APPENDIX AVAILABLE ON THE HEI WEB SITE

Research Report 167

Assessment and Statistical Modeling of the Relationship Between Remotely Sensed Aerosol Optical Depth and $PM_{2.5}$ in the Eastern United States

Christopher J. Paciorek and Yang Liu

Appendix E. R Code for Flexible Buffer Modeling

Correspondence may be addressed to Dr. Christopher J. Paciorek, Department of Statistics, 367 Evans Hall, University of California, Berkeley, CA 94720; e-mail: *paciorek@stat.berkeley.edu*.

Although this document was produced with partial funding by the United States Environmental Protection Agency under Assistance Award CR–83234701 to the Health Effects Institute, it has not been subjected to the Agency's peer and administrative review and therefore may not necessarily reflect the views of the Agency, and no official endorsement by it should be inferred. The contents of this document also have not been reviewed by private party institutions, including those that support the Health Effects Institute; therefore, it may not reflect the views or policies of these parties, and no endorsement by them should be inferred.

This document was reviewed by the HEI Health Review Committee but did not undergo the HEI scientific editing and production process.

E. R Code for Flexible Buffer Modeling

```
# this function creates the Z matrix for fitting the distance effect
# using random effects, essentially tricking the lme part of gamm()
# to do the smoothing for us
makeZmatrix=function(sourceLocations,maxDistance=500,numKnots=30,
  sourceStrength=NULL,receptorLocations,gridLocations=NULL,
  maxNumSources=1000){
  # sourceLocations should be a 2-column matrix of source locations
  # maxDistance is maximum distance in meters at which we model
  #   any effect of a source on a receptor
  # numKnots is number of knots to use in penalized spline
  # sourceStrength is source strength, or NULL if all sources are
  #   treated equally
  # receptorLocations is 2-column matrix receptor locations in same
  #   coordinate system as sourceLocations
  # gridLocations is 2-column matrix defining a fine grid across the
  #   study area for plotting purposes
  # maxNumSources is max number of sources within maxDistance of a
  #   receptor - need this constrained for allocating space
  if(is.null(sourceStrength)){
    sourceStrength=rep(1,nrow(sourceLocations))
  }

  n=nrow(dataLocations)
  m=nrow(gridLocations)
  pointDistsReceptor=matrix(0,nr=n,maxNumSegments)
  pointDistsGrid=matrix(0,nr=nrow(gridLocations),maxNumSegments)
  pointEmitsReceptor=matrix(0,nr=n,maxNumSegments)
  pointEmitsGrid=matrix(0,nr=nrow(gridLocations),maxNumSegments)

  receptorLocations=data.frame(receptorLocations)

  # calculate distances to sources and strengths of those sources
  #   for each receptor location
  for(i in 1:n){
    tmp=rdist(receptorLocations[i,],sourceLocations)
    ll=length(tmp[tmp<maxDistance])
    if(ll){
      pointDistsReceptor[i,1:ll]=(tmp[tmp<maxDistance])
      pointEmitsReceptor[i,1:ll]=sourceStrength[tmp<maxDistance]
    }
    if(i%%1000==0){print(i)}
  }
  # calculate distances to sources and strengths of those sources
  #   for each grid location
  for(i in 1:m){
    tmp=rdist(gridLocations[i,],sourceLocations)
    ll=length(tmp[tmp<maxDistance])
    if(ll){
      pointDistsGrid[i,1:ll]=(tmp[tmp<maxDistance])
      pointEmitsGrid[i,1:ll]=sourceStrength[tmp<maxDistance]
    }
    if(i%%1000==0){print(i)}
  }
```

```
# create fixed effect vectors
U=rowSums(pointDistsReceptor*pointEmitsReceptor)
Ugrid=rowSums(pointDistsGrid*pointEmitsGrid)

knots=seq(0,maxDistance,len=numKnots)
k=length(knots)

omega=abs((outer(knots,knots,"-"))^3)
diag(omega)=0
omega.svd=try(svd(omega))
if(is.null(attr(omega.svd,"class"))){
  sqrt.omega=omega.svd$v%*%diag(sqrt(1/omega.svd$d))%*%t(omega.svd$u)
} else{
  stop("Error: error in SVD; rounded value was numerically p.d.")
}
# create random effects matrices for receptor locations
Z=matrix(0,nr=n,nc=k)
for(i in 1:maxNumSources){
  Ztmp=abs((outer(pointDistsReceptor[,i],knots,"-"))^3)
  Z=Z+(Ztmp%*%sqrt.omega)*pointEmitsReceptor[,i]
}
# create random effects matrices for grid locations
Zgrid=matrix(0,nr=m,nc=k)
for(i in 1:maxNumSources){
  Ztmp=abs((outer(pointDistsGrid[,i],knots,"-"))^3)
  Zgrid=Zgrid+(Ztmp%*%sqrt.omega)*pointEmitsGrid[,i]
}

# preparatory calculations to ensure zero contribution at exactly
#   maxDistance
Zmax=matrix(0,nr=n,nc=k)
Ztmp=matrix(abs((maxDistance-knots)^3),nr=n,nc=k,byrow=T)
Ztmp=Ztmp%*%sqrt.omega
for(i in 1:maxNumSources){
  Zmax=Zmax+Ztmp*pointEmitsReceptor[,i]
}

Ztmp=matrix(abs((maxDistance-knots)^3),nr=m,nc=k,byrow=T)
Ztmp=Ztmp%*%sqrt.omega
ZgridMax=matrix(0,nr=m,nc=k)
for(i in 1:maxNumSourceLocations){
  ZgridMax=ZgridMax+Ztmp*pointEmitsGrid[,i]
}

# ensure zero contribution exactly at maxDistance
U=U-maxDistance*rowSums(pointEmitsReceptor)
Ugrid=Ugrid-maxDistance*rowSums(pointEmitsGrid)
Z=Z-Zmax
Zgrid=Zgrid-ZgridMax

# set up fixed effects vector and random effects matrix for a
#   grid of distance values (0,maxDistance) for plotting
#   effect of a unit source with distance
Upoint=Ugridded=seq(0,maxDistance,len=200)
Ztmp=abs((outer(Upoint,knots,"-"))^3)
omega=abs((outer(knots,knots,"-"))^3)
diag(omega)=0
omega.svd=try(svd(omega))
```

```
  if(is.null(attr(omega.svd,"class"))){
    sqrt.omega=omega.svd$v%*%diag(sqrt(1/omega.svd$d))%*%t(omega.svd$u)
  } else{
    stop("Error: error in SVD; rounded value was numerically p.d.")
  }
  Zpoint=Ztmp%*%sqrt.omega
  Ztmp=matrix(abs((maxDistance-knots)^3),nr=nrow(Zpoint),nc=k,byrow=T)
  Zpoint=Zpoint-Ztmp%*%sqrt.omega
  Upoint=Upoint-maxDistance

  return(list(U=U,Ugrid=Ugrid,Z=Z,Zgrid=Zgrid,Upoint=Upoint,
              Zpoint=Zpoint,Ugridded=Ugridded))
}   # end of makeZmatrix()


Zlist=makeZmatrix(sourceLocations=sourceLocations,maxDistance=500,
  numKnots=numKnots,sourceStrength=sourceStrength,
  receptorLocations=receptorLocations,gridLocations=gridLocations)
# assume that necessary data objects already exist to be called as
#   arguments to makeZmatrix

# assume dataset 'dat' exists with outcome and explanatory vars

dat$dummy=as.factor(rep(1,nrow(dat)))
# needed to trick lme() by having only one group

Z=Zlist$Z # for some reason gamm can't use Zlist$Z directly, so
          #   need to assign to new matrix
Zgrid=Zlist$Zgrid
U=Zlist$U
Ugrid=Zlist$Ugrid
Zpoint=Zlist$Zpoint
Ugridded=Zlist$Ugridded
Upoint=Zlist$Upoint

# assume that y output vector and Xmat design matrix exists;
#   X can be replaced with the usual right hand side of a gam()
#   model formula
mod=gamm(y~Xmat,random=list(dummy=pdIdent(~-1+Z)),data=dat)

beta.hat <- mod$lme$coef$fixed['XU']
b.hat <- unlist(mod$lme$coef$random$dummy)

meanFun=Upoint*beta.hat+c(Zpoint%*%t(b.hat))
# this plots an estimate of the contribution of one unit of
#   traffic on a grid of distances over (0,maxDistance)
plot(Ugridded,Upoint*beta.hat+c(Zpoint%*%t(b.hat)),xlab='distance')

# code here is for uncertainty calculation; we need to manipulate
#   the gamm() output to calculate the expression in (D3) in
#   Section D.2.3)
# note that the Vp output of mod$gam is (in latex):
#   \sigma^2 (X^T(\frac{Z^T G Z}{\sigma^2}+I)^{-1}X+S)^{-1} and we
#   need S, which is the block of \hat{B} (8.3) that corresponds
#   to the penalized components of any smooth terms in the mean
#   model
# also, presumably for numerical reasons, the blocks of the resulting
#   estimate of S corresponding to unpenalized mean parameters are
```

```
#   not zero, so we need to enforce this manually

k=0 # need to specify how many design matrix columns there are
    #   for the smooth terms in the gamm() call mean representation,
    #   e.g. 299 if specify k=300 for a spatial smooth in mean
    #   function
pX=length(mod$gam$coef)  # number of mean model coefficients
p=pX+numKnots    # total number of coefficients in model
X=predict(mod$gam,type='lpmatrix')  # design matrix for mean model
C=cbind(X,Z)  # full design matrix
Bhat=matrix(0,nr=ncol(C),nc=ncol(C)) # Bhat in (8.3)
last=pX  # index of end of smooth term design matrix columns for
         #   __penalized__ coefficients; BE CAREFUL HERE: some
         #   of the last columns may correspond to unpenalized
         #   coefficients, in which case these columns should
         #   not be included here, so that the Bhat block
         #   corresponding to these coefficients forced to be
         #   zero; if you have multiple smooth terms in the mean,
         #   you may need to deal with such columns interspersed
         #   in X and zero out the corresponding blocks in Bhat
first=pX-k+1  # index of start of smooth term design matrix columns
ind=first:last
tau2=exp(attr(mod$lme$apVar,'Pars')[1])^2
   # the random effects variance
varY=Z%*%t(Z)*tau2/mod$gam$sig2  # (Z^T G Z)/sig2 in the notation
                                 # of Ruppert, Wand and Carroll (RWC)
diag(varY)=diag(varY)+1 # (Z^T G Z + R)/sig2 in RWC notation;
#  W^-1 in the notation of Wood (variance of Y divided by
#  sig2 since Vp is sig2 * everything else
S=solve(mod$gam$Vp/mod$gam$sig2)-t(X)%*%solve(varY,X)
   # extract implicit smoothing matrix of the fitted model
   #   from the Vp output of gam()
Bhat[ind,ind]=S[ind,ind]  # block corresponding to smooth
   # terms in mean model (the penalized coefficients); note that
   # upper left block corresponding to fixed effects is forced to
   # be all zeroes, as should be any block for unpenalized
   # coefficients of the smooth terms
Bhat[(pX+1):p,(pX+1):p]=diag(rep(1/tau2,numKnots))
   # block corresponding to random effects should be
   #   (1/tau2) times the identity
LtInv=chol((t(C)%*%C+Bhat)/mod$gam$sig2)
  # Cholesky of precision matrix of fixed and random effects
Uposition=which(names(mod$gam$coef)=='U')
T=1000 # number of samples of decay function to
       #   draw from approximate Bayesian posterior
smp=backsolve(LtInv,matrix(rnorm(p*T),nr=p,nc=T))
betaSamples=beta.hat+smp[Uposition,]
  # pick off random sample of decay function fixed effect
bSamples=c(b.hat)+smp[(pX+1):p,]
  # pick off random samples of decay function random effects
smp=matrix(Upoint,nc=1)%*%matrix(betaSamples,nr=1)+Zpoint%*%bSamples
qu=apply(smp,1,quantile,c(.025,.975))  # pointwise 95% confidence
  #  (credible, really) interval for decay function
```