StableClim - regional regression analysis

Stuart C Brown

Contents

Global and regional regression analysis

The code below reads in each of the 19 pre-processed pre-industrial control simulations, and then for maximally overlapping 100-yr windows performs regional regressions using GLS (OLS if GLS fails) at a variety of scales. All data is cosine weighted and masked as appropriate.

Finally, the data is bootstrapped ($n = 1000$) to account for differences in simulation run length of each of the scenarios.

Note, that this process is repeated for the TraCE-21ka paleo-climate simulation, and for each of the 19 spliced historical/RCP models. For the spliced simulations, the process is repeated for each of the four RCP scenarios. The only difference between the TraCE-21 and spliced historical/future, and the piControl regressions is there is no bootstrap. See the manuscript for details.

Set-up

Set up the various directories etc. that will be used in the processing. Need a list of model names, rcp scenarios, and variables to iterate through

```
library(raster)
library(sf)
library(data.table)
library(ncdf4)
library(nlme)
library(gtools)
library(parallel)
library(doSNOW)

# List of models
model_list <- list.dirs("../data/CMIP5/historical/ts/",
  full.names = FALSE,
  recursive = FALSE
)
# Remove the CNRM-CM5 model
## Issue with volcanic forcing (see https://doi.org/10.1073/pnas.1210514109)
model_list <- model_list[!grepl("CNRM-CM5", model_list, perl = TRUE)]
scenarios <- "piControl"
clim_vars <- "ts"

# Shapefiles for regional analysis
wallReg <- read_sf("../data/shp/WallaceRegions.shp", crs = 4326)
wallReg <- st_zm(wallReg)
wallReg$Descriptio <- NULL
wallReg <- wallReg[wallReg$Name != "Polynesian", ] ## remove polynesia

ipccReg <- read_sf("../data/shp/WG1AR5_Annex1_DissolveRegions.shp")

land_sea_mask <-
  raster("../data/Outputs/01_intermediate_outputs/ensemble_mask.grd")
ant_cover <- crop(land_sea_mask, y = extent(-180, 180, -90, -55))
ant_cover[ant_cover == 1] <- 2
ant_cover <- extend(ant_cover, extent(-180, 180, -90, 90))
land_sea_mask <- cover(ant_cover, land_sea_mask)
```

Regression functions

These functions perform the GLS(AR1) regressions on the global weighted mean temperatures. If for some reason the GLS(AR1) model fails, an equivalent model using OLS will be run instead.

```
ar1_fun <- function(x, ...) {
  require(nlme)
  if (sum(is.na(x[, 2])) >= 1) {
    stop("Missing global GMT values!")
  } else {
    # perform gls(AR1) regression for the global data
    mod <- nlme::gls(GMT ~ Year,
      data = x, na = na.omit, method = "REML",
      verbose = FALSE,
      correlation = corAR1(form = ~Year, fixed = FALSE),
      control = glsControl(
        tolerance = 1e-3, msTol = 1e-3,
        opt = c("nlminb", "optim"),
        singular.ok = TRUE, maxIter = 1000,
        msMaxIter = 1000
      )
    )
    ## extract the slope
    slope <- mod$coefficients[2]
    ## calculate the model variability
    var <- sd(mod$residuals)
    ## return all the layers we need
    return(data.frame("Slope" = slope, "Var" = var, "Method" = "GLS"))
  }
}

# As above, but this function performs simple OLS
# this is only used if gls(AR1) returns an error
ols_fun <- function(x, ...) {
  if (sum(is.na(x[, 2])) >= 1) {
    stop("Missing global GMT values!")
  } else {
    mod <- stats::lm(GMT ~ Year, data = x)
    slope <- mod$coefficients[2]
    var <- sd(mod$residuals)
    return(data.frame("Slope" = slope, "Var" = var, "Method" = "OLS"))
  }
}
```

Peforming global regressions for the pre-industrial control simulations

This section of code performs global gls(AR1) regressions on the pre-industrial control data

```
scen <- scenarios[1]
var <- clim_vars[1]
files <- list.files("C:/tmp/cdo_outputs/", "\\.nc$", full.names = TRUE)
files <- files[grepl(scen, files)]

# cosine latitudes for weights
w <- cos(init(raster(res = 2.5), "y") * (pi / 180))

# parallel progress bar
pb <- txtProgressBar(max = length(model_list), style = 3)
progress <- function(n) setTxtProgressBar(pb, n)
opts <- list(progress = progress)

cls <- detectCores() - 1
cls <- makeCluster(cls)
registerDoSNOW(cls)

foreach(
  m = seq_along(model_list),
  .options.snow = opts,
  .inorder = TRUE,
  .packages = c("raster", "nlme", "data.table", "sf")
) %dopar% {
  model <- model_list[m]
  clim_ras <- readAll(brick(files[m]))
  ## Hadley models have issues with massive changes in GMT for some time steps.
  ## See the pre-processed data for details.
  if (model == "HadGEM2-CC") {
    clim_ras <- clim_ras[[-1]]
  }
  if (model == "HadGEM2-ES") {
    clim_ras <- clim_ras[[-c(1, 242)]]
  }
  ## replace z-index
  clim_ras <- setZ(clim_ras, z = seq(1, nlayers(clim_ras), 1), name = "Year")
  ## define the windows and starting positions
  window <- 100
  step <- 1
  starts <- seq(1, to = nlayers(clim_ras) - (window - 1), by = step)
  ## Loop through the starts and perform regressions
  ### Global
  mod_list <- lapply(X = starts, FUN = function(s, ...) {
    e <- s + window - 1
    sRas <- clim_ras[[s:e]]
```

```
  Year <- getZ(sRas)[1]
  ## Global
  ### Global does not include Antarctica!
  wgtVal <- sRas * w
  wgtMean <- cellStats(mask(wgtVal, land_sea_mask, maskvalue = 2), sum) /
    cellStats(mask(w, land_sea_mask, maskvalue = 2), sum)
  glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
  resGlobe <- tryCatch(
    ## if no error- the funTrAR1Global() function will perform as normal
    ar1_fun(glob),
    error = function(err) {
      err
    }
  )
  ## Use OLS if GLS fails, and report the error
  if (inherits(resGlobe, "error")) {
    warning(paste0(
      "GLS regression failed for model ",
      model, ". OLS used for window ", s, ":", e
    ))
    resGlobe <- ols_fun(glob)
  }
  resGlobe$GMT <- round(mean(glob$GMT), 2)
  resGlobe$RegionType <- "Global"
  resGlobe$Region <- "Land/Sea"
  ## Global Land
  wgtVal <- mask(sRas * w, land_sea_mask, maskvalue = 1, inverse = TRUE)
  wgtMean <- cellStats(wgtVal, sum) /
    cellStats(mask(w, land_sea_mask, maskvalue = 1, inverse = TRUE), sum)
  glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
  resLand <- tryCatch(
    ar1_fun(glob),
    error = function(err) {
      err
    }
  )
  if (inherits(resLand, "error")) {
    warning(paste0(
      "GLS regression failed for model ",
      model, ". OLS used for window ", s, ":", e
    ))
    resLand <- ols_fun(glob)
  }
  resLand$GMT <- round(mean(glob$GMT), 2)
  resLand$RegionType <- "Global"
  resLand$Region <- "Land"
  ## Global Ocean
```

```
  wgtVal <- mask(sRas * w, land_sea_mask, maskvalue = 0, inverse = TRUE)
  wgtMean <- cellStats(wgtVal, sum) /
    cellStats(mask(w, land_sea_mask, maskvalue = 0, inverse = TRUE), sum)
  glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
  resOcean <- tryCatch(
    ar1_fun(glob),
    error = function(err) {
      err
    }
  )
  if (inherits(resOcean, "error")) {
    warning(paste0(
      "GLS regression failed for model ",
      model, ". OLS used for window ", s, ":", e
    ))
    resOcean <- ols_fun(glob)
  }
  resOcean$GMT <- round(mean(glob$GMT), 2)
  resOcean$RegionType <- "Global"
  resOcean$Region <- "Ocean"
  ## Bind Global results
  resGlobe <- rbindlist(l = list(resGlobe, resLand, resOcean))
  ## Wallace Regions
  for (region in unique(wallReg$Name)) {
    subReg <- wallReg[wallReg$Name == region, ]
    wgtVal <- mask(sRas * w, subReg)
    wgtMean <- cellStats(wgtVal, sum) / cellStats(mask(w, subReg), sum)
    glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
    resWall <- tryCatch(
      ar1_fun(glob),
      error = function(err) {
        err
      }
    )
    if (inherits(resWall, "error")) {
      warning(paste0(
        "GLS regression failed for model ",
        model, ". OLS used for window ", s, ":", e,
        ". Region: ", region
      ))
      resWall <- ols_fun(glob)
    }
    resWall$GMT <- round(mean(glob$GMT), 2)
    resWall$RegionType <- "Wallace"
    resWall$Region <- region
    resGlobe <- rbindlist(l = list(resGlobe, resWall))
    rm(resWall)
```

```
  }
  ## Holt realms
  for (region in unique(wallReg$HoltRealm)) {
    subReg <- wallReg[wallReg$HoltRealm == region, ]
    wgtVal <- mask(sRas * w, subReg)
    wgtMean <- cellStats(wgtVal, sum) / cellStats(mask(w, subReg), sum)
    glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
    resHolt <- tryCatch(
      ar1_fun(glob),
      error = function(err) {
        err
      }
    )
    if (inherits(resHolt, "error")) {
      warning(paste0(
        "GLS regression failed for model ",
        model, ". OLS used for window ", s, ":", e,
        ". Region: ", region
      ))
      resHolt <- ols_fun(glob)
    }
    resHolt$GMT <- round(mean(glob$GMT), 2)
    resHolt$RegionType <- "Holt realms"
    resHolt$Region <- region
    resGlobe <- rbindlist(l = list(resGlobe, resHolt))
    rm(resHolt)
  }
  ## IPCC regions
  for (region in unique(ipccReg$RegionName)) {
    if (region == "Pacific Islands region") {
      next()
    }
    subReg <- ipccReg[ipccReg$RegionName == region, ]
    wgtVal <- mask(sRas * w, subReg)
    ## IPCC regions are land only, so need to mask out oceans
    wgtVal <- mask(wgtVal, land_sea_mask, maskvalue = 0)
    wgtMean <- cellStats(wgtVal, sum) / cellStats(mask(w, subReg), sum)
    glob <- data.frame(GMT = unlist(wgtMean), Year = seq(s, e, 1))
    resIPCC <- tryCatch(
      ar1_fun(glob),
      error = function(err) {
        err
      }
    )
    if (inherits(resIPCC, "error")) {
      warning(paste0(
        "GLS regression failed for model ",
```

```
          model, ". OLS used for window ", s, ":", e,
          ". Region: ", region
        ))
        resIPCC <- ols_fun(glob)
      }
      resIPCC$GMT <- round(mean(glob$GMT), 2)
      resIPCC$RegionType <- "IPCC"
      resIPCC$Region <- region
      resGlobe <- rbindlist(l = list(resGlobe, resIPCC))
      rm(resIPCC)
    }
    resGlobe$Start <- s
    resGlobe$End <- e
    resGlobe$Model <- model
    ## re-arrange columns
    resGlobe <- resGlobe[, c(9, 7, 8, 5, 6, 4, 1:3)]
    fwrite(resGlobe,
      file =
        paste0(
          "../data/Outputs/piControl_regional_reg_",
          model, "_temp.csv"
        ),
      row.names = FALSE, append = TRUE
    )
    ## return to list
    return(resGlobe)
  })
  ## unlist the results to a data.table and write to disk
  mod_res <- rbindlist(mod_list)
  fwrite(mod_res,
    file =
      paste0("../data/Outputs/piControl_regional_reg_", model, ".csv"),
    row.names = FALSE, append = FALSE
  )
}
stopCluster(cls)
registerDoSEQ()
```

Combine all results

As we are only accounting for differences in simulation run length the original CDF for each model is used, and then a random sample equal to the difference between length of the CDF and the maximum number of centuries for all models is taken. This bootstrap procedure is performed 1000 times for each model. This preserves existing intra-model variability, while discounting run lengths. A final CDF using this bootstrapped data ensures that the internal variability of each of the models has been preserved and contributes equally to the CDF.

```
## Combine all results
piRegFiles <- list.files("../data/Outputs/", "\\.csv$", full.names = TRUE)
piRegFiles <- piRegFiles[!grepl("_temp.csv$", piRegFiles)]
piRegFiles <- piRegFiles[!grepl("bootstrap", piRegFiles)]
glob_data <- rbindlist(lapply(piRegFiles, fread))

glob_data

slope_summary <- glob_data[, .(
  MinSlope = min(Slope, na.rm = TRUE),
  MaxSlope = max(Slope, na.rm = TRUE),
  MeanSlope = mean(Slope, na.rm = TRUE),
  ## type = 6 == NIST standard
  Q10 = quantile(Slope, .10, type = 6),
  Q20 = quantile(Slope, .20, type = 6),
  Q25 = quantile(Slope, .25, type = 6),
  Q50 = quantile(Slope, .50, type = 6),
  Q75 = quantile(Slope, .75, type = 6),
  Q80 = quantile(Slope, .80, type = 6),
  Q90 = quantile(Slope, .90, type = 6),
  NumCents = max(Start) - min(Start)
),
by = c("RegionType", "Region", "Model")
]
slope_summary

pdf("../data/Outputs/ecdf_piRegressions_regional.pdf",
  onefile = TRUE, paper = "a4r"
)
for (i in unique(glob_data$RegionType)) {
  s1 <- glob_data[RegionType == i, ]
  for (j in unique(s1$Region)) {
    s2 <- s1[Region == j, ]
    print(Hmisc::Ecdf(s2$Slope * 100,
      group = s2$Model,
      xlab = "°C/Century", ylab = "f(x)",
      lwd = 0.75,
      col =
        colorRampPalette(RColorBrewer::brewer.pal(11, "Spectral"))(19),
```

```
      label.curves = list(method = "arrow", cex = .8),
      main = paste(i, j, collapse = "_")
    ))
    print(Hmisc::Ecdf(s2$Slope * 100,
      add = TRUE,
      q = seq(0.1, 0.9, 0.1), subtitles = FALSE,
      col = "black", lwd = 1
    ))
  }
}
dev.off()
```

Bootstrap

Data needs to be boostrappted to account for differences in simulation length

```
# Bootstrap
## CCSM4 has the longest piControl run. How many windows?
maxWindows <- max(glob_data[which(glob_data$Model == "CCSM4"), "Start"])

for (i in unique(glob_data$RegionType)) {
  s1 <- glob_data[RegionType == i, ]
  for (j in unique(s1$Region)) {
    message("Bootstrapping: ", paste(i, j, collapse = " "))
    pb <- txtProgressBar(max = 1000, style = 3)
    progress <- function(n) setTxtProgressBar(pb, n)
    opts <- list(progress = progress)
    s2 <- s1[Region == j, ]
    cls <- parallel::detectCores() - 4
    cls <- parallel::makeCluster(cls)
    registerDoSNOW(cls)
    boot_res <- foreach(
      b = seq_along(1:1000), .inorder = TRUE,
      .packages = c("data.table"), .errorhandling = "remove",
      .noexport = c("ant_cover", "ipccReg", "land_sea_mask", "wallReg"),
      .combine = function(...) rbindlist(list(...)),
      .options.snow = opts
    ) %dopar% {
      piBoot <- s2[0, 1:ncol(glob_data)]
      piBoot[, "Bootstrap" := integer()]
      for (m in unique(s2$Model)) {
        nonBoot <- s2[Model == m, ]
        samples <- sample(
          x = seq(1:nrow(nonBoot)),
          size = maxWindows - nrow(nonBoot),
          replace = TRUE
        )
```

```
        dfBootModel <- rbindlist(
          l =
            list(nonBoot, nonBoot[samples, ])
        )[, Bootstrap := as.integer(b)]
        piBoot <- rbindlist(l = list(piBoot, dfBootModel))
      }
      return(piBoot)
    }
    stopCluster(cls)
    registerDoSEQ()
    invisible(rm(pb))
    outname <- paste(i, j, collapse = "_")
    outname <- gsub(
      "\\/", "_",
      gsub(" ", "_", outname)
    )
    message("Saving output: ", outname)

    print(boot_res[order(boot_res$Bootstrap), ][, .(count = .N), by = Model])
    ## Add rows ID's and years
    boot_res[, rowID := seq_len(.N),
      by = list(Model, Bootstrap)
    ][, Year := seq(from = 1, by = 1, length.out = max(rowID)),
      by = list(Model, Bootstrap)
    ]
    saveRDS(boot_res,
      file = paste0(
        "../data/Outputs/bootstrap_slopes_piControl_",
        outname, ".RDS"
      ),
      compress = TRUE
    )
    invisible({
      rm(piBoot_DF, boot_res, outname, s2)
      gc()
    })
  }
  invisible({
    rm(s1)
    gc()
  })
}
```

The piControl global/regional regressions can now be thresholded to identify different rates of natural climate change

```
fil <- list.files("../data/Outputs/",
  pattern = "bootstrap_slopes_piControl_",
  full.names = TRUE
)

library(pbapply)

## Breaks for trend thresholds
break_seq <- c(0.01, 0.025, seq(0.05, 0.95, 0.05), 0.975, 0.99)

global_list <- pblapply(X = fil, function(x) {
  modelResults <- setDT(readRDS(x))
  pos_idx <- which(modelResults$Slope > 0) ## warming
  neg_idx <- which(modelResults$Slope <= 0) ## cooling
  all_periods_signed <- modelResults[, c(as.list(
    quantile(Slope, probs = break_seq)
  ),
  avg = mean(Slope), n = .N
  ),
  by = c("Model", "RegionType", "Region")
  ]
  all_periods_signed <- all_periods_signed[, lapply(.SD, mean),
    by = c("RegionType", "Region"),
    .SDcols = names(all_periods_signed)[-c(1:3)]
  ][
    ,
    "Type" := "All Periods [signed]"
  ]
  all_periods_abs <- modelResults[, c(as.list(
    quantile(abs(Slope), probs = break_seq)
  ),
  avg = mean(abs(Slope)), n = .N
  ),
  by = c("Model", "RegionType", "Region")
  ]
  all_periods_abs <- all_periods_abs[, lapply(.SD, mean),
    by = c("RegionType", "Region"),
    .SDcols = names(all_periods_abs)[-c(1:3)]
  ][
    ,
    Type := "All Periods [absolute]"
  ]
  warm_periods <- modelResults[pos_idx, c(as.list(
    quantile(Slope, probs = break_seq)
  ),
```

```
  avg = mean(Slope), n = .N
  ),
  by = c("Model", "RegionType", "Region")
  ]
  warm_periods <- warm_periods[, lapply(.SD, mean),
    by = c("RegionType", "Region"),
    .SDcols = names(warm_periods)[-c(1:3)]
  ][
    ,
    Type := "Warm periods"
  ]
  cool_periods <- modelResults[neg_idx, c(as.list(
    quantile(Slope, probs = break_seq)
  ),
  avg = mean(Slope), n = .N
  ),
  by = c("Model", "RegionType", "Region")
  ]
  cool_periods <- cool_periods[, lapply(.SD, mean),
    by = c("RegionType", "Region"),
    .SDcols = names(cool_periods)[-c(1:3)]
  ][, Type := "Cool periods"]
  return(rbindlist(list(
    all_periods_signed, all_periods_abs,
    warm_periods, cool_periods
  )))
})

global_list <- rbindlist(global_list)
global_list <- split(global_list[, 1:26], global_list$Type)
names(global_list)
global_list

# save thresholds
saveRDS(global_list,
  "../data/Outputs/Final_DB/piControl_thresholds.RDS",
  compress = TRUE
)
```