HW5

ASG

2023-03-22

2.9

a

Mokotow

```
compileCode <- TRUE

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

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
# Set MCMC parameters:

nWarm <- 1000            # Length of burn-in.
nKept <- 1000            # Size of the kept sample.
nThin <- 1               # Thinning factor.

# Load Warsaw apartment data:

data(WarsawApts)

# Set graphics parameters:

cex.axisVal <- 1.8 ; cex.labVal <- 1.8
cex.legendVal <- 1.5 ; lwdVal <- 2

# Standardize both predictor and response variable
# and set hyperparameter values:
```

```
xOrig <- WarsawApts$construction.date
yOrig <- WarsawApts$areaPerMzloty
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
treatIndic <- as.numeric(as.character(WarsawApts$district) != "Mokotow")

sigmaBeta <- 1e5 ; Au <- 1e5 ; Aeps <- 1e5

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

X <- cbind(rep(1,length(y)),x,treatIndic,treatIndic*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))

# Set dimension variables:

n <- length(y)
ncZ <- ncol(Z)

# Specify model in Stan:

WarsawAptSimpFacByCurvModel <-
   'data
   {
      int<lower=1> n;           int<lower=1> ncZ;
      vector[n] y;              matrix[n,4] X;
      matrix[n,ncZ] Z ;         vector[n] treatIndic;
      real<lower=0> sigmaBeta;
      real<lower=0> Aeps;       real<lower=0> Au;
   }
   parameters
   {
      vector[4] beta;
      vector[ncZ] uControl;           vector[ncZ] uTreatmt;
      real<lower=0> sigmaUcontrol;    real<lower=0> sigmaUtreatmt;
      real<lower=0> sigmaEps;
   }
   model
   {
      for (i in 1:n)
         y[i] ~ normal((dot_product(beta,X[i])
                        + dot_product(uControl,((1-treatIndic[i])*Z[i]))
                        + dot_product(uTreatmt,(treatIndic[i]*Z[i]))),sigmaEps);
      uControl ~ normal(0,sigmaUcontrol) ; uTreatmt ~ normal(0,sigmaUtreatmt);
      beta ~ normal(0,sigmaBeta);
      sigmaEps ~ cauchy(0,Aeps);
      sigmaUcontrol ~ cauchy(0,Au); sigmaUtreatmt ~ cauchy(0,Au);
```

```
   }'

# Set up input data:

allData <- list(n = n,ncZ = ncZ,y = y,X = X,Z = Z,treatIndic = treatIndic,
                sigmaBeta = 1e5,Aeps = 1e5,Au = 1e5)

# Compile code for model if required:

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

```
## Trying to compile a simple C file

## 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 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000644 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 6.44 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:                0.000955 seconds (Sampling)
## Chain 1:                0.000956 seconds (Total)
## Chain 1:
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Perform MCMC:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000466 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.66 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 35.0823 seconds (Warm-up)
## Chain 1:                11.8869 seconds (Sampling)
## Chain 1:                46.9692 seconds (Total)
## Chain 1:
```

```
## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
# Extract relevant MCMC sampes:

betaMCMCM <- NULL
for (j in 1:4)
{
   charVar <- paste("beta[",as.character(j),"]",sep = "")
   betaMCMCM <- rbind(betaMCMCM,extract(stanObj,charVar,permuted = FALSE))
}
uControlMCMCM <- NULL  ;  uTreatmtMCMCM <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uControl[",as.character(k),"]",sep = "")
   uControlMCMCM <- rbind(uControlMCMCM,extract(stanObj,charVar,permuted = FALSE))

   charVar <- paste("uTreatmt[",as.character(k),"]",sep = "")
   uTreatmtMCMCM <- rbind(uTreatmtMCMCM,extract(stanObj,charVar,permuted = FALSE))
}
sigmaEpsMCMCM <- as.vector(extract(stanObj,"sigmaEps",permuted = FALSE))
sigmaUcontrolMCMCM <- as.vector(extract(stanObj,"sigmaUcontrol",permuted = FALSE))
sigmaUtreatmtMCMCM <- as.vector(extract(stanObj,"sigmaUtreatmt",permuted = FALSE))

# Plot data and fitted curves:

ng <- 201
cex.labVal <- 1.8
xLow <- min(x) ; xUpp <- max(x)
xg <- seq(xLow,xUpp,length = ng)

Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots = intKnots,range.x = c(a,b))



fhatCntrlMCMCM <- Xg%*%betaMCMCM[c(1,2),] + Zg%*%uControlMCMCM


fCntrlgM <- apply(fhatCntrlMCMCM,1,mean)
credLowCntrlM <- apply(fhatCntrlMCMCM,1,quantile,0.025)
credUppCntrlM <- apply(fhatCntrlMCMCM,1,quantile,0.975)

x <- (xOrig - mean.x)/sd.x

# Convert grids to original units:

xgOrig <- sd.x*xg + mean.x
```

```
fCntrlgOrigM <- sd.y*fCntrlgM + mean.y
credLowCntrlOrigM <- sd.y*credLowCntrlM + mean.y
credUppCntrlOrigM <- sd.y*credUppCntrlM + mean.y
```

Wola

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

# Set MCMC parameters:

nWarm <- 1000            # Length of burn-in.
nKept <- 1000            # Size of the kept sample.
nThin <- 1               # Thinning factor.

# Load Warsaw apartment data:

data(WarsawApts)

# Set graphics parameters:

cex.axisVal <- 1.8 ; cex.labVal <- 1.8
cex.legendVal <- 1.5 ; lwdVal <- 2

# Standardize both predictor and response variable
# and set hyperparameter values:

xOrig <- WarsawApts$construction.date
yOrig <- WarsawApts$areaPerMzloty
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
treatIndic <- as.numeric(as.character(WarsawApts$district) != "Wola")

sigmaBeta <- 1e5 ; Au <- 1e5 ; Aeps <- 1e5

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

X <- cbind(rep(1,length(y)),x,treatIndic,treatIndic*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))

# Set dimension variables:

n <- length(y)
ncZ <- ncol(Z)

# Specify model in Stan:
```

```
WarsawAptSimpFacByCurvModel <-
   'data
   {
      int<lower=1> n;           int<lower=1> ncZ;
      vector[n] y;              matrix[n,4] X;
      matrix[n,ncZ] Z ;         vector[n] treatIndic;
      real<lower=0> sigmaBeta;
      real<lower=0> Aeps;       real<lower=0> Au;
   }
   parameters
   {
      vector[4] beta;
      vector[ncZ] uControl;           vector[ncZ] uTreatmt;
      real<lower=0> sigmaUcontrol;    real<lower=0> sigmaUtreatmt;
      real<lower=0> sigmaEps;
   }
   model
   {
      for (i in 1:n)
         y[i] ~ normal((dot_product(beta,X[i])
                      + dot_product(uControl,((1-treatIndic[i])*Z[i]))
                      + dot_product(uTreatmt,(treatIndic[i]*Z[i]))),sigmaEps);
      uControl ~ normal(0,sigmaUcontrol) ; uTreatmt ~ normal(0,sigmaUtreatmt);
      beta ~ normal(0,sigmaBeta);
      sigmaEps ~ cauchy(0,Aeps);
      sigmaUcontrol ~ cauchy(0,Au); sigmaUtreatmt ~ cauchy(0,Au);
   }'

# Set up input data:

allData <- list(n = n,ncZ = ncZ,y = y,X = X,Z = Z,treatIndic = treatIndic,
                sigmaBeta = 1e5,Aeps = 1e5,Au = 1e5)

# Compile code for model if required:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000467 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.67 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: 0 seconds (Warm-up)
```

```
## Chain 1:                0.00094 seconds (Sampling)
## Chain 1:                0.00094 seconds (Total)
## Chain 1:

## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Perform MCMC:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000468 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.68 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 66.5123 seconds (Warm-up)
## Chain 1:                32.1865 seconds (Sampling)
## Chain 1:                98.6988 seconds (Total)
## Chain 1:

## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. S
```

```
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.07, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
# Extract relevant MCMC sampes:

betaMCMCW <- NULL
for (j in 1:4)
{
   charVar <- paste("beta[",as.character(j),"]",sep = "")
   betaMCMCW <- rbind(betaMCMCW,extract(stanObj,charVar,permuted = FALSE))
}
uControlMCMCW <- NULL  ;  uTreatmtMCMCW <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uControl[",as.character(k),"]",sep = "")
   uControlMCMCW <- rbind(uControlMCMCW,extract(stanObj,charVar,permuted = FALSE))

   charVar <- paste("uTreatmt[",as.character(k),"]",sep = "")
   uTreatmtMCMCW <- rbind(uTreatmtMCMCW,extract(stanObj,charVar,permuted = FALSE))
}
sigmaEpsMCMCW <- as.vector(extract(stanObj,"sigmaEps",permuted = FALSE))
sigmaUcontrolMCMCW <- as.vector(extract(stanObj,"sigmaUcontrol",permuted = FALSE))
sigmaUtreatmtMCMCW <- as.vector(extract(stanObj,"sigmaUtreatmt",permuted = FALSE))

# Plot data and fitted curves:

ng <- 201
cex.labVal <- 1.8
xLow <- min(x) ; xUpp <- max(x)
xg <- seq(xLow,xUpp,length = ng)

Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots = intKnots,range.x = c(a,b))



fhatCntrlMCMCW <- Xg%*%betaMCMCW[c(1,2),] + Zg%*%uControlMCMCW


fCntrlgW <- apply(fhatCntrlMCMCW,1,mean)
```

```
credLowCntrlW <- apply(fhatCntrlMCMCW,1,quantile,0.025)
credUppCntrlW <- apply(fhatCntrlMCMCW,1,quantile,0.975)

x <- (xOrig - mean.x)/sd.x

# Convert grids to original units:

xgOrig <- sd.x*xg + mean.x

fCntrlgOrigW <- sd.y*fCntrlgW + mean.y
credLowCntrlOrigW <- sd.y*credLowCntrlW + mean.y
credUppCntrlOrigW <- sd.y*credUppCntrlW + mean.y
```

Zoliborz

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

# Set MCMC parameters:

nWarm <- 1000            # Length of burn-in.
nKept <- 1000            # Size of the kept sample.
nThin <- 1               # Thinning factor.

# Load Warsaw apartment data:

data(WarsawApts)

# Set graphics parameters:

cex.axisVal <- 1.8 ; cex.labVal <- 1.8
cex.legendVal <- 1.5 ; lwdVal <- 2

# Standardize both predictor and response variable
# and set hyperparameter values:

xOrig <- WarsawApts$construction.date
yOrig <- WarsawApts$areaPerMzloty
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
treatIndic <- as.numeric(as.character(WarsawApts$district) != "Zoliborz")

sigmaBeta <- 1e5 ; Au <- 1e5 ; Aeps <- 1e5

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

X <- cbind(rep(1,length(y)),x,treatIndic,treatIndic*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))

# Set dimension variables:

n <- length(y)
ncZ <- ncol(Z)

# Specify model in Stan:

WarsawAptSimpFacByCurvModel <-
   'data
   {
      int<lower=1> n;           int<lower=1> ncZ;
      vector[n] y;              matrix[n,4] X;
      matrix[n,ncZ] Z ;         vector[n] treatIndic;
      real<lower=0> sigmaBeta;
      real<lower=0> Aeps;       real<lower=0> Au;
   }
   parameters
   {
      vector[4] beta;
      vector[ncZ] uControl;           vector[ncZ] uTreatmt;
      real<lower=0> sigmaUcontrol;    real<lower=0> sigmaUtreatmt;
      real<lower=0> sigmaEps;
   }
   model
   {
      for (i in 1:n)
         y[i] ~ normal((dot_product(beta,X[i])
                        + dot_product(uControl,((1-treatIndic[i])*Z[i]))
                        + dot_product(uTreatmt,(treatIndic[i]*Z[i]))),sigmaEps);
      uControl ~ normal(0,sigmaUcontrol) ; uTreatmt ~ normal(0,sigmaUtreatmt);
      beta ~ normal(0,sigmaBeta);
      sigmaEps ~ cauchy(0,Aeps);
      sigmaUcontrol ~ cauchy(0,Au); sigmaUtreatmt ~ cauchy(0,Au);
   }'

# Set up input data:

allData <- list(n = n,ncZ = ncZ,y = y,X = X,Z = Z,treatIndic = treatIndic,
                sigmaBeta = 1e5,Aeps = 1e5,Au = 1e5)

# Compile code for model if required:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
```

```
## Chain 1: Gradient evaluation took 0.000472 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.72 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: 0 seconds (Warm-up)
## Chain 1:                0.000948 seconds (Sampling)
## Chain 1:                0.000948 seconds (Total)
## Chain 1:
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Perform MCMC:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000468 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.68 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
```

```
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 43.6157 seconds (Warm-up)
## Chain 1:                15.7841 seconds (Sampling)
## Chain 1:                59.3999 seconds (Total)
## Chain 1:

## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.05, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```r
# Extract relevant MCMC sampes:

betaMCMCZ <- NULL
for (j in 1:4)
{
   charVar <- paste("beta[",as.character(j),"]",sep = "")
   betaMCMCZ <- rbind(betaMCMCZ,extract(stanObj,charVar,permuted = FALSE))
}
uControlMCMCZ <- NULL  ;  uTreatmtMCMCZ <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uControl[",as.character(k),"]",sep = "")
   uControlMCMCZ <- rbind(uControlMCMCZ,extract(stanObj,charVar,permuted = FALSE))

   charVar <- paste("uTreatmt[",as.character(k),"]",sep = "")
   uTreatmtMCMCZ <- rbind(uTreatmtMCMCZ,extract(stanObj,charVar,permuted = FALSE))
}
sigmaEpsMCMCZ <- as.vector(extract(stanObj,"sigmaEps",permuted = FALSE))
sigmaUcontrolMCMCZ <- as.vector(extract(stanObj,"sigmaUcontrol",permuted = FALSE))
sigmaUtreatmtMCMCZ <- as.vector(extract(stanObj,"sigmaUtreatmt",permuted = FALSE))

# Plot data and fitted curves:

ng <- 201
cex.labVal <- 1.8
xLow <- min(x) ; xUpp <- max(x)
```

```
xg <- seq(xLow,xUpp,length = ng)

Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots = intKnots,range.x = c(a,b))



fhatCntrlMCMCZ <- Xg%*%betaMCMCZ[c(1,2),] + Zg%*%uControlMCMCZ


fCntrlgZ <- apply(fhatCntrlMCMCZ,1,mean)
credLowCntrlZ <- apply(fhatCntrlMCMCZ,1,quantile,0.025)
credUppCntrlZ <- apply(fhatCntrlMCMCZ,1,quantile,0.975)

x <- (xOrig - mean.x)/sd.x

# Convert grids to original units:

xgOrig <- sd.x*xg + mean.x

fCntrlgOrigZ <- sd.y*fCntrlgZ + mean.y
credLowCntrlOrigZ <- sd.y*credLowCntrlZ + mean.y
credUppCntrlOrigZ <- sd.y*credUppCntrlZ + mean.y
```

Srodmiescie

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

# Set MCMC parameters:

nWarm <- 1000            # Length of burn-in.
nKept <- 1000            # Size of the kept sample.
nThin <- 1               # Thinning factor.

# Load Warsaw apartment data:

data(WarsawApts)

# Set graphics parameters:

cex.axisVal <- 1.8 ; cex.labVal <- 1.8
cex.legendVal <- 1.5 ; lwdVal <- 2

# Standardize both predictor and response variable
# and set hyperparameter values:

xOrig <- WarsawApts$construction.date
yOrig <- WarsawApts$areaPerMzloty
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
treatIndic <- as.numeric(as.character(WarsawApts$district) != "Srodmiescie")

sigmaBeta <- 1e5 ; Au <- 1e5 ; Aeps <- 1e5

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

X <- cbind(rep(1,length(y)),x,treatIndic,treatIndic*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))

# Set dimension variables:

n <- length(y)
ncZ <- ncol(Z)

# Specify model in Stan:

WarsawAptSimpFacByCurvModel <-
   'data
   {
      int<lower=1> n;           int<lower=1> ncZ;
      vector[n] y;              matrix[n,4] X;
      matrix[n,ncZ] Z ;         vector[n] treatIndic;
      real<lower=0> sigmaBeta;
      real<lower=0> Aeps;       real<lower=0> Au;
   }
   parameters
   {
      vector[4] beta;
      vector[ncZ] uControl;           vector[ncZ] uTreatmt;
      real<lower=0> sigmaUcontrol;    real<lower=0> sigmaUtreatmt;
      real<lower=0> sigmaEps;
   }
   model
   {
      for (i in 1:n)
         y[i] ~ normal((dot_product(beta,X[i])
                        + dot_product(uControl,((1-treatIndic[i])*Z[i]))
                        + dot_product(uTreatmt,(treatIndic[i]*Z[i]))),sigmaEps);
      uControl ~ normal(0,sigmaUcontrol) ; uTreatmt ~ normal(0,sigmaUtreatmt);
      beta ~ normal(0,sigmaBeta);
      sigmaEps ~ cauchy(0,Aeps);
      sigmaUcontrol ~ cauchy(0,Au); sigmaUtreatmt ~ cauchy(0,Au);
   }'

# Set up input data:

allData <- list(n = n,ncZ = ncZ,y = y,X = X,Z = Z,treatIndic = treatIndic,
```

```
                sigmaBeta = 1e5,Aeps = 1e5,Au = 1e5)

# Compile code for model if required:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000481 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.81 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:                0.000945 seconds (Sampling)
## Chain 1:                0.000946 seconds (Total)
## Chain 1:
```

```
## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Perform MCMC:

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

```
##
## SAMPLING FOR MODEL 'bccf1fc6b263ef6312a5e171463ddea2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000466 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.66 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  500 / 2000 [ 25%]  (Warmup)
```

```
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 42.9366 seconds (Warm-up)
## Chain 1:                11.3445 seconds (Sampling)
## Chain 1:                54.2812 seconds (Total)
## Chain 1:
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

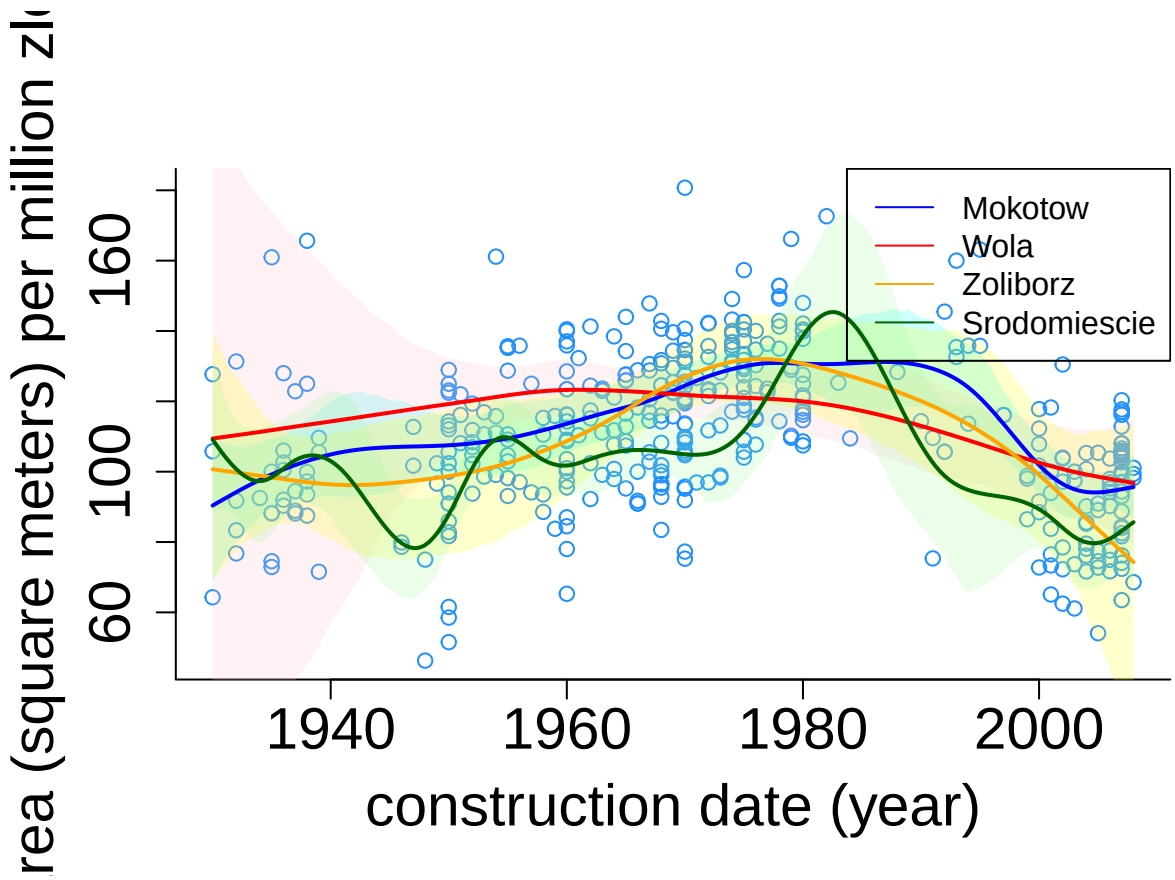
```r
# Extract relevant MCMC sampes:

betaMCMCS <- NULL
for (j in 1:4)
{
   charVar <- paste("beta[",as.character(j),"]",sep = "")
   betaMCMCS <- rbind(betaMCMCS,extract(stanObj,charVar,permuted = FALSE))
}
uControlMCMCS <- NULL  ;  uTreatmtMCMCS <- NULL
for (k in 1:ncZ)
{
   charVar <- paste("uControl[",as.character(k),"]",sep = "")
   uControlMCMCS <- rbind(uControlMCMCS,extract(stanObj,charVar,permuted = FALSE))

   charVar <- paste("uTreatmt[",as.character(k),"]",sep = "")
   uTreatmtMCMCS <- rbind(uTreatmtMCMCS,extract(stanObj,charVar,permuted = FALSE))
}
sigmaEpsMCMCS <- as.vector(extract(stanObj,"sigmaEps",permuted = FALSE))
sigmaUcontrolMCMCS <- as.vector(extract(stanObj,"sigmaUcontrol",permuted = FALSE))
sigmaUtreatmtMCMCS <- as.vector(extract(stanObj,"sigmaUtreatmt",permuted = FALSE))

# Plot data and fitted curves:
```

```
ng <- 201
cex.labVal <- 1.8
xLow <- min(x) ; xUpp <- max(x)
xg <- seq(xLow,xUpp,length = ng)

Xg <- cbind(rep(1,ng),xg)
Zg <- ZOSull(xg,intKnots = intKnots,range.x = c(a,b))



fhatCntrlMCMCS <- Xg%*%betaMCMCS[c(1,2),] + Zg%*%uControlMCMCS


fCntrlgS <- apply(fhatCntrlMCMCS,1,mean)
credLowCntrlS <- apply(fhatCntrlMCMCS,1,quantile,0.025)
credUppCntrlS <- apply(fhatCntrlMCMCS,1,quantile,0.975)

x <- (xOrig - mean.x)/sd.x

# Convert grids to original units:

xgOrig <- sd.x*xg + mean.x

fCntrlgOrigS <- sd.y*fCntrlgS + mean.y
credLowCntrlOrigS <- sd.y*credLowCntrlS + mean.y
credUppCntrlOrigS <- sd.y*credUppCntrlS + mean.y
```

b

```
par(mai = c(1.02,0.9,0.82,0.42))
plot(xOrig,yOrig,type = "n",bty = "l",xlab = "construction date (year)",
     ylab = "area (square meters) per million zloty",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)
points(xOrig,yOrig,col = "dodgerblue")




polygon(c(xgOrig,rev(xgOrig)),c(credLowCntrlOrigM,rev(credUppCntrlOrigM)),
        col =adjustcolor("cyan",alpha.f=0.2),border = FALSE)
polygon(c(xgOrig,rev(xgOrig)),c(credLowCntrlOrigW,rev(credUppCntrlOrigW)),
        col =adjustcolor("pink",alpha.f=0.2),border = FALSE)
polygon(c(xgOrig,rev(xgOrig)),c(credLowCntrlOrigZ,rev(credUppCntrlOrigZ)),
        col =adjustcolor("yellow",alpha.f=0.2),border = FALSE)
polygon(c(xgOrig,rev(xgOrig)),c(credLowCntrlOrigS,rev(credUppCntrlOrigS)),
        col =adjustcolor("palegreen",alpha.f=0.2),border = FALSE)

lines(xgOrig,fCntrlgOrigM,col = "blue",lwd = 2)


lines(xgOrig,fCntrlgOrigW,col = "red",lwd = 2)
```

```
lines(xgOrig,fCntrlgOrigZ,col = "orange",lwd = 2)


lines(xgOrig,fCntrlgOrigS,col = "darkgreen",lwd = 2)


legend("topright",legend = c("Mokotow","Wola","Zoliborz","Srodomiescie"),
       col = c("blue","red","orange","darkgreen"),lty=1)
```



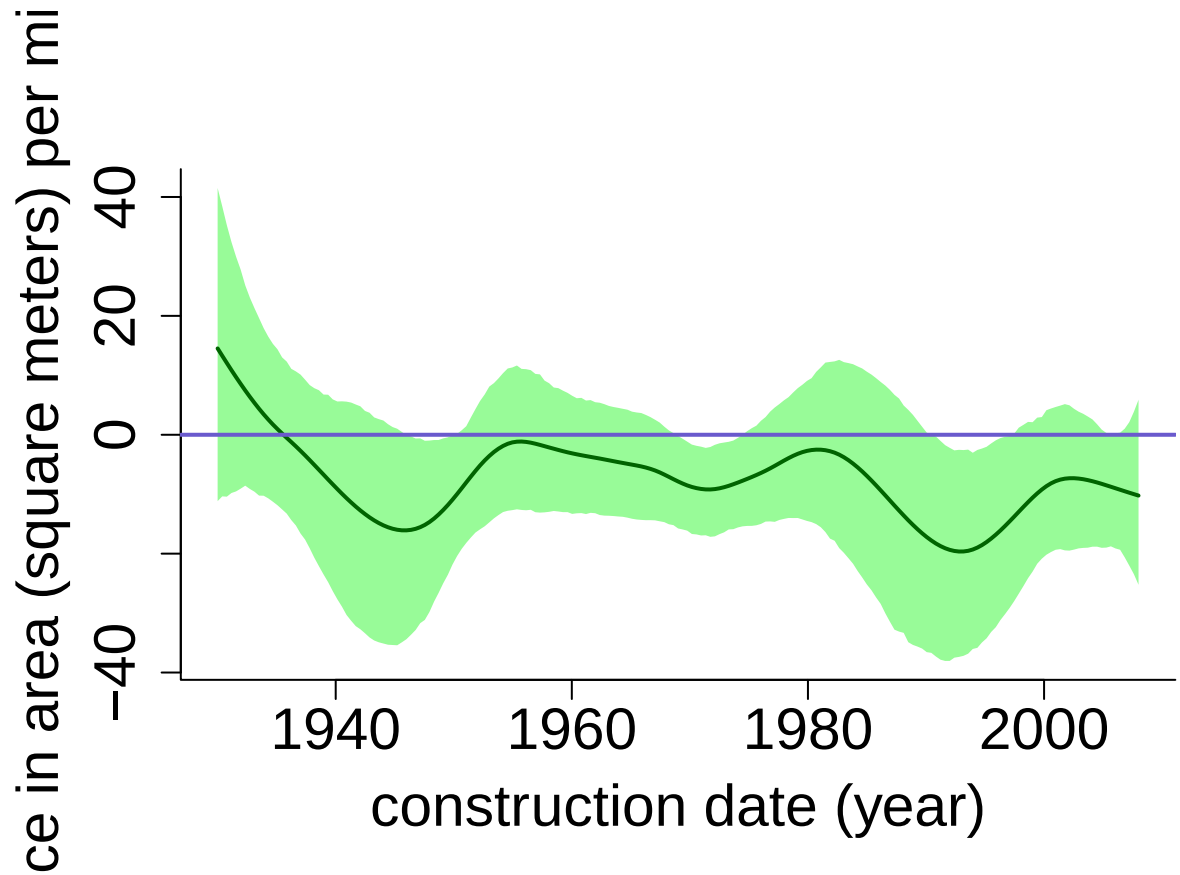
c

I did not know how to code for two districtiwise constrast function and its credibility function. Hence instead I am plotting the contrast function for each districts compared to the rest of the distrcits Combined.

Mokotow vs Rest

```
estFunCol <- "darkgreen";
varBandCol <- "palegreen"

ContrastMCMC <- Xg%*%betaMCMCM[c(3,4),] + Zg%*%(uTreatmtMCMCM-uControlMCMCM)

credLower <- apply(ContrastMCMC,1,quantile,0.025)
```

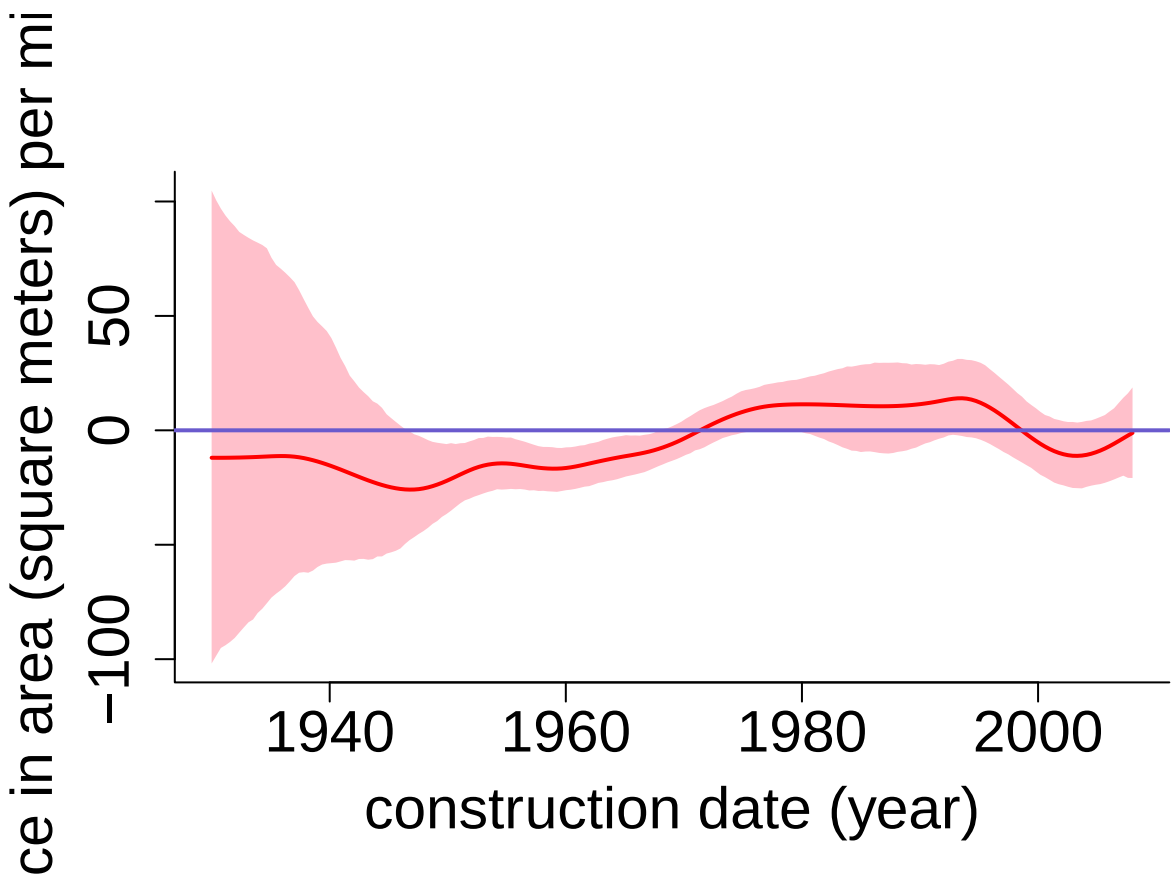
```
credUpper <- apply(ContrastMCMC,1,quantile,0.975)
Contrastg <- apply(ContrastMCMC,1,mean)

# Convert to original units:

ContrastgOrig <- sd.y*Contrastg
credLowerOrig <- sd.y*credLower
credUpperOrig <- sd.y*credUpper

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig,credUpperOrig)),
     xlab = "construction date (year)",
     ylab = " difference in area (square meters) per million zloty",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig,rev(credUpperOrig)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



Wola vs rest

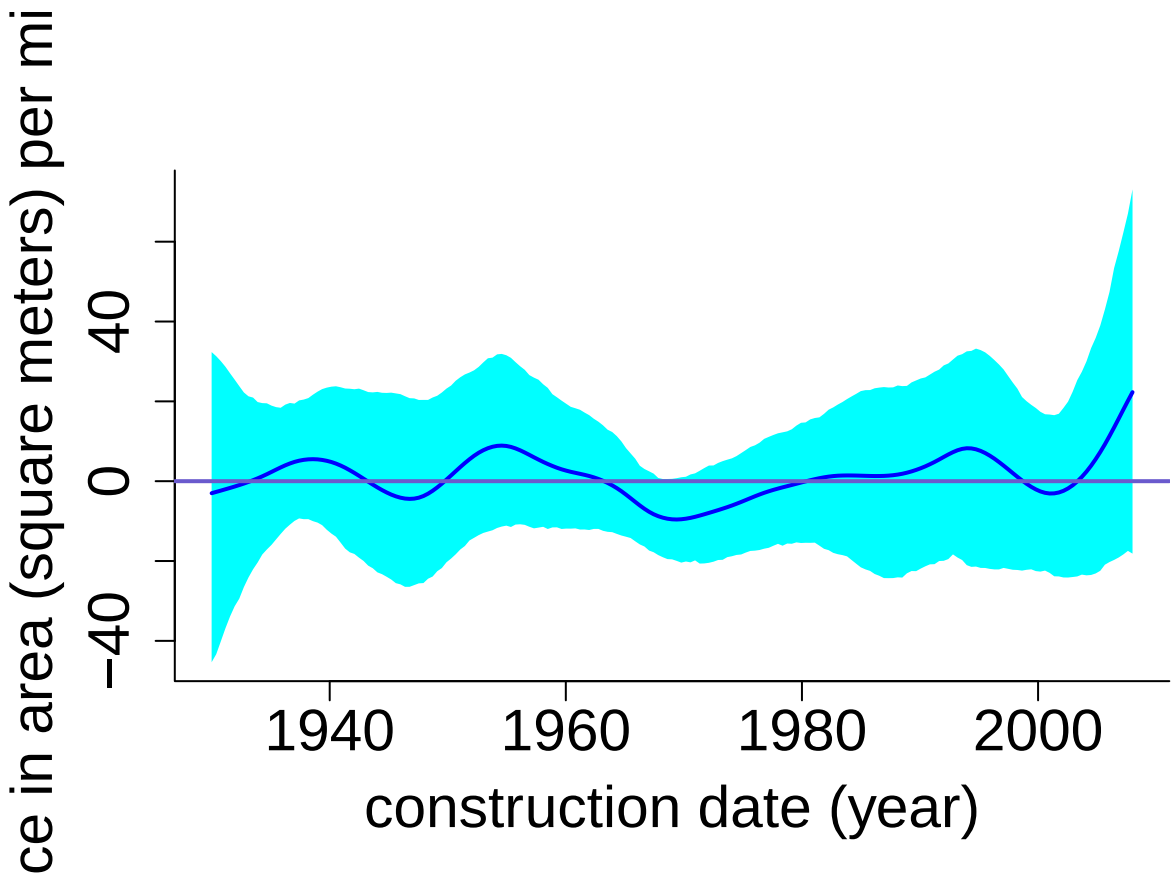
```
estFunCol <- "red";
varBandCol <- "pink"
```

```
ContrastMCMC <- Xg%*%betaMCMCW[c(3,4),] + Zg%*%(uTreatmtMCMCW-uControlMCMCW)

credLower <- apply(ContrastMCMC,1,quantile,0.025)
credUpper <- apply(ContrastMCMC,1,quantile,0.975)
Contrastg <- apply(ContrastMCMC,1,mean)

# Convert to original units:

ContrastgOrig <- sd.y*Contrastg
credLowerOrig <- sd.y*credLower
credUpperOrig <- sd.y*credUpper

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig,credUpperOrig)),
     xlab = "construction date (year)",
     ylab = " difference in area (square meters) per million zloty",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig,rev(credUpperOrig)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



Zoliborz vs Rest

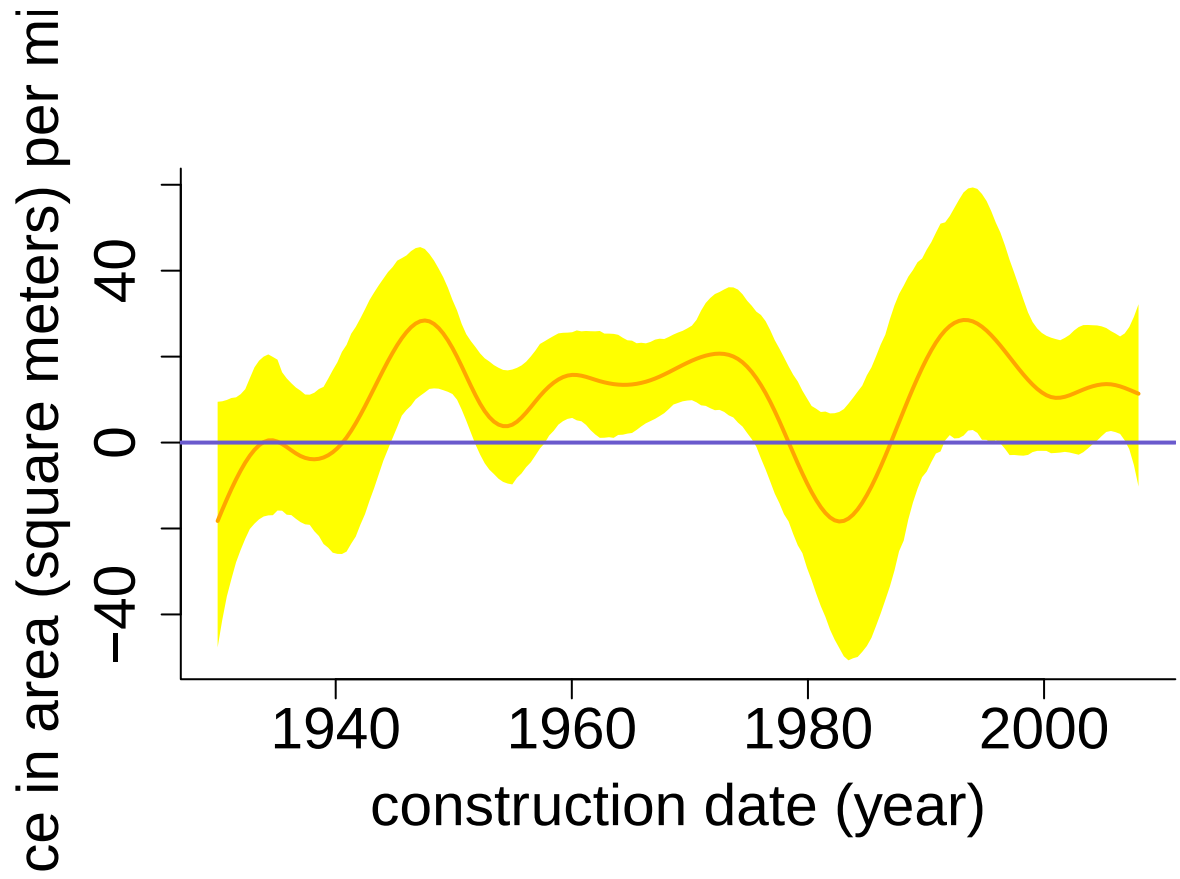
```
estFunCol <- "blue";
varBandCol <- "cyan"

ContrastMCMC <- Xg%*%betaMCMCZ[c(3,4),] + Zg%*%(uTreatmtMCMCZ-uControlMCMCZ)

credLower <- apply(ContrastMCMC,1,quantile,0.025)
credUpper <- apply(ContrastMCMC,1,quantile,0.975)
Contrastg <- apply(ContrastMCMC,1,mean)

# Convert to original units:

ContrastgOrig <- sd.y*Contrastg
credLowerOrig <- sd.y*credLower
credUpperOrig <- sd.y*credUpper

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig,credUpperOrig)),
     xlab = "construction date (year)",
     ylab = " difference in area (square meters) per million zloty",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)

polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig,rev(credUpperOrig)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



Srodmiescie vs rest

```
estFunCol <- "orange";
varBandCol <- "yellow"

ContrastMCMC <- Xg%*%betaMCMCS[c(3,4),] + Zg%*%(uTreatmtMCMCS-uControlMCMCS)

credLower <- apply(ContrastMCMC,1,quantile,0.025)
credUpper <- apply(ContrastMCMC,1,quantile,0.975)
Contrastg <- apply(ContrastMCMC,1,mean)

# Convert to original units:

ContrastgOrig <- sd.y*Contrastg
credLowerOrig <- sd.y*credLower
credUpperOrig <- sd.y*credUpper

par(mfrow = c(1,1),mai = c(1.02,0.9,0.82,0.42))
plot(0,0,type = "n",bty = "l",xlim = range(xgOrig),
     range(c(credLowerOrig,credUpperOrig)),
     xlab = "construction date (year)",
     ylab = " difference in area (square meters) per million zloty",
     cex.lab = cex.labVal,cex.axis = cex.axisVal)
```

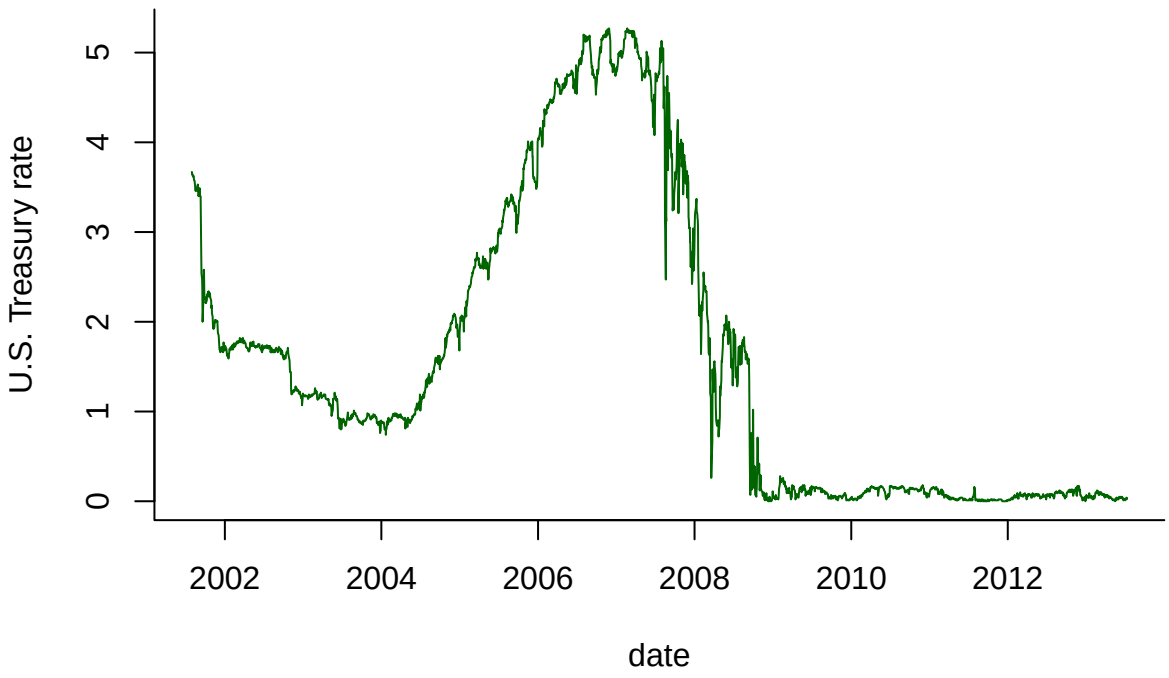
```
polygon(c(xgOrig,rev(xgOrig)),c(credLowerOrig,rev(credUpperOrig)),
        col = varBandCol,border = FALSE)

lines(xgOrig,ContrastgOrig,lwd = 2,col = estFunCol)
abline(0,0,col = "slateblue",lwd = 2)
```



2.10

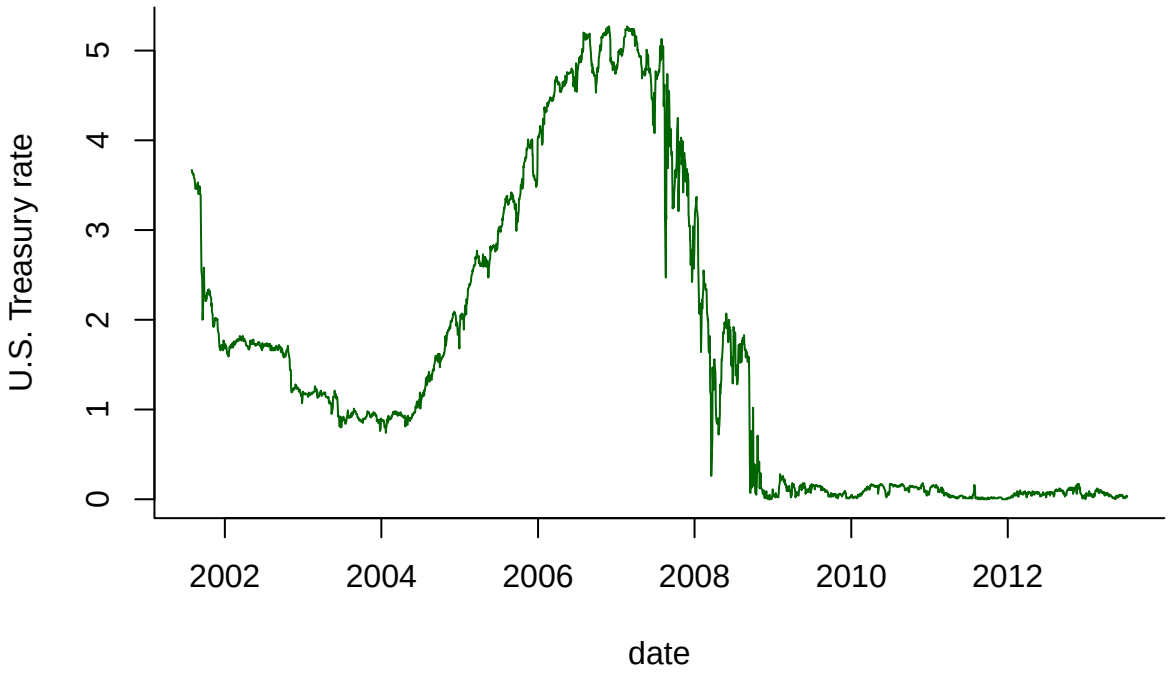
a

```
library(HRW) ; data(TreasuryRate)
date <- (as.Date(TreasuryRate$date,"%m/%d/%Y")[!is.na(TreasuryRate$rate)])
r <- TreasuryRate$rate[!is.na(TreasuryRate$rate)]
plot(date,r,type = "l",bty = "l",col = "darkgreen",xlab = "date",ylab="U.S. Treasury rate")
```



b

```
########## R script: WarsawAptsBayes ##########

# For obtaining a Bayesian penalized spline
# fit to the Warsaw apartment data using Stan
# via the package rstan. Approximate Bayesian
# inference is achieved via Markov chain
# Monte Carlo (MCMC).

# Last changed: 05 JAN 2019

# Load required packages:

library(HRW) ; data(TreasuryRate)
date <- (as.Date(TreasuryRate$date,"%m/%d/%Y")[!is.na(TreasuryRate$rate)])
r <- TreasuryRate$rate[!is.na(TreasuryRate$rate)]
plot(date,r,type = "l",bty = "l",col = "darkgreen",xlab = "date",ylab="U.S. Treasury rate")
```




```
# Standardize both predictor and response variable
# and set hyperparameter values:

yOrig=r[-1]-r[-length(r)]
xOrig=r[-length(r)]
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
sigmaGamma<-1e-5; Bv<-1e-5;
# 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 <- 10
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)

numIntKnotsB <- 100
intKnotsB <-  quantile(unique(x),seq(0,1,length=numIntKnots+2)
                      [-c(1,numIntKnots+2)])
W <- ZOSull(x,intKnots=intKnots,range.x=c(a,b))
```

```
ncW <- ncol(W)

# 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;
      int<lower=1> ncW;        matrix[n,ncW] W;
      real<lower=0> sigmaGamma;
      real<lower=0> Bv;
   }
   parameters
   {
      vector[2] beta;           vector[ncZ] u;
      real<lower=0> sigmaeps;   real<lower=0> sigmau;
      vector[2] gamma;           vector[ncW] v;
      real<lower=0> sigmav;
   }
   transformed parameters
   {
      vector[n] f; // f function
      vector[n] g; // g function
      f = X*beta + Z*u;
      g = exp(X*gamma + W*v);
   }
   model
   {
      y ~ normal(f,sqrt(g));
      u ~ normal(0,sigmau); beta ~ normal(0,sigmaBeta);
      sigmaeps ~ cauchy(0,Aeps); sigmau ~ cauchy(0,Au);
      sigmav ~ cauchy(0,Bv); gamma ~ normal(0,sigmaGamma);
      v ~ normal(0,sigmav);
   }'

# 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,
                sigmaGamma=sigmaGamma,Bv=Bv,
                W=W,ncW=ncW)

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

compileCode <- TRUE

# Compile code for model if required:

if (compileCode)
```

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

```
## Trying to compile a simple C file

## 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 '48929cf195e671b44407208a7ff18bd6' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.001104 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 11.04 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: 3e-06 seconds (Warm-up)
## Chain 1:                0.001895 seconds (Sampling)
## Chain 1:                0.001898 seconds (Total)
## Chain 1:

## Warning: There were 1 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Set MCMC sample size parameters:
```

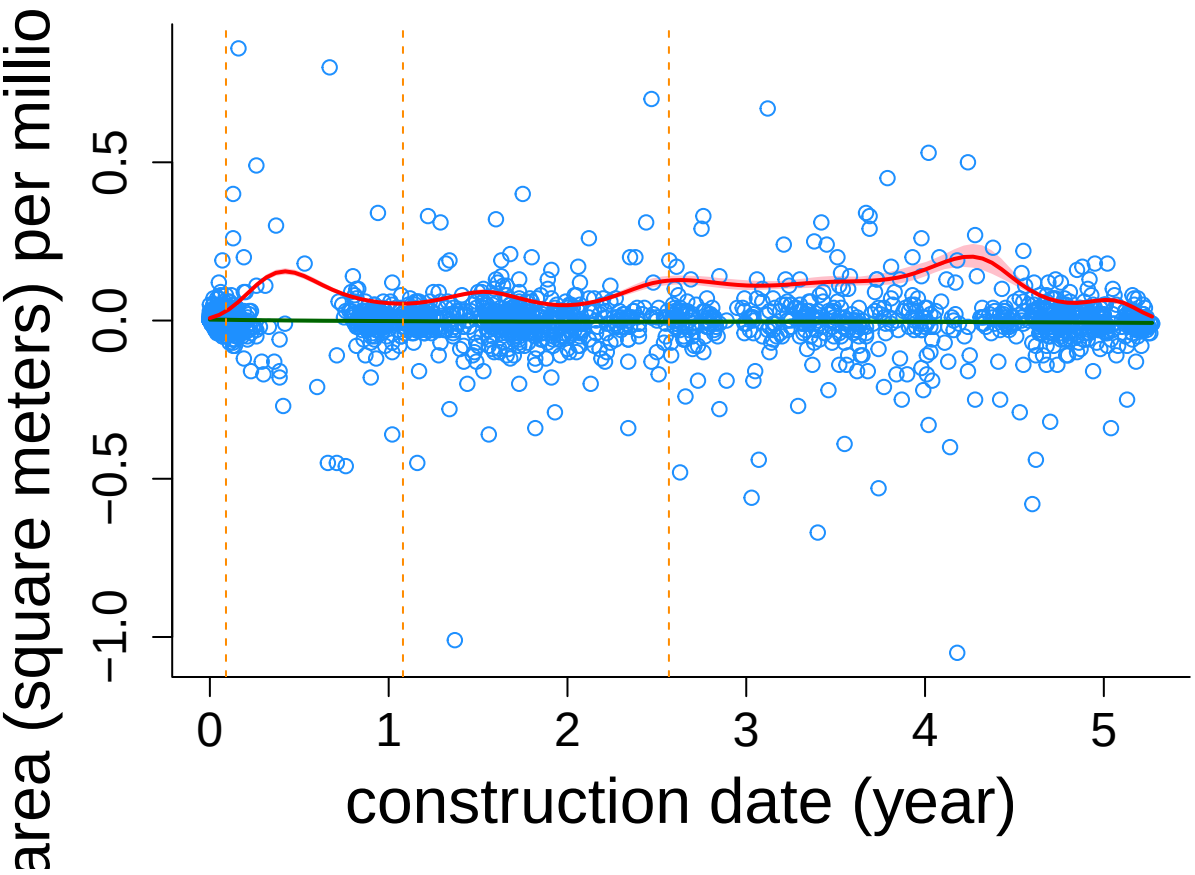
```r
nWarm <- 1000         # Length of warm-up.
nKept <- 2000         # Size of the kept sample.
nThin <- 2            # Thinning factor.

# Obtain MCMC samples for each parameter using Stan:

initFun <- function()
   return(list(sigmau=1,sigmaeps=0.7,beta=rep(0,2),u=rep(0,ncZ),sigmav=1,gamma=rep(0,2),v=rep(0,ncW)))

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 '48929cf195e671b44407208a7ff18bd6' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000528 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 5.28 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: 485.737 seconds (Warm-up)
## Chain 1:                1006.13 seconds (Sampling)
## Chain 1:                1491.87 seconds (Total)
## Chain 1:

## Warning: There were 870 transitions after warmup that exceeded the maximum treedepth. Increase max_t
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

## Warning: There were 1 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 1.12, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```r
# 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))
}
gammaMCMC <- NULL
for (l in 1:2)
{
  charVar <- paste("gamma[",as.character(l),"]",sep="")
  gammaMCMC <- rbind(gammaMCMC,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))
}
vMCMC <- NULL
for (m in 1:ncW)
{
  charVar <- paste("v[",as.character(m),"]",sep="")
  vMCMC <- rbind(vMCMC,extract(stanObj,charVar,permuted=FALSE))
}
```

```
sigmaepsMCMC <- as.vector(extract(stanObj,"sigmaeps",permuted=FALSE))
sigmauMCMC <- as.vector(extract(stanObj,"sigmau",permuted=FALSE))
sigmavMCMC <- as.vector(extract(stanObj,"sigmav",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))
Wg <- ZOSull(xg,intKnots=intKnotsB,range.x=c(a,b))
fhatMCMC <- Xg%*%betaMCMC + Zg%*%uMCMC
ghatMCMC <- exp(Xg%*%gammaMCMC+Wg%*%vMCMC)

# 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)

ghatMCMCOrig <- sqrt(ghatMCMC)*sd.y + mean.y
ghatgOrig <- apply(ghatMCMCOrig,1,mean)
credLowerg <- apply(ghatMCMCOrig,1,quantile,0.025)
credUpperg <- apply(ghatMCMCOrig,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="construction date (year)",
     ylab="area (square meters) per million zloty",
     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(credLowerg,rev(credUpperg)),
        col="pink",border=FALSE)
polygon(c(xgOrig,rev(xgOrig)),c(credLower,rev(credUpper)),
        col="palegreen",border=FALSE)
points(xOrig,yOrig,col="dodgerblue")
lines(xgOrig,fhatgOrig,col="darkgreen",lwd=2)
lines(xgOrig,ghatgOrig,col="red",lwd=2)

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")
```

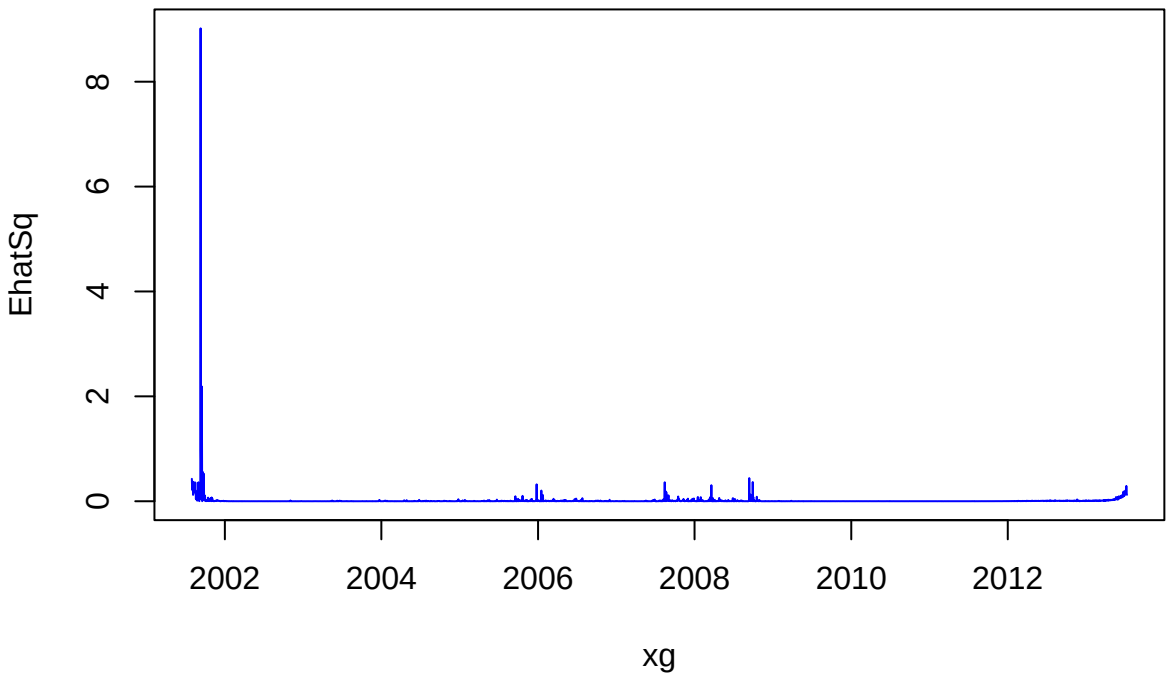


c

```
## Standardized residuals test

ng <- 2987
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))
Wg <- ZOSull(xg,intKnots=intKnotsB,range.x=c(a,b))
fhatMCMC <- Xg%*%betaMCMC + Zg%*%uMCMC
ghatMCMC <- exp(Xg%*%gammaMCMC+Wg%*%vMCMC)
fhatMCMCOrig<-apply(fhatMCMC,1,mean)
ghatMCMCOrig<-apply(ghatMCMC,1,mean)
EhatSq<-((yOrig-fhatMCMCOrig)/sqrt(ghatMCMCOrig))**2
```

```
## Warning in yOrig - fhatMCMCOrig: longer object length is not a multiple of
## shorter object length
```

```
xg<-seq(min(date),max(date),length=2987)
plot(xg,EhatSq,col="white")
lines(xg,EhatSq,col="blue")
```



According to this curve , the mean is definitely noty constant especially during the time period between 2006-2008.

d

Reference : http://cran.nexr.com/web/packages/bayesplot/vignettes/visual-mcmc-diagnostics.html

Some diagnostics that we look at to check the estimated model parameters like sigmaepsmu and functionHats of g() and f() are the following:

1. Traceplots for tge sigma parameters : We can use traceplots to time series plot for the Markov chains corresponding to the draws of sigmaeps and sigmamu.
2. for the convergence of the MC for the model and especially the fhat and sqrt(ghat) , we can also use the Rhat statistics will also stan uses internally. Using Rhat we can check if our model is converging properly or not by comparing it to other randomly initialized chains
3. we can also visualize the autocorrelation function line plot especialluy for the function estimate parameters fhat and ghat. WE can check the extremity of each parameter's markov chain by acf_plot function.
4. I also something called as HMC NUTS(No u turn Sampler) can be used for parameter diagnostics.