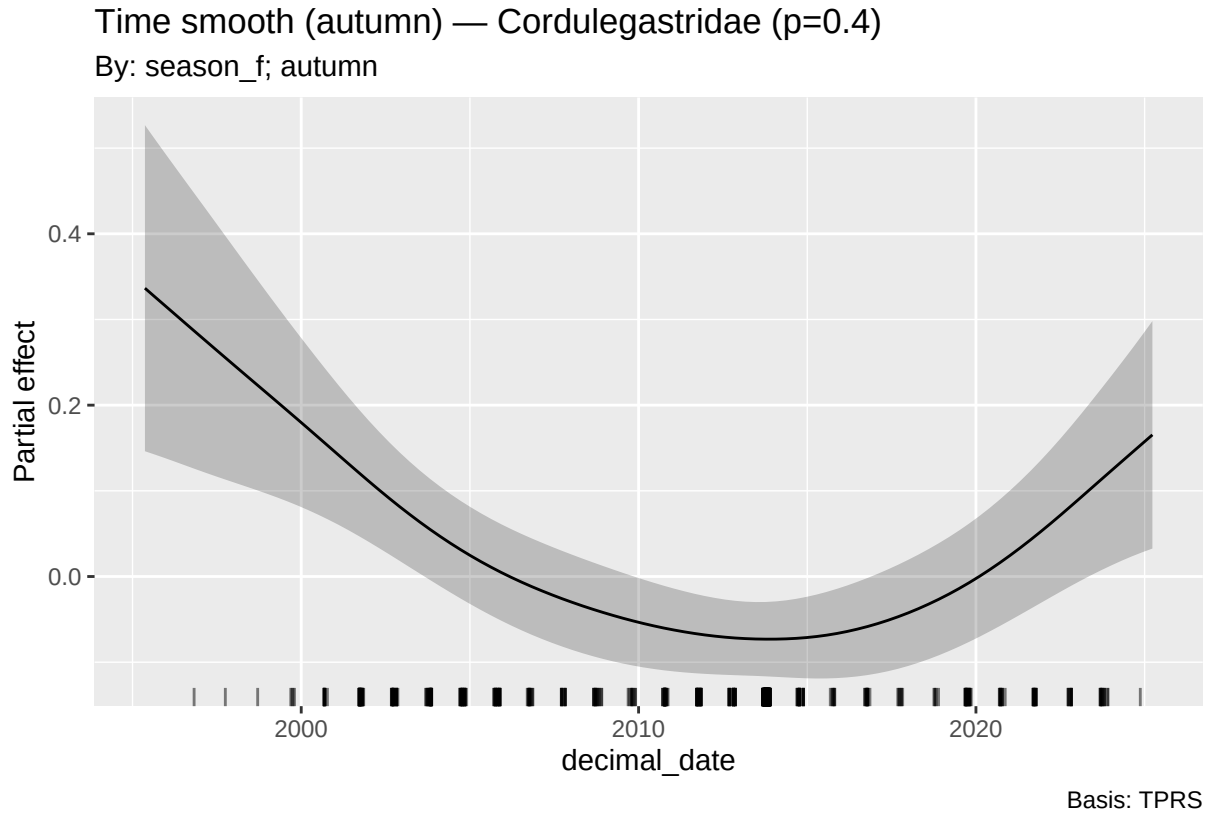

QQ plot — RQ residuals (Cordulegastridae, p=0.4)
Sample Quantiles
Theoretical Quantiles
−4
−2
0
2
4
6
−3
−2
−1
0
1
2
3


RQR vs Fitted (sqrt/p−scale) — Cordulegastridae (p=0.4)
Randomised quantile residual
5.0
2.5
0.0
−2.5
0.5
1.0
1.5
2.0
Fitted mean (sqrt/p−scale)


Histogram of RQ residuals — Cordulegastridae (p=0.4)
Frequency
100
50
0
−5.0
−2.5
0.0
2.5
5.0
RQ residual


ACF of RQ residuals — Cordulegastridae (p=0.4)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag


Time smooth (spring) — Cordulegastridae (p=0.4)
By: season_f; spring
0.3
0.2
0.1
0.0
−0.1
Partial effect
2000
2010
2020
decimal_date
Basis: TPRS




```
##
## [Cordulegastridae] SD(RQR)=1.00 | Dev.expl=72.1% | AIC=1552.7 | N=1064
```

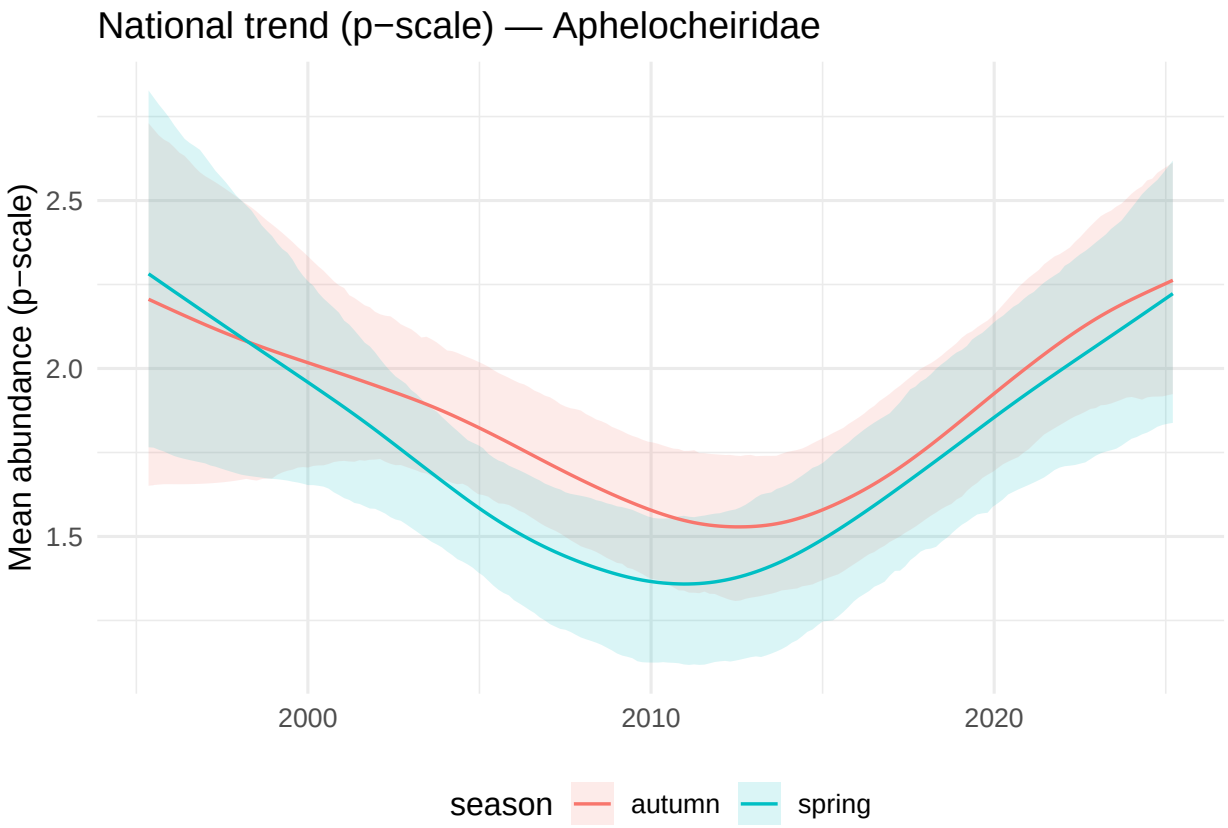
```
# ============================================================
# National marginal trend prediction from final cnorm models
# ------------------------------------------------------------
# What this script does
# 1) Takes each fitted cnorm model in `final_results` (one per family).
# 2) Builds a date grid and generates *marginal* predictions:
#      - excludes the site random effect via `exclude = "s(SITE_ID.F)"`
#      - predicts separately for spring and autumn, plus a 50/50 seasonal average
# 3) Simulates parameter uncertainty from the model's covariance to produce
#    95% intervals on the p-scale, and back-transforms to a count-scale index.
# 4) Binds all families together, writes a CSV, and (optionally) plots.
#
# Notes
# - The "count-scale" here is a back-transformed index: (E[Y^p])^(1/p).
#   It is intended for relative comparisons over time; it is not an unbiased
#   estimator of E[Y] on the raw-count scale.
# - We set the site factor to a *real observed* level then exclude the RE,
#   which avoids "new level" warnings while still giving marginal curves.
# - Plots are optional; set MAKE_PLOTS <- FALSE to skip.
# ============================================================
```

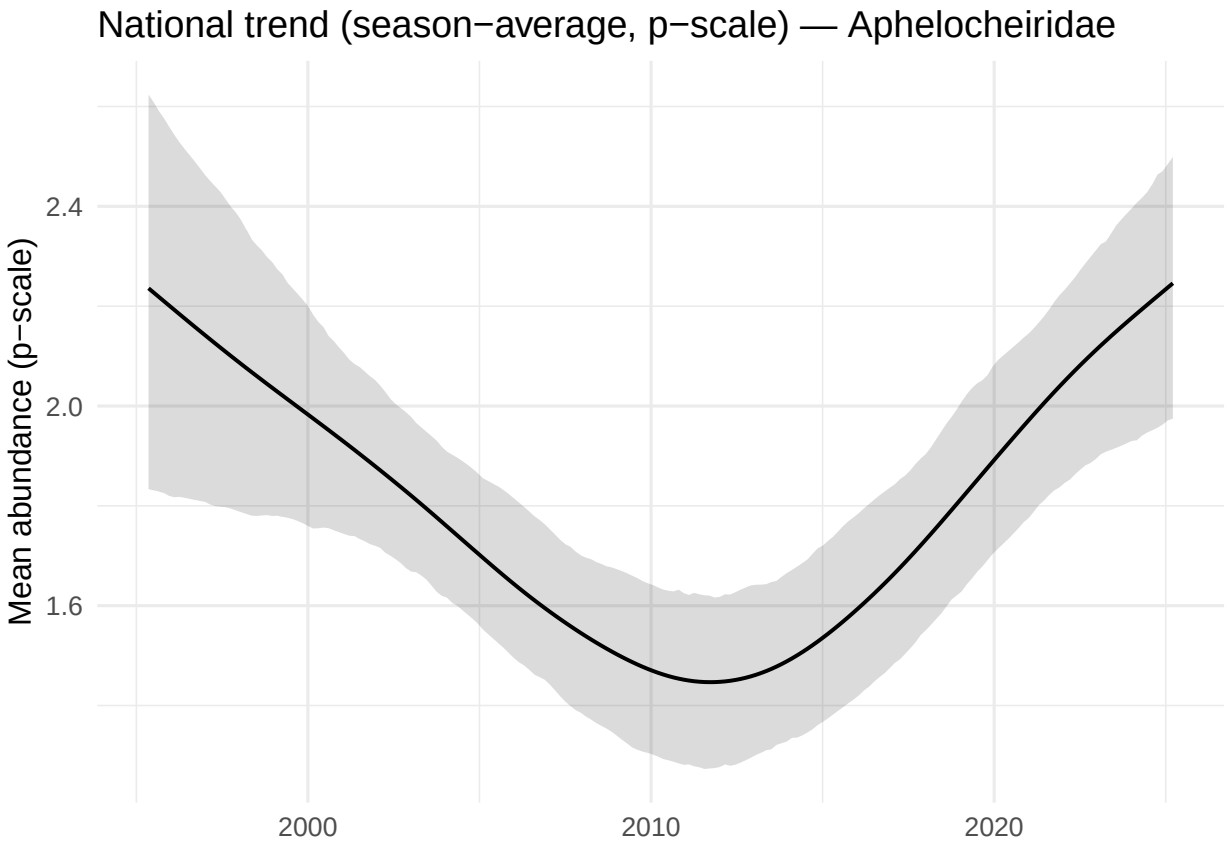
```r
MAKE_PLOTS <- TRUE

# --- helpers -----------------------------------------------------
# Extract the fitted Normal sigma used by mgcv's cnorm() family from the model object.
# (Kept here for completeness; this script works off the linear predictor draws.)
get_sigma_cnorm <- function(fit) {
  fam_str <- fit$family$family
  m <- regexpr("cnorm\\(([^)]+)\\)", fam_str)
  if (m > 0) as.numeric(sub("cnorm\\(([^)]+)\\)", "\\1", regmatches(fam_str, m))) else 1.0
}

# Draw from the posterior of the linear predictor ( ) at `newdata`
# by simulating coefficients ~ N(b, V) and multiplying by the model matrix.
# - `exclude` can drop specific smooths (e.g., the site RE) from the lpmatrix.
# - Returns a matrix [nsim × n_new] of linear predictor draws on the model's p-scale.
posterior_mu <- function(fit, newdata, nsim = 500, exclude = NULL, seed = 123, unconditional = FALSE) {
  X <- predict(fit, newdata = newdata, type = "lpmatrix", exclude = exclude)
  b <- coef(fit)
  V <- vcov(fit, unconditional = unconditional)
  set.seed(seed)
  B <- MASS::mvrnorm(n = nsim, mu = b, Sigma = V)
  as.matrix(B %*% t(X))
}

# Create a regular date grid spanning the observed sample dates in a family dataset.
# Adds decimal_date for the time smooth.
make_grid <- function(df, n = 200) {
  rng <- range(df$SAMPLE_DATE, na.rm = TRUE)
  dates <- seq(rng[1], rng[2], length.out = n)
  tibble(
    SAMPLE_DATE = dates,
    decimal_date = lubridate::decimal_date(dates)
  )
}

# Summarise a matrix of draws into mean and 95% CI on the p-scale,
# then back-transform to a count-scale *index* (^(1/p)).
# Includes the family/season/date fields for plotting/aggregation.
summarise_draws <- function(mu_draws, family, season, dates, p) {
  qs <- t(apply(mu_draws, 2, quantile, probs = c(0.025, 0.5, 0.975), na.rm = TRUE))
  mu <- colMeans(mu_draws, na.rm = TRUE)
  tibble::tibble(
    family = family,
    season = season,
    SAMPLE_DATE = dates,
    decimal_date = lubridate::decimal_date(dates),
    mu_p  = mu,            # mean on p-scale
    lo_p  = qs[,1],        # 2.5% p-scale
    med_p = qs[,2],        # 50%  p-scale
    hi_p  = qs[,3],        # 97.5% p-scale
    mu_count = pmax(mu, 0)^(1/p),       # back-transform index
    lo_count = pmax(qs[,1], 0)^(1/p),   # back-transform index (lower)
    hi_count = pmax(qs[,3], 0)^(1/p)    # back-transform index (upper)
```

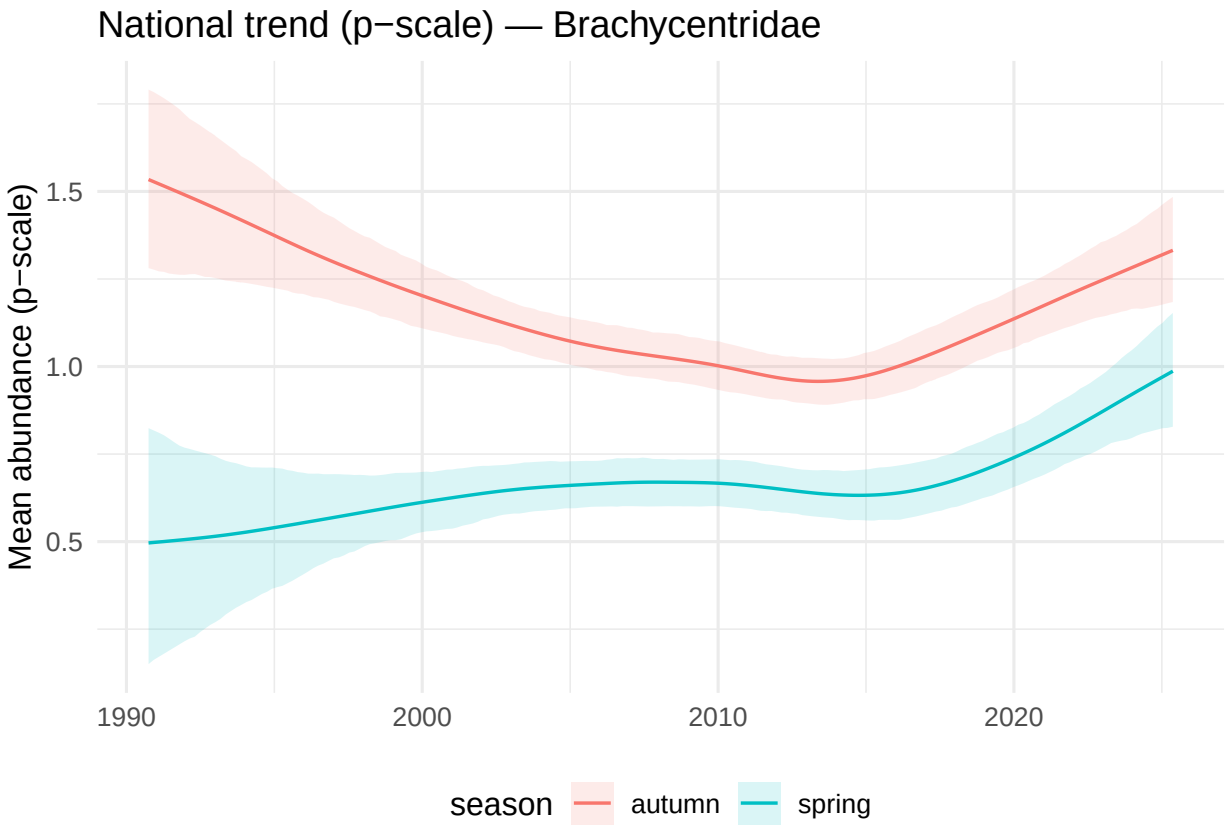
```
  )
}

# For one family's fitted model:
# - Build a time grid, predict spring & autumn marginal curves (exclude site RE),
# - Optionally compute a season-average curve by averaging spring and autumn draws,
# - Return a tidy tibble of summaries on both p-scale and back-transformed index.
predict_marginal <- function(res, nsim = 500, seed = 123, make_avg = TRUE) {
  fit <- res$model
  df  <- res$data
  fam <- res$family
  pwr <- res$p

  grid <- make_grid(df, n = 200)

  # Choose a *real* site level to define SITE_ID.F in newdata, then exclude the RE.
  # This avoids factor-level warnings while producing marginal predictions.
  first_site <- as.character(df$SITE_ID.F[1])
  site_levels <- levels(df$SITE_ID.F)
  if (!first_site %in% site_levels) first_site <- site_levels[1]

  seasons <- c("spring","autumn")
  out_list <- vector("list", length(seasons))

  for (i in seq_along(seasons)) {
    newd <- grid %>%
      dplyr::mutate(
        season_f = factor(seasons[i], levels = c("spring","autumn")),
        SITE_ID.F = factor(first_site, levels = site_levels)
      )
    # Exclude the site random effect to get population-level (marginal) curves.
    draws <- posterior_mu(fit, newd, nsim = nsim, exclude = "s(SITE_ID.F)", seed = seed + i)
    out_list[[i]] <- summarise_draws(draws, fam, seasons[i], newd$SAMPLE_DATE, pwr)
  }

  res_seasons <- dplyr::bind_rows(out_list)

  if (!make_avg) return(res_seasons)

  # Make a season-average by averaging draws (preserves uncertainty properly).
  newd_spr <- grid %>%
    dplyr::mutate(season_f = factor("spring", levels = c("spring","autumn")),
                  SITE_ID.F = factor(first_site, levels = site_levels))
  newd_aut <- grid %>%
    dplyr::mutate(season_f = factor("autumn", levels = c("spring","autumn")),
                  SITE_ID.F = factor(first_site, levels = site_levels))

  Dspr <- posterior_mu(fit, newd_spr, nsim = nsim, exclude = "s(SITE_ID.F)", seed = seed + 101)
  Daut <- posterior_mu(fit, newd_aut, nsim = nsim, exclude = "s(SITE_ID.F)", seed = seed + 102)
  Davg <- (Dspr + Daut) / 2

  res_avg <- summarise_draws(Davg, fam, "avg", grid$SAMPLE_DATE, pwr)
```

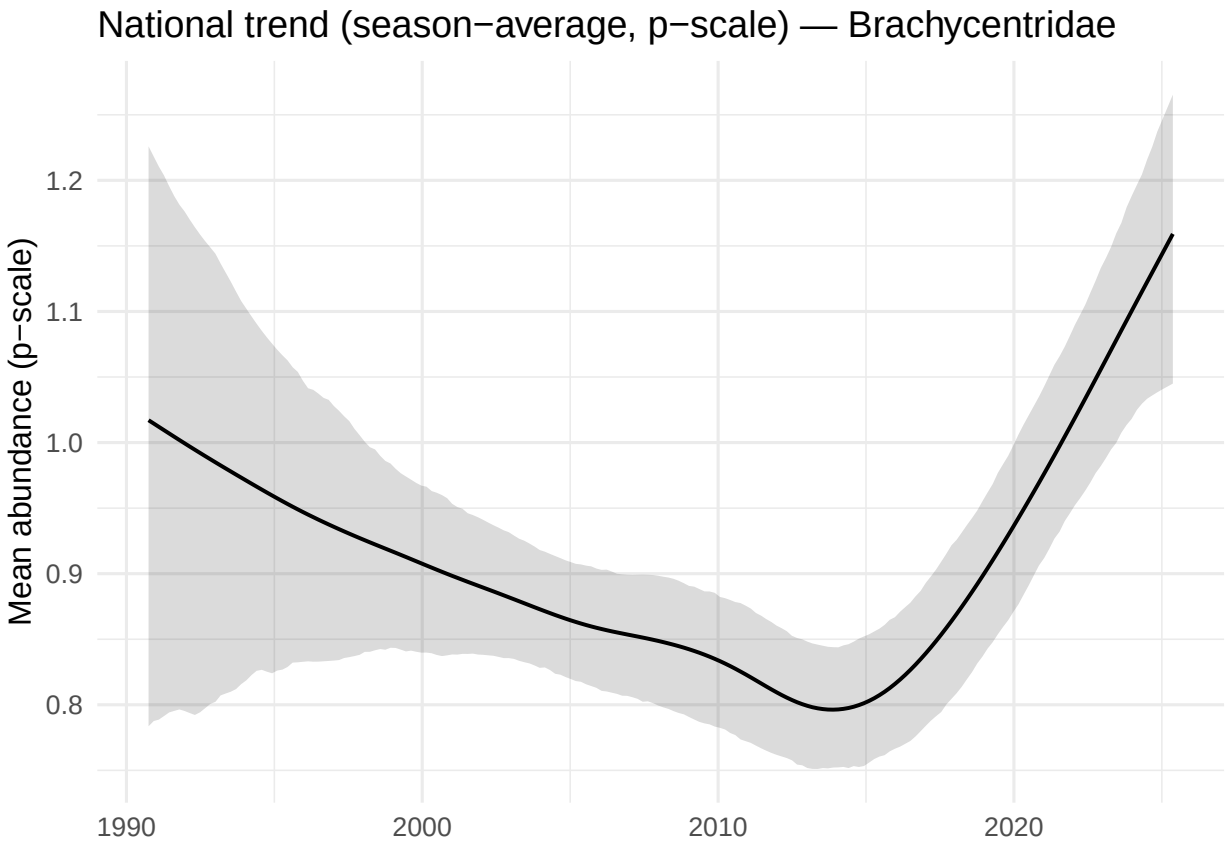
```r
  dplyr::bind_rows(res_seasons, res_avg)
}

# ---- run for all families ---------------------------------------
# Safety: require `final_results` to exist and be a list of fitted family objects.
stopifnot(exists("final_results"), is.list(final_results))

preds_marg_list <- lapply(final_results, function(res) {
  # Wrap in try() so one failing family does not stop the rest; keeps family name in error.
  try(predict_marginal(res, nsim = 600, seed = 2025), silent = TRUE)
})

# Drop any families that failed (e.g., due to missing fields)
preds_marg_list <- purrr::compact(preds_marg_list)

# Bind all families together and retain the key columns for saving/plotting
preds_marg_tbl <- dplyr::bind_rows(preds_marg_list) %>%
  dplyr::select(family, season, SAMPLE_DATE, mu_p, lo_p, hi_p, mu_count, lo_count, hi_count)

# Persist results for the report
dir.create("outputs", showWarnings = FALSE, recursive = TRUE)
readr::write_csv(preds_marg_tbl, file.path("outputs", "cnorm_predictions_marginal.csv"))

# ---- minimal plots (optional) ----------------------------------
# For each family:
#   1) Show spring & autumn marginal curves with 95% ribbons on the p-scale.
#   2) If the seasonal average was computed, show that as a single curve.
if (MAKE_PLOTS) {
  fams <- unique(preds_marg_tbl$family)
  for (fam in fams) {
    dfp <- preds_marg_tbl %>% dplyr::filter(family == fam)

    # Spring & autumn curves (p-scale)
    p1 <- ggplot(dfp %>% dplyr::filter(season %in% c("spring","autumn")),
                 aes(SAMPLE_DATE, mu_p, colour = season, fill = season)) +
      geom_ribbon(aes(ymin = lo_p, ymax = hi_p), alpha = 0.15, colour = NA) +
      geom_line(linewidth = 0.6) +
      labs(title = paste0("National trend (p-scale) - ", fam),
           x = NULL, y = "Mean abundance (p-scale)") +
      theme_minimal(base_size = 12) +
      theme(legend.position = "bottom")
    print(p1)

    # Season-average (if present)
    if ("avg" %in% unique(dfp$season)) {
      p2 <- ggplot(dfp %>% dplyr::filter(season == "avg"),
                   aes(SAMPLE_DATE, mu_p)) +
        geom_ribbon(aes(ymin = lo_p, ymax = hi_p), alpha = 0.18) +
        geom_line(linewidth = 0.7) +
        labs(title = paste0("National trend (season-average, p-scale) - ", fam),
             x = NULL, y = "Mean abundance (p-scale)") +
        theme_minimal(base_size = 12)
      print(p2)
```

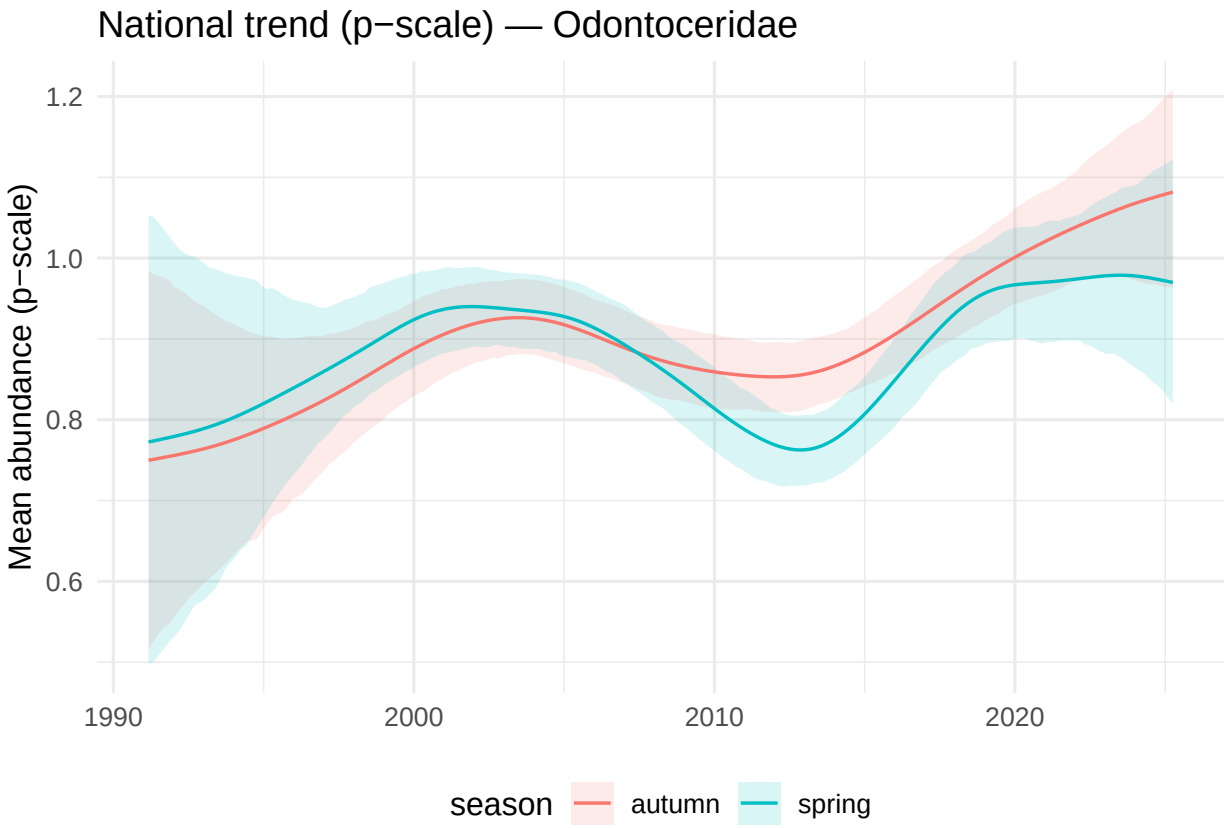
```
    }
  }
}
```

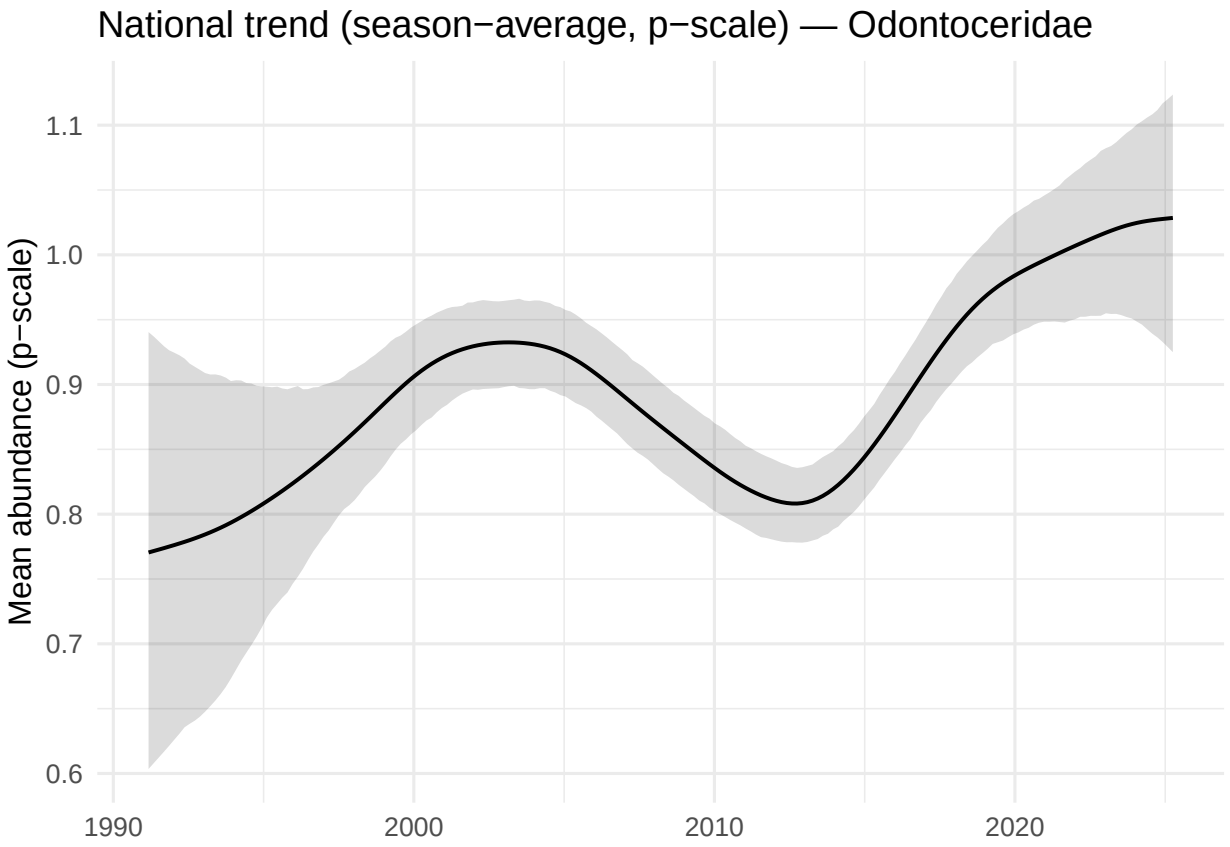

National trend (p−scale) — Aphelocheiridae
Mean abundance (p−scale)
2.5
2.0
1.5
2000
2010
2020
season
autumn
spring

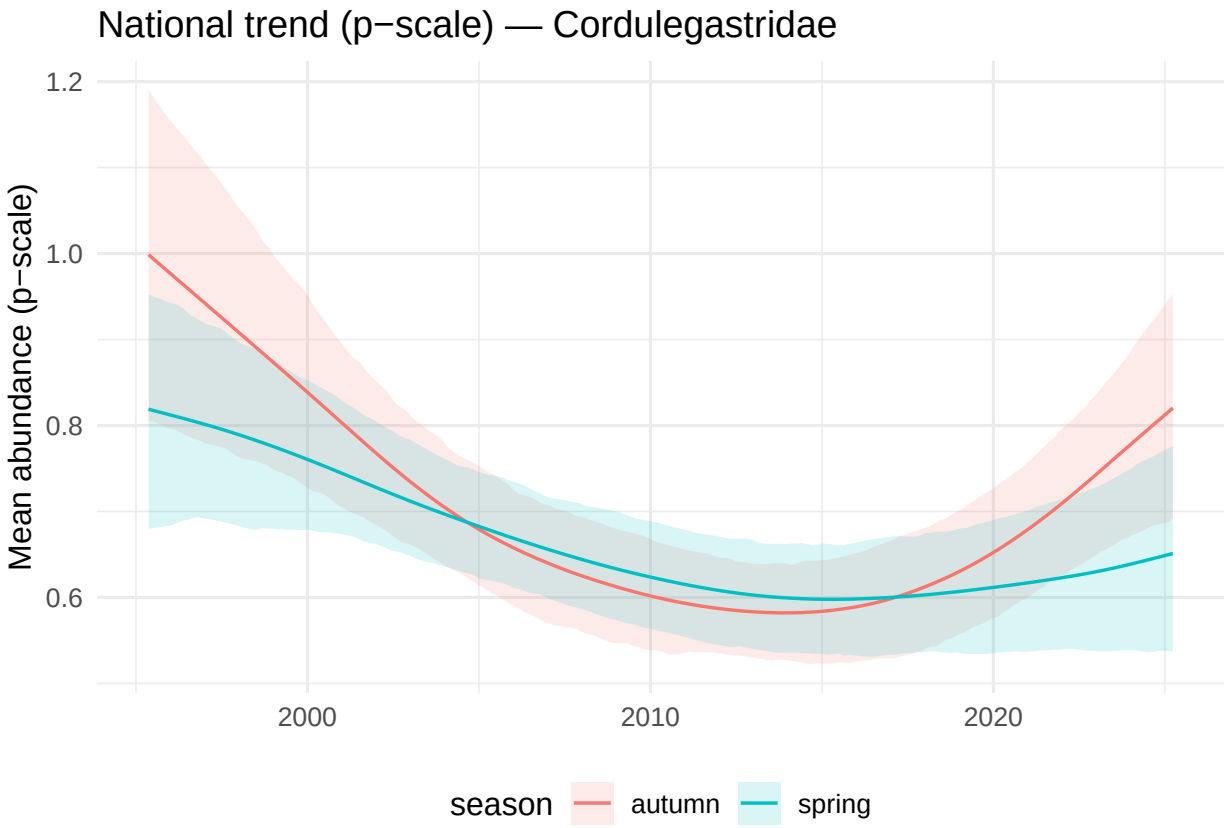

National trend (season−average, p−scale) — Aphelocheiridae
Mean abundance (p−scale)
2.4
2.0
1.6
2000
2010
2020

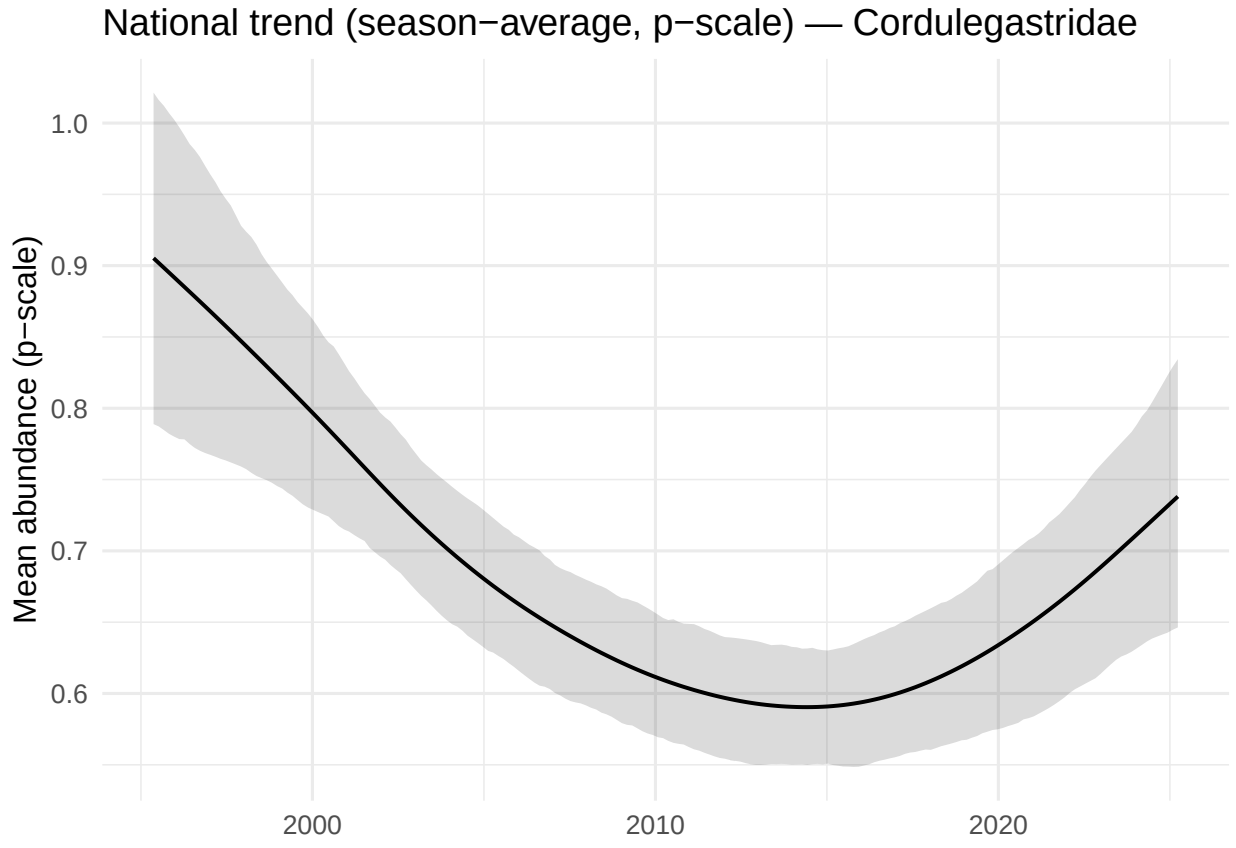

National trend (p−scale) — Brachycentridae
Mean abundance (p−scale)
1.5
1.0
0.5
1990
2000
2010
2020
season
autumn
spring


National trend (season−average, p−scale) — Brachycentridae
Mean abundance (p−scale)
1.2
1.1
1.0
0.9
0.8
1990
2000
2010
2020


National trend (p−scale) — Odontoceridae
Mean abundance (p−scale)
1.2
1.0
0.8
0.6
1990
2000
2010
2020
season
autumn
spring


National trend (season−average, p−scale) — Odontoceridae
Mean abundance (p−scale)
1.1
1.0
0.9
0.8
0.7
0.6
1990
2000
2010
2020

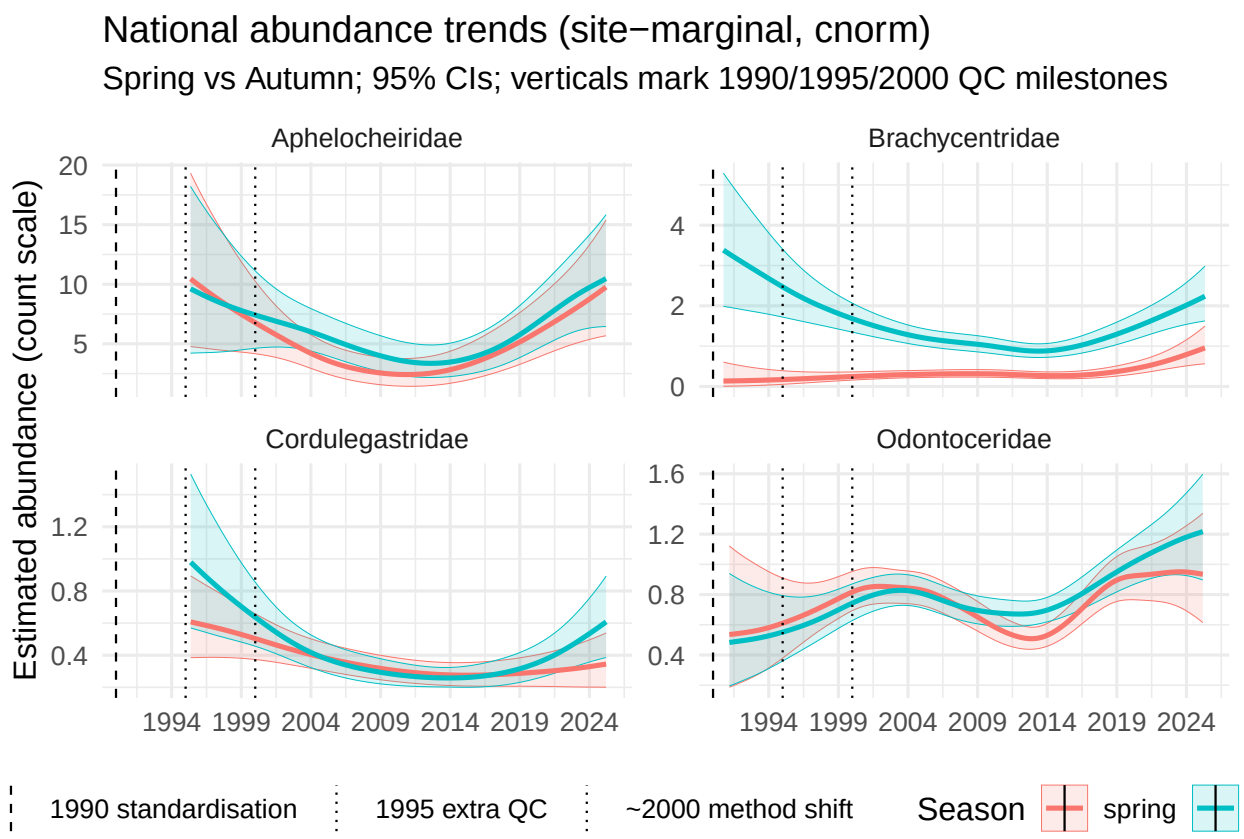

National trend (p−scale) — Cordulegastridae
Mean abundance (p−scale)
1.2
1.0
0.8
0.6
2000
2010
2020
season
autumn
spring




```
# ================================================================
# National trend curves (site-marginal) - FIXED (no mgcv::smooths)
# ----------------------------------------------------------------
# Purpose:
# • Produce national (population-level) trend curves per family and season
#   from final cnorm models, marginalising over site random effects.
# • Add 95% intervals and annotate key QC dates (1990, 1995, ~2000).
# Notes:
# • Uses the fitted model's factor levels to avoid "new level" issues.
# • Excludes the site RE via `exclude =` when predicting.
# • Back-transforms from p-scale to a count-scale *index* (^(1/p)).
# ================================================================

# --- helper: one family ---
nat_curve_one <- function(res) {
  m   <- res$model     # fitted cnorm model (bam)
  df  <- res$data      # modelling data used for the fit
  pwr <- res$p         # power transform used in cnorm (for back-transform)
  fam <- res$family    # family name (string)

  # Factor levels exactly as in the fitted model
  mf          <- stats::model.frame(m)       # extract model frame (to read factor levels)
  site_levels <- levels(mf$SITE_ID.F)        # levels for the site random effect factor
  seas_levels <- levels(mf$season_f)         # levels for the season factor (spring/autumn)

  # Monthly grid across observed span
```

```r
  dates <- seq(min(df$SAMPLE_DATE, na.rm = TRUE),   # start date
               max(df$SAMPLE_DATE, na.rm = TRUE),   # end date
               by = "1 month")                      # monthly resolution

  newd <- tidyr::expand_grid(
    SAMPLE_DATE = dates,                             # prediction dates (grid)
    season_f    = factor(seas_levels, levels = seas_levels)  # both seasons
  ) %>%
    mutate(
      decimal_date = lubridate::decimal_date(SAMPLE_DATE),   # time covariate for smooth
      SITE_ID.F    = factor(site_levels[1], levels = site_levels) # valid dummy site level
    )

  # Exclude site RE to marginalise over sites (use smooth labels from model)
  sm_labels <- vapply(m$smooth, function(s) s$label, character(1))  # names of smooth terms
  sm_ex     <- sm_labels[grepl("^s\\(SITE_ID\\.F", sm_labels)]      # pick the site RE smooth label(s)
  if (!length(sm_ex)) sm_ex <- NULL                                 # if no site RE smooth, leave NULL

  pr <- predict(m, newdata = newd, type = "response", se.fit = TRUE, exclude = sm_ex)
  # ^ Predict on the model (p) scale; `exclude` drops site RE to get the marginal curve
  #   se.fit=TRUE returns standard errors for 95% intervals

  # back-transform from power scale to count scale
  back <- function(x) pmax(0, x)^(1 / pwr)   # safe back-transform (non-negative)

  newd %>%
    mutate(mu_t = as.numeric(pr$fit),        # mean on p-scale
           se_t = as.numeric(pr$se.fit),     # SE on p-scale
           lo_t = mu_t - 1.96 * se_t,        # 95% lower (p-scale)
           hi_t = mu_t + 1.96 * se_t,        # 95% upper (p-scale)
           mu   = back(mu_t),                # back-transformed mean (index)
           lo   = back(lo_t),                # back-transformed lower (index)
           hi   = back(hi_t),                # back-transformed upper (index)
           family = fam)                     # carry family label
}

# Build for all families
nat_df <- map_dfr(final_results, nat_curve_one)
# ^ Apply to each fitted family result and row-bind to one tidy table

# Save tidy series (explicit dplyr::select to avoid masking)
dir.create("outputs", showWarnings = FALSE, recursive = TRUE)
readr::write_csv(
  dplyr::select(nat_df,
                family,
                season = season_f,           # rename for output clarity
                date   = SAMPLE_DATE,
                mu_count = mu,               # back-transformed mean index
                lo_count = lo,               # back-transformed 95% lower
                hi_count = hi),              # back-transformed 95% upper
  "outputs/national_trends_site_marginal.csv"
)
```

```r
# QC vertical lines via a data frame (handles linetype mapping cleanly)
vline_df <- tibble::tibble(
  x  = as.Date(c("1990-01-01","1995-01-01","2000-01-01")),  # key milestone dates
  lt = factor(c("1990 standardisation","1995 extra QC","~2000 method shift"),
              levels = c("1990 standardisation","1995 extra QC","~2000 method shift"))
)

# Plot
ggplot(nat_df, aes(SAMPLE_DATE, mu, colour = season_f, fill = season_f)) +
  geom_ribbon(aes(ymin = lo, ymax = hi), alpha = 0.15, linewidth = 0) +      # 95% CI ribbon (pale)
  geom_line(linewidth = 0.9) +                                               # mean curve
  facet_wrap(~ family, scales = "free_y", ncol = 2) +                        # panels per family
  geom_vline(data = vline_df, aes(xintercept = x, linetype = lt),linewidth = 0.4,
             colour = "black", show.legend = TRUE) +
  scale_linetype_manual(values = c("1990 standardisation" = 2,               # dashed/dotted styles
                                   "1995 extra QC"      = 3,
                                   "~2000 method shift" = 3),
                        name = NULL) +
  labs(title = "National abundance trends (site-marginal, cnorm)",
       subtitle = "Spring vs Autumn; 95% CIs; verticals mark 1990/1995/2000 QC milestones",
       x = NULL, y = "Estimated abundance (count scale)",
       colour = "Season", fill = "Season") +
  scale_x_date(date_breaks = "5 years", date_labels = "%Y") +                # tidy year ticks
  theme_minimal(base_size = 12) +
  theme(legend.position = "bottom")                                          # legend below
```



```r
theme_set(theme_minimal(base_size = 12))

# ------------------------------------------------------------
# 1) Presence map: early vs recent (facets)
# ------------------------------------------------------------
map_presence_2win <- function(fam,
                              years_early  = 1990:1994,
                              years_recent = 2020:2024) {

  pa <- models_data[[fam]]$pa
  if (!"taxon_present" %in% names(pa))
    pa$taxon_present <- pa$TOTAL_NUMBER > 0

  pres_site <- pa %>%
    dplyr::mutate(year = lubridate::year(SAMPLE_DATE),
                  period = dplyr::case_when(
                    year %in% years_early  ~ "Early",
                    year %in% years_recent ~ "Recent",
                    TRUE ~ NA_character_
                  )) %>%
    dplyr::filter(!is.na(period)) %>%
    dplyr::group_by(SITE_ID, period) %>%
    dplyr::summarise(pres = mean(as.integer(taxon_present) > 0),
                     n = dplyr::n(), .groups = "drop")

  pts <- dplyr::select(sites_clean, SITE_ID, FULL_EASTING, FULL_NORTHING) %>%
    dplyr::inner_join(pres_site, by = "SITE_ID") %>%
    sf::st_as_sf(coords = c("FULL_EASTING","FULL_NORTHING"), crs = 27700) %>%
    sf::st_transform(4326)

  uk <- rnaturalearth::ne_countries(scale = "medium", returnclass = "sf") |>
        dplyr::filter(admin %in% c("United Kingdom", "Ireland"))

  ggplot() +
    geom_sf(data = uk, fill = "grey95", colour = "grey70", linewidth = 0.2) +
    geom_sf(data = pts,
            aes(colour = pres, size = sqrt(pmax(n,1))),
            alpha = 0.85) +
    scale_colour_viridis_c(name = "Presence rate",
                           limits = c(0,1), labels = scales::label_percent(accuracy = 1)) +
    scale_size(range = c(0.6, 4), guide = guide_legend(title = "√ samples")) +
    coord_sf(xlim = c(-6, 2.2), ylim = c(49.5, 56.7)) +
    facet_wrap(~ period, nrow = 1) +
    labs(title = paste("Presence of", fam, "- early vs current"),
         subtitle = "Colour = share of samples with the family; size = √(samples at site)",
         caption = "EA open data (England sites); basemap: Natural Earth") +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank())
}

# ------------------------------------------------------------
# 2) Abundance maps (p-scale): early vs recent + change
# ------------------------------------------------------------
```

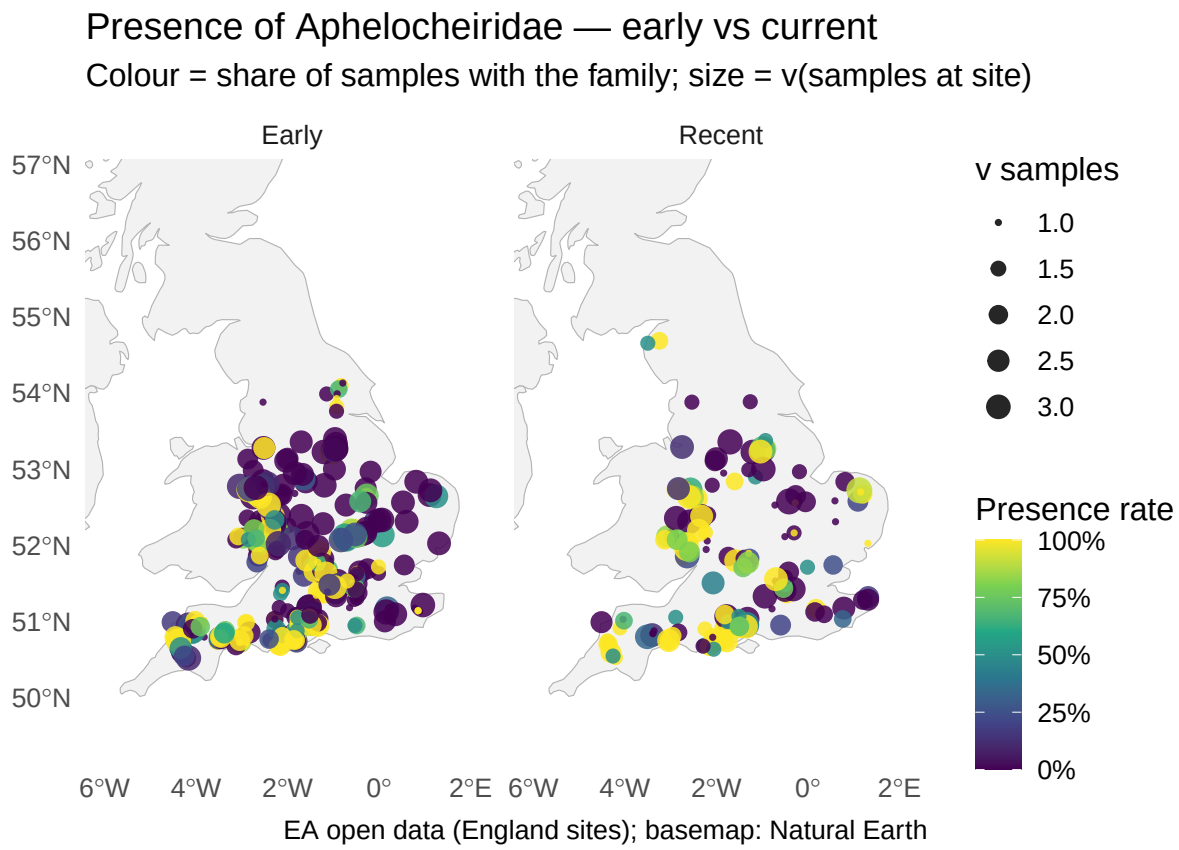
```r
plot_abundance_maps <- function(fam,
                                early_year_max = 2005,
                                bbox = c(xmin = -6.5, xmax = 2.5, ymin = 50, ymax = 56.5)) {
  stopifnot(fam %in% names(final_results))
  fit <- final_results[[fam]]$model
  df  <- final_results[[fam]]$data

  if (!"SITE_ID" %in% names(df)) stop("final_results[[fam]]$data must contain SITE_ID")
  if (!"season_f" %in% names(df)) stop("final_results[[fam]]$data must contain season_f")
  if (!"decimal_date" %in% names(df)) stop("final_results[[fam]]$data must contain decimal_date")
  if (!"SAMPLE_DATE" %in% names(df)) stop("final_results[[fam]]$data must contain SAMPLE_DATE")
  if (!"SITE_ID.F" %in% names(df)) df$SITE_ID.F <- factor(df$SITE_ID)

  site_period <- df %>%
    dplyr::mutate(period = dplyr::if_else(lubridate::year(SAMPLE_DATE) <= early_year_max,
                                          "early","recent")) %>%
    dplyr::group_by(SITE_ID, period) %>%
    dplyr::summarise(dec = mean(decimal_date), .groups = "drop") %>%
    tidyr::pivot_wider(names_from = period, values_from = dec) %>%
    dplyr::filter(!is.na(early), !is.na(recent))

  site_ids  <- site_period$SITE_ID
  site_fac  <- factor(site_ids, levels = levels(df$SITE_ID.F))
  levs_seas <- levels(df$season_f)

  newd <- tidyr::expand_grid(
    SITE_ID   = site_ids,
    SITE_ID.F = site_fac,
    season_f  = factor(levs_seas, levels = levs_seas),
    when      = c("Early","Recent")
  ) %>%
    dplyr::left_join(dplyr::rename(site_period, Early = early, Recent = recent),
                     by = "SITE_ID") %>%
    dplyr::mutate(decimal_date = ifelse(when == "Early", Early, Recent)) %>%
    dplyr::select(SITE_ID, SITE_ID.F, season_f, when, decimal_date)

  # Keep site REs (exclude = NULL)
  newd$mu_p <- as.numeric(predict(fit, newdata = newd, type = "response", exclude = NULL))

  site_mu <- newd %>%
    dplyr::group_by(SITE_ID, when) %>%
    dplyr::summarise(mu_p = mean(mu_p), .groups = "drop") %>%
    tidyr::pivot_wider(names_from = when, values_from = mu_p) %>%
    dplyr::mutate(delta = Recent - Early)

  coords <- sites_clean %>%
    dplyr::distinct(SITE_ID, FULL_EASTING, FULL_NORTHING) %>%
    dplyr::filter(!is.na(FULL_EASTING), !is.na(FULL_NORTHING)) %>%
    sf::st_as_sf(coords = c("FULL_EASTING","FULL_NORTHING"), crs = 27700) %>%
    sf::st_transform(4326) %>%
    dplyr::mutate(lon = sf::st_coordinates(geometry)[,1],
                  lat = sf::st_coordinates(geometry)[,2]) %>%
    sf::st_drop_geometry()
```

```
  plot_df <- dplyr::inner_join(site_mu, coords, by = "SITE_ID")

  lims <- stats::quantile(c(plot_df$Early, plot_df$Recent), c(.05, .95), na.rm = TRUE)
  D    <- stats::quantile(abs(plot_df$delta), .95, na.rm = TRUE)

  uk <- rnaturalearth::ne_countries(scale = "medium",
                                    country = "United Kingdom",
                                    returnclass = "sf") |>
        sf::st_transform(4326)

  base_map <- list(
    ggplot2::geom_sf(data = uk, fill = "grey95", colour = "grey70", linewidth = 0.3),
    ggplot2::coord_sf(xlim = c(bbox["xmin"], bbox["xmax"]),
                      ylim = c(bbox["ymin"], bbox["ymax"])),
    ggplot2::theme(panel.grid.major = element_line(linewidth = 0.2))
  )

  p_early <- ggplot2::ggplot() + base_map +
    ggplot2::geom_point(data = plot_df, ggplot2::aes(lon, lat, colour = Early),
                        size = 1.8, alpha = .9) +
    ggplot2::scale_colour_viridis_c(limits = lims, name = "Mean abundance\n(p-scale)") +
    ggplot2::labs(title = paste("Early (", early_year_max, ") -", fam))

  p_recent <- ggplot2::ggplot() + base_map +
    ggplot2::geom_point(data = plot_df, ggplot2::aes(lon, lat, colour = Recent),
                        size = 1.8, alpha = .9) +
    ggplot2::scale_colour_viridis_c(limits = lims, name = "Mean abundance\n(p-scale)") +
    ggplot2::labs(title = paste("Recent (", early_year_max + 1, ") -", fam))

  p_delta <- ggplot2::ggplot() + base_map +
    ggplot2::geom_point(data = plot_df, ggplot2::aes(lon, lat, colour = delta),
                        size = 2.0, alpha = .95) +
    ggplot2::scale_colour_gradientn(
      colours = c("#084594", "#F0F0F0", "#C51B8A"),  # negative, zero, positive
      values  = scales::rescale(c(-D, 0, D)),
      limits  = c(-D, D),
      labels  = scales::percent,
      name    = "Δ abundance\n(recent − early)",
      oob     = scales::squish
    ) +
    ggplot2::labs(title = paste("Change (recent − early) -", fam))

  list(early = p_early, recent = p_recent, change = p_delta)
}

# ---------------------------------------------------------------
# 3) Presence maps for specific years + a change map
# ---------------------------------------------------------------
.uk_basemap <- rnaturalearth::ne_countries(scale = "medium", returnclass = "sf") |>
  dplyr::filter(admin %in% c("United Kingdom","Ireland"))

.sites_ll <- sites_clean |>
  dplyr::select(SITE_ID, FULL_EASTING, FULL_NORTHING) |>
```

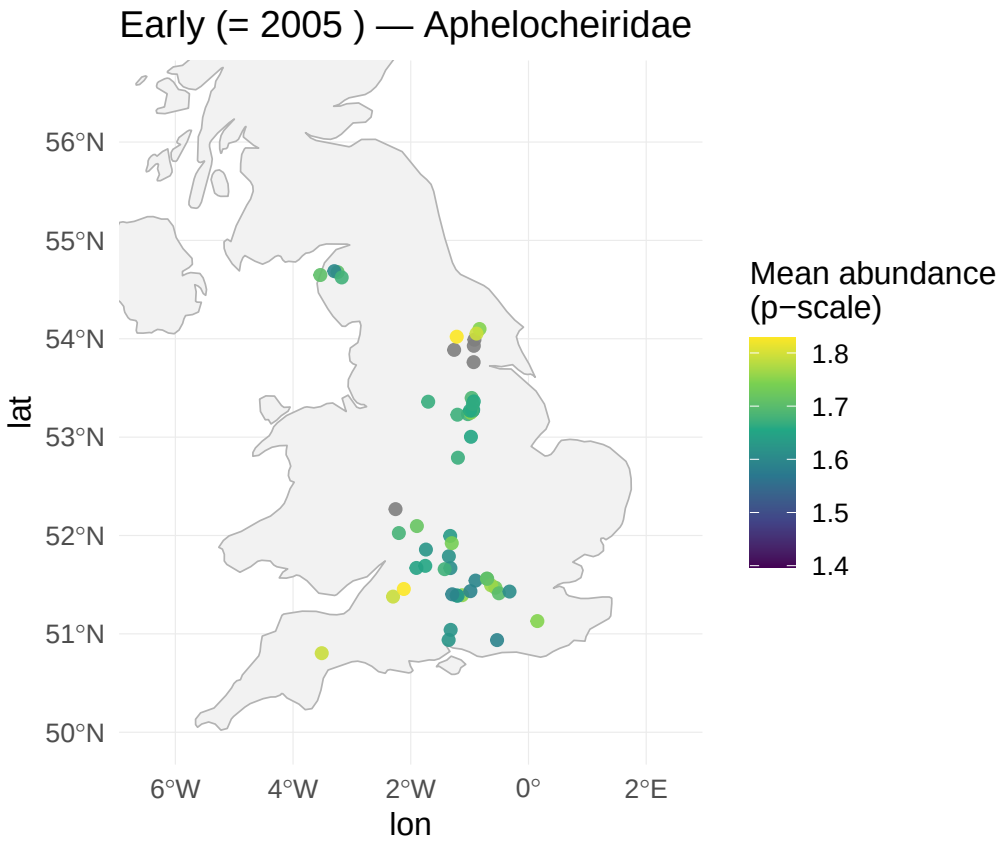
```
  dplyr::filter(!is.na(FULL_EASTING), !is.na(FULL_NORTHING)) |>
  sf::st_as_sf(coords = c("FULL_EASTING","FULL_NORTHING"), crs = 27700) |>
  sf::st_transform(4326) |>
  dplyr::mutate(
    lon = sf::st_coordinates(geometry)[,1],
    lat = sf::st_coordinates(geometry)[,2]
  ) |>
  sf::st_drop_geometry()

presence_window <- function(fam, year, window = 2) {
  rng <- as.Date(sprintf("%d-01-01", year + c(-window, window)))
  models_data[[fam]]$pa |>
    dplyr::select(SITE_ID, SAMPLE_DATE, taxon_present) |>
    dplyr::filter(SAMPLE_DATE >= rng[1], SAMPLE_DATE <= rng[2]) |>
    dplyr::group_by(SITE_ID) |>
    dplyr::summarise(
      n_samples = dplyr::n(),
      presence  = mean(taxon_present, na.rm = TRUE),
      .groups = "drop"
    ) |>
    dplyr::left_join(.sites_ll, by = "SITE_ID") |>
    dplyr::filter(!is.na(lon), !is.na(lat)) |>
    dplyr::mutate(size_sqrt = sqrt(pmax(n_samples, 1)))
}

plot_presence_year <- function(df_year, fam, year, window = 2) {
  ggplot() +
    geom_sf(data = .uk_basemap, fill = "grey97", colour = "grey75", linewidth = 0.3) +
    geom_point(data = df_year,
               aes(lon, lat, colour = presence, size = size_sqrt),
               alpha = 0.9, stroke = 0.15) +
    coord_sf(xlim = c(-6, 2.2), ylim = c(50, 56.5), expand = FALSE) +
    scale_colour_viridis_c(
      name = "Presence",
      labels = scales::percent_format(accuracy = 1),
      limits = c(0,1), oob = scales::squish
    ) +
    scale_size_continuous(
      name   = "sqrt(samples)",
      range  = c(1.2, 4.2),
      breaks = sqrt(c(1,2,4,9)),
      labels = c("1","2","4","9")
    ) +
    guides(size = guide_legend(override.aes = list(alpha = 1))) +
    labs(
      title    = sprintf("Presence of %s around %d (+/- %dy)", fam, year, window),
      subtitle = "Colour = share of samples with family present; size ~ sqrt(samples at site)",
      caption  = "EA open data (England sites); basemap: Natural Earth",
      x = NULL, y = NULL
    ) +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank())
}
```

```
presence_change_map <- function(fam,
                                early = c(1988, 1992),
                                late  = c(2023, 2027)) {
  pa <- models_data[[fam]]$pa |>
    dplyr::select(SITE_ID, SAMPLE_DATE, taxon_present)

  early_df <- pa |>
    dplyr::filter(SAMPLE_DATE >= as.Date(sprintf("%d-01-01", early[1])),
                  SAMPLE_DATE <= as.Date(sprintf("%d-12-31", early[2]))) |>
    dplyr::group_by(SITE_ID) |>
    dplyr::summarise(p_early = mean(taxon_present), n_early = dplyr::n(), .groups = "drop")

  late_df  <- pa |>
    dplyr::filter(SAMPLE_DATE >= as.Date(sprintf("%d-01-01", late[1])),
                  SAMPLE_DATE <= as.Date(sprintf("%d-12-31", late[2]))) |>
    dplyr::group_by(SITE_ID) |>
    dplyr::summarise(p_late = mean(taxon_present), n_late = dplyr::n(), .groups = "drop")

  dd <- dplyr::inner_join(early_df, late_df, by = "SITE_ID") |>
    dplyr::mutate(
      delta     = p_late - p_early,
      n_tot     = n_early + n_late,
      size_sqrt = sqrt(pmax(n_tot, 1))
    ) |>
    dplyr::left_join(.sites_ll, by = "SITE_ID") |>
    dplyr::filter(!is.na(lon), !is.na(lat))

  ggplot() +
    # slightly darker basemap so mid colour pops (was "grey97")
    geom_sf(data = .uk_basemap, fill = "grey90", colour = "grey75", linewidth = 0.3) +
    geom_point(data = dd, aes(lon, lat, colour = delta, size = size_sqrt), alpha = 0.9) +
    coord_sf(xlim = c(-6, 2.2), ylim = c(50, 56.5), expand = FALSE) +
    # non-white midpoint so zero-change doesn't vanish on the map
    scale_colour_gradientn(
     colours = c("#313695", "#4575B4", "#FDE725", "#F46D43", "#A50026"),
     values  = scales::rescale(c(-1, -0.25, 0, 0.25, 1)),
     limits  = c(-1, 1),
     labels  = scales::percent,
     name    = "Δ presence",
     oob     = scales::squish
     ) +
    scale_size_continuous(
      name = "sqrt(total samples)",
      range = c(1.2, 4.2),
      breaks = sqrt(c(1,2,4,9)),
      labels = c("2","4","9",">9")
    ) +
    labs(
      title    = sprintf("Change in presence - %s", fam),
      subtitle = sprintf("Early: %d-%d vs Late: %d-%d",
                         early[1], early[2], late[1], late[2]),
      caption  = "Colour = change in presence rate; size ~ sqrt(total samples in both windows).",
      x = NULL, y = NULL
```

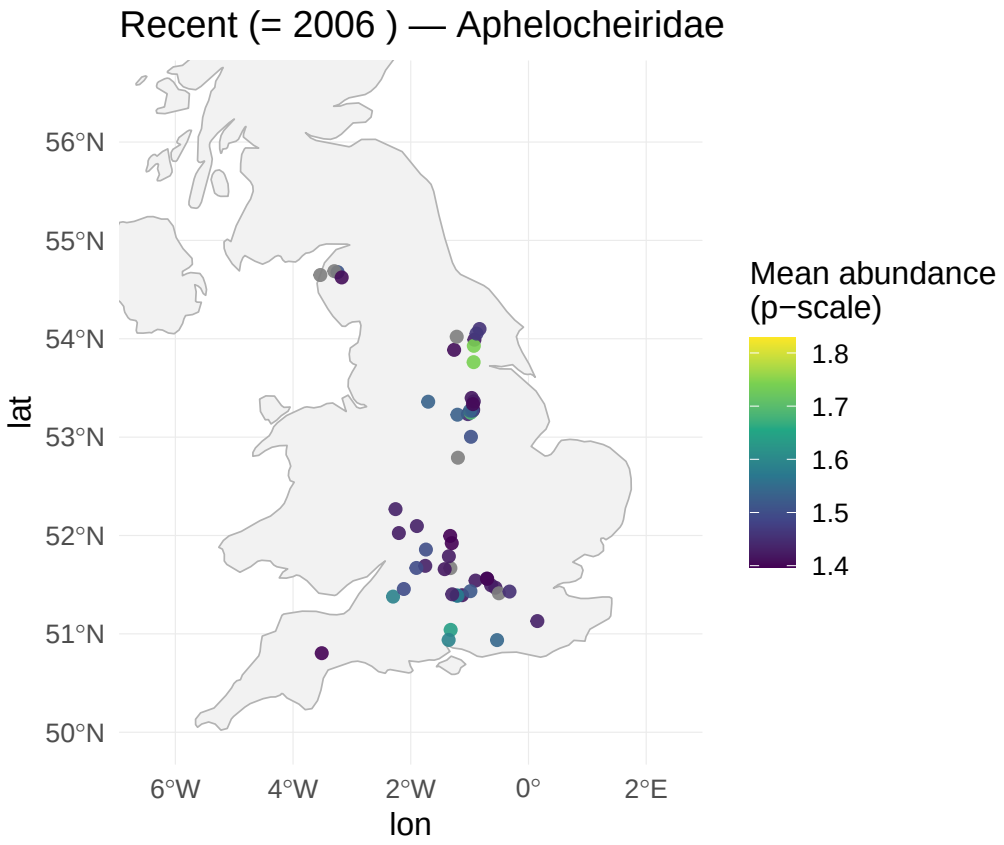
```
    ) +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank())
}

# ============================
# Example calls (PRINT them!)
# ============================
fam_example <- "Aphelocheiridae"

p_fac <- map_presence_2win(fam_example)
print(p_fac)
```



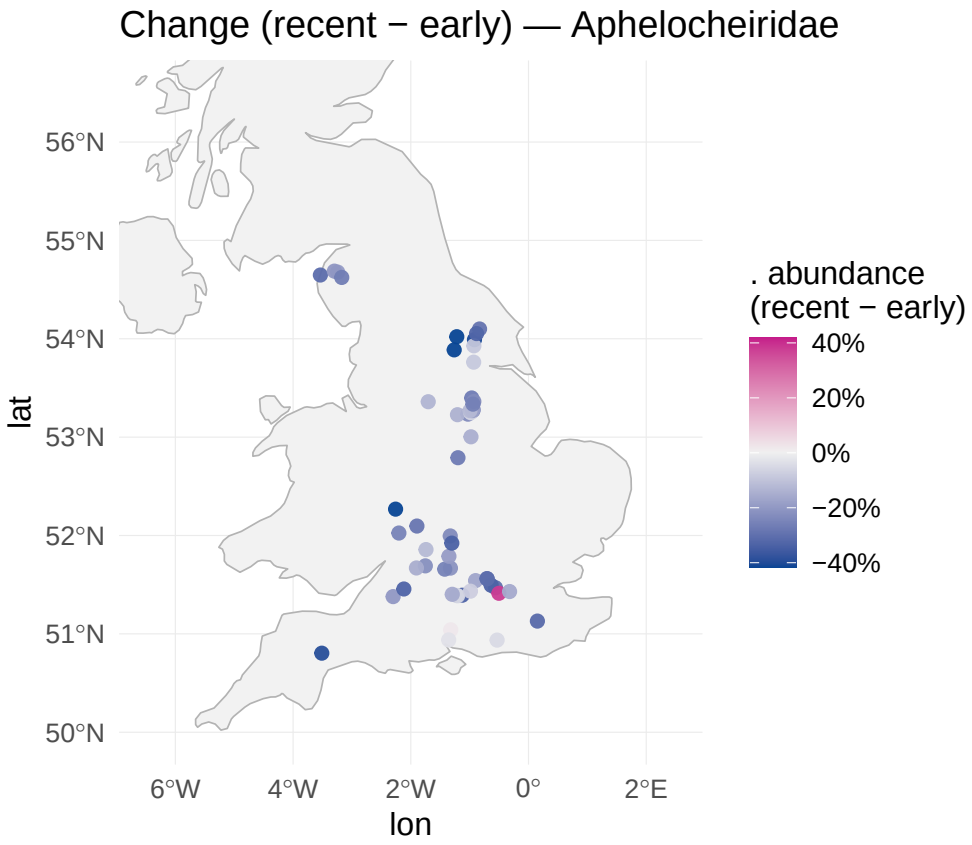

```
ab <- plot_abundance_maps(fam_example)
print(ab$early)
```



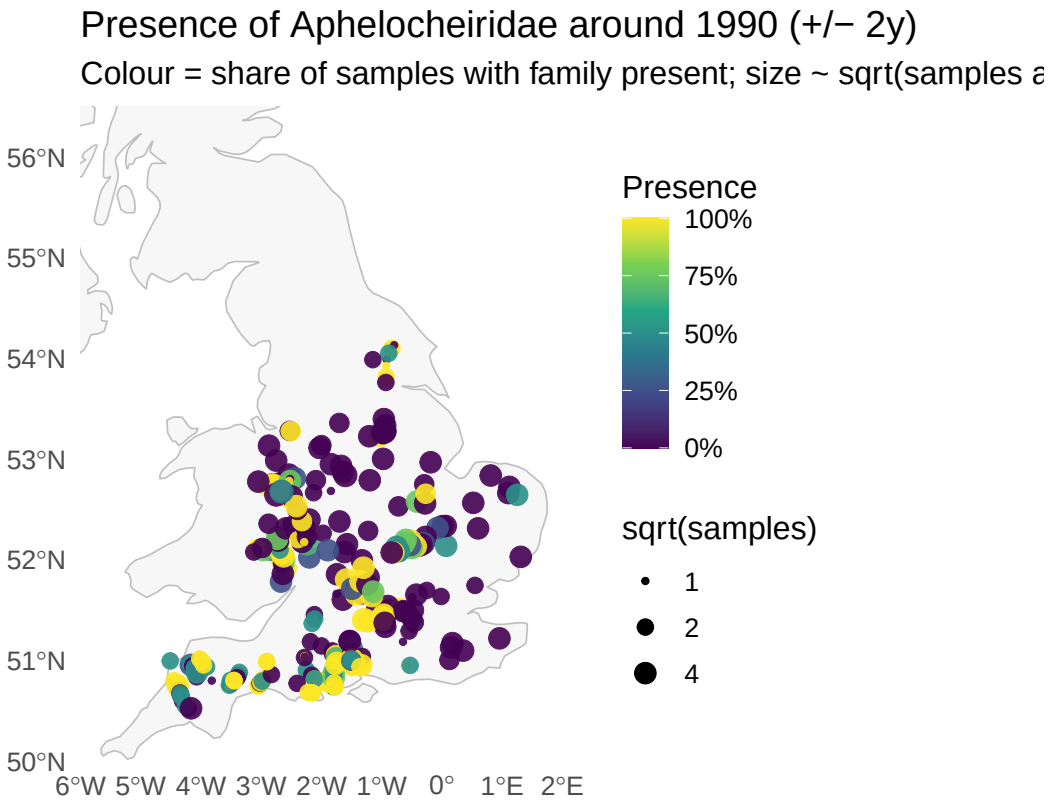

```
print(ab$recent)
```



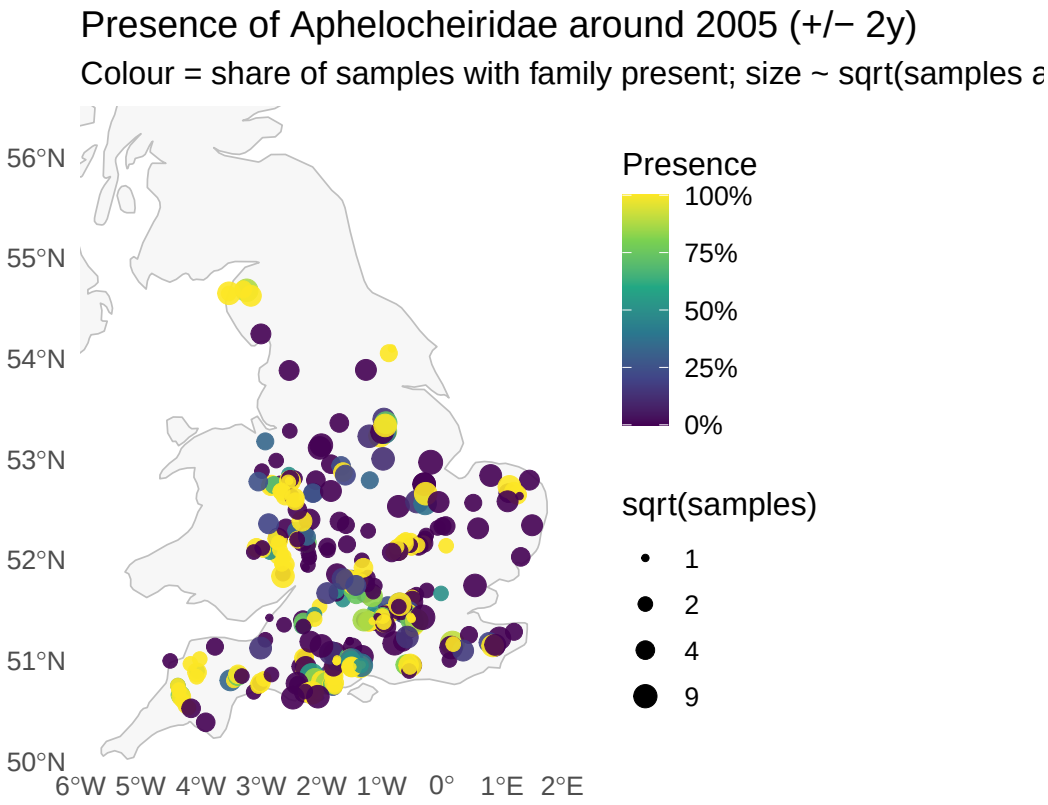

```
print(ab$change)
```



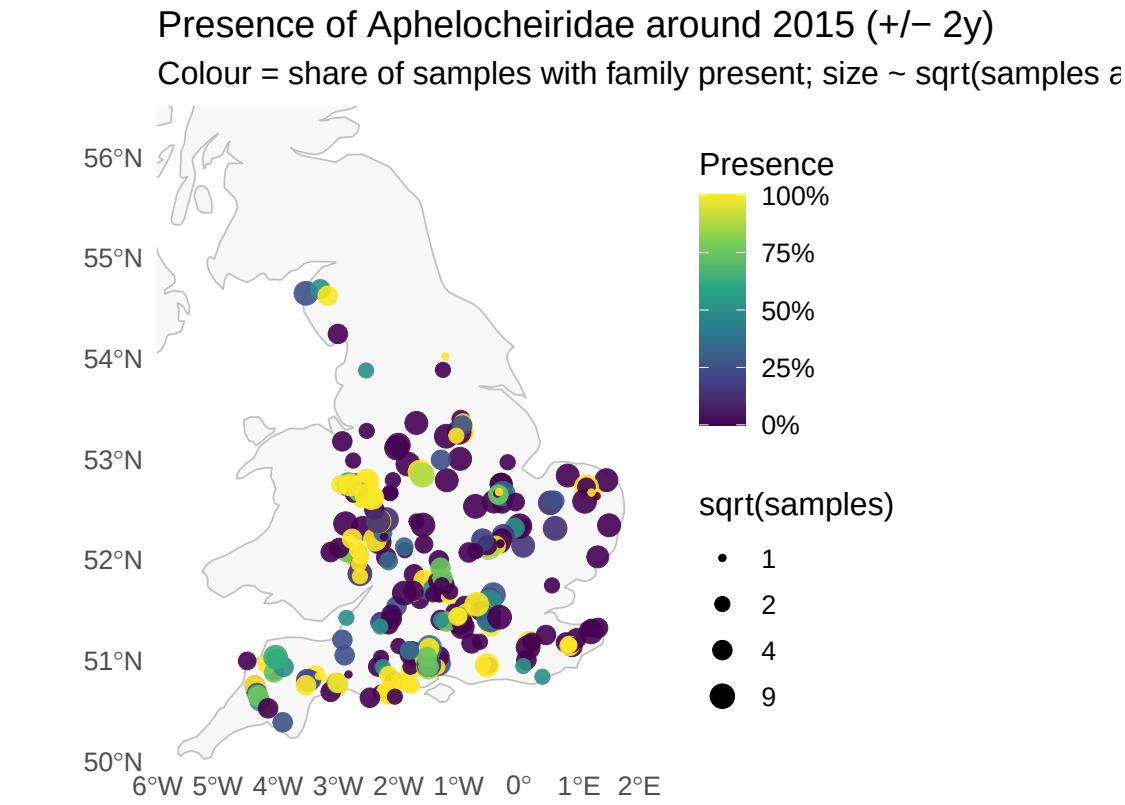

```
yrs <- c(1990, 2005, 2015, 2025)
plots_years <- lapply(yrs, function(yr) {
  dfy <- presence_window(fam_example, yr, window = 2)
  plot_presence_year(dfy, fam_example, yr, window = 2)
})
for (g in plots_years) print(g)
```



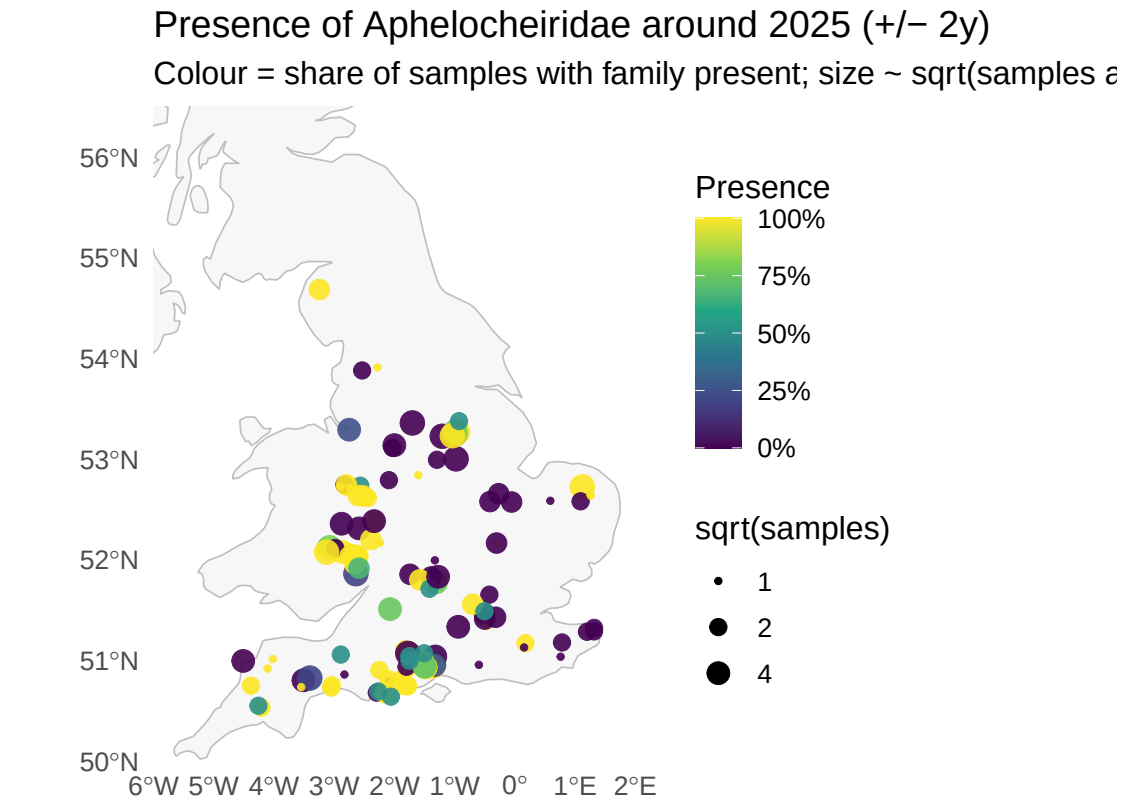
EA open data (England sites); basemap: Natural Earth



EA open data (England sites); basemap: Natural Earth

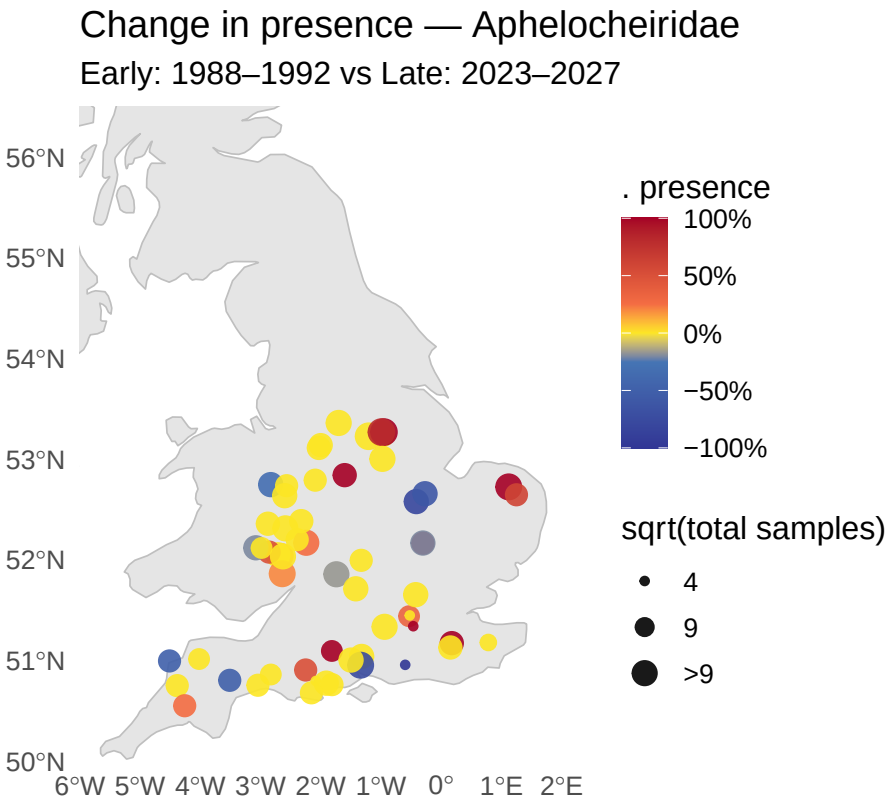


EA open data (England sites); basemap: Natural Earth



EA open data (England sites); basemap: Natural Earth

```
print(presence_change_map(fam_example))
```



ge in presence rate; size ~ sqrt(total samples in both windows).

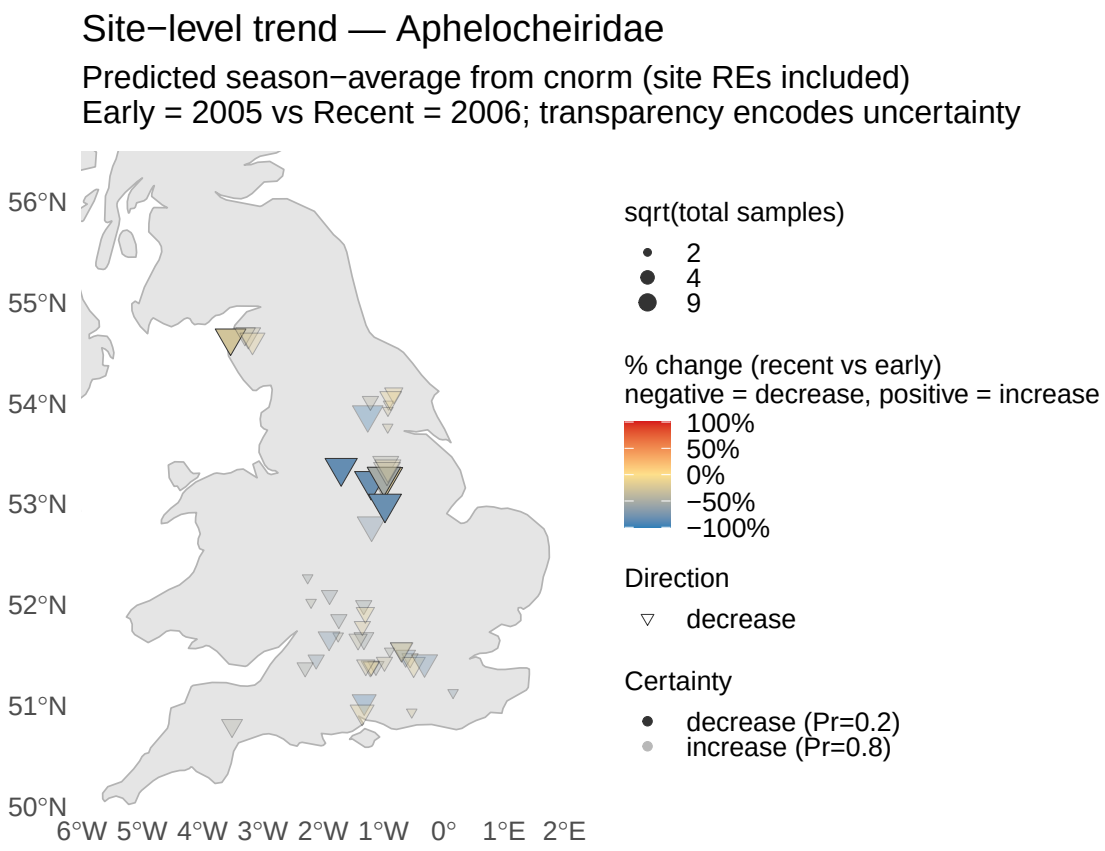
```
# --- helper for posterior draws of fitted mean on p-scale (includes site RE) ---
posterior_mu <- function(fit, newdata, nsim = 600, seed = 123) {
  X <- predict(fit, newdata = newdata, type = "lpmatrix")
  b <- coef(fit)
  V <- vcov(fit, unconditional = FALSE)
  set.seed(seed)
  B <- MASS::mvrnorm(n = nsim, mu = b, Sigma = V)
  as.matrix(B %*% t(X))                          # nsim x n_new
}

# --- compute site-level change + significance from your cnorm result object ---
site_trend_with_sig <- function(res, early_year_max = 2005, nsim = 600) {
  m   <- res$model
  df  <- res$data
  fam <- res$family
  p   <- res$p

  stopifnot(all(c("SITE_ID","season_f","decimal_date","SAMPLE_DATE") %in% names(df)))
  if (!"SITE_ID.F" %in% names(df)) df$SITE_ID.F <- factor(df$SITE_ID)

  # average season within site/period at the *prediction* stage
  site_period <- df %>%
    dplyr::mutate(period = dplyr::if_else(lubridate::year(SAMPLE_DATE) <= early_year_max,
                                          "early","recent")) %>%
    dplyr::group_by(SITE_ID, period) %>%
    dplyr::summarise(
```

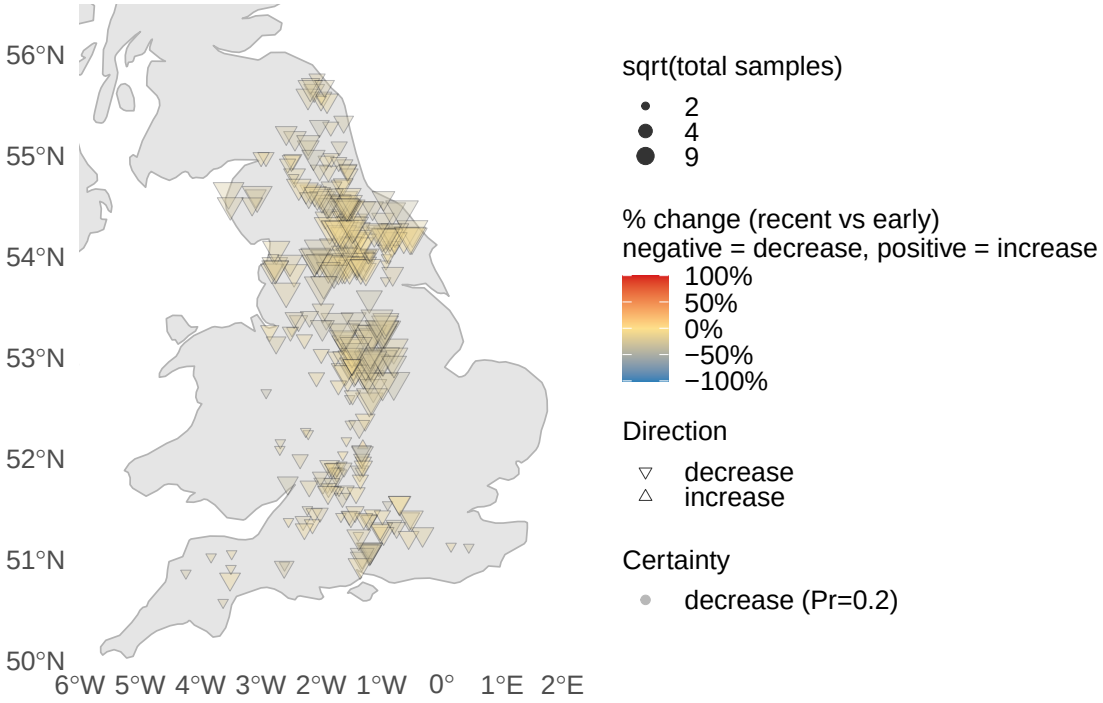
```
    dec = mean(decimal_date),
    n   = dplyr::n(),
    .groups = "drop"
  ) %>%
  tidyr::pivot_wider(names_from = period, values_from = c(dec, n)) %>%
  dplyr::filter(!is.na(dec_early), !is.na(dec_recent))

base_grid <- tidyr::expand_grid(
  SITE_ID   = site_period$SITE_ID,
  SITE_ID.F = factor(site_period$SITE_ID, levels = levels(df$SITE_ID.F)),
  season_f  = factor(levels(df$season_f), levels = levels(df$season_f))
)

new_early  <- dplyr::mutate(dplyr::left_join(base_grid, site_period, by = "SITE_ID"),
                            decimal_date = dec_early)
new_recent <- dplyr::mutate(dplyr::left_join(base_grid, site_period, by = "SITE_ID"),
                            decimal_date = dec_recent)

# posterior draws (includes site RE); average seasons by site inside the draws
D_e <- posterior_mu(m, new_early,  nsim = nsim, seed = 2025)
D_r <- posterior_mu(m, new_recent, nsim = nsim, seed = 2026)

# reshape to (nsim x sites x seasons) then average across seasons
n_sites   <- dplyr::n_distinct(base_grid$SITE_ID)
n_seasons <- length(levels(df$season_f))
idx <- matrix(1:ncol(D_e), nrow = n_seasons)  # columns grouped by site*season
E_mean <- sapply(split(seq_len(ncol(D_e)), rep(1:n_sites, each = n_seasons)),
                 function(cols) rowMeans(D_e[, cols, drop = FALSE]))
R_mean <- sapply(split(seq_len(ncol(D_r)), rep(1:n_sites, each = n_seasons)),
                 function(cols) rowMeans(D_r[, cols, drop = FALSE]))

# back-transform *after* differencing on p-scale
diff_p   <- R_mean - E_mean                          # nsim x sites (p-scale)
prob_inc <- colMeans(diff_p > 0)                     # Pr(recent > early)
mu_e     <- colMeans(E_mean)
mu_r     <- colMeans(R_mean)

# back to count-scale indices for % change
back <- function(x) pmax(0, x)^(1/p)
early_count  <- back(mu_e)
recent_count <- back(mu_r)
delta_pct    <- (recent_count / pmax(early_count, 1e-9)) - 1

# combine with metadata + coords
out <- dplyr::tibble(
  SITE_ID    = site_period$SITE_ID,
  early      = mu_e, recent = mu_r,
  delta_pct  = delta_pct,
  prob_inc   = prob_inc,
  n_total    = site_period$n_early + site_period$n_recent
) %>%
  dplyr::left_join(
    sites_clean %>%
```

```
        dplyr::distinct(SITE_ID, FULL_EASTING, FULL_NORTHING) %>%
        dplyr::filter(!is.na(FULL_EASTING), !is.na(FULL_NORTHING)) %>%
        sf::st_as_sf(coords = c("FULL_EASTING","FULL_NORTHING"), crs = 27700) %>%
        sf::st_transform(4326) %>%
        dplyr::mutate(lon = sf::st_coordinates(geometry)[,1],
                      lat = sf::st_coordinates(geometry)[,2]) %>%
        sf::st_drop_geometry(),
      by = "SITE_ID"
    ) %>%
    dplyr::mutate(
      size_sqrt = sqrt(pmax(n_total, 1)),
      sig = dplyr::case_when(
        prob_inc >= 0.8 ~ "increase",
        prob_inc <= 0.2 ~ "decrease",
        TRUE            ~ "uncertain"
      ),
      family = fam
    ) %>%
    dplyr::filter(!is.na(lon), !is.na(lat))
  out
}

# --- plot one family with significance greying + clearer legend text ---
plot_trend_map_one <- function(fam_name, early_year_max = 2005,
                               bbox = c(xmin = -6, xmax = 2.2, ymin = 50, ymax = 56.5)) {

  # Basemap
  uk_sf <- rnaturalearth::ne_countries(scale = "medium", returnclass = "sf") %>%
    dplyr::filter(admin %in% c("United Kingdom","Ireland")) %>%
    sf::st_transform(4326)

  # Your per-site change table (uses your existing helper)
  pts <- site_trend_with_sig(final_results[[fam_name]], early_year_max = early_year_max)

  # Add explicit direction for a shape legend
  pts <- pts %>%
    dplyr::mutate(
      dir = dplyr::case_when(
        delta_pct >  0 ~ "increase",
        delta_pct <  0 ~ "decrease",
        TRUE           ~ "no change"
      ),
      dir = factor(dir, levels = c("decrease","no change","increase"))
    )

  ggplot() +
    geom_sf(data = uk_sf, fill = "grey90", colour = "grey70", linewidth = 0.3) +
    # Use filled shapes so direction triangles are clear; grey outline for contrast
    geom_point(
      data = pts,
      aes(lon, lat, fill = delta_pct, size = size_sqrt, alpha = sig, shape = dir),
      colour = "grey20", stroke = 0.25, show.legend = TRUE
    ) +
```

```
    coord_sf(xlim = c(bbox["xmin"], bbox["xmax"]),
             ylim = c(bbox["ymin"], bbox["ymax"]), expand = FALSE) +
    # Diverging fill: blue = decrease, red = increase (no white midpoint)
    scale_fill_gradient2(
      low  = "#2C7FB8", mid = "#FEE08B", high = "#D7191C",
      midpoint = 0, limits = c(-1, 1),
      labels = scales::percent_format(accuracy = 1),
      name   = "% change (recent vs early)\nnegative = decrease, positive = increase"
    ) +
    # Direction legend (shapes):  decrease,  ~no change,  increase
    scale_shape_manual(
      values = c("decrease" = 25, "no change" = 21, "increase" = 24),
      name   = "Direction"
    ) +
    # Alpha still shows certainty bins from your helper ("increase/decrease/uncertain")
    scale_alpha_manual(
      values = c(decrease = 1, increase = 1, uncertain = 0.35),
      name   = "Certainty",
      labels = c("decrease (Pr 0.2)", "increase (Pr 0.8)", "uncertain"),
      guide  = guide_legend(override.aes = list(fill = "grey60", colour = "grey20", shape = 21))
    ) +
    scale_size_continuous(
      name   = "sqrt(total samples)",
      range  = c(1.2, 4.2),
      breaks = sqrt(c(2,4,9)),
      labels = c("2","4","9"),
      guide  = guide_legend(override.aes = list(alpha = 1, fill = "grey60", shape = 21))
    ) +
    labs(
      title    = paste0("Site-level trend - ", fam_name),
      subtitle = "Predicted season-average from cnorm (site REs included)\nEarly  2005 vs Recent  2006
      x = NULL, y = NULL
    ) +
    theme_minimal(base_size = 12) +
    theme(
      panel.grid       = element_blank(),
      legend.box       = "vertical",
      legend.position  = "right",
      legend.title     = element_text(size = 10),
      legend.key.height= unit(0.55, "lines"),
      legend.spacing.y = unit(2, "pt"),
      plot.subtitle    = element_text(margin = margin(b = 8))
    ) +
    # Order guides: sample support → % change → direction → certainty
    guides(
      size   = guide_legend(order = 1),
      fill   = guide_colourbar(order = 2),
      shape  = guide_legend(order = 3),
      alpha  = guide_legend(order = 4)
    )
}
```

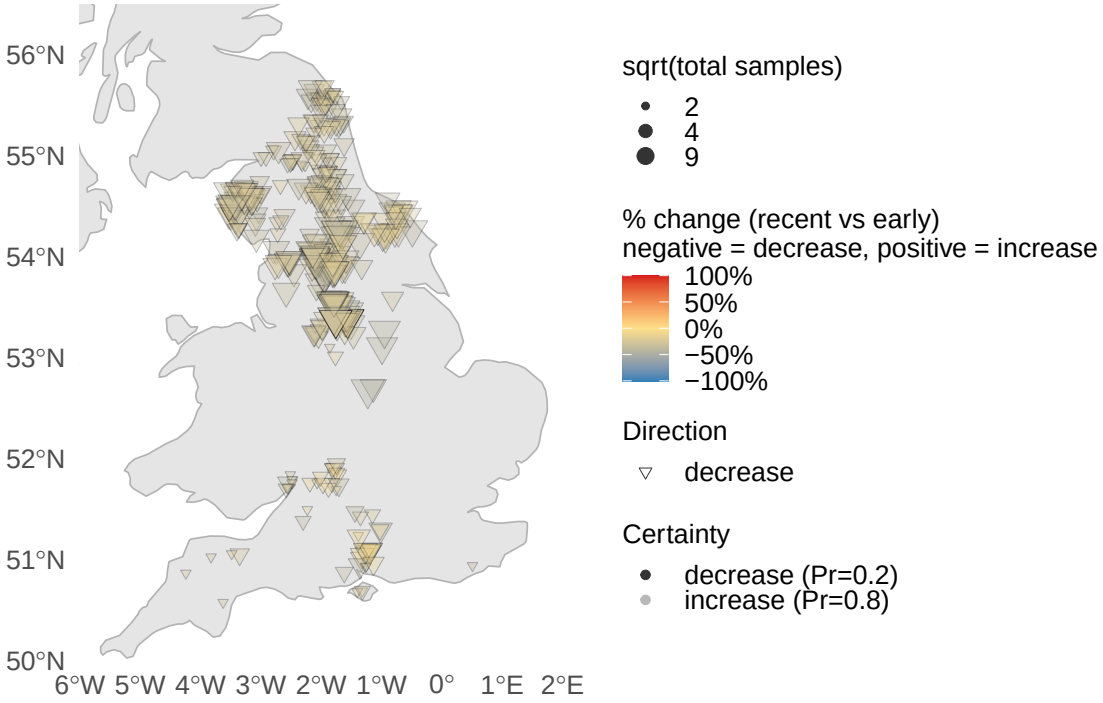
```r
# ---- call it per family (prints four separate plots) ----
FAMS <- c("Aphelocheiridae","Brachycentridae","Odontoceridae","Cordulegastridae")
plots <- lapply(FAMS, plot_trend_map_one)
for (p in plots) print(p)
```

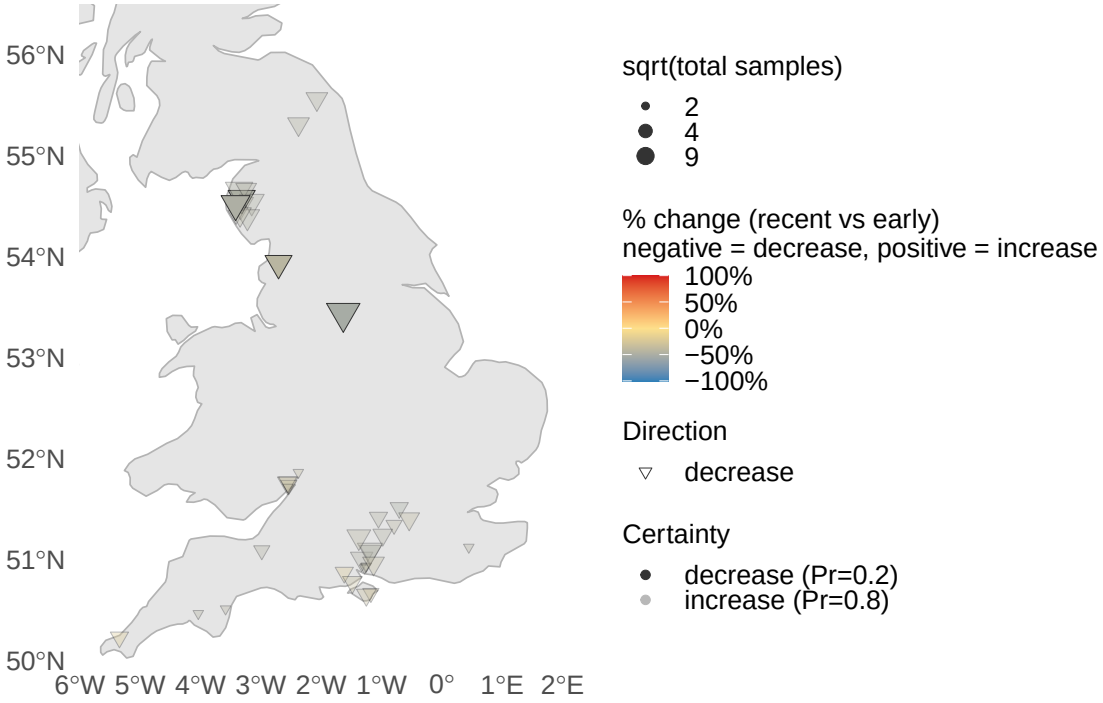



Site−level trend — Brachycentridae
Predicted season−average from cnorm (site REs included)
Early = 2005 vs Recent = 2006; transparency encodes uncertainty
56°N
55°N
54°N
53°N
52°N
51°N
50°N
6°W 5°W 4°W 3°W 2°W 1°W 0° 1°E 2°E
sqrt(total samples)
2
4
9
% change (recent vs early)
negative = decrease, positive = increase
100%
50%
0%
−50%
−100%
Direction
decrease
increase
Certainty
decrease (Pr=0.2)

Site−level trend — Odontoceridae

Predicted season−average from cnorm (site REs included)
Early = 2005 vs Recent = 2006; transparency encodes uncertainty




Site−level trend — Cordulegastridae
Predicted season−average from cnorm (site REs included)
Early = 2005 vs Recent = 2006; transparency encodes uncertainty
56°N
55°N
54°N
53°N
52°N
51°N
50°N
6°W 5°W 4°W 3°W 2°W 1°W 0° 1°E 2°E
sqrt(total samples)
2
4
9
% change (recent vs early)
negative = decrease, positive = increase
100%
50%
0%
−50%
−100%
Direction
decrease
Certainty
decrease (Pr=0.2)
increase (Pr=0.8)