Looking at CMIP using R and Yellowstone.

Douglas Nychka, Dorit Hammerling, Sophia Chen
National Center for Atmospheric Research
Summary

- R Basics
- Fitting the extremes in a data set
- Doing things in parallel
- `Foreach` as a quick way to parallel R
- `Rmpi` as a more general strategy
- Running on YS in batch.

Leverage the usefulness of R for data analysis with the ease of embarrassingly parallel operations on a node of Yellowstone.
R

- R is a *data analysis* command line language that can run in either interactive or batch mode.

- Free and runs on many different systems – including a core on Yellowstone.

- Many contributed packages for specific methods often created by experts in the field. *extRemes*.

- Many good tutorials online.
Some CMIP model output

We will demo R in interactive mode on Yellowstone.

On Yellowstone:

# make a copy of the tutorial directory
cp -r /glade/u/home/nychka/HPC4Stats .
cd HPC4Stats/CMIPExample
module load R/3.0.1
# run R in interactive mode in this directory
R
Some R basics

# load the extremes toolkit library
library( extRemes)
# load some data that was already created in R
load("CMIPExample.rda")
# take a look at workspace
ls()
# size of sliceData
dim( sliceData)
# copy one of the grid boxes
# (1,1) grid box and all times
Y<- dataSlice[1,1,]
# normalize to cm/day,
  Y<- Y*3600*24/10
# set a threshold as the .99 quantile
tailProb<- .01
  threshold<- quantile( Y, 1- tailProb)
print( threshold)
Generalized Pareto

- This is a complementary approach to block maxima

- For data above a given threshold ($\mu$) fit a probability density with the form:

  $$f(x) = \frac{1}{\sigma[1 + \frac{\xi (x-\mu)}{\sigma}](1/\xi+1)}$$

  for $x \geq \mu$.

- $\sigma$ – scale parameter
- $\xi$ – shape parameter

- We are ignoring all the data below the threshold to just fit the tail.

- Having selected the threshold, estimate $\sigma$ and $\xi$ by maximum likelihood.
Boulder daily precipitation

The graph shows the Boulder daily precipitation over the years from 1900 to 2000. The x-axis represents the years, and the y-axis represents the precipitation in centimeters. The data points are clustered around 23.1 cm, indicating a significant amount of precipitation.

The graph also includes a vertical arrow pointing to the 23.1 cm mark, highlighting this value.

The graph is produced by D. Nychka using CMIP and R.
Generalized Pareto Fit:

Fit to observations > 2 cm with 95% CI for 25 year return level

Generalized Pareto: depends on three parameters:

- (1) scale, (2) shape and (3) probability of exceeding threshold.
- With these one can find all quantiles, means and return levels.
The generalized Pareto fit

```r
GPFit <- fevd( Y, threshold=threshold, type="GP", method="MLE")

# list out some stuff from this object
print( GPFit)
print( GPFit$results)
print( GPFit$results$par)
return.level( GPFit, 100, do.ci=TRUE)

# save plot to pdf file
 pdf( "Test1.pdf", width=7, height=7)
 plot( GPFit)
 dev.off()
```

Now grab the pdf file Test1.pdf from Yellowstone and take a look
GP fit diagnostics (Test1.pdf)

fevd(x = Y, threshold = threshold, type = "GP", method = "MLE")
A GEV example

Fit a distribution to the annual maxima using the GEV distribution (consistent with the GP).

# some R arcana to get the annual maxima --
# tapply is for apply a function to data based on categories (years)
yearIndicator <- floor((tm - min(tm))/365.25)
Ymax <- tapply(Y, yearIndicator, "max")
GEVFit <- fevd(Ymax, type="GEV",
               method="MLE")

print(GEVFit)
print(GEVFit$results)
print(GEVFit$results$par)
return.level(GEVFit, 100, do.ci=TRUE)
A parallel for loop

A simple way to convert serial loop in R code to a parallel loop.

1 LIBRARIES Load all the libraries you need

2 SETUP
• Read in some common variables (e.g. lon and lats from grid).
• Define any input functions
• Define objects to control computation

3 CREATE CLUSTER

4 FOR LOOP

5 SAVE RESULTS

Complete script is the file foreach_CMIP_YS.R
# load libraries
library(ncdf4)
library(fields)
library(extRemes)
library(parallel)
library(doParallel)
library(foreach)
SETUP

##### Initial data read ######

dataHandle <- nc_open(dataFileName)  # location of data file relative to
lon<- ncvar_get(dataHandle, "lon")
lat<- ncvar_get(dataHandle,"lat")
tm<- ncvar_get(dataHandle,"time")

nc_close(dataHandle)
What is in a name?

"pr_day_CCSM4_historical_r1i1p1_19550101-19891231.nc"

> summary.ncdf( dataHandle )

DIMENSIONS

time has size 12775
lat has size 192
lon has size 288
bnds has size 2

VARIABLES

1 : time_bnds has size 2 12775
2 : lat_bnds has size 2 192
3 : lon_bnds has size 2 288
4 : pr has size 288 192 12775
CREATE CLUSTER

Machine dependent but handled by the loaded packages.

##### Cluster setup #####

```r
cl <- makePSOCKcluster(numCores) # Create cluster
everRegisterDoParallel(cl)
```

I really don’t understand this magic!
The foreach parallel loop

There is a normal outer loop over the latitudes. Within a latitude task:

```r
outSummary[latindex,,] <- foreach (lonindex = 1:dim(lon),
                                .combine=rbind) %dopar%{

    library(extRemes)  # library that is needed in inner loop
    Y <- dataset[lonindex,]
    # convert to cm/day
    Y <- Y * 3600*24/10
    threshold <- quantile(Y, 1- tailProb)
    frac <- sum(Y > threshold) / length(Y)
    GPFit <- fevd(Y, threshold=threshold, type="GP",
                  method="MLE")
    ReturnLevel <- return.level(GPFit, returnLevelYear, do.ci=TRUE)
    # last line is the returned vector -- this is weird to me!
    c(threshold, GPFit$results$par, frac = frac, ReturnLevel )
}
```
Actually running on YS

Need to submit the Rscript as a batch file:

```bash
#!/bin/bash
#BSUB -J foreach
#BSUB -q small
#BSUB -n 16
#BSUB -P P86850053
#BSUB -W 00:40
#BSUB -o %J.out
#BSUB -e %J.err

module load R/3.0.1
# first file is the R script second will hold output from the
# master session.
R CMD BATCH --no-save foreach_CMIP_YS.R $LSB_JOBID.Rout
```
library( Rmpi)

- A convenient way with R to run many R tasks on many cores. Little knowledge of MPI and parallel computing needed.

- Uses a supervisor/worker model:
  All are full fledged R sessions but the worker sessions only receive instructions from the supervisor.

- The task assigned to each worker is an R function that is passed a unique index. The index is used to determine exactly what task to do. (In our case the index determines a range of grid boxes.)
To avoid too much I/O

For working on time slices use many workers and read each day of output in parallel.

For working on time series assign a worker to each geographical region and read in the time series for just that region.
Our problem:

Fit the Pareto to every grid box of a CMIP experiment.

1 Started by a short batch script and for this example stay within a Yellowstone node using 16 workers.

2 NAMELIST Supervisor R batch session reads in an R namelist (.rnl) that has the information for the computation.

3 SETUP Supervisor sets up the problem and defines some common information

X Rmpi / doTask Spawn workers and broadcast common information
Assign each worker a equal share of the regions.

4 COMBINE Supervisor collates the results and saves them.
What needs to be done

1 *lsf Batch Script* (bsubYS.lsf)

- Set the number of cores (16) and HPC4Stats — the ”hook”

2 *NAMELIST* (CMIP1.rnl)
- Filenames and important variables (e.g. `tailProb <- .01`)
- Info to broadcast to workers
- Sources the specific functions (e.g. `doTask ,combineList`)

3 *SETUP* (setupCMIP.R)
- Sets up indexing and bookkeeping — can be nasty!

4 *COMBINE* (combineList)
Defined in the functions sourced — a convenient way to put the results into an single 3D array (lon, lat, values). Saves time later!
What happens

**Batch output files:**
- `.Rout` From the supervisor R session
- `.out` From `lsf`
- `.err` Should be empty!

**Analysis:**
SETUP configured to write an R binary data set (`result.rda`) in user scratch
A better output name should be chosen!
Take a look

In R

```r
load("/glade/scratch/nychka/result2.rda")
ls()
library(fields)
pdf("Test4.pdf", width=8, height=6)
image.plot( gridData$x, gridData$y, outObject[,,6],
            xlab="", ylab="")
map("world2", col="magenta", add=TRUE)
dev.off()
q()
```
100 year returns
Summary

- R runs on Yellowstone and is effective for embarrassingly parallel data analysis

- Some examples for R Code that is flexible and also easy to document what has been done.

- Easy to test/develop parallel R code on a laptop before YS applications.

Future:

Exploit shared memory on a node

Create flexible tools for assembling times series over multiple files.
Thank you!