manipulate

Johnson T.F.

Manipulating biodiversity data. This markdown is dependent on first running 'data_compile.Rmd'

Required packages and functions

This workflow is developed in R (R Core Team 2022) using the `tidyverse` suite of packages (Wickham et al. 2019), in addition using `arrow` (Richardson et al 2023) for efficient data storage, `rotl` (Michonneau et al 2023) to access the API of the Open Tree of Life, `ape` (Paradis et al 2023) and `phytools` (Revell 2023 2023) to manipulate phylogenies, and `geosphere` (Hijmans 2022 2023) to manipulate coordinates. For R version, platform, package versions, etc. see Reproducibility.

Load required packages:

```
library(dplyr)
library(arrow)
library(rotl)
library(tidyverse)
library(ape)
library(phytools)
library(geosphere)
library(data.table)
```

We also require a function for later analyses, designed to scale matrices to a maximum of 1 and minimum of zero.

```
norm_range <- function(x){
  (x-min(x, na.rm = T))/(max(x, na.rm = T)-min(x, na.rm = T))
}
```

Load and tidy data

Here we draw from the data compiled in data_compile.Rmd using the arrow R package.

```
compiled_data <- arrow::read_parquet("../data/derived_data/compiled_data.parquet") %>% as_tibble()
```

Across some of the datasets we compiled, the population abundance data are not resolved to the species level. For instance, some trends will just be attributed to 'Genera sp.'. For our analyses, we require all data are resolved to the species level, so here we remove all records without a sufficient taxonomic resolution.

```
compiled_data$species = gsub("^\\s+|\\s+$", "", compiled_data$species)
compiled_data$name_count = sapply(strsplit(compiled_data$species, " "), length)
compiled_data = subset(compiled_data, name_count == 2)
compiled_data$species = paste0(compiled_data$species, " ")
compiled_data$species = gsub(" spec. ", " unknownspecies", compiled_data$species)
compiled_data$species = gsub(" sp. ", " unknownspecies", compiled_data$species)
compiled_data$species = gsub(" sp ", " unknownspecies", compiled_data$species)
compiled_data$spec_resolve = grepl("unknownspecies", compiled_data$species, fixed = TRUE)
compiled_data = subset(compiled_data, spec_resolve == F)
compiled_data$species = substr(compiled_data$species,1,nchar(compiled_data$species)-1)
compiled_data$name_count = NULL
compiled_data$spec_resolve = NULL
```

Report the number of species and unique sites in each dataset after this data subsetting

```
count_1 = compiled_data%>%
  group_by(dataset_id) %>%
  summarise(N_obv = n(), N_spec = n_distinct(species), N_site = n_distinct(site))
```

Compile phylogenies

Here we search for phylogenies (topologies may be a fairier name as the phylogenies lack branch lengths - although this is something we estimate later in the workflow) from the Open Tree of Life. This is hashtagged out as it takes a few days to run. We provide the data file.

```
#phylo_list = list()
#for(a in unique(compiled_data$dataset_id)){
#  message(paste0("Extracting tree for: ", a))
#  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
#  tx_search = tnrs_match_names(names = unique(tmp_df$species), context_name = "All life")
#  message("... taxa search complete...")
#  ott_in_tree = ott_id(tx_search)[is_in_tree(ott_id(tx_search))]
#  message("... ott id's extracted...")
#  tr = tol_induced_subtree(ott_ids = ott_in_tree)
#  message("... tree compiled...")
#  tr = ape::compute.brlen(tr)
#  phylo_list[[a]] = list(tx_search,ott_in_tree,tr)
#}

#saveRDS(phylo_list, "../data/derived_data/trees.rds")
phylo_list = readRDS("../data/derived_data/trees.rds")
```

Next we remove population trends for species lacking phylogenetic information. This step risks imposing a bias as species without phylogeny information may be more at risk of having a particular trend type (e.g. declining). This is something we are unable to assess in the current manuscript.

```
# Remove species without a phylogeny match
compiled_data2 = NULL
for(a in unique(compiled_data$dataset_id)){
  message(paste0("Removing species without a tip: ", a))
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
```

```
  spec_list = sub(".*ott", "", phylo_list[[a]][[3]]$tip.label)
  phylo_list[[a]][[1]] = phylo_list[[a]][[1]][phylo_list[[a]][[1]]$ott_id %in% spec_list, ]
  tmp_df = left_join(tmp_df, unique(phylo_list[[a]][[1]][,c(2,4)]), by = c("species" = "unique_name"))
  tmp_df = tmp_df[which(!is.na(tmp_df$ott_id)),]
  compiled_data2 = rbind(compiled_data2, tmp_df)
}
compiled_data = compiled_data2
rm(compiled_data2)
```

Report the number of species and unique sites in each dataset after this data subsetting

```
count_2 = compiled_data %>%
  group_by(dataset_id) %>%
  summarise(N_obv = n(), N_spec = n_distinct(species), N_site = n_distinct(site))
```

Keeping high quality data

Next we remove population trends which contain zeros. These zeros may simply represent missed detections. But could also represent extreme cases like extinction or recolonisation, which likely possess a different process and can have eccessive influecne on models (as shown by Leung 2020. Clustered vs catastropic declines. Nature.

We have also opted to constrain the datasets, only keeping the 50% most longest time-series. In some cases, this still results in population time-series with only 2 abundances, so for all datasets - except the small CaPTrends and TimeFISH, which have less than 50,000 population time series - we also impose a further constraint, only keeping trends that contain 5 or more population estimates as this will improve the reliability of trend estimation. In the datasets with less than 50000 populations, this further subsetting would have left the datasets with very few populations, and so was deemed undesirable.

```
compiled_data2 = NULL
for(a in unique(compiled_data$dataset_id)){
  message(paste0("Extracting highest quality data for: ", a))
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  tmp_df = subset(tmp_df, !is.na(latitude))
  tmp_df = subset(tmp_df, !is.na(longitude))


  #For the very large datasets, we reduce
  if(nrow(tmp_df) >= 250000){
      tmp_df$lat_round = round(tmp_df$latitude,0)
      tmp_df$lon_round = round(tmp_df$longitude,0)
  } else {
      tmp_df$lat_round = tmp_df$latitude
      tmp_df$lon_round = tmp_df$longitude
  }
  tmp_df$lon_round = ifelse(tmp_df$lon_round == -180, 180, tmp_df$lon_round)


  tmp_df$site = paste0(tmp_df$site,"_",tmp_df$latitude,"_", tmp_df$longitude)
  tmp_df$coords = paste0(tmp_df$lat_round,"_", tmp_df$lon_round)
  tmp_df$site_spec = paste0(tmp_df$site,tmp_df$species,sep = "_")
  tmp_df = tmp_df %>% group_by(site_spec) %>% filter(!any(abundance < 0))
```

```
  tmp_df = tmp_df %>% group_by(site_spec) %>% filter(!any(is.na(abundance)))
  tmp_df = tmp_df %>% group_by(site_spec) %>% filter(!any(is.nan(abundance)))
  tmp_df = tmp_df %>% group_by(site_spec) %>% filter(!any(is.infinite(abundance)))
  tmp_df_mn = tmp_df %>%
    group_by(site_spec, species, site, coords, date) %>%
    summarise(mn_abundance = mean(abundance))
  tmp_df_gaps = tmp_df_mn %>%
    group_by(site_spec, species, site, coords) %>%
    summarise(miss_perc = n()/((max(date) - min(date))+1), N = n(), range = (max(date) - min(date)), zer
  tmp_df_gapss = tmp_df_gaps
  tmp_df_gaps = subset(tmp_df_gapss, N > 1)
  tmp_df_gaps = subset(tmp_df_gaps, zero == 0)
  tmp_df_gaps = subset(tmp_df_gaps, miss_perc == 1)

  df_len = nrow(tmp_df)

    if(df_len < 50000){
  } else {
    cutoff = ifelse(median(tmp_df_gaps$range, na.rm = T) < 5, 5, median(tmp_df_gaps$range, na.rm = T))
    tmp_df_gaps = subset(tmp_df_gaps, range > cutoff)
  }

  tmp_df_mn = left_join(tmp_df_mn, tmp_df_gaps)
  tmp_df_mn = subset(tmp_df_mn, !is.na(N))
  tmp_df_mn = left_join(tmp_df_mn, unique(tmp_df[,c("dataset_id","coords","latitude", "lat_round","longi
  compiled_data2 = rbind(compiled_data2, tmp_df_mn)
}
compiled_data = compiled_data2
saveRDS(compiled_data, "../data/derived_data/compiled_data.rds")
rm(compiled_data2)
```

Report the number of species and unique sites in each dataset after this data subsetting

```
count_3 = compiled_data %>%
  group_by(dataset_id) %>%
  summarise(N_obv = n(), N_spec = n_distinct(species), N_site = n_distinct(site))
```

Data for modelling_core.R, modelling_sensitivity_structure.R, modelling_prediction.R

This involves removing species from our phylogeny that were excluded in the data cleaning process. We then convert this phylogeny in into a variance-covaraince matrix describing distance in branch lengths between species pairs. After this we derive the distance between sites using the Haversine distance matrix. We extract spatial regions for the hierarchical terms using latitude and longitudes. These spatial regions are sensistive to varying spatial scales in datasets i.e. in BioTIME, 10degree grids are used. In TimeFISH, 0.1degree grids are used. To extract genera hierarchical terms, we extract the parent node of each tip. Finally, we conduct any data transformations, save any matrices as sparse to improve computational efficiency of INLA, and save the cooresponding dataset for modelling

```
analysis_list = list()
summarise_data = NULL
for(a in unique(compiled_data$dataset_id)){
  message(paste0("Preparing data for modelling: ", a))
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  tmp_df$tips_chr = gsub(" ", "_", tmp_df$species)
  tmp_df$tips_chr = paste0(tmp_df$tips_chr, "_ott", tmp_df$ott_id)
  spec_list = unique(tmp_df$tips_chr)
  tr = drop.tip(phylo_list[[a]][[3]], setdiff(phylo_list[[a]][[3]]$tip.label, spec_list))
  tr$edge.length = ifelse(tr$edge.length == 0, sort(unique(tr$edge.length))[2], tr$edge.length)
  phy_mat_trim = vcv.phylo(tr, corr = T)
  phy_mat_trim = phy_mat_trim[, colSums(is.na(phy_mat_trim)) != nrow(phy_mat_trim)]
  phy_mat_trim = phy_mat_trim[rowSums(is.na(phy_mat_trim)) != ncol(phy_mat_trim), ]
  phy_id = rownames(phy_mat_trim)
  phy_mat_trim = solve(phy_mat_trim)
  tmp_df = tmp_df %>%
    left_join(tibble(tips_chr = rownames(phy_mat_trim),
                     tips_code = 1:nrow(phy_mat_trim)))
  colnames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
  rownames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
  tmp_df$tips_code2 = tmp_df$tips_code

  taxo = NULL
  for(b in tr$tip.label){
    tmp_taxo = data.frame(
      tips_chr = as.character(b),
      genus_code = getParent(tr, which(tr$tip.label==b)))
    taxo = rbind(taxo, tmp_taxo)
  }
  taxo$genus_code = as.numeric(as.factor(taxo$genus_code))
  tmp_df = left_join(tmp_df, taxo)
  phy_mat_trim = phy_mat_trim

  tmp_df$region_code = as.numeric(as.factor(paste0(10*round(tmp_df$lat_round/10), "_", 10*round(tmp_df$
  if(length(unique(tmp_df$region_code)) < 5) {
    tmp_df$region_code = as.numeric(as.factor(paste0(5*round(tmp_df$lat_round/5), "_", 5*round(tmp_df$l
  } else {
  }
  if(length(unique(tmp_df$region_code)) < 5) {
    tmp_df$region_code = as.numeric(as.factor(paste0(2*round(tmp_df$lat_round/2), "_", 2*round(tmp_df$l
  } else {
  }
  if(length(unique(tmp_df$region_code)) < 5) {
    tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round), "_", round(tmp_df$lon_rou
  } else {
  }
  if(length(unique(tmp_df$region_code)) >= 5) {
  } else {
    tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round,1), "_", round(tmp_df$lon_r
  }

  tmp_df$site_id = as.numeric(as.factor(tmp_df$coords))
  spa_df = unique(tmp_df[,c("site_id","lat_round","lon_round")])
```

```
spa_df = subset(spa_df, !is.na(lat_round) & !is.na(lon_round))
spa_df
spa_mat_trim = as.matrix(distm(spa_df[,c(3,2)], fun = distHaversine))/1000000
spa_mat_trim = norm_range(spa_mat_trim)
spa_mat_trim = abs(spa_mat_trim - 1)
spa_mat_trim = ifelse(spa_mat_trim == 0, sort(unique(as.vector(spa_mat_trim)))[2], spa_mat_trim)
spa_id = spa_df$site_id
colnames(spa_mat_trim) = spa_id
rownames(spa_mat_trim) = spa_id
spa_mat_trim = spa_mat_trim[rowSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim), ]
spa_mat_trim = spa_mat_trim[, colSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim)]
spa_id = colnames(spa_mat_trim)
spa_mat_trim = solve(spa_mat_trim)

tmp_df$site_chr = as.character(tmp_df$site_id)
tmp_df = tmp_df %>%
  left_join(tibble(site_chr = rownames(spa_mat_trim),
                   site_code = 1:nrow(spa_mat_trim)))
colnames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
rownames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
tmp_df$site_code2 = tmp_df$site_code

tmp_df$site_spec_code = as.numeric(as.factor(tmp_df$site_spec))
tmp_df$site_spec_code2 = tmp_df$site_spec_code

phy_mat_trim = as(phy_mat_trim, "sparseMatrix")
spa_mat_trim = as(spa_mat_trim, "sparseMatrix")

tmp_df = tmp_df %>%
  group_by(site_spec) %>%
  mutate(
    log_abundance = log(mn_abundance),
    cent_abundance = log(mn_abundance) - mean(log(mn_abundance)),
    mean_log = mean(log(mn_abundance)),
    year_centre = date - mean(date),
    year3 = (date - min(date))+1,
    mean_year = mean(date))
tmp_df$year2 = tmp_df$year_centre

newdata <- tmp_df[c(1:100),]
newdata[] = NA
newdata$year_centre = seq(min(tmp_df$year_centre), max(tmp_df$year_centre), length.out = 100)
tmp_df = rbind(tmp_df, newdata)

analysis_list[[a]][[1]] = tmp_df
analysis_list[[a]][[2]] = phy_mat_trim
analysis_list[[a]][[3]] = spa_mat_trim

tmp_summary = data.frame(
  observations = nrow(tmp_df),
  populations = length(unique(tmp_df$site_spec)),
  species = length(unique(tmp_df$tips_code)),
  sites = length(unique(tmp_df$site_code))
```

```
  )
  summarise_data = rbind(summarise_data, tmp_summary)

}
saveRDS(analysis_list, "../data/derived_data/analysis_list.rds")
```

Data for modelling_phylo.R

Here we create a dataset to assess sensitivity to phylogeny quality. We use Open Tree of Life and TimeTree, and identify all common species across the phylogenies. We then conduct all of the same data preperation steps as above '#Data for modelling_core.R, modelling_sensitivity_structure.R, modelling_prediction.R'.

```
species_list = list()
for(a in unique(compiled_data$dataset_id)){
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  specs = unique(tmp_df[,c("species", "ott_id")])
  species_list[[a]] = specs
  write.table(specs$species, paste0("../data/derived_data/",a,"_species.txt"), quote = F, row.names = F
}

phylo_list1 = readRDS("../data/derived_data/trees.rds")

phylo_list2 = list()
for(a in unique(compiled_data$dataset_id)){
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  specs = unique(tmp_df$species)
  tr = read.newick(paste0("../data/derived_data/",a,"_species.nwk"))
  tips_match = data.frame(species = tr$tip.label)
  specs = species_list[[a]]
  specs$species = gsub(" ", "_", specs$species)
  tips_match = left_join(tips_match, specs)
  tips_match$new_tip = paste0(tips_match$species, "_ott", tips_match$ott_id)
  tr$tip.label = tips_match$new_tip
  phylo_list1[[a]][[3]] = drop.tip(phylo_list1[[a]][[3]], setdiff(phylo_list1[[a]][[3]]$tip.label, tr$t
  phylo_list2[[a]] = drop.tip(tr, setdiff(tr$tip.label, phylo_list1[[a]][[3]]$tip.label))
}

analysis_list = list()
summarise_data_phylo = NULL
for(a in unique(compiled_data$dataset_id)){
  message(paste0("Preparing data for modelling: ", a))
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  tmp_df$tips_chr = gsub(" ", "_", tmp_df$species)
  tmp_df$tips_chr = paste0(tmp_df$tips_chr, "_ott", tmp_df$ott_id)
  spec_list = unique(tmp_df$tips_chr)
  tr = drop.tip(phylo_list1[[a]][[3]], setdiff(phylo_list1[[a]][[3]]$tip.label, spec_list))
  tr$edge.length = ifelse(tr$edge.length == 0, sort(unique(tr$edge.length))[2], tr$edge.length)
  phy_mat_trim = vcv.phylo(tr, corr = T)
  phy_mat_trim = phy_mat_trim[, colSums(is.na(phy_mat_trim)) != nrow(phy_mat_trim)]
  phy_mat_trim = phy_mat_trim[rowSums(is.na(phy_mat_trim)) != ncol(phy_mat_trim), ]
  phy_id = rownames(phy_mat_trim)
```

```
phy_mat_trim = solve(phy_mat_trim)
tmp_df = tmp_df %>%
  left_join(tibble(tips_chr = rownames(phy_mat_trim),
                   tips_code = 1:nrow(phy_mat_trim)))
colnames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
rownames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
tmp_df$tips_code2 = tmp_df$tips_code

taxo = NULL
for(b in tr$tip.label){
  tmp_taxo = data.frame(
    tips_chr = as.character(b),
    genus_code1 = getParent(tr, which(tr$tip.label==b)))
  taxo = rbind(taxo, tmp_taxo)
}
taxo$genus_code1 = as.numeric(as.factor(taxo$genus_code1))
tmp_df = left_join(tmp_df, taxo)
phy_mat_trim1 = phy_mat_trim

tr = drop.tip(phylo_list2[[a]], setdiff(phylo_list2[[a]]$tip.label, spec_list))
tr$edge.length = ifelse(tr$edge.length == 0, sort(unique(tr$edge.length))[2], tr$edge.length)
phy_mat_trim = vcv.phylo(tr, corr = T)
phy_mat_trim = phy_mat_trim[, colSums(is.na(phy_mat_trim)) != nrow(phy_mat_trim)]
phy_mat_trim = phy_mat_trim[rowSums(is.na(phy_mat_trim)) != ncol(phy_mat_trim), ]
phy_id = rownames(phy_mat_trim)
phy_mat_trim = solve(phy_mat_trim)
tmp_df = tmp_df %>%
  left_join(tibble(tips_chr = rownames(phy_mat_trim),
                   tips_code = 1:nrow(phy_mat_trim)))
colnames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
rownames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
tmp_df$tips_code2 = tmp_df$tips_code

taxo = NULL
for(b in tr$tip.label){
  tmp_taxo = data.frame(
    tips_chr = as.character(b),
    genus_code2 = getParent(tr, which(tr$tip.label==b)))
  taxo = rbind(taxo, tmp_taxo)
}
taxo$genus_code2 = as.numeric(as.factor(taxo$genus_code2))
tmp_df = left_join(tmp_df, taxo)
phy_mat_trim2 = phy_mat_trim
tmp_df = subset(tmp_df, !is.na(tips_code))

tmp_df$region_code = as.numeric(as.factor(paste0(10*round(tmp_df$lat_round/10), "_", 10*round(tmp_df$l
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(5*round(tmp_df$lat_round/5), "_", 5*round(tmp_df$l
} else {
}
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(2*round(tmp_df$lat_round/2), "_", 2*round(tmp_df$l
} else {
```

```
}
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round), "_", round(tmp_df$lon_rou
} else {
}
if(length(unique(tmp_df$region_code)) >= 5) {
} else {
  tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round,1), "_", round(tmp_df$lon_r
}

tmp_df$site_id = as.numeric(as.factor(tmp_df$coords))
spa_df = unique(tmp_df[,c("site_id","lat_round","lon_round")])
spa_df = subset(spa_df, !is.na(lat_round) & !is.na(lon_round))
spa_df
spa_mat_trim = as.matrix(distm(spa_df[,c(3,2)], fun = distHaversine))/1000000
spa_mat_trim = norm_range(spa_mat_trim)
spa_mat_trim = abs(spa_mat_trim - 1)
spa_mat_trim = ifelse(spa_mat_trim == 0, sort(unique(as.vector(spa_mat_trim)))[2], spa_mat_trim)
spa_id = spa_df$site_id
colnames(spa_mat_trim) = spa_id
rownames(spa_mat_trim) = spa_id
spa_mat_trim = spa_mat_trim[rowSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim), ]
spa_mat_trim = spa_mat_trim[, colSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim)]
spa_id = colnames(spa_mat_trim)
spa_mat_trim = solve(spa_mat_trim)

tmp_df$site_chr = as.character(tmp_df$site_id)
tmp_df = tmp_df %>%
  left_join(tibble(site_chr = rownames(spa_mat_trim),
                   site_code = 1:nrow(spa_mat_trim)))
colnames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
rownames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
tmp_df$site_code2 = tmp_df$site_code

tmp_df$site_spec_code = as.numeric(as.factor(tmp_df$site_spec))
tmp_df$site_spec_code2 = tmp_df$site_spec_code

phy_mat_trim1 = as(phy_mat_trim1, "sparseMatrix")
phy_mat_trim2 = as(phy_mat_trim2, "sparseMatrix")
spa_mat_trim = as(spa_mat_trim, "sparseMatrix")

tmp_df = tmp_df %>%
  group_by(site_spec) %>%
  mutate(
    log_abundance = log(mn_abundance),
    cent_abundance = log(mn_abundance) - mean(log(mn_abundance)),
    mean_log = mean(log(mn_abundance)),
    year_centre = date - mean(date),
    year3 = (date - min(date))+1,
    mean_year = mean(date))
tmp_df$year2 = tmp_df$year_centre

newdata <- tmp_df[c(1:100),]
```

```
  newdata[] = NA
  newdata$year_centre = seq(min(tmp_df$year_centre), max(tmp_df$year_centre), length.out = 100)
  tmp_df = rbind(tmp_df, newdata)

  analysis_list[[a]][[1]] = tmp_df
  analysis_list[[a]][[2]] = phy_mat_trim1
  analysis_list[[a]][[3]] = spa_mat_trim
  analysis_list[[a]][[4]] = phy_mat_trim2

  tmp_summary = data.frame(
    observations = nrow(tmp_df),
    populations = length(unique(tmp_df$site_spec)),
    species = length(unique(tmp_df$tips_code)),
    sites = length(unique(tmp_df$site_code))

  )
  summarise_data_phylo = rbind(summarise_data_phylo, tmp_summary)

}
saveRDS(analysis_list, "../data/derived_data/analysis_list_phylo.rds")
```

Data for modellling Fig 4

Here we prepare the BioTIME data to predict abundance trends for North America. To make predicitons in INLA, you have to specify the data structure priori. This involves specifying the spatial sites you wish to predict. All other data processing is identical to '#Data for modelling_core.R, modelling_sensitivity_structure.R, modelling_prediction.R'

```
analysis_list = list()
for(a in unique(compiled_data$dataset_id)[1]){
  message(paste0("Preparing data for modelling: ", a))
  tmp_df = subset(compiled_data, compiled_data$dataset_id == a)
  tmp_df$tips_chr = gsub(" ", "_", tmp_df$species)
  tmp_df$tips_chr = paste0(tmp_df$tips_chr, "_ott", tmp_df$ott_id)
  spec_list = unique(tmp_df$tips_chr)
  tr = drop.tip(phylo_list[[a]][[3]], setdiff(phylo_list[[a]][[3]]$tip.label, spec_list))
  tr$edge.length = ifelse(tr$edge.length == 0, sort(unique(tr$edge.length))[2], tr$edge.length)
  phy_mat_trim = vcv.phylo(tr, corr = T)
  phy_mat_trim = phy_mat_trim[, colSums(is.na(phy_mat_trim)) != nrow(phy_mat_trim)]
  phy_mat_trim = phy_mat_trim[rowSums(is.na(phy_mat_trim)) != ncol(phy_mat_trim), ]
  phy_id = rownames(phy_mat_trim)
  phy_mat_trim = solve(phy_mat_trim)
  tmp_df = tmp_df %>%
    left_join(tibble(tips_chr = rownames(phy_mat_trim),
                     tips_code = 1:nrow(phy_mat_trim)))
  colnames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
  rownames(phy_mat_trim) = 1:dim(phy_mat_trim)[1]
  tmp_df$tips_code2 = tmp_df$tips_code

  taxo = NULL
  for(b in tr$tip.label){
    tmp_taxo = data.frame(
```

```
    tips_chr = as.character(b),
    genus_code = getParent(tr, which(tr$tip.label==b)))
  taxo = rbind(taxo, tmp_taxo)
}
taxo$genus_code = as.numeric(as.factor(taxo$genus_code))
tmp_df = left_join(tmp_df, taxo)

newdata <- tmp_df[c(1:5822),]
newdata[] = NA

map_coords = expand.grid(seq(20,60,by = 1),seq(-130,-60,by = 1))
map_coords = data.frame(
  coords = paste0(map_coords$Var1,"_",map_coords$Var2),
  latitude =  map_coords$Var1,
  longitude = map_coords$Var2,
  lat_round = map_coords$Var1,
  lon_round = map_coords$Var2,
  date = 2000
)
map_coords2 = map_coords
map_coords$date = 2001

map_coords = rbind(map_coords, map_coords2)
newdata$latitude = map_coords$latitude
newdata$longitude = map_coords$longitude
newdata$lat_round = map_coords$latitude
newdata$lon_round = map_coords$longitude
newdata$coords = paste0(map_coords$latitude,"_",newdata$longitude)
newdata$date = map_coords$date
newdata$tips_code = 230
newdata$tips_code2 = 230
newdata$genus_code = 139
newdata$site_spec = paste0("predict_am_rob_",newdata$latitude,newdata$longitude)
tmp_df = rbind(tmp_df, newdata)

tmp_df$region_code = as.numeric(as.factor(paste0(10*round(tmp_df$lat_round/10), "_", 10*round(tmp_df$
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(5*round(tmp_df$lat_round/5), "_", 5*round(tmp_df$lo
} else {
}
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(2*round(tmp_df$lat_round/2), "_", 2*round(tmp_df$lo
} else {
}
if(length(unique(tmp_df$region_code)) < 5) {
  tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round), "_", round(tmp_df$lon_rou
} else {
}
if(length(unique(tmp_df$region_code)) >= 5) {
} else {
  tmp_df$region_code = as.numeric(as.factor(paste0(round(tmp_df$lat_round,1), "_", round(tmp_df$lon_r
}
```

```
  tmp_df$site_id = as.numeric(as.factor(tmp_df$coords))

  spa_df = unique(tmp_df[,c("site_id","lat_round","lon_round")])
  spa_df = subset(spa_df, !is.na(lat_round) & !is.na(lon_round))
  spa_mat_trim = as.matrix(distm(spa_df[,c(3,2)], fun = distHaversine))/1000000
  spa_mat_trim = norm_range(spa_mat_trim)
  spa_mat_trim = abs(spa_mat_trim - 1)
  spa_mat_trim = ifelse(spa_mat_trim == 0, sort(unique(as.vector(spa_mat_trim)))[2], spa_mat_trim)
  spa_id = spa_df$site_id
  colnames(spa_mat_trim) = spa_id
  rownames(spa_mat_trim) = spa_id
  spa_mat_trim = spa_mat_trim[rowSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim), ]
  spa_mat_trim = spa_mat_trim[, colSums(is.na(spa_mat_trim)) != ncol(spa_mat_trim)]
  spa_id = colnames(spa_mat_trim)
  spa_mat_trim = solve(spa_mat_trim)

  tmp_df$site_chr = as.character(tmp_df$site_id)
  tmp_df = tmp_df %>%
    left_join(tibble(site_chr = rownames(spa_mat_trim),
                     site_code = 1:nrow(spa_mat_trim)))
  colnames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
  rownames(spa_mat_trim) = 1:dim(spa_mat_trim)[1]
  tmp_df$site_code2 = tmp_df$site_code

  tmp_df$site_spec_code = as.numeric(as.factor(tmp_df$site_spec))
  tmp_df$site_spec_code2 = tmp_df$site_spec_code

  phy_mat_trim = as(phy_mat_trim, "sparseMatrix")
  spa_mat_trim = as(spa_mat_trim, "sparseMatrix")

  tmp_df = tmp_df %>%
    group_by(site_spec) %>%
    mutate(
      log_abundance = log(mn_abundance),
      cent_abundance = log(mn_abundance) - mean(log(mn_abundance)),
      mean_log = mean(log(mn_abundance)),
      year_centre = date - mean(date),
      year3 = (date - min(date))+1,
      mean_year = mean(date))
  tmp_df$year2 = tmp_df$year_centre

  analysis_list[[a]][[1]] = tmp_df
  analysis_list[[a]][[2]] = phy_mat_trim
  analysis_list[[a]][[3]] = spa_mat_trim

}
saveRDS(analysis_list, "../data/derived_data/analysis_list_predict2.rds")
```

Reproducibility

Date rendered

```
## [1] "2024-02-07 17:51:02 GMT"
```

Session info

```
## R version 4.2.3 (2023-03-15)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] data.table_1.14.8 geosphere_1.5-18  phytools_1.5-1    maps_3.4.1
##  [5] ape_5.7-1         lubridate_1.9.2   forcats_1.0.0     stringr_1.5.1
##  [9] purrr_1.0.1       readr_2.1.4       tidyr_1.3.0       tibble_3.2.1
## [13] ggplot2_3.4.4     tidyverse_2.0.0   rotl_3.1.0        arrow_12.0.1
## [17] dplyr_1.1.2
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.7              bit64_4.0.5             jsonlite_1.8.7
##  [4] foreach_1.5.2           assertthat_0.2.1        expm_0.999-7
##  [7] sp_2.1-1                yaml_2.3.7              progress_1.2.2
## [10] numDeriv_2016.8-1.1     pillar_1.9.0            lattice_0.20-45
## [13] glue_1.6.2              quadprog_1.5-8          phangorn_2.11.1
## [16] digest_0.6.33           colorspace_2.1-0        htmltools_0.5.5
## [19] Matrix_1.5-3            XML_3.99-0.14           pkgconfig_2.0.3
## [22] rncl_0.8.7              scales_1.2.1            tzdb_0.4.0
## [25] optimParallel_1.0-2     timechange_0.2.0        combinat_0.0-8
## [28] generics_0.1.3          withr_2.5.2             cli_3.6.1
## [31] mnormt_2.1.1            magrittr_2.0.3          crayon_1.5.2
## [34] evaluate_0.21           fansi_1.0.5             doParallel_1.0.17
## [37] nlme_3.1-162            MASS_7.3-58.2           tools_4.2.3
## [40] prettyunits_1.1.1       hms_1.1.3               lifecycle_1.0.4
## [43] munsell_0.5.0           plotrix_3.8-2           compiler_4.2.3
## [46] clusterGeneration_1.3.7 rlang_1.1.2             grid_4.2.3
## [49] iterators_1.0.14        rstudioapi_0.15.0       igraph_1.5.0
## [52] rmarkdown_2.23          gtable_0.3.4            codetools_0.2-19
## [55] rentrez_1.2.3           R6_2.5.1                knitr_1.43
## [58] fastmap_1.1.1           bit_4.0.5               utf8_1.2.4
## [61] fastmatch_1.1-3         stringi_1.8.1           parallel_4.2.3
## [64] Rcpp_1.0.11             vctrs_0.6.4             scatterplot3d_0.3-44
## [67] tidyselect_1.2.0        xfun_0.39               coda_0.19-4
```