PARE Model

Carlynn Fagnant, Julia C. Schedler, Katherine B. Ensor

2023-07-14

Table of contents

Point-to-Area Random Effects

Link to this project's Github Repo (contains all datasets).

This site is a guided tutorial on fitting the Point-to-Area Random Effects model as part of the paper "Spatial-Temporal Extreme Modeling for Point-to-Area Random Effects (PARE)", presented at SDSS 2023 and submitted to the accompanying special issue of the Journal of Data Science.

This paper concerns how to model extreme values when data are available at the point level, but results are desired at the area level.

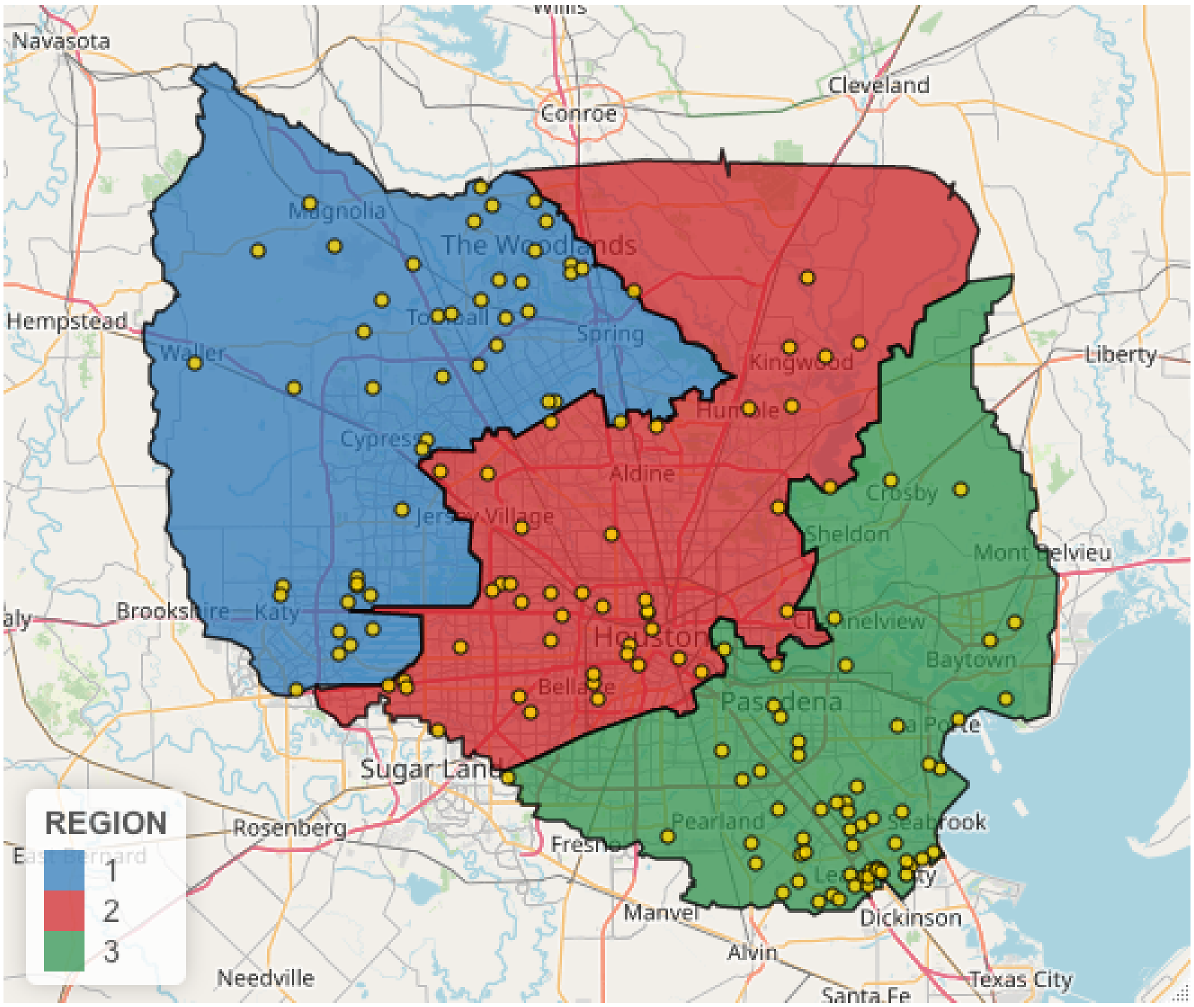


Figure 1: A map of the rainfall gauges (point-level data) and the hydrologic regions (areal data) used in the analysis.

1 PARE model

1.1 Load data, functions, and packages

```
library(dplyr)
library(sf)
library(spdep)
library(extRemes)
library(ggplot2)

thresh = 253

stations <- read.csv("Data/station_info.csv")
stat_nos <- stations[,1]

source("Code/HelperFunctions_PARE/format_data.R")
source("Code/HelperFunctions_PARE/jitter_n_sym.R")
source("Code/HelperFunctions_PARE/get_sym_car_mat.R")
source("Code/HelperFunctions_PARE/get_par_car.R")
source("Code/HelperFunctions_PARE/get_se.R")
source("Code/HelperFunctions_PARE/pars_to_rl.R")
source("Code/HelperFunctions_PARE/rl_with_ci.R")
source("Code/HelperFunctions_PARE/make_varcov_mats_car.R")
source("Code/HelperFunctions_PARE/rl_ci_to_plot_vec.R")
source("Code/HelperFunctions_PARE/create_mat.R")
source("Code/HelperFunctions_PARE/win_rl_to_plot_dat.R")
source("Code/HelperFunctions_PARE/get_latex_table.R")
```

1.2 Format time series of GPD fits

```
## load watershed data
ws_regs <- sf::read_sf("Data/watershed_region_updated")
st_crs(ws_regs) <-sf::st_crs(2278)

## window object generated in Fit_GPD.qmd
## Load rolling GPD fits (takes a bit)
window <- readRDS("Data/window_1day_dclust_updated.rds")  # list of lists (82 x 601) of GP
window_CVM <- readRDS("Data/window_CVM_updated.rds")
window_AD <- readRDS("Data/window_AD_updated.rds")

## Extract windows (note varying # stations)
dat_win_22 <- format_data(window[[22]], stations, ws_regs, window_CVM[[22]]) # 34
dat_win_52 <- format_data(window[[52]], stations, ws_regs, window_CVM[[52]]) # 28
dat_win_82 <- format_data(window[[82]], stations, ws_regs, window_CVM[[82]]) # 149

## save values (faster than reading entire window[[]] object)
save(dat_win_22, file = "Data/dat_win_22")
save(dat_win_52, file = "Data/dat_win_52")
save(dat_win_82, file = "Data/dat_win_82")
```

1.3 Construct spatial weight matrices

Generating the Hausdorff matrix

The `hausMat` function and additional documentation can be found in the `hausdorff` Github repository.

```
# (make sure ws_regs is projected first)
distMat <- hausMat(ws_regs, f1 = 0.5)
distMat/5280

# save output
# saveRDS(distMat, file = "~/Documents/GitHub/Spatial_Extreme_Value_Modeling/Data/hMat_m
```

```
# read in previously computed Hausdorff matrix and convert units to miles
hMat <- readRDS(file = "Data/hMat_med.rds")
```

```r
hMat_miles <- hMat/5280

## Create weight matrix for each window
## Accounts for different numbers of stations (varying point-to-area structure)
## Jiters the values to give valid weight matrix
set.seed(24)
D_22 <- get_sym_car_mat(dat_win_22, hMat_miles)
D_52 <- get_sym_car_mat(dat_win_52, hMat_miles)
D_82 <- get_sym_car_mat(dat_win_82, hMat_miles)
```

1.4 Fit models to each window for each parameter

To obtain region-level estimates of the GPD parameters, fit a conditional auto-regressive model for each GPD parameter for each window, a total of 9 models.

```r
## window 22
## Fit models
car_shape_22 <- spatialreg::spautolm(shape ~ -1 + Reg1 + Reg2 + Reg3,
                                     data = dat_win_22, family="CAR",
                                     listw=mat2listw(1/D_22, style = "C"))
car_ln.scale_22 <- spatialreg::spautolm(log(scale) ~ -1 + Reg1 + Reg2 + Reg3,
                                        data = dat_win_22, family="CAR",
                                        listw=mat2listw(1/D_22,style = "C"))
car_rate_22 <- spatialreg::spautolm(rate ~ -1 + Reg1 + Reg2 + Reg3,
                                    data = dat_win_22, family="CAR",
                                    listw=mat2listw(1/D_22, style = "C"))
## extract parameter estimates
par_22 <- get_par_car(car_ln.scale_22, car_shape_22, car_rate_22)
se_22 <- get_se(car_ln.scale_22, car_shape_22, car_rate_22)[3,]
car_varcov_22 <- make_varcov_mats_car(car_ln.scale_22, car_shape_22)  # se of scale 8.5, 4
st_devs_22 <- data.frame(Reg1 = c(sqrt(diag(car_varcov_22[[1]])), se_22[1]),
                         Reg2 = c(sqrt(diag(car_varcov_22[[2]])), se_22[2]),
                         Reg3 = c(sqrt(diag(car_varcov_22[[3]])), se_22[3]))
row.names(st_devs_22) <- row.names(par_22)



## window 52
## Fit models
car_shape_52 <- spatialreg::spautolm(shape ~ -1 + Reg1 + Reg2 + Reg3,
```

```
                                    data = dat_win_52, family="CAR",
                                    listw=mat2listw(1/D_52, style = "C"))
car_ln.scale_52 <- spatialreg::spautolm(log(scale) ~ -1 + Reg1 + Reg2 + Reg3,
                                       data = dat_win_52, family="CAR",
                                       listw=mat2listw(1/D_52, style = "C"))
car_rate_52 <- spatialreg::spautolm(rate ~ -1 + Reg1 + Reg2 + Reg3,
                                   data = dat_win_52, family="CAR",
                                   listw=mat2listw(1/D_52, style = "C"))
## Extract parameter estimates
par_52 <- get_par_car(car_ln.scale_52, car_shape_52, car_rate_52)
se_52 <- get_se(car_ln.scale_52, car_shape_52, car_rate_52)[3,]
car_varcov_52 <- make_varcov_mats_car(car_ln.scale_52, car_shape_52)  # se of scale 8.5, 4
st_devs_52 <- data.frame(Reg1 = c(sqrt(diag(car_varcov_52[[1]])), se_52[1]),
                         Reg2 = c(sqrt(diag(car_varcov_52[[2]])), se_52[2]),
                         Reg3 = c(sqrt(diag(car_varcov_52[[3]])), se_52[3]))

## window 82
## fit models
car_shape_82 <- spatialreg::spautolm(shape ~ -1 + Reg1 + Reg2 + Reg3,
                                    data = dat_win_82, family="CAR",
                                    listw=mat2listw(1/D_82, style = "C"))
car_ln.scale_82 <- spatialreg::spautolm(log(scale) ~ -1 + Reg1 + Reg2 + Reg3,
                                       data = dat_win_82, family="CAR",
                                       listw=mat2listw(1/D_82, style = "C"))
car_rate_82 <- spatialreg::spautolm(rate ~ -1 + Reg1 + Reg2 + Reg3,
                                   data = dat_win_82, family="CAR",
                                   listw=mat2listw(1/D_82, style = "C"))
## Extract parameter estiamtes
par_82 <- get_par_car(car_ln.scale_82, car_shape_82, car_rate_82)
se_82 <- get_se(car_ln.scale_82, car_shape_82, car_rate_82)[3,]
car_varcov_82 <- make_varcov_mats_car(car_ln.scale_82, car_shape_82)  # se of scale 8.5, 4
st_devs_82 <- data.frame(Reg1 = c(sqrt(diag(car_varcov_82[[1]])), se_82[1]),
                         Reg2 = c(sqrt(diag(car_varcov_82[[2]])), se_82[2]),
                         Reg3 = c(sqrt(diag(car_varcov_82[[3]])), se_82[3]))
```

1.4.1 Table of parameter estimates

```
[1] "Parameters"

              Reg1         Reg2         Reg3
scale 230.85855709 200.15626351 198.45979895
```

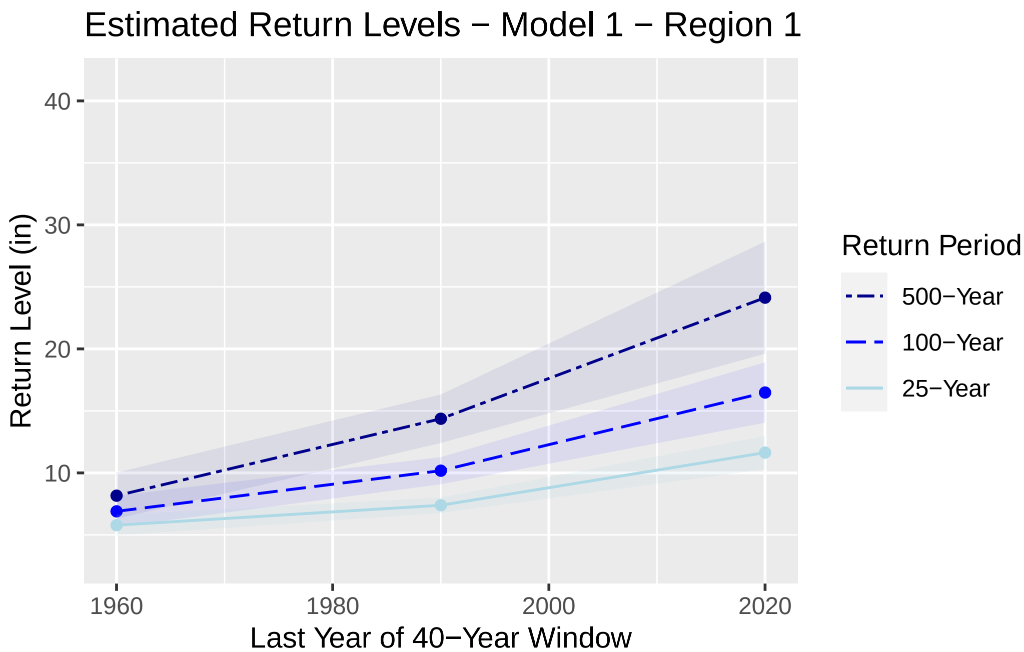
```
shape  -0.01912063   0.05329551   0.14923817
rate    0.02814622   0.03138875   0.03426008

[1] "Standard Errors"

             Reg1         Reg2         Reg3
scale 5.3213140942 3.0221032163 4.5999740248
shape 0.0313947819 0.0204812330 0.0315739471
rate  0.0005872697 0.0003877783 0.0005903755

[1] "Parameters"

             Reg1        Reg2         Reg3
scale 171.95531347 202.3189569 217.80371166
shape   0.17232481   0.1340430   0.16194601
rate    0.02981064   0.0317399   0.03388214

[1] "Standard Errors"

         Reg1         Reg2         Reg3
1 4.4827004098 2.7176004289 6.6253934853
2 0.0131459687 0.0067930158 0.0153266094
3 0.0006522969 0.0003391368 0.0007591396

[1] "Parameters"

             Reg1         Reg2         Reg3
scale 215.07848444 236.68456472 223.98647293
shape   0.20456458   0.16327472   0.18610913
rate    0.05510062   0.04333857   0.05629901

[1] "Standard Errors"

        Reg1        Reg2        Reg3
1 4.323191261 4.894934168 3.699433095
2 0.016361115 0.016841083 0.013420936
3 0.002272369 0.002339033 0.001864008
```

1.5 Calculate return levels + CIs

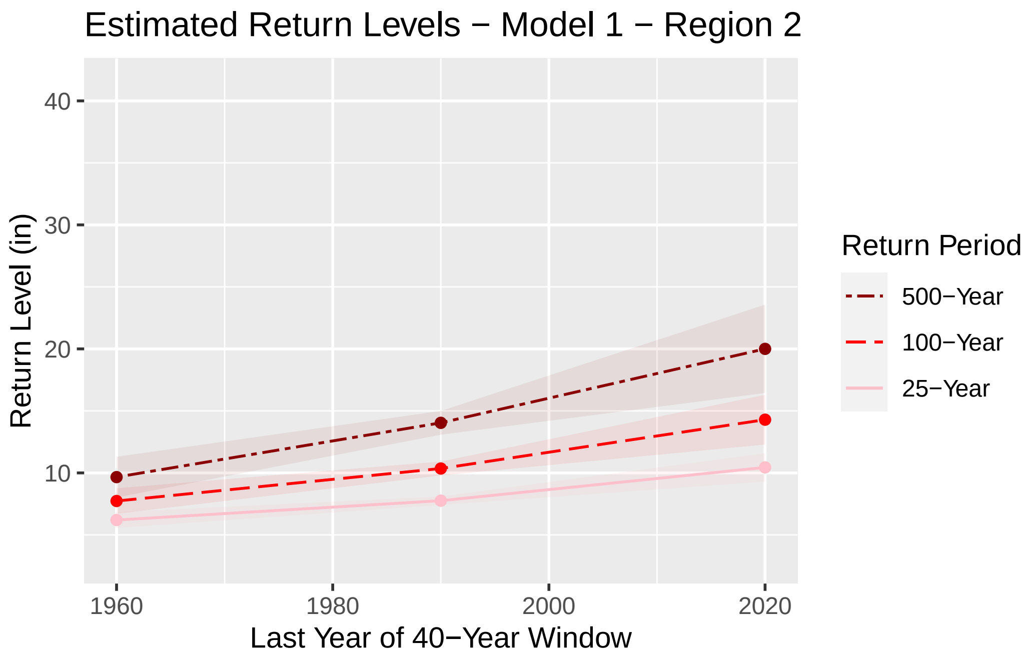
```r
### Estimates and CIs for visuals
car_rl_25_22  <- rl_with_ci(par_mat = par_22,
                            varcov_list = car_varcov_22,
                            return_period = 25,
                            type = "ci", alpha = 0.05)/254
car_rl_100_22 <- rl_with_ci(par_22, car_varcov_22, 100, "ci")/254
car_rl_500_22 <- rl_with_ci(par_22, car_varcov_22, 500, "ci")/254

car_rl_25_52  <- rl_with_ci(par_52, car_varcov_52, 25, "ci")/254
car_rl_100_52 <- rl_with_ci(par_52, car_varcov_52, 100, "ci")/254
car_rl_500_52 <- rl_with_ci(par_52, car_varcov_52, 500, "ci")/254

car_rl_25_82  <- rl_with_ci(par_82, car_varcov_82, 25, "ci")/254
car_rl_100_82 <- rl_with_ci(par_82, car_varcov_82, 100, "ci")/254
car_rl_500_82 <- rl_with_ci(par_82, car_varcov_82, 500, "ci")/254

win_1 <- rl_ci_to_plot_vec(car_rl_25_22, car_rl_100_22, car_rl_500_22)
win_2 <- rl_ci_to_plot_vec(car_rl_25_52, car_rl_100_52, car_rl_500_52)
win_3 <- rl_ci_to_plot_vec(car_rl_25_82, car_rl_100_82, car_rl_500_82)

plot_dat <- win_rl_to_plot_dat(win_1, win_2, win_3)
reg1 <- plot_dat[[1]] # Region 1 data frame for plotting
reg2 <- plot_dat[[2]]
reg3 <- plot_dat[[3]]
```

1.5.1 Visualize return levels

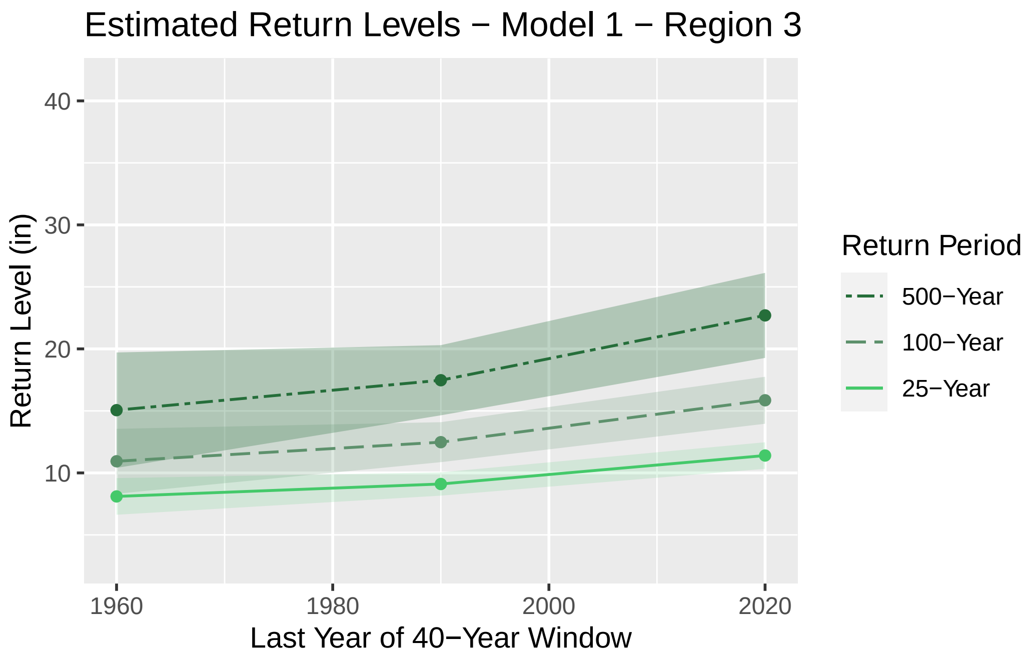
```r
ggplot(data=reg1, aes(x=c(1960, 1990, 2020))) +
    coord_cartesian(ylim = c(3, 41.5)) +
    geom_line(aes(y=RL_500, color="500-Year", linetype = "500-Year")) + geom_point(aes(y=R
    geom_line(aes(y=RL_100, color="100-Year", linetype = "100-Year")) + geom_point(aes(y=R
    geom_line(aes(y=RL_25, color="25-Year", linetype = "25-Year")) + geom_point(aes(y=RL_2
    labs(x="Last Year of 40-Year Window", y="Return Level (in)", title="Estimated Return L
    scale_colour_manual(name = "Return Period", values = c('500-Year' = "darkblue", '100-Y
    scale_linetype_manual(values = c("500-Year" =
    "twodash", "100-Year" = "longdash", "25-Year" = "solid"), breaks = c('500-Year', '100-
```




```
ggplot(data=reg2, aes(x=c(1960, 1990, 2020))) +
  coord_cartesian(ylim = c(3, 41.5)) +
  geom_line(aes(y=RL_500, color="500-Year", linetype = "500-Year")) + geom_point(aes(y=RL_
  geom_line(aes(y=RL_100, color="100-Year", linetype = "100-Year")) + geom_point(aes(y=RL_
  geom_line(aes(y=RL_25, color="25-Year", linetype = "25-Year")) + geom_point(aes(y=RL_25,
  labs(x="Last Year of 40-Year Window", y="Return Level (in)", title="Estimated Return Lev
  scale_colour_manual(name = "Return Period", values = c('500-Year' = "darkred", '100-Year
                      breaks = c('500-Year', '100-Year', '25-Year'))+
    scale_linetype_manual(values = c("500-Year" =
    "twodash", "100-Year" = "longdash", "25-Year" = "solid"), breaks = c('500-Year', '100-
```




```
ggplot(data=reg3, aes(x=c(1960, 1990, 2020))) +
    coord_cartesian(ylim = c(3, 41.5)) +
    geom_line(aes(y=RL_500, color="500-Year", linetype = "500-Year")) + geom_point(aes(y=R
    geom_line(aes(y=RL_100, color="100-Year", linetype = "100-Year")) + geom_point(aes(y=R
    geom_line(aes(y=RL_25, color="25-Year" , linetype = "25-Year")) + geom_point(aes(y=RL_
    labs(x="Last Year of 40-Year Window", y="Return Level (in)", title="Estimated Return L
    scale_colour_manual(name = "Return Period",
                        values = c('500-Year' = "#256e3a", '100-Year' = "#5d916c", '25-Yea
                        breaks = c('500-Year', '100-Year', '25-Year')) +
    scale_linetype_manual(values = c("500-Year" =
    "twodash", "100-Year" = "longdash", "25-Year" = "solid"), breaks = c('500-Year', '100-
```



1.5.2 Table of return levels

```r
## Table 6
car_rl_25_22_se  <- rl_with_ci(par_mat = par_22,
                               varcov_list = car_varcov_22,
                               return_period = 25,
                               type = "se", alpha = 0.05)/254
car_rl_100_22_se <- rl_with_ci(par_22, car_varcov_22, 100, "se")/254
car_rl_500_22_se <- rl_with_ci(par_22, car_varcov_22, 500, "se")/254

## latex output
car_rl_22_list <- list(car_rl_25_22_se, car_rl_100_22_se, car_rl_500_22_se)
get_latex_table_RL(car_rl_22_list)
```

```
&25-yr RL&5.781(0.4269)&6.2(0.3274)&8.102(0.7539)\\
&100-yr RL&6.9(0.645)&7.733(0.5268)&10.938(1.3344)\\
&500-yr RL&8.161(0.9508)&9.66(0.8373)&15.059(2.3736)\\
```

```r
rbind(t(car_rl_25_22_se), t(car_rl_100_22_se), t(car_rl_500_22_se))
```

```
                Reg1      Reg2       Reg3
25-yr RL  5.7812602 6.2000477  8.1017057
SE        0.4268857 0.3273666  0.7538876
100-yr RL 6.8995253 7.7328867 10.9382561
SE        0.6450349 0.5268412  1.3344426
500-yr RL 8.1611487 9.6604967 15.0588530
SE        0.9508076 0.8372663  2.3736256
```

```
## Table 7
car_rl_25_52_se  <- rl_with_ci(par_52, car_varcov_52, 25, "se")/254
car_rl_100_52_se <- rl_with_ci(par_52, car_varcov_52, 100, "se")/254
car_rl_500_52_se <- rl_with_ci(par_52, car_varcov_52, 500, "se")/254

# Latex output
car_rl_52_list <- list(car_rl_25_52_se, car_rl_100_52_se, car_rl_500_52_se)
get_latex_table_RL(car_rl_52_list)
```

```
&25-yr RL&7.391(0.3201)&7.759(0.1733)&9.104(0.4793)\\
&100-yr RL&10.177(0.5587)&10.354(0.2883)&12.477(0.8223)\\
&500-yr RL&14.367(0.9991)&14.038(0.4891)&17.472(1.4436)\\
```

```
rbind(t(car_rl_25_52_se), t(car_rl_100_52_se), t(car_rl_500_52_se))
```

```
                Reg1       Reg2       Reg3
25-yr RL   7.3910220  7.7591772  9.1036876
SE         0.3201394  0.1732745  0.4792676
100-yr RL 10.1767483 10.3537503 12.4771511
SE         0.5586972  0.2883097  0.8222935
500-yr RL 14.3667665 14.0375739 17.4724334
SE         0.9991024  0.4890933  1.4436359
```

```
## TABLE 3
car_rl_25_82_se  <- rl_with_ci(par_82, car_varcov_82, 25, "se")/254
car_rl_100_82_se <- rl_with_ci(par_82, car_varcov_82, 100, "se")/254
car_rl_500_82_se <- rl_with_ci(par_82, car_varcov_82, 500, "se")/254

# Latex output
car_rl_82_list <- list(car_rl_25_82_se, car_rl_100_82_se, car_rl_500_82_se)
get_latex_table_RL(car_rl_82_list)
```

```
&25-yr RL&11.634(0.6874)&10.442(0.5857)&11.399(0.5461)\\
&100-yr RL&16.48(1.2435)&14.291(1.0239)&15.856(0.9666)\\
&500-yr RL&24.131(2.3097)&20.002(1.8186)&22.7(1.7504)\\
```

```r
rbind(t(car_rl_25_82_se), t(car_rl_100_82_se), t(car_rl_500_82_se))
```

```
                Reg1       Reg2       Reg3
25-yr RL  11.6342489 10.4422599 11.3992557
SE         0.6873749  0.5856544  0.5461398
100-yr RL 16.4795690 14.2913625 15.8560177
SE         1.2435110  1.0239273  0.9666489
500-yr RL 24.1305826 20.0023744 22.7002292
SE         2.3097267  1.8186075  1.7503929
```

2 Block Kriging and Regional Max code

The code in the previous few files has been curated for ease-of-use with a focus on the PARE model, which is the novel part of the paper.

Code for all three models PARE, Block Kriging, and Regional Max methods the full analysis as originally included in [?] is available on Github.

3 Fitting GPD to moving windows

The methodology used here is described in [?]. If fitting to more rolling windows than described in the paper this material accompanies, you will need to run this code to generate the object `window`.

3.1 Setup and Declustering

```
# Recall PRCP = Precipitation is measured as "tenths of mm"   (254 = 1 inch)
# To get inches: x/254
# To get mm: x/10

library(lubridate)
library(extRemes)
library(stats)
library(dplyr)
library(xts)
library(gnFit)
library(ismev)
# library(tseries)
# library(trend)
# library(astsa)
# library(ggmap)


# Load in the precipitation data
precip <- read.csv("../scrape-NOAA-data/Data/all_stations_precip_UDP_updated_2021.csv")

### Declustering
precip_dclust = precip
for ( i in seq(from=2, to=length(precip[1,]))){
  station = precip[,i]
  non_na = which(station>=0)
  station[is.na(station)] <- 0
```

```
  dec = extRemes::decluster(station, threshold=0, clusterfun = "max", replace.with = 0)
  stat_clus = as.numeric(dec)

  station = stat_clus
  precip_dclust[non_na,i] = station[non_na]
}

### format
precip_dclust[,1] <- lubridate::ymd(as.character(precip_dclust[,1])) #put Date column into
precip_dclust <- as.data.frame(precip_dclust) #make it a data frame

### Set threshold
thresh <- 253   # for 1-day
```

3.2 Fit GPD to entire series by station

```
## NOTE: suppressing some warnings and errors which do not impact the analysis-- they are

# Distribution Fits --------------------------------------------------------

numstat <- numrow <- 601 #stations
#numcol <- 79 #moving windows (this number will change with the new data since we have mor
numcol <- (2020-39)-1900 +1
  #2020-39 will get the starting year for the last window
  #1900 is the starting year but it is not counted in the math, hence the +1

source("Code/HelperFunctions_GPD/fitgpdR.R")

# Fit to entire station length ---------------------------------------------

# Fitting GPD to entire length of data for each station
fullfits <- fitgpdR(precip_dclust, thresh)
# which(is.na(fullfits))  # only a few are NA



source("Code/HelperFunctions_GPD/gof.R")
source("Code/HelperFunctions_GPD/data_no_na.R")
source("Code/HelperFunctions_GPD/gof_fix_error.R")
```

```
fulldata_no_na <- data_no_na(precip_dclust, thresh)  # saving data corresponding to fullfi

## Goodness of fit measures
CVMp <- ADp <- NULL
for(h in c(1:numstat)){ # for some reason 481 was not working with gof fn, so use gof_fix_
  if(!sum(is.na(fullfits[[h]]))){
    gof_h <- gof_fix_error(fulldata_no_na[[h]], dist="gpd", pr=fullfits[[h]]$results$par,
    CVMp[h] <- gof_h$Wpval
    ADp[h]  <- gof_h$Apval
  }else{
    CVMp[h] <- ADp[h]  <- NA
  }
}
```

3.3 GPD moving window fits

This code chunk generates the "window" file which is the starting point for the PARE model. It will take some time to run and the resulting file is about 1.6 Gb.

```
# All Stations - 40-Year Windows ------------------------------------------

# Eventually we will want these fits for all of the 40-year windows, but for now we can ju


### Uncomment if you want to run from scratch- Note: will take about an hour
### You can instead load the .rds file below to load a saved copy (see readRDS line below)

# ### 40-YEAR MOVING WINDOWS //////////////
#numcol <- 79 #windows of 40
numcol <- (2020-39)-1900 +1
startday <- "-01-01"
endday <- "-12-31"
window <- list()
labelyr <- NULL   # end year of window

# to access window j, gpdfit for station i, use window[[j]][[i]]

startyr <- 1900
## Takes awhile to run
# for(j in 1:numcol){
```

```
#   start <- paste0(startyr, startday)
#   endyr <- startyr + 39
#   end <- paste0(endyr, endday)
#   labelyr[j] <- lubridate::year(as.Date(end))
#
#   start <- which(precip_dclust$Date==start)  #finding indexes corresponding to start & e
#   end   <- which(precip_dclust$Date==end)
#   sub <- precip_dclust[start:end, ]  #subset data to those 40 years
#   window[[j]] <- fitgpdR(sub, thresh)   # GPD fit with extRemes package
#
#   startyr <- startyr + 1
# }
#

## Large file-- takes time to save/load
#saveRDS(window, file="Data/window_1day_dclust_updated.rds")

#window <- readRDS(file="Data/window_1day_dclust_updated.rds")
## file too big for quarto to load-- could try some of the solutions listed
```

3.4 Calculate return levels

Using the GPD parameter estimates to calculate the return level estimates.

```
# Saving Return Levels for Easy Plotting ----------------------------------

# in mm
# this is set up differently for easier plotting
# saving vector of RLs for each station across 79 windows

trend_RL <- list()
for(i in 1:numstat){
  RL <- NULL
  for(j in 1:numcol){
    fit <- window[[j]][[i]]
    if(!sum(is.na(fit)) == TRUE){
      RL[j] <- extRemes::return.level(fit, return.period=100)/10  # divide by 10 to get mm
    }else{
      RL[j] <- NA
    }
```

```
  }
  trend_RL[[i]] <- RL
}

# trend_RL[[588]] # Hobby

trend_RL_25 <- list()
trend_RL_100 <- list()
trend_RL_500 <- list()
for(i in 1:numstat){
  RL_25 <- RL_100 <- RL_500 <- NULL
  for(j in 1:numcol){
    fit <- window[[j]][[i]]
    if(!sum(is.na(fit)) == TRUE){
      RL_25[j] <- extRemes::return.level(fit, return.period=25)/10  # divide by 10 to get
      RL_100[j] <- extRemes::return.level(fit, return.period=100)/10  # divide by 10 to ge
      RL_500[j] <- extRemes::return.level(fit, return.period=500)/10  # divide by 10 to ge
    }else{
      RL_25[j] <- RL_100[j] <- RL_500[j] <- NA
    }
  }
  trend_RL_25[[i]]  <- RL_25
  trend_RL_100[[i]] <- RL_100
  trend_RL_500[[i]] <- RL_500
}


#labelyr <- 1939:2017
labelyr <- 1939:2020
```

3.5 Goodness of fit measures

This code chunk performs goodness of fit tests to determine for each of the rolling windows which of the stations have achieved a good fit to the GPD distribution. These goodness-of-fit results are used in the extreme value model fitting process as part of the data cleaning. The spatial models are only applied to stations which do not show evidence of a lack of GPD fit for a particular window.

```r
# Goodness of Fit ----------------------------------------------------------
# Now going to do GOF for ALL stations, for EACH WINDOW FIT
### Should save RDS to access saved output instead of having to run again
window_CVM <- matrix(nrow = numstat, ncol = numcol)
window_AD <- matrix(nrow = numstat, ncol = numcol)
for(j in 1:numcol){
  CVMp <- ADp <- NULL
  for(i in 1:numstat){
    fit <- window[[j]][[i]]
    if(!sum(is.na(fit)) == TRUE){
      gof_h <- gof_fix_error(fit$x, dist="gpd", pr=fit$results$par, threshold=thresh)   #
      CVMp[i] <- gof_h$Wpval
      ADp[i] <- gof_h$Apval
    }else{
      CVMp[i] <- ADp[i] <- NA
    }
  }
  window_CVM[, j] <- CVMp
  window_AD[, j] <- ADp
}
window_CVM <- as.data.frame(window_CVM)
window_AD <- as.data.frame(window_AD)

 # saveRDS(window_CVM, file="Data/window_CVM_updated.rds")
 # saveRDS(window_AD, file="Data/window_AD_updated.rds")

window_CVM <- readRDS("Data/window_CVM_updated.rds")
window_AD <- readRDS("Data/window_AD_updated.rds")

# window_CVM[i, j] gives CVM (Cramer Von Mises) p-value for station i, window j  (similar
# window[[j]][[i]] gives gpd fit output for station i, window j
```