HW4

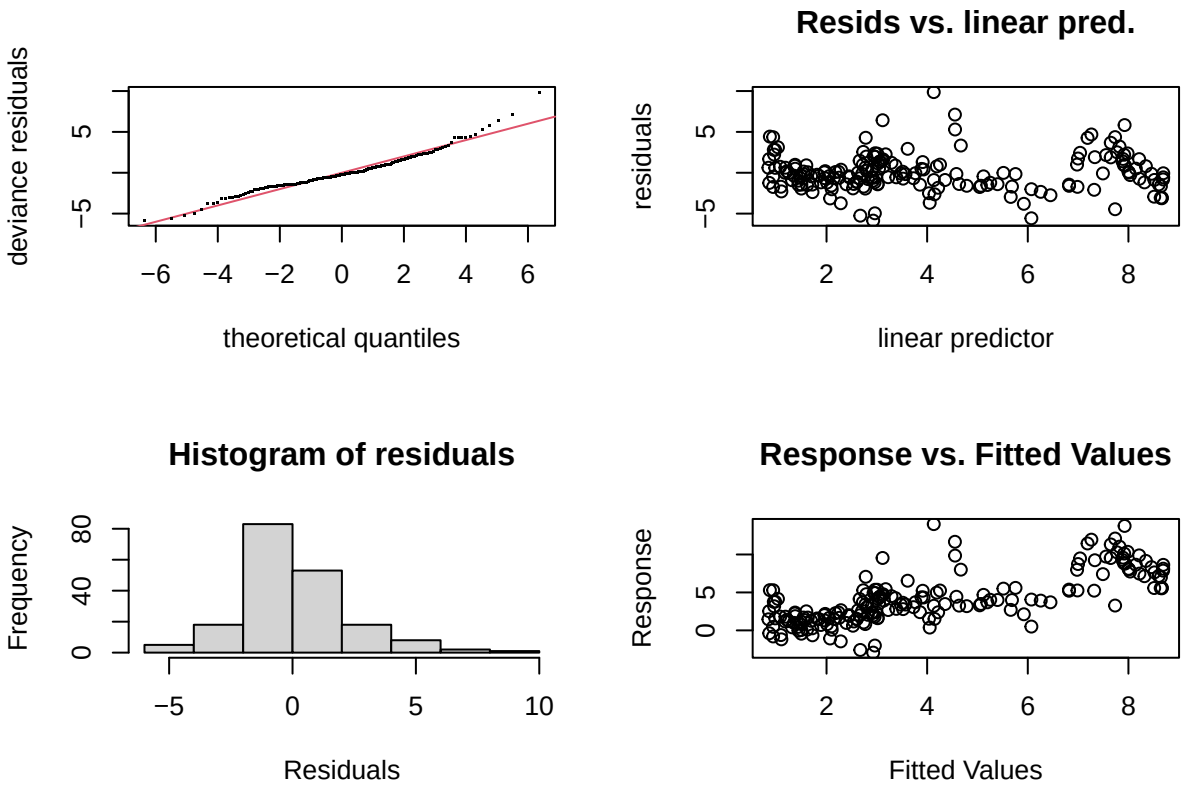
ASG

2023-03-15

2.4

a

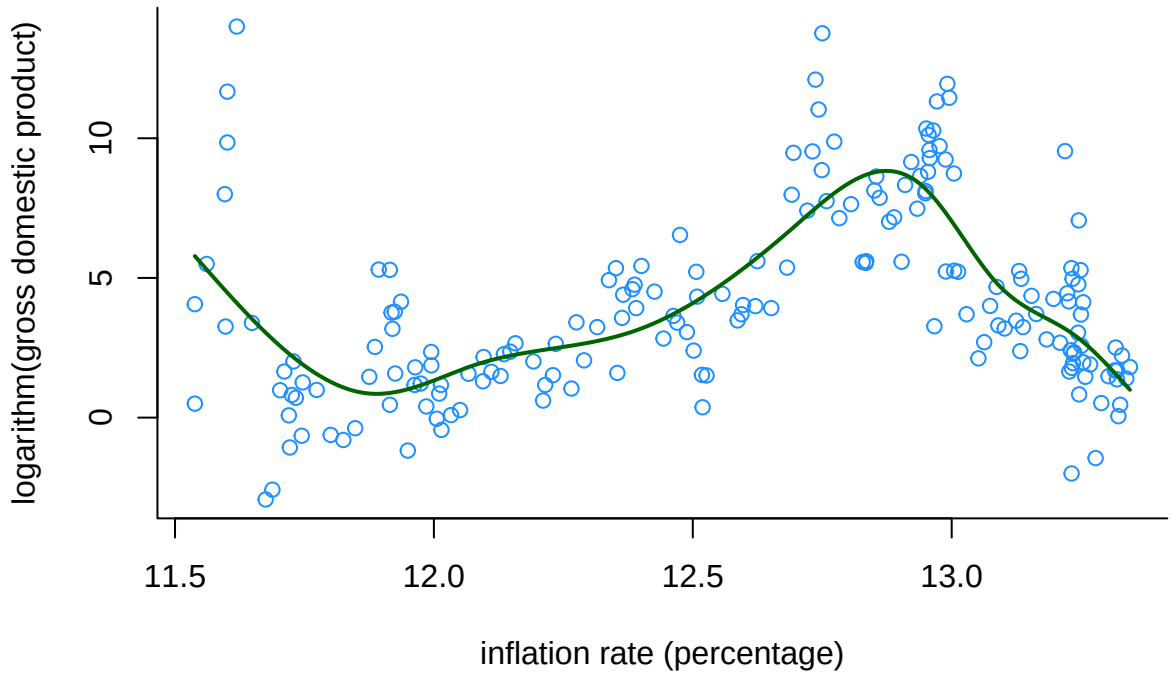
```
library(Ecdat) ; data(Tbrate); library(mgcv)
x <- as.data.frame(Tbrate)$y
y <- as.data.frame(Tbrate)$pi
fitGAMcrChk <- gam(y ~ s(x,bs = "cr"))
gam.check(fitGAMcrChk)
```




```
##
## Method: GCV   Optimizer: magic
```

```
## Smoothing parameter selection converged after 6 iterations.
## The RMS GCV score gradient at convergence was 2.432575e-06 .
## The Hessian was positive definite.
## Model rank =  10 / 10
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##        k'  edf k-index p-value
## s(x) 9.00 7.42    0.61  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

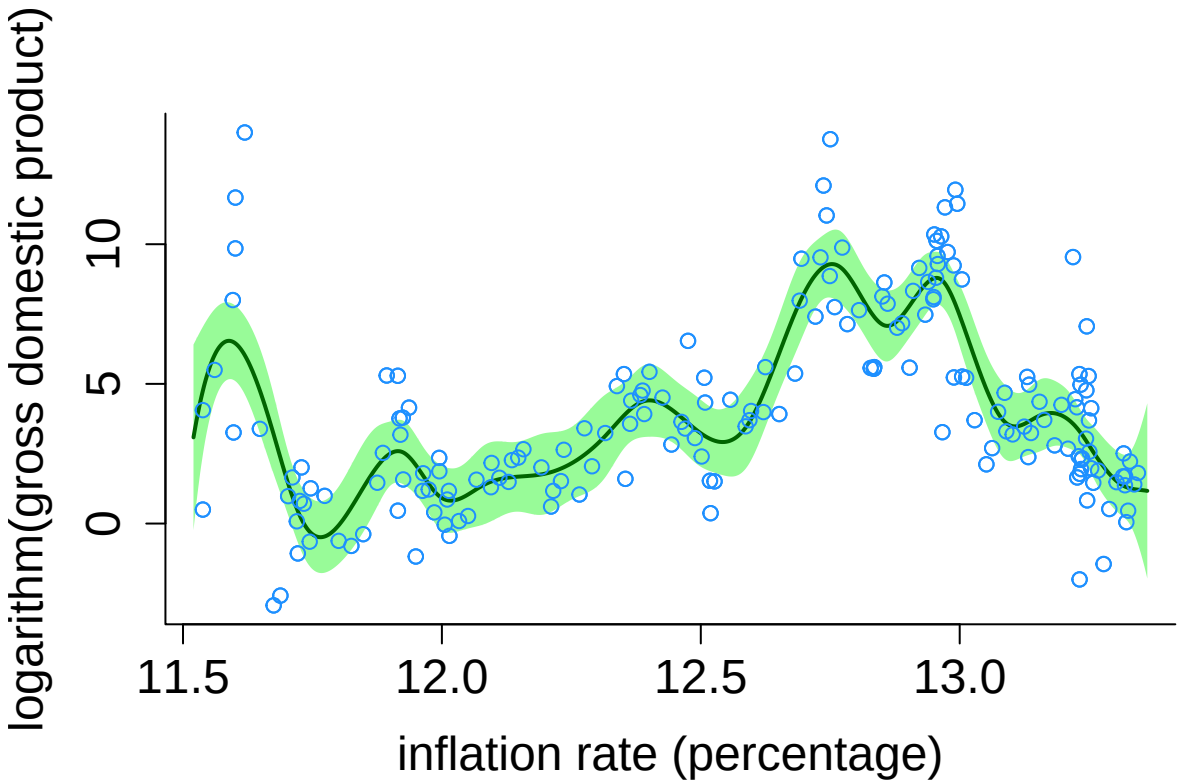
```
fitGAMcr <- gam(y ~ s(x,bs = "cr",k=9))
xg <- seq(min(x),max(x),length = 1001)
fHatgGAMcr <- predict(fitGAMcr,newdata = data.frame(x = xg))
plot(x,y,bty = "l",col = "dodgerblue",xlab = "inflation rate (percentage)",ylab = "logarithm(gross domes
lines(xg,fHatgGAMcr,col = "darkgreen",lwd=2)
```



b

```
library(nlme);library(HRW);library(Ecdat);
x <- as.data.frame(Tbrate)$y
y <- as.data.frame(Tbrate)$pi
numIntKnots <- 23
intKnots <- quantile(unique(x),seq(0,1,length=(numIntKnots+2))[-c(1,(numIntKnots+2))])
```

```
a <- 1.01*min(x) - 0.01*max(x)
b <- 1.01*max(x) - 0.01*min(x)
Z <- ZOSull(x,range.x = c(a,b),intKnots = intKnots)

dummyID <- factor(rep(1,length(x)))
fit <- lme(y ~ x,random = list(dummyID = pdIdent(~-1+Z)))
betaHat <- fit$coef$fixed ; uHat <- unlist(fit$coef$random)
sigsqepsHat <- fit$sigma^2
sigsquHat <- as.numeric(VarCorr(fit)[1,1])
ng <- 1001 ; xg <- seq(a,b,length=ng)
Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,range.x = c(a,b),intKnots = intKnots)
fHatg <- as.vector(Xg%*%betaHat + Zg%*%uHat)
plot(x,y,bty = "l",xlab = "inflation rate (percentage)",ylab = "logarithm(gross domestic product)",col =
Cg <- cbind(rep(1,ng),xg,Zg)
C <- cbind(rep(1,length(y)),x,Z)
D <- diag(c(0,0,rep(1,ncol(Z))))
sdg <- sqrt(sigsqepsHat)*sqrt(diag(Cg%*%solve(crossprod(C)+(sigsqepsHat/sigsquHat)*D,t(Cg))))
CIlowg <- fHatg - 2*sdg ; CIuppg <- fHatg + 2*sdg
polygon(c(xg,rev(xg)),c(CIlowg,rev(CIuppg)),col = "palegreen",border = FALSE)
lines(xg,fHatg,col = "darkgreen",lwd = 2)
points(x,y,col = "dodgerblue")
abline(v = 1980,lty=2,col = "darkorange")
```



c

```
library(HRW)  ;  library(rstan)

library(Ecdat) ; data(Tbrate) ;
xOrig <- as.data.frame(Tbrate)$y
yOrig <- as.data.frame(Tbrate)$pi
mean.x <- mean(xOrig)  ; sd.x <- sd(xOrig)
mean.y <- mean(yOrig)  ; sd.y <- sd(yOrig)
x <- (xOrig - mean.x)/sd.x
y <- (yOrig - mean.y)/sd.y
sigmaBeta <- 1e5 ; Au <- 1e5 ; Aeps <- 1e5

# Obtain linear and spline basis design matrices (X and Z):

X <- cbind(rep(1,length(y)),x)
aOrig <- min(xOrig) ; bOrig <- max(xOrig)
a <- (aOrig - mean.x)/sd.x  ;  b <- (bOrig - mean.x)/sd.x
numIntKnots <- 25
intKnots <-  quantile(unique(x),seq(0,1,length=numIntKnots+2)
                      [-c(1,numIntKnots+2)])
Z <- ZOSull(x,intKnots=intKnots,range.x=c(a,b))
ncZ <- ncol(Z)

# Specify model in Stan:

npRegModel <-
   'data
   {
      int<lower=1> n;          int<lower=1> ncZ;
      vector[n] y;             matrix[n,2] X;
      matrix[n,ncZ] Z;         real<lower=0> sigmaBeta;
      real<lower=0> Au;        real<lower=0> Aeps;
   }
   parameters
   {
      vector[2] beta;           vector[ncZ] u;
      real<lower=0> sigmaeps;   real<lower=0> sigmau;
   }
   model
   {
      y ~ normal(X*beta + Z*u,sigmaeps);
      u ~ normal(0,sigmau); beta ~ normal(0,sigmaBeta);
      sigmaeps ~ cauchy(0,Aeps); sigmau ~ cauchy(0,Au);
   }'

# Store data in a list in format required by Stan:

allData <- list(n=length(x),ncZ=ncZ,y=y,X=X,Z=Z,
                sigmaBeta=sigmaBeta,Au=Au,Aeps=Aeps)

# Set flag for code compilation (needed if
# running script first time in current session) :

compileCode <- TRUE
```

```r
# Compile code for model if required:

if (compileCode)
   stanCompilObj <- stan(model_code=npRegModel,data=allData,
                         iter=1,chains=1)
```

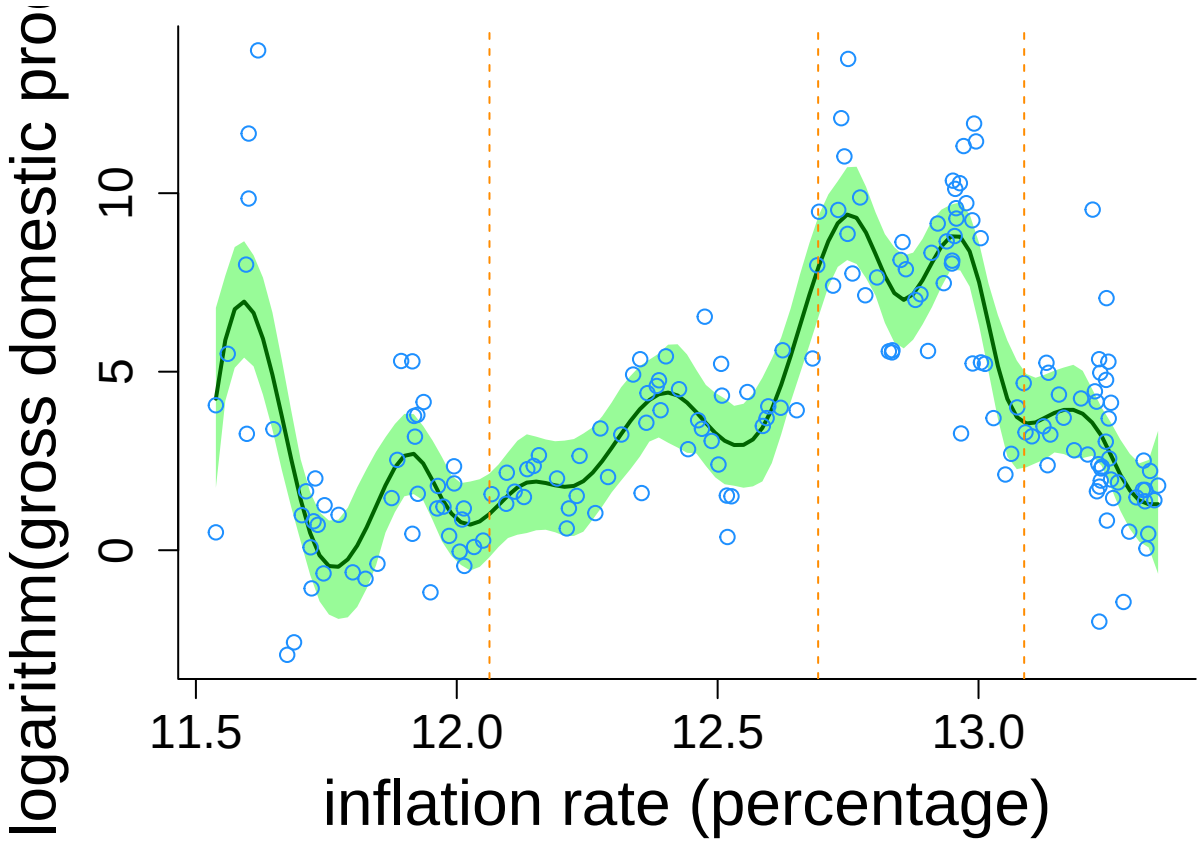
```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.2/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '6722fcaecc9957e96126809b797cdb81' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000102 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.02 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 1 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 1e-06 seconds (Warm-up)
## Chain 1:                8.5e-05 seconds (Sampling)
## Chain 1:                8.6e-05 seconds (Total)
## Chain 1:
```

```r
# Set MCMC sample size parameters:

nWarm <- 1000        # Length of warm-up.
nKept <- 2000        # Size of the kept sample.
nThin <- 2           # Thinning factor.

# Obtain MCMC samples for each parameter using Stan:
```

```r
initFun <- function()
   return(list(sigmau=1,sigmaeps=0.7,beta=rep(0,2),u=rep(0,ncZ)))

stanObj <-  stan(model_code=npRegModel,data=allData,warmup=nWarm,
                 iter=(nWarm+nKept),chains=1,thin=nThin,refresh=100,
                 fit=stanCompilObj,init=initFun,seed=13)
```

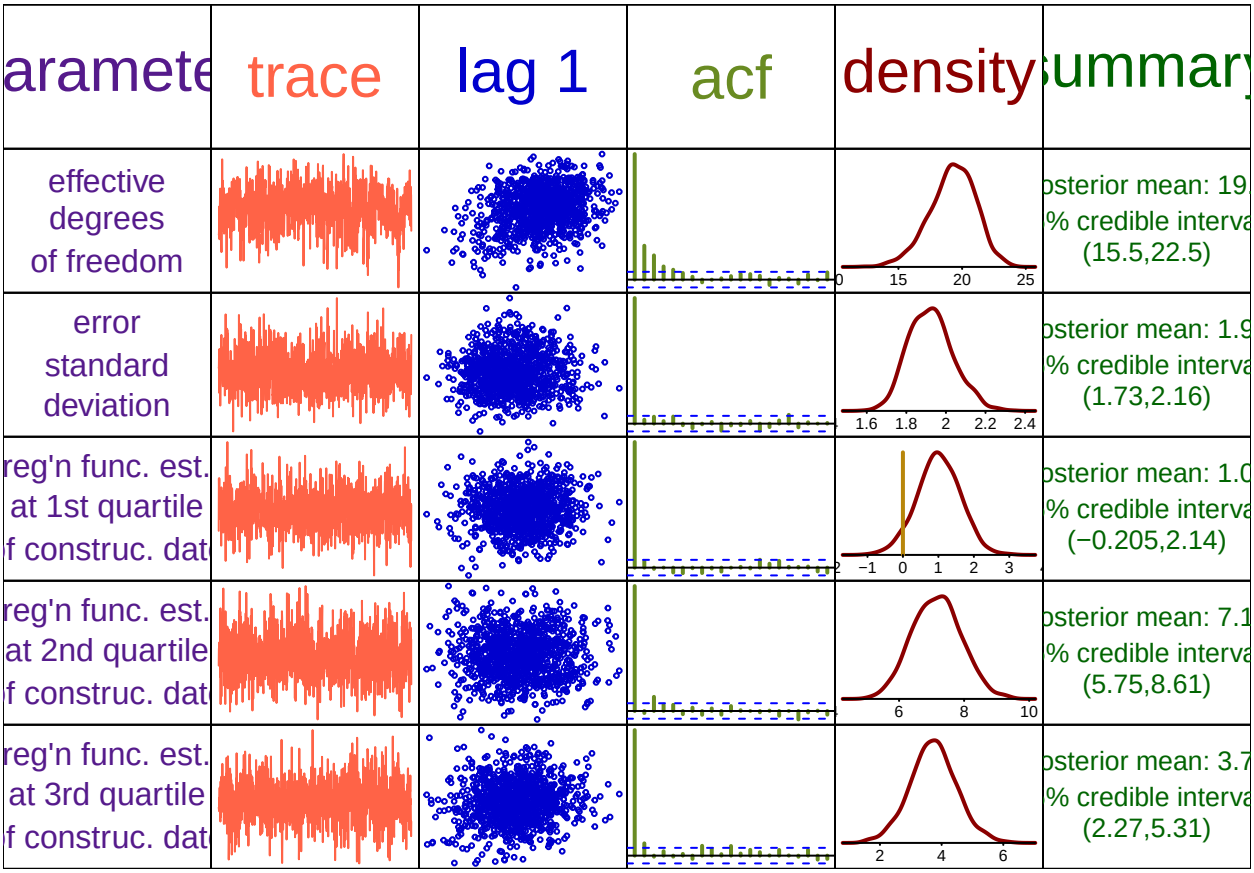
```
##
## SAMPLING FOR MODEL '6722fcaecc9957e96126809b797cdb81' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.38 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 3000 [  3%]  (Warmup)
## Chain 1: Iteration:  200 / 3000 [  6%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 3000 [ 13%]  (Warmup)
## Chain 1: Iteration:  500 / 3000 [ 16%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  700 / 3000 [ 23%]  (Warmup)
## Chain 1: Iteration:  800 / 3000 [ 26%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1100 / 3000 [ 36%]  (Sampling)
## Chain 1: Iteration: 1200 / 3000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1400 / 3000 [ 46%]  (Sampling)
## Chain 1: Iteration: 1500 / 3000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1700 / 3000 [ 56%]  (Sampling)
## Chain 1: Iteration: 1800 / 3000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2000 / 3000 [ 66%]  (Sampling)
## Chain 1: Iteration: 2100 / 3000 [ 70%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2300 / 3000 [ 76%]  (Sampling)
## Chain 1: Iteration: 2400 / 3000 [ 80%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2600 / 3000 [ 86%]  (Sampling)
## Chain 1: Iteration: 2700 / 3000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2900 / 3000 [ 96%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.11464 seconds (Warm-up)
## Chain 1:                0.616428 seconds (Sampling)
## Chain 1:                2.73107 seconds (Total)
## Chain 1:
```

```
# Extract relevant MCMC samples:

betaMCMC <- NULL
for (j in 1:2)
{
   charVar <- paste("beta[",as.character(j),"]",sep="")
   betaMCMC <- rbind(betaMCMC,extract(stanObj,charVar,permuted=FALSE))
}
uMCMC <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("u[",as.character(k),"]",sep="")
   uMCMC <- rbind(uMCMC,extract(stanObj,charVar,permuted=FALSE))
}
sigmaepsMCMC <- as.vector(extract(stanObj,"sigmaeps",permuted=FALSE))
sigmauMCMC <- as.vector(extract(stanObj,"sigmau",permuted=FALSE))

# Obtain MCMC samples of regression curves over a fine grid:

ng <- 101
xgOrig <- seq(aOrig,bOrig,length=ng)
xg <- (xgOrig - mean.x)/sd.x
Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots=intKnots,range.x=c(a,b))
fhatMCMC <- Xg%*%betaMCMC + Zg%*%uMCMC

# Convert fhatMCMC matrix to original scale:

fhatMCMCOrig <- fhatMCMC*sd.y + mean.y
fhatgOrig <- apply(fhatMCMCOrig,1,mean)
credLower <- apply(fhatMCMCOrig,1,quantile,0.025)
credUpper <- apply(fhatMCMCOrig,1,quantile,0.975)

# Display the fit:

par(mai=c(1,1.1,0.1,0.1))
cex.labVal <- 2   ;   cex.axisVal <- 1.5
plot(xOrig,yOrig,type="n",xlab = "inflation rate (percentage)",ylab = "logarithm(gross domestic product)
     bty="l",xlim=range(xgOrig),ylim=range(c(credLower,credUpper,yOrig)),
     cex.lab=cex.labVal,cex.axis=cex.axisVal)
polygon(c(xgOrig,rev(xgOrig)),c(credLower,rev(credUpper)),
        col="palegreen",border=FALSE)
lines(xgOrig,fhatgOrig,col="darkgreen",lwd=2)
points(xOrig,yOrig,col="dodgerblue")
abline(v=quantile(xOrig,0.25),lty=2,col="darkorange")
abline(v=quantile(xOrig,0.50),lty=2,col="darkorange")
abline(v=quantile(xOrig,0.75),lty=2,col="darkorange")
```




```
# Obtain samples from the posterior distribution of the
# effective degrees of freedom:

X <- cbind(rep(1,length(x)),x)
Z <- ZOSull(x,intKnots=intKnots,range.x=c(a,b))
CTC <- crossprod(cbind(X,Z)) ; Dmat <- diag(c(0,0,rep(1,ncol(Z))))
lambdaMCMC <- (sigmaepsMCMC/sigmauMCMC)^2
EDFMCMC <- rep(NA,length(lambdaMCMC))
for (i in 1:length(lambdaMCMC))
   EDFMCMC[i] <- sum(diag(solve(CTC+lambdaMCMC[i]*Dmat,CTC)))

# Convert error standard deviation MCMC sample to the orginal units:

sigmaepsOrigMCMC <- sd.y*sigmaepsMCMC

# Do some summaries and diagnostic checking of the MCMC:

indQ1 <- length(xgOrig[xgOrig<quantile(xOrig,0.25)])
indQ2 <- length(xgOrig[xgOrig<quantile(xOrig,0.50)])
indQ3 <- length(xgOrig[xgOrig<quantile(xOrig,0.75)])
fhatOrigQ1MCMC <- fhatMCMCOrig[indQ1,]
fhatOrigQ2MCMC <- fhatMCMCOrig[indQ2,]
fhatOrigQ3MCMC <- fhatMCMCOrig[indQ3,]
MCMClist <- list(cbind(EDFMCMC,sigmaepsOrigMCMC,
                  fhatOrigQ1MCMC,fhatOrigQ2MCMC,fhatOrigQ3MCMC))
parNamesVal <- list(c("effective","degrees","of freedom"),
```

```
                    c("error","standard","deviation"),
                    c("reg'n func. est.","at 1st quartile","of construc. date"),
                    c("reg'n func. est.","at 2nd quartile","of construc. date"),
                    c("reg'n func. est.","at 3rd quartile","of construc. date"))

summMCMC(MCMClist,parNames=parNamesVal)
```




```
# Obtain chain summaries via the monitor() function:

myMCMCarray <- array(0,dim=c(length(sigmaepsMCMC),1,5))
myMCMCarray[,1,1] <- EDFMCMC
myMCMCarray[,1,2] <- sigmaepsOrigMCMC
myMCMCarray[,1,3] <- fhatOrigQ1MCMC
myMCMCarray[,1,4] <- fhatOrigQ2MCMC
myMCMCarray[,1,5] <- fhatOrigQ3MCMC
monitorAnswer <- monitor(myMCMCarray,warmup=0,print=FALSE)
dimnames(monitorAnswer)[[1]] <- c("EDF","err. st. dev.","f(Q_1)","f(Q_2)","f(Q_3)")
print(signif(monitorAnswer,4))
```

```
##                mean  se_mean     sd    2.5%     25%     50%    75%  97.5%
## EDF          19.250 0.087650 1.7880 15.4700 18.1600 19.3300 20.510 22.540
## err. st. dev.  1.924 0.004023 0.1125  1.7290  1.8440  1.9200  1.993  2.159
## f(Q_1)         1.007 0.020240 0.6100 -0.2053  0.6196  0.9951  1.407  2.141
## f(Q_2)         7.161 0.027410 0.7445  5.7480  6.6440  7.1650  7.649  8.607
## f(Q_3)         3.732 0.026910 0.7519  2.2710  3.2370  3.7320  4.205  5.313
```

```
##               n_eff  Rhat valid       Q5     Q50    Q95 MCSE_Q2.5 MCSE_Q25
## EDF             415 1.002     1 16.08000 19.3300 21.830  0.254500 0.109500
## err. st. dev.   776 1.002     1  1.75500  1.9200  2.129  0.007807 0.005442
## f(Q_1)          908 1.003     1 -0.03073  0.9951  1.988  0.070250 0.026020
## f(Q_2)          736 1.000     1  5.95600  7.1650  8.389  0.041850 0.036450
## f(Q_3)          774 1.005     1  2.47800  3.7320  4.964  0.060680 0.033190
##               MCSE_Q50 MCSE_Q75 MCSE_Q97.5  MCSE_SD Bulk_ESS Tail_ESS
## EDF           0.090100 0.080330   0.140300 0.062020      440      322
## err. st. dev. 0.004188 0.004226   0.007258 0.002847      784      983
## f(Q_1)        0.024180 0.023430   0.081990 0.014780      905      820
## f(Q_2)        0.032250 0.032730   0.041010 0.019720      748      908
## f(Q_3)        0.036040 0.043340   0.058800 0.019320      801      695
```

```
#readline("Hit Enter to continue.\n")

# Obtain multiple chains MCMC samples for each parameter using Stan
# with different initialisations of sigmaeps:

sigmaepsInit <- c(0.7,0.9,1.2)
initFun <- function(chainNum=1)
{
   if(chainNum==1)
   {
      return(list(sigmau=1,sigmaeps=sigmaepsInit[1],beta=rep(0,2),u=rep(0,ncZ)))
   }
   if(chainNum==2)
   {
      return(list(sigmau=1,sigmaeps=sigmaepsInit[2],beta=rep(0,2),u=rep(0,ncZ)))
   }
   if(chainNum==3)
   {
      return(list(sigmau=1,sigmaeps=sigmaepsInit[3],beta=rep(0,2),u=rep(0,ncZ)))
   }
}

numChains <- 3
stanObj <-  stan(model_code=npRegModel,data=allData,
                 warmup=nWarm,iter=(nWarm+nKept),
                 chains=numChains,thin=nThin,
                 refresh=100,fit=stanCompilObj,
                 init=initFun,seed=13)
```

```
##
## SAMPLING FOR MODEL '6722fcaecc9957e96126809b797cdb81' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.36 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 3000 [  3%]  (Warmup)
## Chain 1: Iteration:  200 / 3000 [  6%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
```

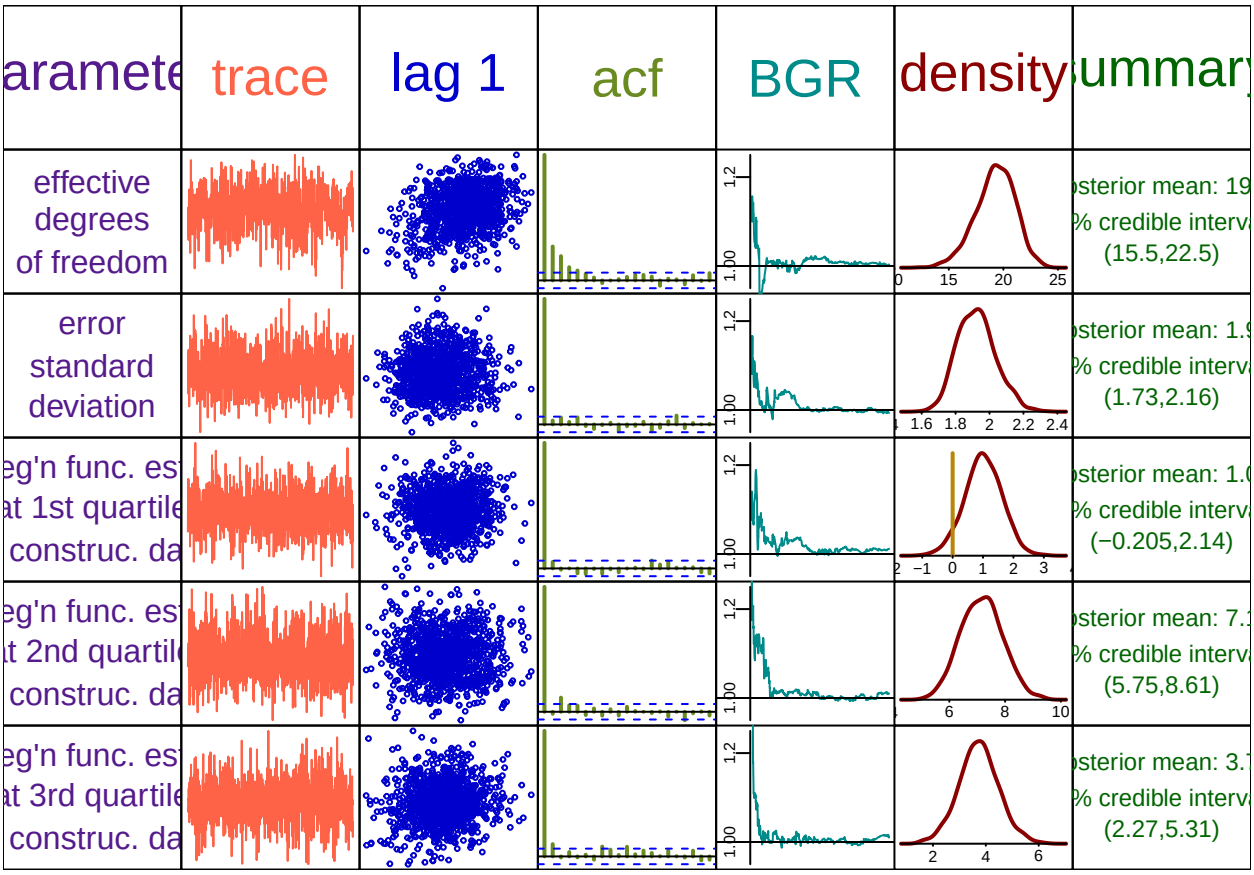
```
## Chain 1: Iteration:  400 / 3000 [ 13%]  (Warmup)
## Chain 1: Iteration:  500 / 3000 [ 16%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  700 / 3000 [ 23%]  (Warmup)
## Chain 1: Iteration:  800 / 3000 [ 26%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1100 / 3000 [ 36%]  (Sampling)
## Chain 1: Iteration: 1200 / 3000 [ 40%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1400 / 3000 [ 46%]  (Sampling)
## Chain 1: Iteration: 1500 / 3000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1700 / 3000 [ 56%]  (Sampling)
## Chain 1: Iteration: 1800 / 3000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2000 / 3000 [ 66%]  (Sampling)
## Chain 1: Iteration: 2100 / 3000 [ 70%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2300 / 3000 [ 76%]  (Sampling)
## Chain 1: Iteration: 2400 / 3000 [ 80%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2600 / 3000 [ 86%]  (Sampling)
## Chain 1: Iteration: 2700 / 3000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2900 / 3000 [ 96%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 2.15711 seconds (Warm-up)
## Chain 1:                0.624253 seconds (Sampling)
## Chain 1:                2.78137 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '6722fcaecc9957e96126809b797cdb81' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.5e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.25 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 2: Iteration:  100 / 3000 [  3%]  (Warmup)
## Chain 2: Iteration:  200 / 3000 [  6%]  (Warmup)
## Chain 2: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 3000 [ 13%]  (Warmup)
## Chain 2: Iteration:  500 / 3000 [ 16%]  (Warmup)
## Chain 2: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  700 / 3000 [ 23%]  (Warmup)
## Chain 2: Iteration:  800 / 3000 [ 26%]  (Warmup)
## Chain 2: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1100 / 3000 [ 36%]  (Sampling)
```

```
## Chain 2: Iteration: 1200 / 3000 [ 40%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1400 / 3000 [ 46%]  (Sampling)
## Chain 2: Iteration: 1500 / 3000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1700 / 3000 [ 56%]  (Sampling)
## Chain 2: Iteration: 1800 / 3000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2000 / 3000 [ 66%]  (Sampling)
## Chain 2: Iteration: 2100 / 3000 [ 70%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2300 / 3000 [ 76%]  (Sampling)
## Chain 2: Iteration: 2400 / 3000 [ 80%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2600 / 3000 [ 86%]  (Sampling)
## Chain 2: Iteration: 2700 / 3000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2900 / 3000 [ 96%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 2.0013 seconds (Warm-up)
## Chain 2:                0.731132 seconds (Sampling)
## Chain 2:                2.73243 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '6722fcaecc9957e96126809b797cdb81' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.27 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 3: Iteration:  100 / 3000 [  3%]  (Warmup)
## Chain 3: Iteration:  200 / 3000 [  6%]  (Warmup)
## Chain 3: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 3000 [ 13%]  (Warmup)
## Chain 3: Iteration:  500 / 3000 [ 16%]  (Warmup)
## Chain 3: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 3: Iteration:  700 / 3000 [ 23%]  (Warmup)
## Chain 3: Iteration:  800 / 3000 [ 26%]  (Warmup)
## Chain 3: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 3: Iteration: 1000 / 3000 [ 33%]  (Warmup)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1100 / 3000 [ 36%]  (Sampling)
## Chain 3: Iteration: 1200 / 3000 [ 40%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1400 / 3000 [ 46%]  (Sampling)
## Chain 3: Iteration: 1500 / 3000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1700 / 3000 [ 56%]  (Sampling)
## Chain 3: Iteration: 1800 / 3000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2000 / 3000 [ 66%]  (Sampling)
```

```
## Chain 3: Iteration: 2100 / 3000 [ 70%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2300 / 3000 [ 76%]  (Sampling)
## Chain 3: Iteration: 2400 / 3000 [ 80%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2600 / 3000 [ 86%]  (Sampling)
## Chain 3: Iteration: 2700 / 3000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2900 / 3000 [ 96%]  (Sampling)
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 2.13229 seconds (Warm-up)
## Chain 3:                0.551206 seconds (Sampling)
## Chain 3:                2.6835 seconds (Total)
## Chain 3:
```

```
sigmaepsOrigMCMC <- vector("list",numChains)
EDFMCMC <- vector("list",numChains)
fhatQ1OrigMCMC <- vector("list",numChains)
fhatQ2OrigMCMC <- vector("list",numChains)
fhatQ3OrigMCMC <- vector("list",numChains)

for (ichn in 1:numChains)
{
   betaMCMC <- NULL
   for (j in 1:2)
   {
      charVar <- paste("beta[",as.character(j),"]",sep="")
      betaMCMC <- rbind(betaMCMC,extract(stanObj,charVar,permuted=FALSE)[,ichn,])
   }

   uMCMC <- NULL
   for (k in 1:ncZ)
   {
      charVar <- paste("u[",as.character(k),"]",sep="")
      uMCMC <- rbind(uMCMC,extract(stanObj,charVar,permuted=FALSE)[,ichn,])
   }

   fhatMCMC <- Xg%*%betaMCMC + Zg%*%uMCMC
   fhatMCMCOrig <- fhatMCMC*sd.y + mean.y

   fhatQ1OrigMCMC[[ichn]] <- fhatMCMCOrig[indQ1,]
   fhatQ2OrigMCMC[[ichn]] <- fhatMCMCOrig[indQ2,]
   fhatQ3OrigMCMC[[ichn]] <- fhatMCMCOrig[indQ3,]

   sigmaepsMCMC <- extract(stanObj,"sigmaeps",permuted=FALSE)[,ichn,]
   sigmauMCMC <- extract(stanObj,"sigmau",permuted=FALSE)[,ichn,]

   lambdaMCMC <- (sigmaepsMCMC/sigmauMCMC)^2
   EDFMCMC[[ichn]] <- rep(NA,length(lambdaMCMC))
   for (i in 1:length(lambdaMCMC))
      EDFMCMC[[ichn]][i] <- sum(diag(solve(CTC+lambdaMCMC[i]*Dmat,CTC)))

   sigmaepsOrigMCMC[[ichn]] <- sd.y*sigmaepsMCMC
```

```
}

MCMClist <- list(chain1=cbind(EDFMCMC[[1]],sigmaepsOrigMCMC[[1]],
                              fhatQ1OrigMCMC[[1]],fhatQ2OrigMCMC[[1]],
                              fhatQ3OrigMCMC[[1]]),
                 chain2=cbind(EDFMCMC[[2]],sigmaepsOrigMCMC[[2]],
                              fhatQ1OrigMCMC[[2]],fhatQ2OrigMCMC[[2]],
                              fhatQ3OrigMCMC[[2]]),
                 chain3=cbind(EDFMCMC[[3]],sigmaepsOrigMCMC[[3]],
                              fhatQ1OrigMCMC[[3]],fhatQ2OrigMCMC[[3]],
                              fhatQ3OrigMCMC[[3]]))


summMCMC(MCMClist,parNames=parNamesVal)
```



##2.8

a

```
n <- 800 ; theta <- 0.5 ; sigmaEps <- 0.2
set.seed(1) ; x <- seq(0,1,length = n);
y <- x + theta*dnorm(x,0.5,0.25) + sigmaEps*rnorm(n)
library(mgcv) ; fitGCV <- gam(y ~ s(x,k = 27))
fitREML <- gam(y ~ s(x,k = 27),method = "REML")
fhatGCV <- fitted(fitGCV) ; fhatREML <- fitted(fitREML)
fTrue <- x + theta*dnorm(x,0.5,0.25)
ASEforGCV <- sum((fhatGCV - fTrue)^2)/n
```

```
ASEforREML <- sum((fhatREML - fTrue)^2)/n
print(c(ASEforGCV,ASEforREML))
```

```
## [1] 0.0001479181 0.0001851418
```

b

```
library(mgcv)
num<-c(30 ,50, 100, 200, 400, 800, 1600, 3200)
theta <- 0.5 ; sigmaEps <- 0.2
ASEnum<-c()
sample<-100
for (j in 1:length(num)){
  n<-num[j]
  ASEforGCV<-c()
  ASEforREML<-c()
  for (i in 1:sample){
    set.seed(i)
    x <- seq(0,1,length = n)
    y <- x + rnorm(200)
    y <- x + theta*dnorm(x,0.5,0.25) + sigmaEps*rnorm(n)
    fitGCV <- gam(y ~ s(x,k = 27))
    fitREML <- gam(y ~ s(x,k = 27),method = "REML")
    fhatGCV <- fitted(fitGCV) ; fhatREML <- fitted(fitREML)
    fTrue <- x + theta*dnorm(x,0.5,0.25)
    ASEforGCV<-c(ASEforGCV,sum((fhatGCV - fTrue)^2)/n)
    ASEforREML<-c(ASEforREML,sum((fhatREML - fTrue)^2)/n)
  }
  ASEnum<-rbind(c(sum(ASEforGCV)/sample,sum(ASEforREML)/sample,n,theta,sigmaEps),ASEnum)
}
print(ASEnum)
```

```
##              [,1]         [,2] [,3] [,4] [,5]
## [1,] 9.777854e-05 0.0001002170 3200  0.5  0.2
## [2,] 1.715133e-04 0.0001728358 1600  0.5  0.2
## [3,] 3.246343e-04 0.0003014563  800  0.5  0.2
## [4,] 6.302637e-04 0.0005547073  400  0.5  0.2
## [5,] 1.123713e-03 0.0010084448  200  0.5  0.2
## [6,] 2.259880e-03 0.0018340298  100  0.5  0.2
## [7,] 4.532657e-03 0.0038742996   50  0.5  0.2
## [8,] 7.983684e-03 0.0061839476   30  0.5  0.2
```

c

```
thetas<-seq(0,1,by=0.1)
sigmaEps <- 0.2 ; n<-800
ASEtheta<-c()
sample<-100
for (j in 1:length(thetas)){
  theta<-thetas[j]
  ASEforGCV<-c()
  ASEforREML<-c()
```

```
  for (i in 1:sample){
    set.seed(i)
    x <- seq(0,1,length = n)
    y <- x + rnorm(200)
    y <- x + theta*dnorm(x,0.5,0.25) + sigmaEps*rnorm(n)
    fitGCV <- gam(y ~ s(x,k = 27))
    fitREML <- gam(y ~ s(x,k = 27),method = "REML")
    fhatGCV <- fitted(fitGCV) ; fhatREML <- fitted(fitREML)
    fTrue <- x + theta*dnorm(x,0.5,0.25)
    ASEforGCV<-c(ASEforGCV,sum((fhatGCV - fTrue)^2)/n)
    ASEforREML<-c(ASEforREML,sum((fhatREML - fTrue)^2)/n)
  }
  ASEtheta<-rbind(c(sum(ASEforGCV)/sample,sum(ASEforREML)/sample,n,theta,sigmaEps),ASEtheta)
}
```

d

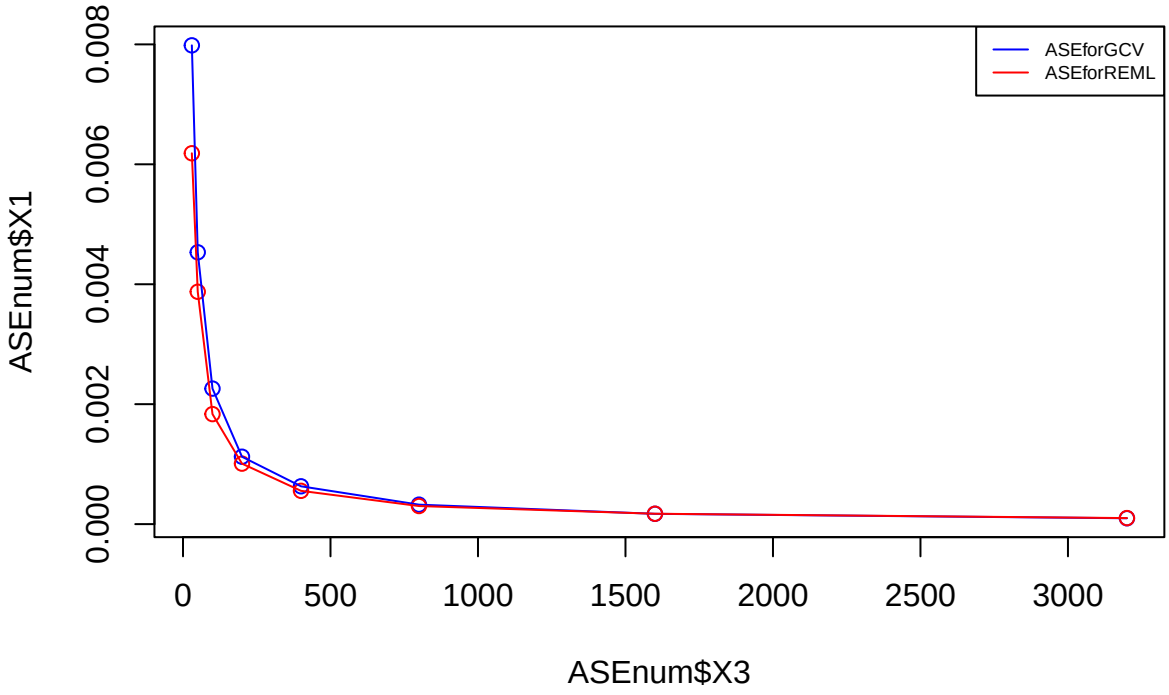
```
theta<-0.5; n<-800
sigmaEpsl <- seq(0.1,1,by=0.1)
ASEsigma<-c()
sample<-100
for (j in 1:length(sigmaEpsl)){
  sigmaEps<-sigmaEpsl[j]
  ASEforGCV<-c()
  ASEforREML<-c()
  for (i in 1:sample){
    set.seed(i)
    x <- seq(0,1,length = n)
    y <- x + rnorm(200)
    y <- x + theta*dnorm(x,0.5,0.25) + sigmaEps*rnorm(n)
    fitGCV <- gam(y ~ s(x,k = 27))
    fitREML <- gam(y ~ s(x,k = 27),method = "REML")
    fhatGCV <- fitted(fitGCV) ; fhatREML <- fitted(fitREML)
    fTrue <- x + theta*dnorm(x,0.5,0.25)
    ASEforGCV<-c(ASEforGCV,sum((fhatGCV - fTrue)^2)/n)
    ASEforREML<-c(ASEforREML,sum((fhatREML - fTrue)^2)/n)
  }
   ASEsigma<-rbind(c(mean(ASEforGCV),mean(ASEforREML),n,theta,sigmaEps),ASEsigma)
}
```

e

```
library(summarytools)
print("X1=ASEforGCV,X2=ASEforREML,X3=num,X4=theta,X4=sigmaeps")
```

```
## [1] "X1=ASEforGCV,X2=ASEforREML,X3=num,X4=theta,X4=sigmaeps"
```

```
ASEnum<-data.frame(ASEnum)
plot(ASEnum$X3,ASEnum$X1,col="blue")
points(ASEnum$X3,ASEnum$X2,col="red")
lines(ASEnum$X3,ASEnum$X1,col="blue")
lines(ASEnum$X3,ASEnum$X2,col="red")
legend('topright',legend=c("ASEforGCV","ASEforREML"),col=c("blue","red"),lty=1, cex=0.6)
```

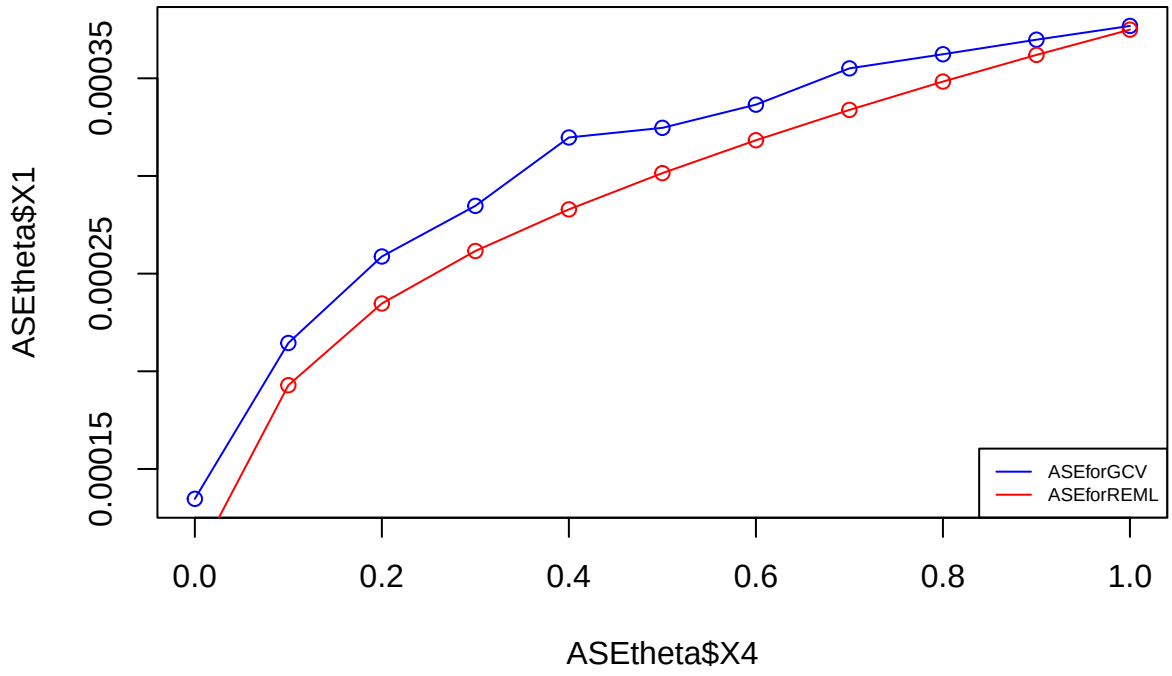



```
dfSummary(ASEnum)
```

```
## Data Frame Summary
## ASEnum
## Dimensions: 8 x 5
## Duplicates: 0
##
## ------------------------------------------------------------------------------------------------------------
## No   Variable    Stats / Values               Freqs (% of Valid)   Graph                 Valid
## ---- ----------- ---------------------------- -------------------- --------------------- ----------
## 1    X1          Mean (sd) : 0 (0)            8 distinct values    II                    8
##      [numeric]   min < med < max:                                   II                    (100.0%)
##                  0 < 0 < 0                                        II
##                  IQR (CV) : 0 (1.3)                               II
##                                                                   II
##                                                                   II
##                                                                   II
##                                                                   II
##
## 2    X2          Mean (sd) : 0 (0)            8 distinct values    II                    8
##      [numeric]   min < med < max:                                   II                    (100.0%)
##                  0 < 0 < 0                                        II
##                  IQR (CV) : 0 (1.2)                               II
##                                                                   II
##                                                                   II
```

```
##                                                                  II
##                                                                  II
##
## 3    X3          Mean (sd) : 797.5 (1106.2)   30 : 1 (12.5%)     II                    8
##      [numeric]   min < med < max:             50 : 1 (12.5%)     II                    (100.0%)
##                  30 < 300 < 3200              100 : 1 (12.5%)    II
##                  IQR (CV) : 912.5 (1.4)       200 : 1 (12.5%)    II
##                                               400 : 1 (12.5%)    II
##                                               800 : 1 (12.5%)    II
##                                               1600 : 1 (12.5%)   II
##                                               3200 : 1 (12.5%)   II
##
## 4    X4          1 distinct value             0.50 : 8 (100.0%)  IIIIIIIIIIIIIIIIIII   8
##      [numeric]                                                                          (100.0%)
##
## 5    X5          1 distinct value             0.20 : 8 (100.0%)  IIIIIIIIIIIIIIIIIII   8
##      [numeric]                                                                          (100.0%)
## ---------------------------------------------------------------------------------------------------------------
```

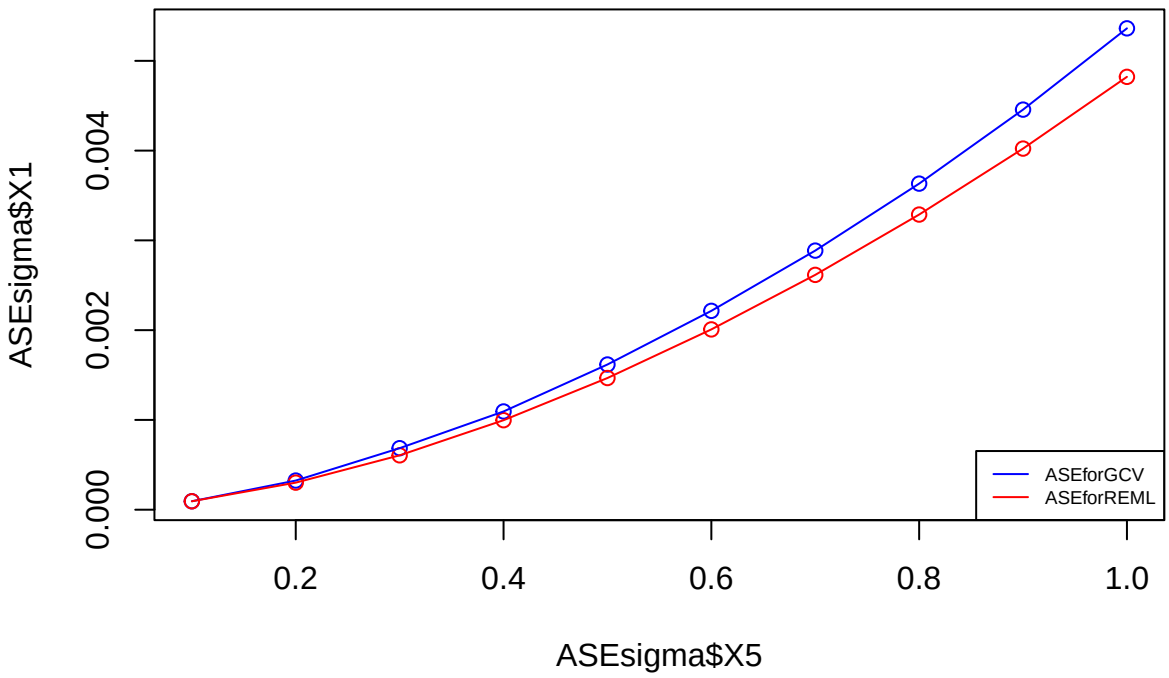
```
ASEtheta<-data.frame(ASEtheta)
plot(ASEtheta$X4,ASEtheta$X1,col="blue")
points(ASEtheta$X4,ASEtheta$X2,col="red")
lines(ASEtheta$X4,ASEtheta$X1,col="blue")
lines(ASEtheta$X4,ASEtheta$X2,col="red")
legend('bottomright',legend=c("ASEforGCV","ASEforREML"),col=c("blue","red"),lty=1, cex=0.6)
```



```
dfSummary(ASEtheta)
```

```
## Data Frame Summary
## ASEtheta
## Dimensions: 11 x 5
## Duplicates: 0
##
## ------------------------------------------------------------------------------------------------------------
## No   Variable    Stats / Values          Freqs (% of Valid)   Graph                  Valid      Miss
## ---- ----------- ---------------------- -------------------- ---------------------- ---------- -----
## 1    X1          Mean (sd) : 0 (0)       11 distinct values            :           11         0
##      [numeric]   min < med < max:                                      : :          (100.0%)   (0.0
##                  0 < 0 < 0                                        . : :
##                  IQR (CV) : 0 (0.2)                               : : :
##                                                             :    : : : :
##
## 2    X2          Mean (sd) : 0 (0)       11 distinct values            :           11         0
##      [numeric]   min < med < max:                                      :           (100.0%)   (0.0
##                  0 < 0 < 0                                        . : .
##                  IQR (CV) : 0 (0.3)                               : : :
##                                                             : : : : : :
##
## 3    X3          1 distinct value        800 : 11 (100.0%)    IIIIIIIIIIIIIIIIIII    11         0
##      [numeric]                                                                      (100.0%)   (0.0
##
## 4    X4          Mean (sd) : 0.5 (0.3)   11 distinct values   :                      11         0
##      [numeric]   min < med < max:                             : . . . .              (100.0%)   (0.0
##                  0 < 0.5 < 1                                  : : : : :
##                  IQR (CV) : 0.5 (0.7)                         : : : : :
##                                                               : : : : :
##
## 5    X5          1 distinct value        0.20 : 11 (100.0%)   IIIIIIIIIIIIIIIIIII    11         0
##      [numeric]                                                                      (100.0%)   (0.0
## ------------------------------------------------------------------------------------------------------------
```

```
ASEsigma<-data.frame(ASEsigma)
plot(ASEsigma$X5,ASEsigma$X1,col="blue")
points(ASEsigma$X5,ASEsigma$X2,col="red")
lines(ASEsigma$X5,ASEsigma$X1,col="blue")
lines(ASEsigma$X5,ASEsigma$X2,col="red")
legend('bottomright',legend=c("ASEforGCV","ASEforREML"),col=c("blue","red"),lty=1, cex=0.6)
```




```
dfSummary(ASEsigma)
```

```
## Data Frame Summary
## ASEsigma
## Dimensions: 10 x 5
## Duplicates: 0
##
## ------------------------------------------------------------------------------------------------------------
## No   Variable    Stats / Values          Freqs (% of Valid)   Graph                 Valid      Miss
## ---- ----------- ---------------------- -------------------- ---------------------- ---------- -----
## 1    X1          Mean (sd) : 0 (0)       10 distinct values   II                    10         0
##      [numeric]   min < med < max:                            II                    (100.0%)   (0.0
##                  0 < 0 < 0                                  II
##                  IQR (CV) : 0 (0.8)                         II
##                                                             II
##                                                             II
##                                                             II
##                                                             II
##                                                             II
##                                                             II
##
## 2    X2          Mean (sd) : 0 (0)       10 distinct values   II                    10         0
##      [numeric]   min < med < max:                            II                    (100.0%)   (0.0
##                  0 < 0 < 0                                  II
##                  IQR (CV) : 0 (0.8)                         II
```

```
##                                                                  II
##                                                                  II
##                                                                  II
##                                                                  II
##                                                                  II
##                                                                  II
##
## 3    X3          1 distinct value        800 : 10 (100.0%)    IIIIIIIIIIIIIIIIIII   10         0
##      [numeric]                                                                      (100.0%)   (0.0%
##
## 4    X4          1 distinct value        0.50 : 10 (100.0%)   IIIIIIIIIIIIIIIIIII   10         0
##      [numeric]                                                                      (100.0%)   (0.0%
##
## 5    X5          Mean (sd) : 0.6 (0.3)   0.10 : 1 (10.0%)     II                    10         0
##      [numeric]   min < med < max:        0.20 : 1 (10.0%)     II                    (100.0%)   (0.0%
##                  0.1 < 0.6 < 1           0.30 : 1 (10.0%)     II
##                  IQR (CV) : 0.5 (0.6)    0.40 : 1 (10.0%)     II
##                                          0.50 : 1 (10.0%)     II
##                                          0.60 : 1 (10.0%)     II
##                                          0.70 : 1 (10.0%)     II
##                                          0.80 : 1 (10.0%)     II
##                                          0.90 : 1 (10.0%)     II
##                                          1.00 : 1 (10.0%)     II
## ------------------------------------------------------------------------------------------------------
```

As we increase the number of samples, the mean square error decreases exponentially for both the model based ASE. They both follow a similar trend. although for lower sample size GCV shows a higher ASE loss than REML based fit.

When we wary theta , we can draw the following observation. The GCV based ASE gives a higher value than REML based ASE for all thetas. Moreover the generale trend is that as theta increases so does the ASE for both the model fits.

When we vary sigma we can draw the following conclusions:Again the GCV based ASE provides a higher value than REML based for all sigma. THis means that REML based are better at modelling. We can also see that the ASE increases parabolically as the sigma increases.

3

a

GCV smoothness can often suffer from underfitting issues compared to other. Cannot account easily for missing data and heteroscedasticity. GCV is computationally faster

REML or mixed model based smoothing are computationally slower. Cannot account easily for missing data and heteroscedasticity. They provide much more robustness in smoothing for GAM. As seen above the ASE is also comparatively lower for REML

bayesian based model can be very complicated to code and implement. Speaking from experience Bayesian models can account for missing data

b

for Bayesian penalized splines , a non linear time series data which has groupings and also some missing data. They would be the best case scenario to model such a data. This can be best used when there are

gaps in data collection in certain years due to varied reasons.

Mixed model based penalized spline would be better used for data which has grouping in it. For example a time series data of some kind grouped according to a certain geographical area.

GCV based penalized splines would work best when we have abundance of data and we dont have much risk to overfitting. Datasets with unpenalized parametric terms like categorical predictors would be best suited for this approach.

Normal GCV penalized spline