Accessing LOCA Downscaling Via OPeNDAP and the Geo Data Portal with R.

USGS Office of Water Information

Introduction

In this example, we will use the R programming language to access LOCA data via the OPeNDAP web service interface using the ncdf4 package then use the geoknife package to access LOCA data using the Geo Data Portal as a geoprocessing service. More examples like this can be found at the Geo Data Portal wiki.
A number of other packages are required for this example. They include: tidyr for preparing data to plot; ggplot2 to create the plots; grid and gridExtra to layout multiple plots in the same figure; jsonlite and leaflet to create the simple map of a polygon; and chron, climates, and PCICt to calculate derived climate indices. They are all available from CRAN except climates which can be installed from github.

About LOCA

This new, feely available, statistically downscaled dataset is summarized in depth at the U.C. San Diego home web page: Summary of Projections and Additional Documentation
From the Metadata: LOCA is a statistical downscaling technique that uses past history to add improved fine-scale detail to global climate models. We have used LOCA to downscale 32 global climate models from the CMIP5 archive at a 1/16th degree spatial resolution, covering North America from central Mexico through Southern Canada. The historical period is 1950-2005, and there are two future scenarios available: RCP 4.5 and RCP 8.5 over the period 2006-2100 (although some models stop in 2099). The variables currently available are daily minimum and maximum temperature, and daily precipitation. For more information visit: http://loca.ucsd.edu/ The LOCA data is available due to generous support from the following agencies: The California Energy Commission (CEC), USACE Climate Preparedness and Resilience Program and US Bureau of Reclamation, US Department of Interior/USGS via the Southwest Climate Science Center, NOAA RISA program through the California Nevada Applications Program (CNAP), NASA through the NASA Earth Exchange and Advanced Supercomputing (NAS) Division
Reference: Pierce, D. W., D. R. Cayan, and B. L. Thrasher, 2014: Statistical downscaling using Localized Constructed Analogs (LOCA). Journal of Hydrometeorology, volume 15, page 2558-2585.

OPeNDAP Web Service Data Access

We'll use the LOCA Future dataset available from the cida.usgs.gov thredds server. The OPeNDAP service can be seen here. and the OPeNDAP base url that we will use with ncdf4 is: http://cida.usgs.gov/thredds/dodsC/loca_future. In the code below, we load the OPeNDAP data source and look at some details about it.
NOTE: The code using ncdf4 below will only work on mac/linux installations of ncdf4. Direct OPeNDAP access to datasets is not supported in the Windows version of ncdf4.
library(ncdf4) loca_nc <- nc_open('http://cida.usgs.gov/thredds/dodsC/loca_future') loca_globals <- ncatt_get(loca_nc, varid = 0)

names(loca_globals) # Available global attributes. ## [1] "history" "creation_date" "Conventions" ## [4] "title" "acknowledgment" "Metadata_Conventions" ## [7] "summary" "keywords" "id" ## [10] "naming_authority" "cdm_data_type" "publisher_name" ## [13] "publisher_url" "creator_name" "creator_email" ## [16] "time_coverage_start" "time_coverage_end" "date_modified" ## [19] "date_issued" "project" "publisher_email" ## [22] "geospatial_lat_min" "geospatial_lat_max" "geospatial_lon_min" ## [25] "geospatial_lon_max" "license" names(loca_nc) # top level names of the ncdf4 object. ## [1] "filename" "writable" "id" "safemode" "format" ## [6] "is_GMT" "groups" "fqgn2Rindex" "ndims" "natts" ## [11] "dim" "unlimdimid" "nvars" "var" names(loca_nc$dim) # Dimensions of the ncdf4 object. ## [1] "bnds" "lat" "lon" "time" loca_nc$nvars # How many variables are available from the ncdf4 object. ## [1] 171

From this, we can see the global variables that are available for the dataset as well as the ncdf4 object's basic structure. Notice that the dataset has 171 variables!

Point-based time series data

For this example, we'll look at the dataset in the Colorado Platte Drainage Basin climate division which includes Denver and North Central Colorado. For the examples that use a point, well use 40.2 degrees north latitude and 105 degrees west longitude, along I25 between Denver and Fort Collins.

View Code

library(jsonlite) library(leaflet) getGDPPolygon<-function(attribute,value) { baseURL<-"http://cida.usgs.gov/gdp/geoserver/ows" layer<-"sample:CONUS_Climate_Divisions" filter<-URLencode(paste0("<Filter><PropertyIsEqualTo><PropertyName>", attribute, "</PropertyName><Literal>", value, "</Literal></PropertyIsEqualTo></Filter>")) dataURL<-paste0(baseURL,"?service=WFS&version=1.0.0&request=GetFeature&typeName=", layer,"&filter=",filter,"&outputFormat=application/json&srsName=EPSG:4326") geojson<-fromJSON(txt=readLines(dataURL, warn = FALSE), collapse = "\n",simplifyVector = FALSE) return(geojson) } attribute <- "NAME" value <- "PLATTE DRAINAGE BASIN" analysis_polygon<-getGDPPolygon(attribute = attribute, value = value) coords<-analysis_polygon$features[[1]]$geometry$coordinates[[1]][[1]] coords<-as.data.frame(matrix(unlist(coords),nrow=length(coords),byrow = T)) names(coords)<-c('lon','lat') leafMapLOCA <- leaflet() %>% setView(lng = mean(coords$lon), lat = mean(coords$lat), zoom = 7,) %>% addTiles("http://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer/tile/{z}/{y}/{x}", options=tileOptions(minZoom = 7, maxZoom = 7)) %>% addGeoJSON(analysis_polygon, weight = 5, color = "#ff7800", fill = FALSE) %>% addMarkers(lng = -105, lat = 40.2)

Get time series data for a single cell.

First, let's pull down a time series from our point of interest using ncdf4. To do this, we need to determine which lat/lon index we are interested in then access a variable of data at that index position.

lat_poi <- 40.2 lon_poi <- 360 - 105 # Note LOCA uses 0 - 360 so -105 is 255 degrees. lat_index <- which.min(abs(lat_poi-loca_ncdimlat$vals)) lon_index <- which.min(abs(lon_poi-loca_ncdimlon$vals)) paste('lat',loca_ncdimlat$vals[lat_index],'lon',loca_ncdimlon$vals[lon_index]) ## [1] "lat 40.21875 lon 254.96875" names(loca_nc$var)[1:9] ## [1] "lat_bnds" "lon_bnds" ## [3] "time_bnds" "pr_CCSM4_r6i1p1_rcp45" ## [5] "tasmax_CCSM4_r6i1p1_rcp45" "tasmin_CCSM4_r6i1p1_rcp45" ## [7] "pr_CCSM4_r6i1p1_rcp85" "tasmax_CCSM4_r6i1p1_rcp85" ## [9] "tasmin_CCSM4_r6i1p1_rcp85" start.time <- Sys.time() test_var<-ncvar_get(nc = loca_nc, varid = names(loca_nc$var)[5], start = c(lon_index,lat_index,1), count = c(1,1,-1)) Sys.time() - start.time # See how long this takes. ## Time difference of 1.653601 mins

Time series plot of two scenarios.

In the code above, we requested the fifth variable, or tasmax_CCSM4_r6i1p1_rcp45. Let's request a second variable and see how rcp45 compares to rcp85. For this example, we'll need three additional packages: chron to deal with the NetCDF time index conversion, tidyr to get data ready to plot with the gather function, and ggplot2 to create a plot of the data.

library(chron) library(tidyr) library(ggplot2) test_var2<-ncvar_get(nc = loca_nc, varid = names(loca_nc$var)[8], start = c(lon_index,lat_index,1), count = c(1,1,-1)) plot_dates<-as.POSIXct(chron(loca_ncdimtime$vals, origin=c(month=1, day=1, year=1900))) pData<-gather_(data = data.frame(dates=plot_dates, rcp45=test_var, rcp85=test_var2), key_col = "Pathway", value_col = "data", gather_cols = c("rcp45","rcp85")) ggplot(pData,aes(x=dates,y=data,colour=Pathway,group=Pathway)) + geom_point(alpha=1/15) + geom_smooth() + ylab('Daily Maximum Temperature (degrees C)') + xlab('Year') + ggtitle(paste0('Daily Maximum Temperature from variables:\n', names(loca_nc$var)[5],', ',names(loca_nc$var)[8])) +

theme(plot.title = element_text(face="bold"))

Areal average time series data access with the Geo Data Portal

Using the method shown above, you can access any variable at any lat/lon location for your application. Next, we'll look at how to get areal statistics of the data for the Platte Drainage Basin climate division polygon shown above.

For this, we'll need the package geoknife. In the code below, we setup the webgeom and webdata objects called stencil and fabric respectively. Then we make a request for one timestep to find out the SUM of the cell weights and the total COUNT of cells in the area weighted grid statistics calculation that the Geo Data Portal will perform at the request of geoknife. This tells us the fractional number of grid cells considered in the calculation as well as the total number of cells that will be considered for each time step.

library(geoknife) stencil <- webgeom() # query(stencil,'geoms') # Uncomment to see list of available geoms. geom(stencil) <- 'sample:CONUS_Climate_Divisions' # query(stencil, 'attributes') # Uncomment to see list of available attributes. attribute(stencil) <- 'NAME' # query(stencil, 'values') # Uncomment to see list of available attribute values. values(stencil) <- 'PLATTE DRAINAGE BASIN' fabric <- webdata(url='http://cida.usgs.gov/thredds/dodsC/loca_future') varList<-query(fabric,'variables') variables(fabric) <- "pr_CCSM4_r6i1p1_rcp45" # query(fabric,'times') # Uncomment to see start and end dates. times(fabric) <- c('2006-01-01', '2006-01-01') knife <- webprocess(STATISTICS = c("SUM","COUNT"), wait = TRUE) job_onestep <- geoknife(stencil, fabric, knife) result(job_onestep) ## DateTime PLATTE DRAINAGE BASIN variable ## 1 2006-01-01 12:00:00 1465.039 pr_CCSM4_r6i1p1_rcp45 ## 2 2006-01-01 12:00:00 1616.000 pr_CCSM4_r6i1p1_rcp45 ## statistic ## 1 SUM ## 2 COUNT

Areal intersection calculation summary

This result shows that the SUM and COUNT are 1465.039 and 1616 respectively. This tells us that of the 1616 grid cells that partially overalap the Platte Drainage Basin polygon, the sum of cell weights in the area weighted grid statistics calculation is 1465.039. The Platte Drainage Basin climate division is about 52,200 square kilometers and the LOCA grid cells are roughly 36 square kilometers (1/16th degree). Given this, we would expect about 1450 grid cells in the Platte Drainage Basin polygon which compares will with the Geo Data Portal's 1465.

Process time and request size analysis

Next, we'll look at the time it might take to process this data with the Geo Data Portal. We'll use the geoknife setup from above, but the default knife settings rather than those used above to get the SUM and COUNT. First we'll get a single time step, then a year of data. Then we can figure out how long the spatial interstection step as well as each time step should take.

times(fabric) <- c('2006-01-01', '2006-01-01') start.time <- Sys.time() job_onestep <- geoknife(stencil, fabric, wait = TRUE) time_one_step <- Sys.time() - start.time times(fabric) <- c('2006-01-01', '2007-01-01') start.time <- Sys.time() job_oneyear <- geoknife(stencil, fabric, wait=TRUE) time_one_year <- Sys.time() - start.time time_per_step <- (time_one_year - time_one_step) / 364 steps_per_var <- 365 * (2100-2006) time_per_var <- time_per_step * steps_per_var time_per_int <- time_one_step - time_per_step cat('Precip time is about',as.numeric(time_per_var,units="hours"), 'hours per variable. \nTime for spatial intersection is about',as.numeric(time_per_int), 'seconds.') ## Precip time is about 0.4170167 hours per variable. ## Time for spatial intersection is about 6.68374 seconds.

This result shows about how long we can expect each full variable to take to process and how much of that process is made up by the spatial intersection calculations. As can be seen, the spatial intersection is insignificant compared to the time series data processing, which means running one variable at a time should be ok.

In the case that the spatial intersection takes a lot of time and the data processing is quick, we could run many variables at a time to limit the number of spatial intersections that are performed. In this case, we can just run a single variable per request to geoknife and the Geo Data Portal.

GDP run to get all the data.

Now, lets run the GDP for the whole loca dataset. This will be 168 variables with each variable taking something like a half hour depending on server load. Assuming 30 minutes per variable, **that is 84 hours!** That may seem like a very long time, but considering that each variable is 34310 time steps, that is a throughput of about **19 time steps per second**.

The code below is designed to keep retrying until the processing result for each has been downloaded successfully. For this demonstration, the processing has been completed and all the files are available in the working directory of the script. **NOTE: This is designed to make one request at a time. Please don't make concurrent requests to the GDP, it tends to be slower than executing them in serial.**

times(fabric) <- c('2006-01-01', '2101-01-01') ran_one<-TRUE while(ran_one){ ran_one<-FALSE for(var_to_get in varList) { # print(var_to_get) # Uncomment for status if(!file.exists(paste0('loca_demo/',var_to_get,'.csv'))) { variables(fabric) <- var_to_get job <- try(geoknife(stencil, fabric, wait=TRUE)) try(download(job,paste0('loca_demo/',var_to_get,'.csv'))) ran_one<-TRUE } } } loca_data_platte<-data.frame(dates=parseTimeseries(paste0('loca_demo/',varList[1],'.csv'),delim=',')$DateTime) for(var_to_parse in varList) { loca_data_platte[var_to_parse] <- parseTimeseries(paste0('loca_demo/',var_to_parse,'.csv'), delim=',')[2] }

Derivative calculations

Now that we have all the data downloaded and it has been parsed into a list we can work with, we can do something interesting with it. The code below shows an example that uses the climates package, available on github to generate some annual indices of the daily data we accessed. For this example, we'll look at all the data and 5 derived quantities.

View Code

library(climates) library(PCICt) years<-as.character(loca_data_platte$dates,format='%Y') scenarios<-c('rcp45','rcp85') vars<-c('pr','tmax','tmin') gcms<-c("CCSM4_r6i1p1", "CESM1-BGC_r1i1p1", "CESM1-CAM5_r1i1p1", "CMCC-CMS_r1i1p1", "CMCC-CM_r1i1p1", "CNRM-CM5_r1i1p1", "CSIRO-Mk3-6-0_r1i1p1", "CanESM2_r1i1p1", "EC-EARTH_r8i1p1", "FGOALS-g2_r1i1p1", "GFDL-CM3_r1i1p1", "GFDL-ESM2G_r1i1p1", "GFDL-ESM2M_r1i1p1", "GISS-E2-H_r6i1p3", "GISS-E2-R_r6i1p1", "HadGEM2-AO_r1i1p1", "HadGEM2-CC_r1i1p1", "HadGEM2-ES_r1i1p1", "IPSL-CM5A-LR_r1i1p1", "IPSL-CM5A-MR_r1i1p1", "MIROC-ESM-CHEM_r1i1p1", "MIROC-ESM_r1i1p1", "MIROC5_r1i1p1", "MPI-ESM-LR_r1i1p1", "MPI-ESM-MR_r1i1p1", "MRI-CGCM3_r1i1p1", "NorESM1-M_r1i1p1", "bcc-csm1-1-m_r1i1p1") pList <- list() thresholds <-list(days_tmax_abv_thresh=c(37.7778), days_tmin_blw_thresh=c(0), longest_run_tmax_abv_thresh=c(32.2222), longest_run_prcp_blw_thresh=c(3), cooling_degree_day_thresh=c(18.3333)) for(scenario in scenarios) { pList[[scenario]]<-list() for(thresh in names(thresholds)) { # Set up the empty output data structure. pList[[scenario]][[thresh]]<-data.frame(year=c(as.numeric(years[1]): (as.numeric(years[length(years)])-1))) pList[[scenario]][[thresh]][gcms]<-NA } prVars <- varList[grepl(paste0("pr.*",scenario), varList)] tmaxVars <- varList[grepl(paste0("tasmax.*",scenario), varList)] tminVars <- varList[grepl(paste0("tasmin.*",scenario), varList)] for(year in 1:length(pList[[scenario]][[names(thresholds)[1]]]$year)) { subRows<-which(years %in% pList[[scenario]][[names(thresholds)[1]]]$year[year]) dates_pcict<-as.PCICt(loca_data_platte[subRows,'dates'],cal="gregorian") pr <- loca_data_platte[subRows,prVars] tmax <-loca_data_platte[subRows,tmaxVars] tmin <- loca_data_platte[subRows,tminVars] indices <- daily_indices(tmin = tmin, tmax = tmax, prec = pr, tmean = (tmin+tmax)/2, thresholds = thresholds, time_PCICt = dates_pcict) for(thresh in 1:length(names(thresholds))) { pList[[scenario]][[names(thresholds)[thresh]]][gcms][year,] <- indices[,thresh] } } }

Plot setup

Now we have a data in a structure that we can use to create some plots. First, we define a function from the ggplot2 wiki that allows multiple plots to share a legend.

View Code

grid_arrange_shared_legend <- function(..., ncol = length(list(...)), nrow = 1, position = c("bottom", "right"), top = NULL, legend.text = NULL) { # From https://github.com/hadley/ggplot2/wiki/Share-a-legend-between-two-ggplot2-graphs plots <-list(...) position <- match.arg(position) g <- ggplotGrob(plots[[1]] + theme(legend.position = position, legend.text=legend.text, legend.title=element_blank()))$grobs legend <- g[[which(sapply(g, function(x) x$name) == "guide-box")]] lheight <-sum(legend$height) lwidth <- sum(legend$width) gl <- lapply(plots, function(x) x + theme(legend.position="none")) gl <- c(gl, ncol = ncol, nrow = nrow) combined <- switch(position, "bottom" = arrangeGrob(do.call(arrangeGrob, gl), legend, ncol = 1, heights = unit.c(unit(1, "npc") - lheight, lheight)), "right" = arrangeGrob(do.call(arrangeGrob, gl), legend, ncol = 2, widths = unit.c(unit(1, "npc") - lwidth, lwidth))) combined<-arrangeGrob(combined, top = top) grid.newpage() grid.draw(combined) }

Summary Plots

Now we can create a set of plot configuration options and a set of comparitive plots looking at the RCP45 (aggressive emmisions reduction) and RCP85 (business as usual).

View Code

library(grid) library(gridExtra) plot_setup<-list(days_tmax_abv_thresh= list(thresh=c(37.7778), title='Days per Year with Maximum Temperature Above 100F', yaxis='Days per Year', plottype=geom_point()), days_tmin_blw_thresh= list(thresh=c(0), title='Days per Year with Minimum Temperature Below 32F', yaxis='Days per Year', plottype=geom_line()), longest_run_tmax_abv_thresh= list(thresh=c(32.2222), title='Longest spell of Days per Year with Maximum Temperature above 90F', yaxis='Spell Length in Days', plottype=geom_line()), longest_run_prcp_blw_thresh= list(thresh=c(3), title='Longest Spell of

```
Days per Year with Precipitation Below 3mm', yaxis='Spell Length in Days', plottype=geom_line()), cooling_degree_day_thresh=
list(thresh=c(18.3333), title='Cooling Degree Days per Year using a 65F Threshold', yaxis='Cooling Degree-Days',
plottype=geom_line())) for(thresh in names(plot_setup)) { minPlot<-
min(pList[[scenarios[1]]][[thresh]][2:length(pList[[scenarios[1]]][[thresh]])],pList[[scenarios[2]]][[thresh]][2:length(pList[[scenarios[2]]][[t
maxPlot<-
max(pList[[scenarios[1]]][[thresh]][2:length(pList[[scenarios[1]]][[thresh]])],pList[[scenarios[2]]][[thresh]][2:length(pList[[scenarios[2]]][[t
for(scenario in scenarios) { datset<-pList[[scenario]][[thresh]] pData<-gather_(datset,'gcm','data',names(datset)[2:length(datset)])
plot_setup[[thresh]][[paste0("plotAll",scenario)]] <- ggplot(pData,aes(x=year,y=data,colour=gcm,group=gcm)) +
plot_setup[[thresh]]$plottype + ylab(plot_setup[[thresh]]$yaxis) + xlab('Year') + ggtitle(scenario) + ylim(minPlot, maxPlot)
pData<-data.frame(mean=apply(datset[names(datset)[2:length(datset)]],1,mean)) pData['max']<-
apply(datset[names(datset)[2:length(datset)]],1,max) pData['min']<-apply(datset[names(datset)[2:length(datset)]],1,min)
pData['year']<-datset$year pData<-gather_(pData,'stat','data',c('mean','max','min'))
plot_setup[[thresh]][[paste0("plotStats",scenario)]] <- ggplot(pData,aes(x=year,y=data,colour=stat,group=stat)) +
plot_setup[[thresh]]$plottype + ylab(plot_setup[[thresh]]$yaxis) + xlab('Year') + ggtitle(scenario) + ylim(minPlot, maxPlot) }
plot_setup[[thresh]]$plotStatsrcp45 <- plot_setup[[thresh]]$plotStatsrcp45 + theme(legend.position="none")
plot_setup[[thresh]]$plotStatsrcp85 <- plot_setup[[thresh]]$plotStatsrcp85 + theme(axis.title.y=element_blank())
grid_arrange_shared_legend(plot_setup[[thresh]]$plotStatsrcp45,
plot_setup[[thresh]]$plotStatsrcp85,ncol=2,top=plot_setup[[thresh]]$title) }
```

All GCM Plots

Finally, to give an impression of how much data this example actually pulled in, here are the same plots with all the GCMs shown rather then the ensemble mean, min, and max.

```
for(thresh in names(plot_setup)) { plot_setup[[thresh]]$plotAllrcp45 <- plot_setup[[thresh]]$plotAllrcp45
plot_setup[[thresh]]$plotAllrcp85 <- plot_setup[[thresh]]$plotAllrcp85 + theme(axis.title.y=element_blank())
grid_arrange_shared_legend(plot_setup[[thresh]]$plotAllrcp45, plot_setup[[thresh]]$plotAllrcp85, ncol=2,
top=plot_setup[[thresh]]$title, legend.text=element_text(size = 6)) }
```