Supplementary appendices for MMathStat project by Matthew Capewell

Appendix 1

```
#Example data pre-processing into annual maxima
sanjuan2010=read.table("41053h2010.txt")
sanjuan2011=read.table("41053h2011.txt")
sanjuan2012=read.table("41053h2012.txt")
sanjuan2013=read.table("41053h2013.txt")
sanjuan2014=read.table("41053h2014.txt")
sanjuan2015=read.table("41053h2015.txt")
sanjuan2016=read.table("41053h2016.txt")
sanjuan=rbind(sanjuan2010,sanjuan2011,sanjuan2012,sanjuan2013,
              sanjuan2014,sanjuan2015,sanjuan2016)
sanjuan[sanjuan[,7]==99,7] = NA
temp=0
for(i in 0:6){
  temp[i+1]=max(sanjuan[sanjuan[,1]==2010 + i,7],na.rm=TRUE)
}
sanjuan = temp

#Site 1 return level plot
library(ismev)
A=gev.fit(sanjuan)
Amle=gev.fit(sanjuan)$mle
zA = function(r){Amle[1] - Amle[2]/Amle[3]*(1-(-log(1-1/r))^(-Amle[3]))}
zAse = function(r){
  yr=-log(1-(1/r))
  del=matrix(ncol=1,nrow=3)
  del[1,1]=1
  del[2,1]=-((A$mle[3])**(-1))*(1-(yr**(-A$mle[3])))
  del[3,1]=((A$mle[2])*((A$mle[3])**(-2))*(1-((yr)**(-A$mle[3]))))-
    ((A$mle[2])*((A$mle[3])**(-1))*((yr)**(-(A$mle[3])))*log(yr))
  del.transpose=t(del)
  varcovar = A$cov
  return(sqrt(del.transpose%*%varcovar%*%del))
}

plot(-log(-log(1-1/(2:100))), zA((2:100)),type="l",ylim=c(0,40))
temp1=0;temp2=0
for(r in 2:100){
  temp1[r-1]=zA(r)+1.96*zAse(r)
  temp2[r-1]=zA(r)-1.96*zAse(r)
}
lines(-log(-log(1-1/(2:100))), temp1,lty=2,col=2)
lines(-log(-log(1-1/(2:100))), temp2,lty=2,col=2)
```

Appendix 2

```r
########### GPD BAYES ###########
#n is the number of runs
#dataset is the declustered exceedances
#sigmastart and xistart is where we initiliase the chain
#err eta and err xi are the MH random walk innovation sizes (tune according to acc. rate),
#sdeta and sdxi are the normal prior specifications (should be quite large)
#Thresh is the threshold we used, inputted for return levels
#Lambda is exceedance rate
#Thin is the level of thinning desired, 0 or less results in no thinning
bayes6<-function(n,dataset,sigmastart,xistart,erreta,errxi,sdeta,sdxi,thresh,lambda,thin=0)
{
  k<-length(dataset); #lambda=length(dataset[dataset>thresh])/length(dataset)
  sigma<-sigmastart; xi<-xistart; eta<-log(sigma)
  caneta<-vector("numeric"); caneta[1]<-eta              #Proposal values
  canxi<-vector("numeric");  canxi[1]<-xi
  x<-vector("numeric")                                   #x and y hold the chain
  y<-vector("numeric")
  aprobeta<-vector("numeric")                            #Acceptance probabilities
  aprobxi<-vector("numeric")
  ret=matrix(data=0,nrow=n,ncol=10)                      #Return levels
  x[1]<-caneta[1]                                        #Initialise chain
  y[1]<-canxi[1]
  for(i in 1:10){
    ret[1,i]   =thresh + exp(x[1])/y[1]*(((365.25*6*24)*i*10*lambda)^y[1] -1)
  }
  loglik<-function(k,dataset,ETA,XI)
  {
    m<-min(1+(dataset*(XI/exp(ETA))))
    if(m<0.00001){return(as.double(-1000000))}           #This gives NA's otherwise
    if(XI==0){return( -k*ETA - (1/exp(ETA))*sum(dataset) )}#Need this by definition
    loglik<- -k*ETA - (1+(1/XI))*sum(log(1+((XI*dataset)/exp(ETA))))
    return(loglik)
  }
  for(i in 2:n){
    caneta[i]<-x[i-1]+rnorm(1,0,erreta)                  #Metropolis Normal random walk
    likely<-exp((loglik(k,dataset,caneta[i],y[i-1]))-(loglik(k,dataset,x[i-1],y[i-1])))
    aprobeta[i]<-min(1,(likely*((dnorm(caneta[i],0,sdeta)))/((dnorm(x[i-1],0,sdeta)))))
    u<-runif(1)
    if(u<aprobeta[i]){x[i]<-caneta[i]}
    if(u>=aprobeta[i]){x[i]<-x[i-1]}
    canxi[i]<-y[i-1]+rnorm(1,0,errxi)
    likely2<-exp((loglik(k,dataset,x[i],canxi[i]))-(loglik(k,dataset,x[i],y[i-1])))
    aprobxi[i]<-min(1,(likely2*((dnorm(canxi[i],0,sdxi)))/((dnorm(y[i-1],0,sdxi)))))
```

```
    if(u<aprobxi[i]){y[i]<-canxi[i]}
    if(u>=aprobxi[i]){y[i]<-y[i-1]}
    if(y[i]==0){
      for(j in 1:10){
        ret[i,j]   =thresh + exp(x[i])*log((365.25*6*24)*j*10*lambda)
      }                                          #Gives return levels 10:100
    }else{
      for(j in 1:10){
        ret[i,j]   =thresh + exp(x[i])/y[i]*(((365.25*6*24)*j*10*lambda)^y[i]-1)
      }
    }
  }
  if(thin <= 0){                                 #If no thinning provided (or negative)
    print(head(x))
    results=matrix(ncol=14,nrow=n)
    results[,1]=x
    results[,2]=y
    results[,3]=t(aprobeta)                      #Need to transpose these vectors
    results[,4]=t(aprobxi)
    results[,5:14]=ret[,]
  } else {
    num.thin = floor(n/thin)
    results=matrix(ncol=14,nrow=num.thin)
    for(t in 1:num.thin){
      results[t,1]=x[t*thin]
      results[t,2]=y[t*thin]
      results[t,3]=t(aprobeta)[t*thin]
      results[t,4]=t(aprobxi)[t*thin]
      results[t,5:14]=ret[t*thin,]
    }
  }
  return(results)
}
```

Appendix 3

```
#Our function to decluster the data for arbitrary kappa
decluster <- function(dataset,threshold,kappa){
  #Pre-defining kappa for odd situations, useful for web apps
  if(!length(kappa)) {
    kappa = 10
  }
  x=list()
  z=list()
  j=1
  {
    for(i in (kappa+1):length(dataset)){
      if(dataset[i-kappa]>threshold &
          all(dataset[i:(i-kappa+1)]<=threshold)
```

```
    ){
      x=max(dataset[j:i])
      ifelse(i != length(dataset), j<-i+1, NA)
      z=c(z,x)}}}
  return(as.numeric(z))
}
```

Appendix 4

```
#Predictive return level
z17=vector("numeric")
for (r in 1:1000){          #For return levels 1 to 1000, find predictive distribution
  print(r)
  CDF=function(z){
    GPDCDF   = vector("numeric",length=length(xi.burn))
    brackets = vector("numeric",length=length(xi.burn))
    for(i in 1:length(xi.burn)){
      brackets[i] = max(0,(1 + xi.burn[i]*(z-8.3)/sigma.burn[i]))
      GPDCDF[i] =  (1-length(peaks)/length(hurrseason)*((brackets[i])**(-1/xi.burn[i])))
    }
    return(mean(GPDCDF) -1 + 1/(365.25*6*24*100*r)) #Increments of 100 to speed up computation
  }
  z17[r]=uniroot(CDF,c(4,90))$root                      #Solve for z analytically
}
```

Appendix 5

```
bayesSSE.GPD<-function(n,windspeeds,hyper=c(0,0.001,10,1,10,1),thin=0)
{
  a=hyper[1]; b=hyper[2]; c=hyper[3]
  d=hyper[4]; e=hyper[5]; f=hyper[6]

  start.gammaeta=rep(0,length=12); start.gammaxi=rep(0,length=12)  #Seasonal components
  start.epsiloneta=rep(0,length=5); start.epsilonxi=rep(0,length=5)#Site components

  err.gammaeta = rep(0.06, length=12); err.gammaxi = rep(0.05, length=12)
  err.epsiloneta = rep(0.05, length=5); err.epsilonxi = rep(0.03, length=5)

  X1 = matrix(ncol = 12, nrow = n)   #MCMC output for the seasonal effects (gamma's) for log(sigma)
  X2 = matrix(ncol = 12, nrow = n)   #MCMC output for the seasonal effects (gamma's) for xi
  Y1 = matrix(ncol = 5, nrow = n)    #MCMC output for the site effects (epsilon's) for log(sigma)
  Y2 = matrix(ncol = 5, nrow = n)    #MCMC output for the site effects (epsilon's) for xi
```

```
can.X1 = matrix(ncol = 12, nrow = n)  #Candidates for the seasonal effects - gammaeta
can.X1[1, ] = start.gammaeta
can.X2 = matrix(ncol = 12, nrow = n)  #Candidates for the seasonal effects - gammaxi
can.X2[1, ] = start.gammaxi
can.Y1 = matrix(ncol = 5, nrow = n)  #Candidates for the site effects - epsiloneta
can.Y1[1, ] = start.epsiloneta
can.Y2 = matrix(ncol = 5, nrow = n)  #Candidates for the site effects - epsilonxi
can.Y2[1, ] = start.epsilonxi

#Now we initialise the first row of the MCMC matrices:
X1[1, ] = ...
...

#Matrices for acceptance probabilities:
aprob.gammaeta = ...
...

#Construct thresholds automatically for each site for each season independently:
thresh = matrix(ncol= 5, nrow = 12)
tempholder = hourlyMax #To preserve column names etc.
tempholder[,2:6] =
print("Calculating thresholds and exceedances:")
for(m in 1:12){
  print(paste("Season number ",m))
  for(k in 1:5){
    temporary = vector("numeric")
    temporary = windspeeds[, (k+1)][windspeeds[, 8]==m] #To make subsetting it easier
    thresh[m,k] = quantile(temporary ,0.9) #90% quantile
    for(u in 1:dim(tempholder)[1]){
      if(windspeeds[u, 8]==m){                #Only use relevant month
        temp = windspeeds[u, (k+1)]
        if(temp > thresh[m,k]){tempholder[u, (k+1)] = temp - thresh[m,k]}
        else{tempholder[u, (k+1)] = NA}       #Computes exceedance
      }
    }
  }
} windspeeds = tempholder

#We're using eta = gammaeta + epsiloneta; xi = gammaxi + epsilonxi
loglik<-function(dataset,
                    GAMMAETA,GAMMAXI,EPSILONETA,EPSILONXI)          #Define log likelihood for GPD
{k=length(dataset)
  m<-min(1+(dataset*((GAMMAXI+EPSILONXI)/exp(GAMMAETA+EPSILONETA))))
  if(m<0.00001){return(as.double(-1000000))}          #This gives NA's otherwise
```

```
    if((GAMMAXI+EPSILONXI)==0){return( -k*(GAMMAETA+EPSILONETA) - (1/exp(GAMMAETA+EPSILONETA))*sum(dataset) )}#Need this by definition
    loglik<- -k*(GAMMAETA+EPSILONETA) - (1+(1/(GAMMAXI+EPSILONXI)))*sum(log(1+(((GAMMAXI+EPSILONXI)*dataset)/exp(GAMMAETA+EPSILONETA))))
    return(loglik)}

  #Initialise mu, tau and zeta using the prior means of FCDs
  mu = ...
  ...
  tau[1,1]  = ...
  ...

  #Start Metropolis-Gibbs alg.:
  print("Start the MCMC:")
  for(i in 2:n){
    print(paste("Iteration ",i))

    #Seasonal effects:
    lik.X1 = vector("numeric", 12); lik.X2 = vector("numeric", 12)
    lik.X1[] = 0; lik.X2[] =0
    for(m in 1:12){
      can.X1[i, m] = X1[(i-1), m] + rnorm(1, 0, err.gammaeta[m])
      can.X2[i, m] = X2[(i-1), m] + rnorm(1, 0, err.gammaxi[m])

      for(k in 1:5){
        data.contribution = windspeeds[, (k+1)][windspeeds[, 8]==m]
        data.contribution = data.contribution[!is.na(data.contribution)]
        if(length(data.contribution)==0){lik.cont.X1 = 0; lik.cont.X2 = 0}else{
          lik.cont.X1 = loglik(data.contribution, can.X1[i, m], X2[(i-1), m], Y1[(i-1), k], Y2[(i-1), k]) - loglik(data.contribution, X1[(i-1), m], X2[(i-1), m], Y1[(i-1), k], Y2[(i-1), k])
          lik.cont.X2 = loglik(data.contribution, X1[(i-1), m], can.X2[i, m], Y1[(i-1), k], Y2[(i-1), k]) - loglik(data.contribution, X1[(i-1), m], X2[(i-1), m], Y1[(i-1), k], Y2[(i-1), k])}
        if(is.na(lik.cont.X1) | is.na(lik.cont.X2)){print(k)}
        lik.X1[m] = lik.cont.X1 + lik.X1[m]
        lik.X2[m] = lik.cont.X2 + lik.X2[m]
      }

      aprob.gammaeta[i, m] = min(1, (exp(lik.X1[m])*(dnorm(can.X1[i, m], 0, 1/sqrt(tau[i-1,1])))/(dnorm(X1[(i-1), m], 0, 1/sqrt(tau[i-1,1])))))
      aprob.gammaxi[i, m]  = min(1, (exp(lik.X2[m])*(dnorm(can.X2[i, m], 0, 1/sqrt(tau[i-1,2])))/(dnorm(X2[(i-1), m], 0, 1/sqrt(tau[i-1,2])))))

      u = runif(2,0,1)
      if(u[1] < aprob.gammaeta[i, m]){
        X1[i, m] = can.X1[i, m]}else{X1[i, m] = X1[(i-1), m]}
      if(u[2] < aprob.gammaxi[i, m]){
        X2[i, m] = can.X2[i, m]}else{X2[i, m] = X2[(i-1), m]}
    }

    #Site effects::
    ...
```

```r
    for(j in 1:5){
      ...

      for(k in 1:12){
        ...}

      aprob.epsiloneta[i, j] = ...
      u = runif(2,0,1)
      if(u[1] < aprob.epsiloneta[i, j]){...}else{...}
      if(u[2] < aprob.epsilonxi[i, j]){...}else{...}
    }

    #FCDs:
    tau[i,1]  = ...
    ...

  }
  if(thin <= 0){
    return(...)
  } else {
    num.thin = floor(n/thin)
    ... = ...
    }
    return(...)
  }
}
```

Appendix 6

```r
## An example of data-pre processing
#Creating a "hollow" timestamp matrix
yearscount = c(8760, 8760, 8784, 8760, 8760, 8760, 8784)
timestamp.mat=matrix(0,ncol=5,nrow=sum(yearscount))
tally=0
yearstart=2010
for(y in 1:length(yearscount)){
  timestamp.mat[(tally + 1):(tally + yearscount[y]), 1]=yearstart-1 + y
  sum=0; daysmonths = c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
  if(yearscount[y]==8784){daysmonths[2]=29} else {daysmonths[2]=28}
  for(i in 1:12){
    timestamp.mat[ (tally + sum + 1):(tally + sum + daysmonths[i]*24), 2] = i
    timestamp.mat[ (tally + sum + 1):(tally + sum + daysmonths[i]*24), 3] =
                                                        as.numeric(gl(daysmonths
                                                           [i],24))
    #Hours
    if(yearscount[y]==8784) {timestamp.mat[ (tally + 1):(tally + yearscount[y
        ]), 4] =
                                                          rep(seq(from=0,to=23,
                                                             by=1), 366)}
    else                    {timestamp.mat[ (tally + 1):(tally + yearscount[y
        ]), 4] =
```

```
                                    rep(seq(from=0,to=23,by=1), 365)}
    sum=sum + daysmonths[i]*24
  }
  tally = tally + yearscount[y]
}
timestamp.mat = as.data.frame(timestamp.mat)
timestamp.mat$Datehour <- sprintf("%.4i-%.2i-%.2iT%.2i", timestamp.mat[, 1],
                                  timestamp.mat[, 2], timestamp.mat[, 3],
                                  timestamp.mat[, 4])
hourlyMax = data.frame(rep(0,length(timestamp.mat$Datehour)),0,0,0,0,0)
colnames(hourlyMax) <- c("Time","Site1","Site2", "Site3","Site4","Site5")
hourlyMax$Time <- timestamp.mat$Datehour

#Pre-processing for site 1
sanjuan[sanjuan[, 7] == 99, 7] <- NA
sanjuan$Datehour <- sprintf("%.4i-%.2i-%.2iT%.2i", sanjuan[, 1], sanjuan[, 2],
                            sanjuan[, 3], sanjuan[, 4])
hourlyMax1 <- aggregate(V7 ~ Datehour, sanjuan, max)
for(i in 1:max(dim(hourlyMax1)[1],dim(hourlyMax)[1])){
  if(any(hourlyMax1[,1][i] == hourlyMax[,1])){#Catch any numeric(0), replace with 0
    if(identical(hourlyMax1[which(hourlyMax1[,1][i] == hourlyMax[,1]),2], numeric(0))){
      hourlyMax[which(hourlyMax1[,1][i] == hourlyMax[,1]),2] = 0.1
    } else {hourlyMax[which(hourlyMax1[,1][i] == hourlyMax[,1]),2] = hourlyMax1[i,2]}
  }
}
```