Using DFAclust to analyse bird indices

Load R package

```
library(DFAclust)
#> Le chargement a nécessité le package : TMB
#> Le chargement a nécessité le package : RcppEigen
```

Empirical data: preparing data

The following empirical example corresponds to the empirical analysis in the paper related to this R package. Preparation of the data from the raw data are not in the scope of the package, this is why required packages have to be loaded first. The user can skip this part as the ready-to-use data are already available in the package (i.e. start the vignette at *Empirical data: analysing data*).

Load additionnal R packages

```
require(stringr)
#> Le chargement a nécessité le package : stringr
require(rnaturalearth)
#> Le chargement a nécessité le package : rnaturalearth
require(sp)
#> Le chargement a nécessité le package : sp
require(ggplot2)
#> Le chargement a nécessité le package : ggplot2
require(reshape2)
#> Le chargement a nécessité le package : reshape2
require(plyr)
#> Le chargement a nécessité le package : plyr
require(dplyr)
#> Le chargement a nécessité le package : dplyr
#>
#> Attachement du package : 'dplyr'
#> Les objets suivants sont masqués depuis 'package:plyr':
#>
#>     arrange, count, desc, failwith, id, mutate, rename, summarise,
#>     summarize
#> Les objets suivants sont masqués depuis 'package:stats':
#>
#>     filter, lag
#> Les objets suivants sont masqués depuis 'package:base':
#>
#>     intersect, setdiff, setequal, union
require(emmeans)
#> Le chargement a nécessité le package : emmeans
require(arm)
#> Le chargement a nécessité le package : arm
#> Le chargement a nécessité le package : MASS
#>
#> Attachement du package : 'MASS'
#> L'objet suivant est masqué depuis 'package:dplyr':
#>
#>     select
#> Le chargement a nécessité le package : Matrix
#> Le chargement a nécessité le package : lme4
#>
#> arm (Version 1.13-1, built: 2022-8-25)
#> Working directory is /home/stanislas/Documents/Gitlab_dossier/DFAclust/vignettes
require(see)
#> Le chargement a nécessité le package : see
```

Load and prepare data

Bird data

Download and extract data

Breeding bird data for Sweden can be download from https://www.gbif.org/occurrence/download?dataset_key=91fa1a0d-a208-40aa-8a6e-f2c0beb9b253 (an free account is necessary).Type of download: Darwin Core Archive. This will provide you a text file: “occurrence.txt”.

```
bird_se_raw <- read.csv("raw_data/occurrence.txt", header = T, sep="\t")
```

Subselecting and preparing data

We select columns and row of interest (i.e. birds).

```
bird_se_clean <- bird_se_raw[bird_se_raw$class=="Aves",c("class","order","family",
        "genus","species",

        "specificEpithet","infraspecificEpithet","taxonRank",

        "organismQuantity","decimalLatitude","decimalLongitude",

        "day","month","year","taxonKey","speciesKey","countryCode",

        "level1Gid","level2Gid","iucnRedListCategory")]
```

We add a code by species from species name and check for duplicate in species' code. The link between species name and species name code is stored in a specific dataset `species_data`.

```
species_data <- data.frame(name_long = unique(bird_se_clean$species[bird_se_clean$taxonRank !=
        "GENUS"]))

species_data$code_sp <- paste0(toupper(substr(species_data$name_long, 1, 3)),
                               toupper(substr(sub(".* ", "", species_data$name_long), 1, 3)))

species_data$code_sp[species_data$name_long=="Corvus corax"] <- "CORCOX"

species_data$code_sp[species_data$name_long=="Phylloscopus trochilus"] <- "PHYTRU"

species_data$code_sp[species_data$name_long=="Saxicola rubicola"] <- "SAXRUI"

species_data$code_sp[species_data$name_long=="Sterna paradisaea"] <- "STEPAD"

species_data$genus <- sub(" .*", "", species_data$name_long)

species_data$species <- sub(".* ", "", species_data$name_long)

bird_se_clean <- merge(bird_se_clean, species_data[,c("code_sp", "name_long")],
                       by.x = c("species"), by.y = c("name_long"), all.x = T)

species_data$class <- bird_se_clean$class[match(species_data$code_sp, bird_se_clean$code_sp)]

species_data$order <- bird_se_clean$order[match(species_data$code_sp, bird_se_clean$code_sp)]

species_data$family <- bird_se_clean$family[match(species_data$code_sp, bird_se_clean$code_sp)]

species_data$iucnRedListCategory <-
        bird_se_clean$iucnRedListCategory[match(species_data$code_sp, bird_se_clean$code_sp)]
```

Geographical coordinates and routes

We prepare a specific dataset for coordinates of monitored routes by linking route numbers to their coordinates.

```
route_data <- paste0(bird_se_clean$decimalLatitude, sep="_", bird_se_clean$decimalLongitude)

route_data <- data.frame(code_route = paste0("R",str_pad(1:length(unique(route_data)), 3, pad =
        "0")),
                         coordinate_chr = unique(route_data))

route_data$lat <- as.numeric(sub("_.*", "", route_data$coordinate_chr))

route_data$lon <- as.numeric(sub(".*_", "", route_data$coordinate_chr))
```

We can then display route location on the map.

```
worldmap <- ne_countries(scale = 'medium', type = 'countries',returnclass = 'sf')

sweden_map_wgs84 <- worldmap[worldmap$sovereign=="Sweden",]

sweden_map_moll <- sf::st_transform(sweden_map_wgs84, "+proj=moll")
```

```
sweden_map_swe <- sf::st_transform(sweden_map_wgs84, "+init=epsg:3006")

route_data_coord <- route_data

coordinates(route_data_coord) <- ~lon+lat

proj4string(route_data_coord) <- CRS("+proj=longlat +datum=WGS84")

route_data_coord <- spTransform(route_data_coord, CRSobj = "+proj=moll")

route_data_moll <- as.data.frame(coordinates(route_data_coord))

route_data_moll$code_route <- route_data_coord$code_route

route_data_coord <- spTransform(route_data_coord, CRSobj = "+init=epsg:3006")

route_data_swe <- as.data.frame(coordinates(route_data_coord))

route_data_swe$code_route <- route_data_coord$code_route

ggplot() + geom_sf(data=sweden_map_swe) +
  geom_tile(data = route_data_swe, aes(x=lon, y=lat), width=25000, height=25000, alpha=0.5) +
  theme_void() + coord_sf(datum=NA)
```

We aggregate all the projections into on dataset.

```
route_data <- merge(route_data, route_data_moll, by="code_route", all=T)

route_data <- merge(route_data, route_data_swe, by="code_route", all=T)

names(route_data)[3:8] <- c("lat_wgs", "lon_wgs", "lon_moll", "lat_moll", "lon_swe", "lat_swe")

route_data$coordinate_chr <- NULL
```

We finally associate routes and coordinates with the main dataset on bird occurrence.

```
bird_se_clean <- merge(bird_se_clean, route_data[,c("code_route", "lat_wgs", "lon_wgs")],
                       by.x = c("decimalLatitude", "decimalLongitude"), by.y = c("lat_wgs",
        "lon_wgs"), all.x = T)

route_data$level1Gid <- bird_se_clean$level1Gid[match(route_data$code_route,
        bird_se_clean$code_route)]

route_data$level2Gid <- bird_se_clean$level2Gid[match(route_data$code_route,
        bird_se_clean$code_route)]
```

Finalise bird dataset

We need to incorporate 0s in the dataset when, a given year, a species was not present while the route was monitored.

```
bird_se_clean_tot <- dcast(bird_se_clean, countryCode+code_route+year~code_sp,
                           fun.aggregate = sum, value.var="organismQuantity")

bird_se <- melt(bird_se_clean_tot, id.vars = c("countryCode", "code_route", "year"))

names(bird_se)[4:5] <- c("code_sp", "abund")

bird_se$code_sp <- as.character(bird_se$code_sp)

bird_se <- bird_se[bird_se$code_sp!="NA",]
```

We add information on geographical coordinate and taxa in the final dataset `bird_se`.

```
bird_se <- merge(bird_se, route_data, by="code_route", all.x = T)

bird_se$order <- species_data$order[match(bird_se$code_sp, species_data$code_sp)]

bird_se$family <- species_data$family[match(bird_se$code_sp, species_data$code_sp)]

bird_se$genus <- species_data$genus[match(bird_se$code_sp, species_data$code_sp)]

bird_se$species <- species_data$species[match(bird_se$code_sp, species_data$code_sp)]
```

```r
bird_se$name_long <- species_data$name_long[match(bird_se$code_sp, species_data$code_sp)]

bird_se$iucnRedListCategory <- species_data$iucnRedListCategory[match(bird_se$code_sp,
        species_data$code_sp)]
```

Estimate species time-series

Now that the data are ready, we can estimate species time-series and standard errors of these time-series.

Select period of time

We first remove data from the first two years of the survey (1996 and 1997) because there was a low number of routes monitored in 1996 and 1997.

```r
bird_se_1998 <- droplevels(bird_se[bird_se$year>1997,])
```

Compute species time series

As we said above about the additional packages that needed to be loaded, it is not in the scope of this R package to obtain time-series from raw data and the user will normally have usable time-series. So we need to specify here an additional function adapted from the French Breeding Bird Survey analysis https://www.vigienature.fr/sites/vigienature/files/atoms/files/analysestoceps_0.zip) to estimate time-series from bird abundance data.

```r
get_ts <- function(data_bird_input){

  # d: data for species i

  d <- droplevels(data_bird_input)

  sp <- levels(as.factor(d$code_sp))

  # number of route followed by year

  nb_route <- tapply(rep(1,nrow(d)),d$year,sum)

  # number of route with species i by year

  nb_route_presence <- tapply(ifelse(d$abund>0,1,0),d$year,sum)

  year <- as.numeric(as.character(levels(as.factor(d$year))))

  firstY <- min(year)

  lastY <- max(year)

  timestep <- length(year)-1

  # table for analysis result

  threshold_occurrence <- 3

  tab_ana <- data.frame(year=rep(year,2),val=c(nb_route,nb_route_presence),LL = NA,UL=NA,
                        catPoint=NA,pval=NA,
                        curve=rep(c("route","presence"),each=length(year)))

  tab_ana$catPoint <- ifelse(tab_ana$val == 0,"0", ifelse(tab_ana$val < threshold_occurrence,
                                                       "inf_threshold",NA))


  # abundance by year

  abund <- tapply(d$abund,d$year,sum)

  threshold_abundance <- 5

  tab_fig <- data.frame(year=year,val=abund,LL = NA,UL=NA,catPoint=NA,pval=NA)

  tab_fig$catPoint <- ifelse(tab_fig$val == 0,"0",ifelse(tab_fig$val < threshold_abundance,
                                                      "inf_threshold",NA))

  # remove criteria

  remove_sp <- FALSE
```

```r
# if first year empty

if(tab_fig$val[1]==0){remove_sp <- TRUE}

# if four consecutive years empty

ab_vec <- paste(tab_fig$val,collapse="")

if(str_detect(ab_vec, "0000")){remove_sp <- TRUE}

# if less than three consecutive years

ab_vec2 <- paste(sign(tab_fig$val),collapse="")

if(!str_detect(ab_vec2, "111")){remove_sp <- TRUE}

if(anyNA(tab_fig$catPoint) & anyNA(tab_ana$catPoint[tab_ana$curve=="presence"]) &
      remove_sp==F){

  # GLM abundance variation
  glm1 <- glm(abund~as.factor(code_route)+as.factor(year),data=d,family=quasipoisson)

  sglm1 <- summary(glm1)

  # mean-centered values

  con.mat <- diag(length(year)) - 1/length(year)

  colnames(con.mat) <- year # firstY:lastY

  rg <- ref_grid(glm1, nuisance = 'code_route')

  sglm2 <- summary(contrast(rg, as.data.frame(con.mat)))

  # as link function is log, estimates need to be back transformed from sglm1 (first year set
      to 1 and se to 0)

  coef_yr <- tail(matrix(sglm1$coefficients[,1]), timestep)

  coef_yr <- rbind(1, exp(coef_yr))

  error_yr <- tail(matrix(sglm1$coefficients[,2]), timestep)

  error_yr <- rbind(0, error_yr)*coef_yr # approximated se values

  log_error_yr <- tail(matrix(sglm1$coefficients[,2]), timestep)

  log_error_yr <- rbind(0, log_error_yr)

  pval <- c(1,tail(matrix(coefficients(sglm1)[,4]),timestep))

  # from sglm2 (mean value to 0)

  coef_yr_m0 <- exp(sglm2$estimate)

  error_yr_m0 <- sglm2$SE*coef_yr_m0 # approximated se values

  log_error_yr_m0 <- sglm2$SE

  pval_m0 <- sglm2$p.value

  # CIs

  glm1.sim <- sim(glm1)

  ci_inf_sim <- c(1, exp(tail(apply(coef(glm1.sim),2, quantile, .025), timestep)))

  ci_sup_sim <- c(1, exp(tail(apply(coef(glm1.sim),2, quantile, .975), timestep)))

  thresold_signif <- 0.05

  tab_res <- data.frame(year,val=coef_yr,val_m0=coef_yr_m0,
                      LL=ci_inf_sim,UL=ci_sup_sim,
                      catPoint=ifelse(pval<thresold_signif,"significatif",NA),pval)

  # cleaning out of range CIs

  tab_res$UL <- ifelse(nb_route_presence==0,NA,tab_res$UL)
```

```
    tab_res$UL <-  ifelse(tab_res$UL == Inf, NA,tab_res$UL)

    tab_res$UL <-  ifelse(tab_res$UL > 1.000000e+20, NA,tab_res$UL)

    tab_res$UL[1] <- 1

    tab_res$val <-  ifelse(tab_res$val > 1.000000e+20,1.000000e+20,tab_res$val)

    tab_res$val_m0 <-  ifelse(tab_res$val_m0 > 1.000000e+20,1.000000e+20,tab_res$val_m0)

    # overdispersion index
    dispAn <- sglm1$deviance/sglm1$null.deviance

    # class uncertainity

    if(dispAn > 2 | (median(nb_route_presence)<threshold_occurrence & median(abund)
        <threshold_abundance)) catIncert <- "Uncertain" else catIncert <-"Good"

    vecLib <-  NULL

    if(dispAn > 2 | median(nb_route_presence)<threshold_occurrence) {

      if(median(nb_route_presence)<threshold_occurrence) {

            vecLib <- c(vecLib,"too rare species")

          }

      if(dispAn > 2) {

         vecLib <- c(vecLib,"deviance")

       }
    }

    reason_uncert <-  paste(vecLib,collapse=" and ")

    # Store results

    tab_tot <- data.frame(code_sp=sp, year = tab_res$year, nb_year=timestep,
                          firstY = firstY, lastY = lastY,
                          relative_abundance = tab_res$val,
                          CI_inf = tab_res$LL, CI_sup = tab_res$UL,
                          Standard_error = error_yr,
                          Log_SE = log_error_yr,
                          p_value = tab_res$pval,
                          relative_abundance_m0 = tab_res$val_m0,
                          Standard_error_m0 = error_yr_m0,
                          Log_SE_m0 = log_error_yr_m0,
                          p_value_m0 = pval_m0, signif = !is.na(tab_res$catPoint),
                          nb_route,nb_route_presence,abundance=abund,
                          mediane_occurrence=median(nb_route_presence),
        mediane_ab=median(abund) ,
                          valid = catIncert, uncertanity_reason = reason_uncert)

  }
  else{
    tab_tot <- data.frame(code_sp=sp, year = year, nb_year=timestep,
                          firstY=firstY, lastY=lastY,
                          relative_abundance=NA,
                          CI_inf = NA, CI_sup = NA,
                          Standard_error = NA,
                          p_value = NA,
                          relative_abundance_m0 = NA,
                          Standard_error_m0 = NA,
                          Log_SE_m0 = NA,
                          p_value_m0 = NA,signif = NA,
                          nb_route,nb_route_presence,abundance=abund,
                          mediane_occurrence=median(nb_route_presence),
        mediane_ab=median(abund) ,
                          valid = NA, uncertanity_reason = NA)
  }

  return(tab_tot)
 }
```

We can now compute time-series between 1998 and 2020 for each species in the Swedish Breeding Bird Survey.

```
ts_bird_se_allcountry <- ddply(bird_se_1998, .(code_sp), .fun=get_ts, .progress="text")
```

We can then check, by plotting the estimated time-series, if they are similar to the one produced by the official institution in charge in Sweden Svensk fageltaxering (https://www.fageltaxering.lu.se/resultat/trender).

```
for(i in 1:length(levels(as.factor(bird_se$code_sp)))){

  sp <- levels(as.factor(ts_bird_se_allcountry$code_sp))[i]

  gp <- ggplot(ts_bird_se_allcountry[ts_bird_se_allcountry$code_sp==sp,],
         aes(year, relative_abundance)) + geom_line() + geom_text(x=2010, y=1, label=sp) +
    geom_line(aes(y=CI_inf), linetype="dashed") +
    geom_line(aes(y=CI_sup), linetype="dashed") +
    ylab("Relative abundance") + xlab("Years") +
    theme_modern()

  print(gp)

}
```

Those data (species time-series, their log standrard errors and species names) are already available in the package. Let's load them.

```
data(ts_bird_se_allcountry)
data(species_data)
```

Empirical data: analysing data

DFA cluster analysis for Swedish birds

Now that we have time-series and standard errors for each species, we can launch the analysis.

Farmland birds

We will first run the analysis on farmland birds. We therefore select the 15 farmland species.

```
species_sub <- species_farm <-  droplevels(species_data[species_data$code_sp %in% c(
  "FALTIN","VANVAN","ALAARV","HIRRUS","CORFRU",
  "SAXRUB","SYLCOM","ANTPRA","MOTFLA","LANCOL",
  "STUVUL","LINCAN","EMBCIT","EMBHOR","PASMON"),])
```

We then subselect the corresponding time-series and standard errors for farmland birds.

```
Obs <- ts_bird_se_allcountry[ts_bird_se_allcountry$code_sp %in% species_sub$code_sp,]
```

We finally produce a dataset of species time-series `y_farm_ts` and a dataset of standard errors `obs_se_farm_ts`.

```
y_farm <- dcast(Obs[,c("code_sp","relative_abundance_m0","year")],
           code_sp~year, fun.aggregate = sum, value.var = "relative_abundance_m0")

obs_se_farm <- dcast(Obs[,c("code_sp","Log_SE_m0","year")],
             code_sp~year, fun.aggregate = sum, value.var = "Log_SE_m0")

y_farm_ts <- as.matrix(y_farm[,2:ncol(y_farm)]) # species time series

y_uncert_ts <- as.matrix(obs_se_farm[,2:ncol(obs_se_farm)]) # standard error on time series

rownames(y_farm_ts) <- rownames(y_uncert_ts) <- y_farm$code_sp # add species names as row names
```

We can now run the `prepare_data` function. This function allows the user to log-transform the standard errors if they are not and check if there are missing values in standard error input. It also handles zeros in time-series by replacing zeros by a percentage of the reference year value (1 % by default). `perc_replace` correspond to the proportion of the average index value used to replace zeros in species time-series with `0.01` as default value (1 %). It can also transform the observation error time-series to get their log values if they are initially not in log values (in that case set `se_log` to `TRUE`).

```
data_ready_dfa <- prepare_data(data_ts = y_farm_ts,data_ts_se = y_uncert_ts, se_log = TRUE,
        perc_replace = 0.01)
```

Then the `fit_dfa` function to launch the DFA analysis. Note that, here, we know the number of latent trends `nfac`=3 (by running this function with `nfac`=0), so we specify it which shortens the computation. In general the user will not know the optimal number for `nfac` and in that case, set `nfac`=0 (which is the default) to look for it between `mintrend` and `maxtrend`. `center_option` allows to handle time-series centered according to the first year (`center_option` = 0) or mean-centred, which is the default (`center_option` = 1). `control` is a `list` of control options for `MakeADFun()` (default is list()).

```
dfa_result <- fit_dfa(data_ts = data_ready_dfa$data_ts,data_ts_se = data_ready_dfa$data_ts_se,
        min_year = data_ready_dfa$min_year, max_year = data_ready_dfa$max_year,
        species_name_ordre = data_ready_dfa$species_name_ordre, species_sub = species_farm,
        nfac = 3, mintrend = 1, maxtrend = 5, AIC = TRUE, center_option = 1, silent = TRUE,
        control = list())
#> NLMINB   BFGS NLMINB   BFGS NLMINB   BFGS
#>      0      0      0      0      0      0
#>    NLMINB      BFGS    NLMINB      BFGS    NLMINB      BFGS
#> -232.2488 -232.2488 -232.2488 -232.2488 -232.2488 -232.2488
#>       NLMINB         BFGS       NLMINB         BFGS       NLMINB         BFGS
#> 4.264477e-04 6.615034e-06 3.417204e-04 6.390130e-05 5.178977e-04 1.470864e-05
#> AIC:  -350.49756409878
```

Once the latent trend are estimated, the clustering analysis can be performed using the function `cluster_dfa`.

```
cluster_result <- cluster_dfa(data_dfa = dfa_result, species_sub = species_farm, nboot = 500)
```

And finally the `plot_dfa_result` function to get the results of the whole analysis.

```
dfa_result_plot_farm <- plot_dfa_result(data_dfa = dfa_result, sdRep = cluster_result$sdRep,
        species_sub = species_farm, group_dfa = cluster_result$group_dfa, min_year =
        data_ready_dfa$min_year, species_name_ordre = data_ready_dfa$species_name_ordre)
#> Using name_long as id variables
```

Woodland birds

Before showing the results for farmland birds, we will now run the analysis on woodland birds. We therefore select the 26 woodland species.

```
species_sub <- species_forest <- droplevels(species_data[species_data$code_sp %in% c(
    "ACCNIS","TETBON","TRIOCH","COLOEN","DRYMAR",
   "DRYMIN","NUCCAR","GARGLA","PERATE","LOPCRI","POEPAL","POEMON",
   "SITEUR","CERFAM","TURVIS","PHOPHO","PHYCOL","PHYSIB","REGREG","FICHYP","FICALB",
   "ANTTRI","COCCOC","SPISPI","PYRPYR","EMBRUS"),])
```

We then subselect the corresponding time-series and standard errors for woodland birds.

```
Obs <- ts_bird_se_allcountry[ts_bird_se_allcountry$code_sp %in% species_sub$code_sp,]
```

We finally produce a dataset of species time-series `y_forest` and a dataset of standard errors `obs_se_forest`.

```
y_forest <- dcast(Obs[,c("code_sp","relative_abundance_m0","year")],
           code_sp~year, fun.aggregate = sum, value.var = "relative_abundance_m0")

obs_se_forest <- dcast(Obs[,c("code_sp","Log_SE_m0","year")],
                code_sp~year, fun.aggregate = sum, value.var = "Log_SE_m0")

y_forest_ts <- as.matrix(y_forest[,2:ncol(y_forest)]) # species time series

y_uncert_ts <- as.matrix(obs_se_forest[,2:ncol(obs_se_forest)]) # standard error on time series

rownames(y_forest_ts) <- rownames(y_uncert_ts) <- y_forest$code_sp # add species names as row
        names
```

We can now run the `prepare_data` function.

```
data_ready_dfa <- prepare_data(data_ts = y_forest_ts,data_ts_se = y_uncert_ts, se_log = TRUE,
        perc_replace = 0.01)
```

Then the `fit_dfa` function to launch the DFA analysis. Note that, as for farmland birds, we know the number of latent trends `nfac`=4 (by running this function with `nfac`=0), so we specify it which shortens the computation. In general the user will not know the optimal number for `nfac` and in that case, set `nfac`=0 (which is the default) to look for it.

```
dfa_result <- fit_dfa(data_ts = data_ready_dfa$data_ts,data_ts_se = data_ready_dfa$data_ts_se,
        min_year = data_ready_dfa$min_year, max_year = data_ready_dfa$max_year,
        species_name_ordre = data_ready_dfa$species_name_ordre, species_sub = species_forest,
        nfac = 4, mintrend = 1, maxtrend = 5, AIC = TRUE, center_option = 1, silent = TRUE,
        control = list())
#> NLMINB   BFGS NLMINB   BFGS NLMINB   BFGS
#>      0      0      0      0      0      0
#>    NLMINB      BFGS    NLMINB      BFGS    NLMINB      BFGS
#> -327.4038 -327.4038 -327.4038 -327.4038 -327.4038 -327.4038
#>       NLMINB         BFGS       NLMINB         BFGS       NLMINB         BFGS
#> 6.234733e-04 9.409873e-05 3.669179e-04 5.673957e-05 1.912420e-04 6.884552e-05
#> AIC:  -406.80762263983
```

Then the `cluster_dfa` function to launch the clustering analysis. To make the analysis faster `nboot` is set to 100. By default it is 500.

```
cluster_result <- cluster_dfa(data_dfa = dfa_result, species_sub = species_forest, nboot = 100)
```

And finally the `plot_dfa_result` function to get the results of the whole analysis.

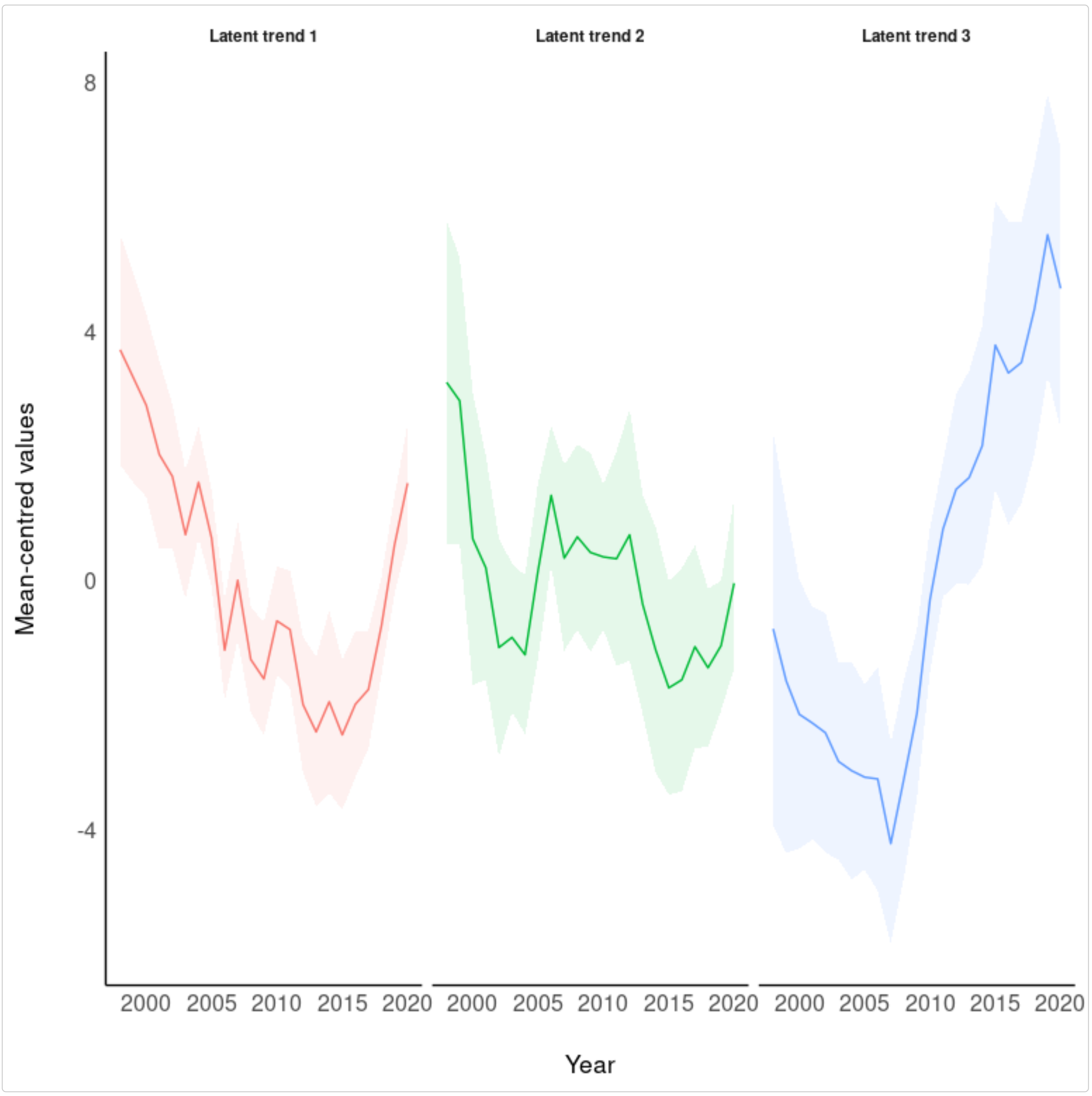
```
dfa_result_plot_forest <- plot_dfa_result(data_dfa = dfa_result, sdRep = cluster_result$sdRep,
        species_sub = species_forest, group_dfa = cluster_result$group_dfa, min_year =
        data_ready_dfa$min_year, species_name_ordre = data_ready_dfa$species_name_ordre)
#> Using name_long as id variables
```

Display results

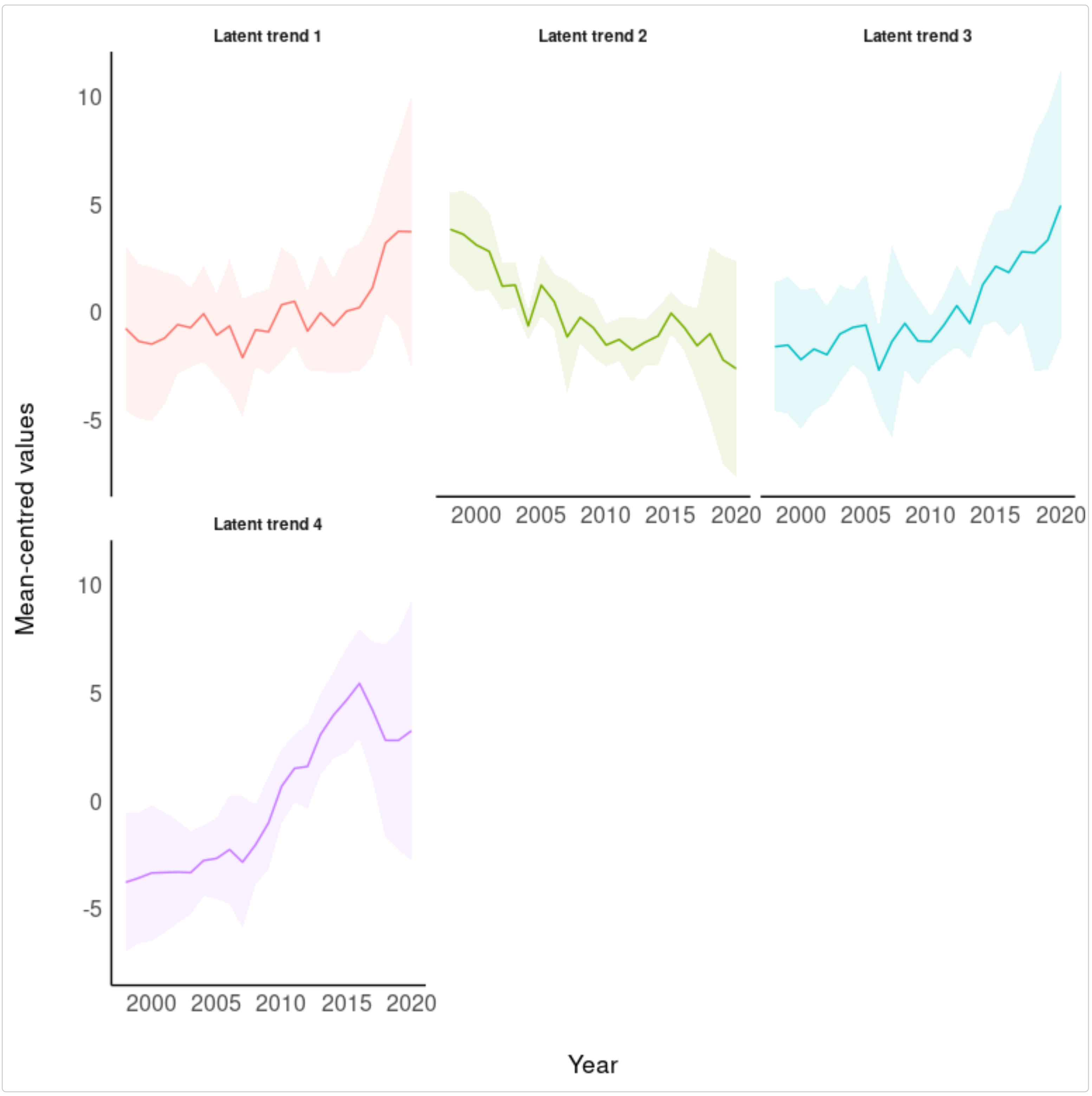
We now can look at the tree tools of the toolbox that are the main results for farmland and woodland birds.

Tool 1: Latent trends

```
dfa_result_plot_farm$plot_tr
```

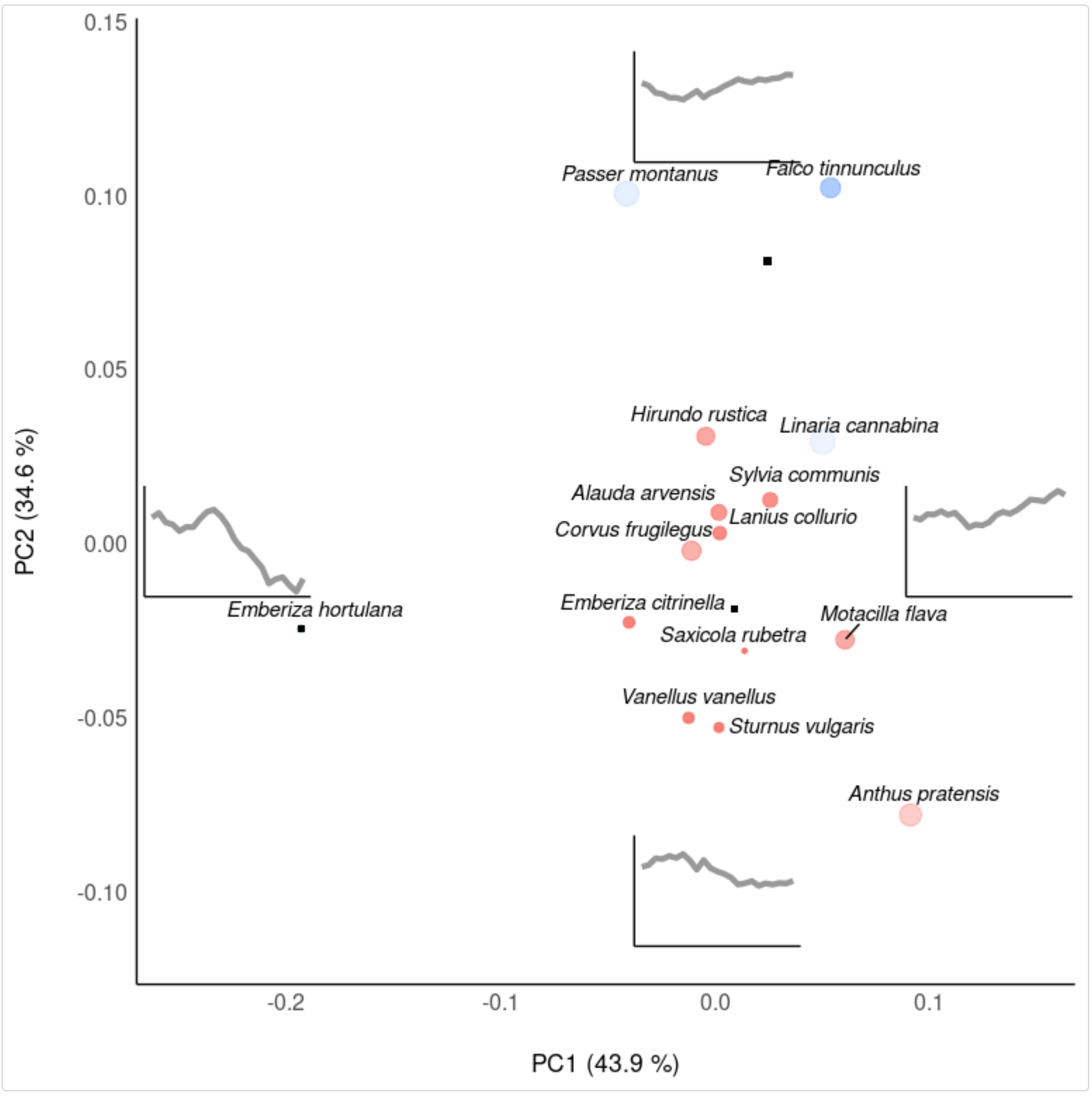


```
dfa_result_plot_forest$plot_tr
```

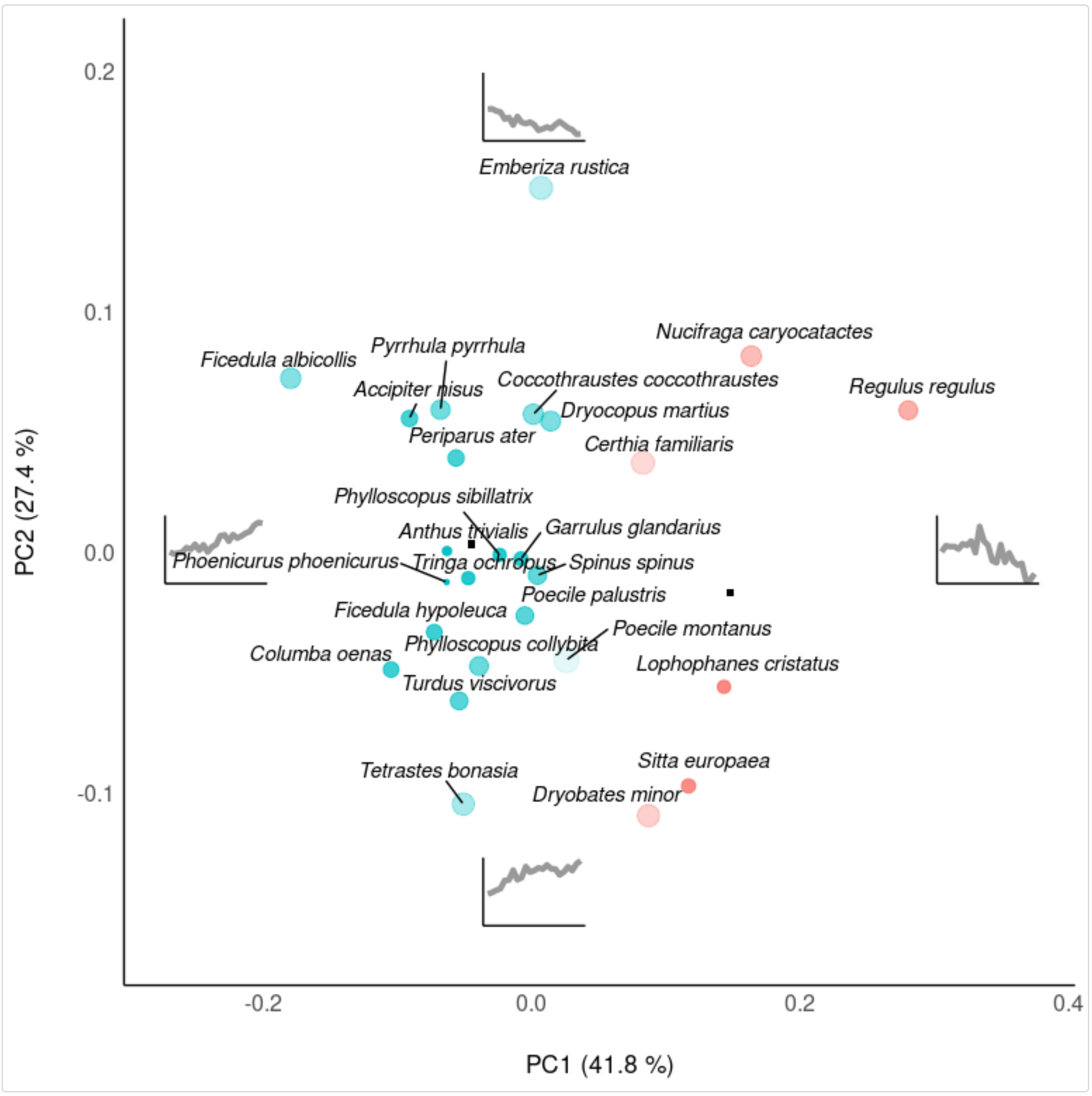


Tools 2 and 3: Ordination and clusters

```
dfa_result_plot_farm$plot_sp_group[[1]]
```



```
dfa_result_plot_forest$plot_sp_group[[1]]
```



0.2
0.1
0.0
-0.1
PC2 (27.4 %)
-0.2
0.0
0.2
0.4
PC1 (41.8 %)
Emberiza rustica
Nucifraga caryocatactes
Pyrrhula pyrrhula
Ficedula albicollis
Accipiter nisus
Coccothraustes coccothraustes
Regulus regulus
Dryocopus martius
Periparus ater
Certhia familiaris
Phylloscopus sibillatrix
Anthus trivialis
Garrulus glandarius
Phoenicurus phoenicurus
Tringa ochropus
Spinus spinus
Poecile palustris
Ficedula hypoleuca
Poecile montanus
Phylloscopus collybita
Columba oenas
Lophophanes cristatus
Turdus viscivorus
Sitta europaea
Tetrastes bonasia
Dryobates minor