Recent Work

Bowen Xiao

May 16, 2018

```
load(".RData")
```

```
library(rstan)
```

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

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

```
## rstan (Version 2.17.3, 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)
```

```
library(loo)
```

```
## This is loo version 2.0.0.
## **NOTE: As of version 2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use th
```

```
library(mice)
```

```
## Loading required package: lattice
```

```
library(Kendall)
library(trend)
library(forecast)
```

Time series decomposition

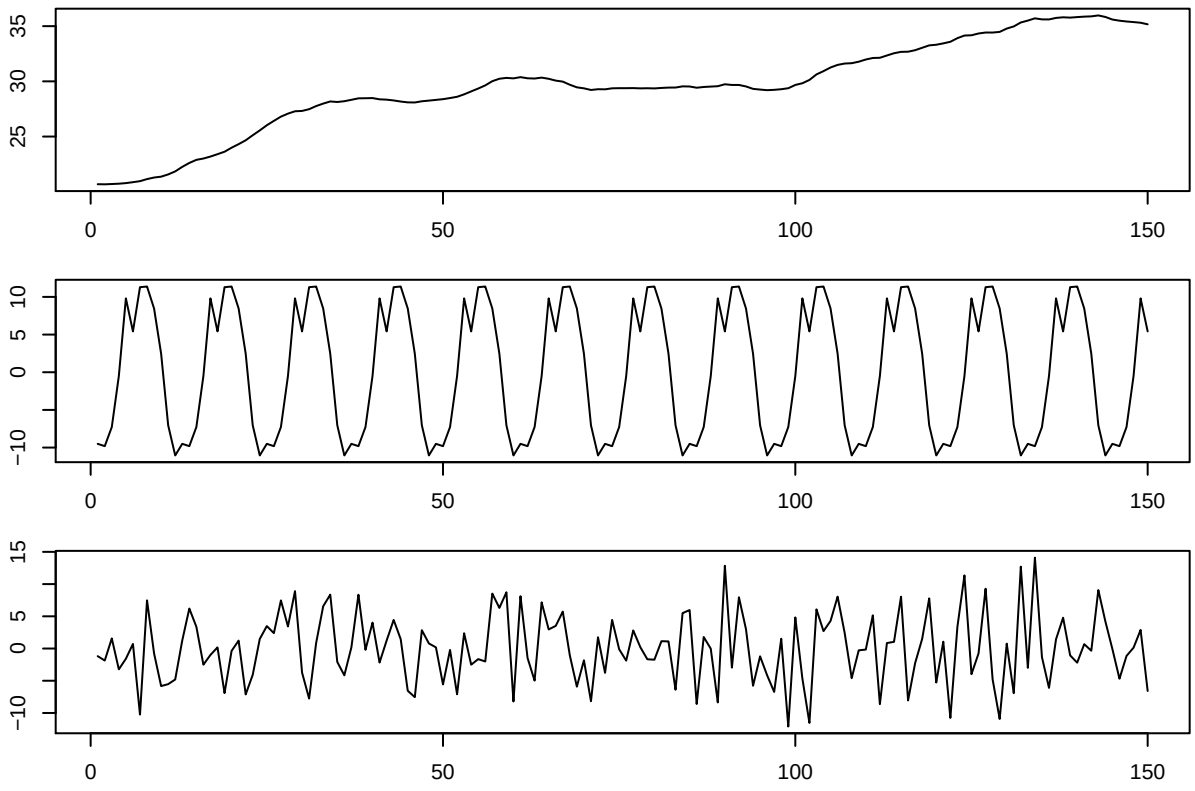
I use GMRF twice to do time series decomposition. Firstly, I use GMRF to fit a smoothing line. I split the line by bandwidth of 12 (since I know the period is 12 month), and avarage the difference with the mean in each piece. So that I get the seasonal component. Secondly, I minus the raw data with seasonal component and do another GMRF fitting on it, which turns out to be the trend component.

```
# first fitting
test<-list(J=150,y=tts)
TT<-stan('src/stan/Gaussian.stan', data=test, chains=nchain, iter=niter, warmup=nburn, thin=nthin,
         control=list(adapt_delta=0.95, max_treedepth=12))
TT.N<-matrix(unlist(extract(TT,pars=c("theta"))),nrow = 2500, byrow = FALSE)
theta<-apply(TT.N,2,median)

# extract seasonal component
temp=c()
for (i in 1:12){temp=c(temp,rep(mean(theta[1:150][(1+12*(i-1)):(12+12*(i-1))]),12))}
temp=c(temp,rep(mean(theta[145:150]),6))
temp=test$y-temp
seasonal<-c()
for (i in 1:12){seasonal<-c(seasonal,mean(temp[seq(i,150,12)]))}
seasonal<-rep(seasonal,length=150)
```

```
# second fitting
test1 <- list(J = 150, y = test$y-seasonal)
TT1<-stan('src/stan/Gaussian1.stan', data=test1, chains=nchain, iter=niter, warmup=nburn, thin=nthin,
          control=list(adapt_delta=0.95, max_treedepth=12))
TT.N1<-matrix(unlist(extract(TT1,pars=c("theta"))),nrow = 2500, byrow = FALSE)
theta1<-apply(TT.N1,2,median)

# extract trend component
trend=theta1
random=tts-seasonal-trend
```

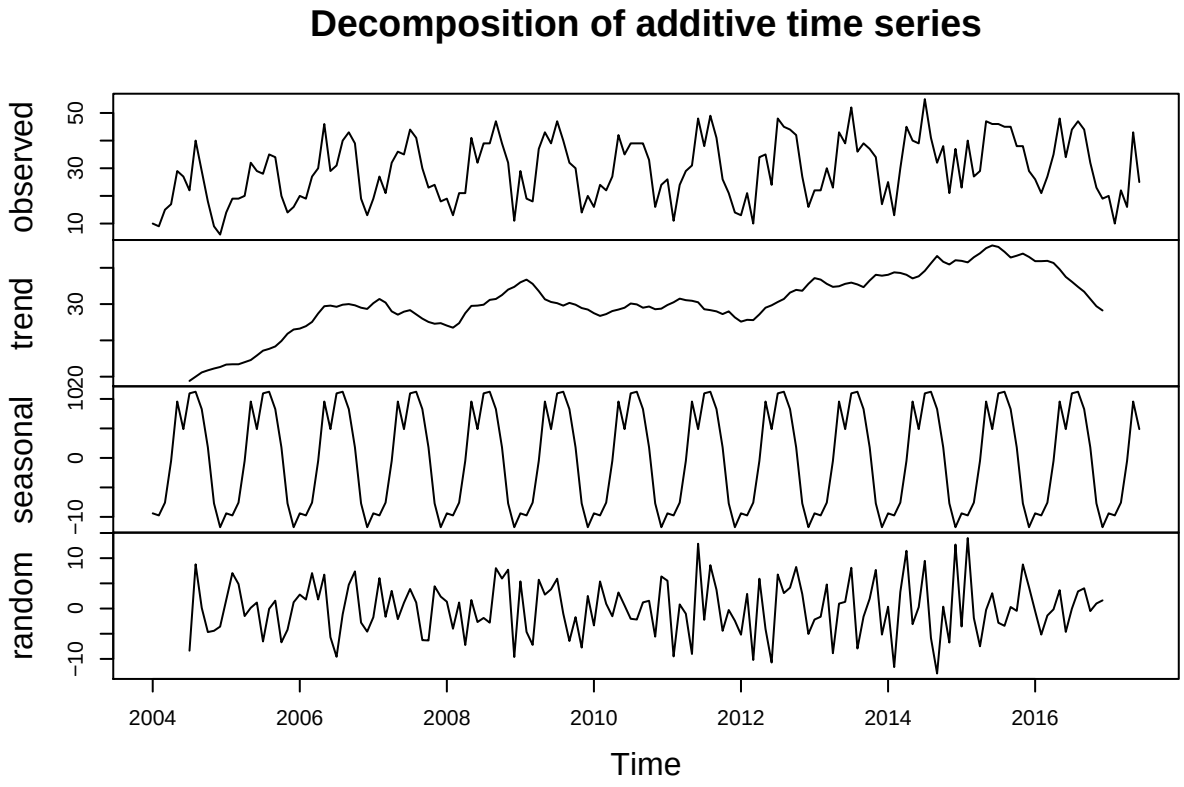
My decomposition will look like the following.

```
par(mfrow=c(3,1), mar=c(2,1.5,1.5,1), oma=c(2,2,0,0))
plot(trend,type='l')
plot(seasonal,type='l')
plot(random,type='l')
```



As a comparison, decomposition based on ARIMA looks like the following.

```
par(mfrow=c(1,1))
plot(decompose(TS))
```



I choose step length based on cross validation. And then I sum up the seasonal component with a linear extension of trend component as my prediction.

```
#step_width<-select_StepLength(trend,seasonal,random)
mean((predict(lm(y~.,data=data.frame(y=trend[(150-step_width+1):150],x=c((150-step_width+1):150))),
              newdata=data.frame(x=c(151:162)))+seasonal[139:150]-TS[151:162])^2)
```

```
## [1] 71.37007
```

Things could be better if I choose to discribe the 'trend' in a parametric form.

```
library(splines)

tr<-data.frame(y=trend,x=1:150)
loocv<-c()
for (i in 3:10){
  cv<-c()
  for (j in 1:150){
    temp<-tr[-j,]
    long_line<-lm(y~bs(x,df=i),data=temp)
    cv<-c(cv,abs(predict(long_line,newdata=data.frame(x=j))-trend[j]))
  }
  loocv<-c(loocv,mean(cv))
}
long_line<-lm(y~bs(x,df=which.min(loocv)+2),data=tr)
mean((seasonal[139:150]+predict(long_line,newdata = data.frame(x=c(151:162)))-TS[151:162])^2)
```

```
## [1] 49.02031
```

Or

```
tcv<-c()
for (i in 3:10){
  cv<-c()
  for (j in 1:5){
    temp<-tr[1:(nrow(tr)-j),]
    long_line<-lm(y~bs(x,df=i),data=temp)
    cv<-c(cv,abs(predict(long_line,newdata=data.frame(x=c((nrow(tr)-j+1):nrow(tr))))-trend[j]))
  }
  tcv<-c(tcv,mean(cv))
}
long_line1<-lm(y~bs(x,df=which.min(tcv)+2),data=tr)
mean((seasonal[139:150]+predict(long_line1,newdata = data.frame(x=c(151:162)))-TS[151:162])^2)
```

```
## [1] 38.21428
```

As a comparison, predictions based on raw sampling, ARIMA and RNN are shown as following.

```
mean((theta[151:162]-TS[151:162])^2)
```

```
## [1] 210.9952
```

```
# arim <- auto.arima(ts(tts,start=c(2004,1),frequency=12),stepwise=FALSE,approximation=FALSE)
# ari<-forecast(arim,h=12)
mean((ari$mean-TS[151:162])^2)
```

```
## [1] 160.4374
```

```
mean((rnn-TS[151:162])^2)
```

```
## [1] 109.9083
```

Bayes diagnostic

Horseshoe prior seems to have a little issue.

```
check_hmc_diagnostics(TT)
```

```
##
## Divergences:
```

```
## 0 of 2500 iterations ended with a divergence.
```

```
##
## Tree depth:
```

```
## 0 of 2500 iterations saturated the maximum tree depth of 12.
```

```
##
## Energy:
```

```
## E-BFMI indicated no pathological behavior.
```

```
check_hmc_diagnostics(TT1)
```

```
##
## Divergences:
```

```
## 0 of 2500 iterations ended with a divergence.
```

```
##
## Tree depth:

## 0 of 2500 iterations saturated the maximum tree depth of 12.

##
## Energy:

## E-BFMI indicated no pathological behavior.
```

```
check_hmc_diagnostics(TTL)
```

```
##
## Divergences:

## 0 of 2500 iterations ended with a divergence.

##
## Tree depth:

## 0 of 2500 iterations saturated the maximum tree depth of 12.

##
## Energy:

## E-BFMI indicated no pathological behavior.
```

```
check_hmc_diagnostics(TTH)
```

```
##
## Divergences:

## 52 of 2500 iterations ended with a divergence (2.08%).
## Try increasing 'adapt_delta' to remove the divergences.

##
## Tree depth:

## 0 of 2500 iterations saturated the maximum tree depth of 12.

##
## Energy:

## E-BFMI indicated no pathological behavior.
```

Gaussian prior seems to be the best choice here.

```
library("loo")
log_lik_1 <- extract_log_lik(TT)
log_lik_2 <- extract_log_lik(TTL)
log_lik_3 <- extract_log_lik(TTH)
loo1 <- loo(log_lik_1)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
loo2 <- loo(log_lik_2)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
## Warning in log(z): NaNs produced
```

```
loo3 <- loo(log_lik_3)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
compare(loo1,loo2,loo3)
```

```
##      elpd_diff elpd_loo se_elpd_loo p_loo  se_p_loo looic  se_looic
## loo1    0.0    -514.2      7.6        71.4    4.9   1028.4   15.3
## loo2   -1.0    -515.2      7.8        72.1    5.3   1030.3   15.5
## loo3   -6.9    -521.1      8.2        74.4    5.6   1042.1   16.4
```

```
lpd_point <- cbind(
  loo1$pointwise[,"elpd_loo"],
  loo2$pointwise[,"elpd_loo"],
  loo3$pointwise[,"elpd_loo"]
)
(stacking_wts <- stacking_weights(lpd_point))
```

```
## Method: stacking
## ------
##        weight
## model1 0.773
## model2 0.227
## model3 0.000
```

```
waic1<-waic(log_lik_1)
```

```
## Warning: 40 (26.7%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

```
waic2<-waic(log_lik_2)
```

```
## Warning: 42 (28.0%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

```
waic3<-waic(log_lik_3)
```

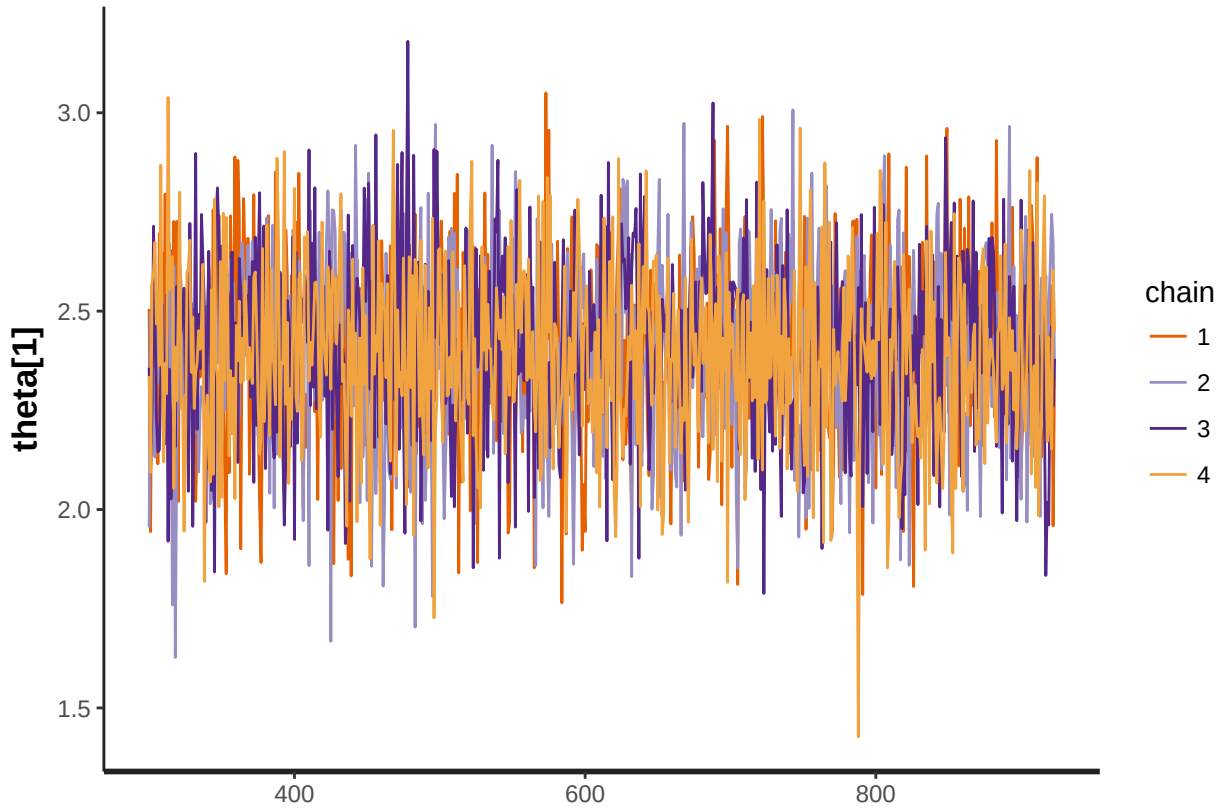
```
## Warning: 48 (32.0%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

```
compare(waic1,waic2,waic3)
```

```
##       elpd_diff elpd_waic se_elpd_waic p_waic se_p_waic waic   se_waic
## waic1    0.0    -499.8       6.6         57.0    3.9     999.6   13.2
## waic2   -2.1    -501.9       6.8         58.8    4.3    1003.8   13.7
## waic3   -8.1    -507.9       7.7         61.3    5.0    1015.8   15.5
```

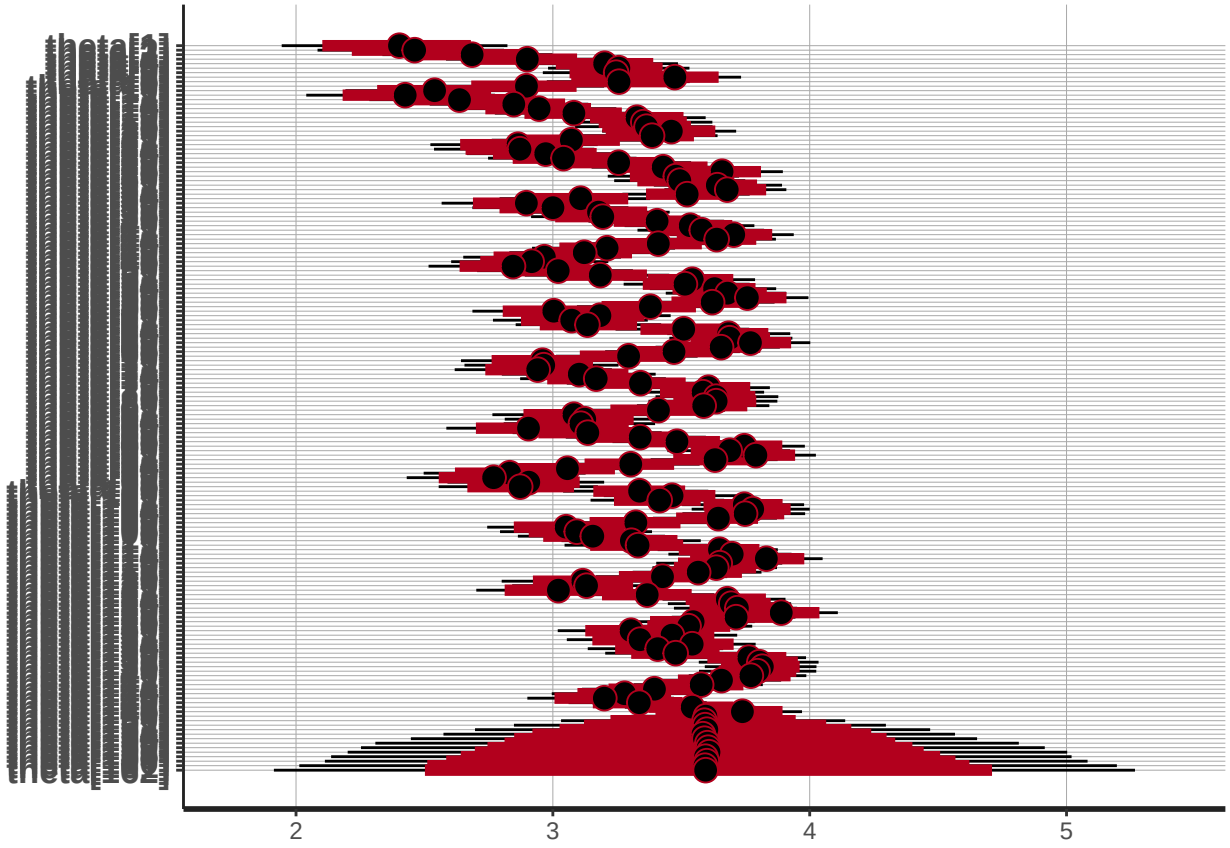
And some poor-organized plot for GMRF based on Gaussian prior.

```
traceplot(TT,pars=c('theta[1]'))
```



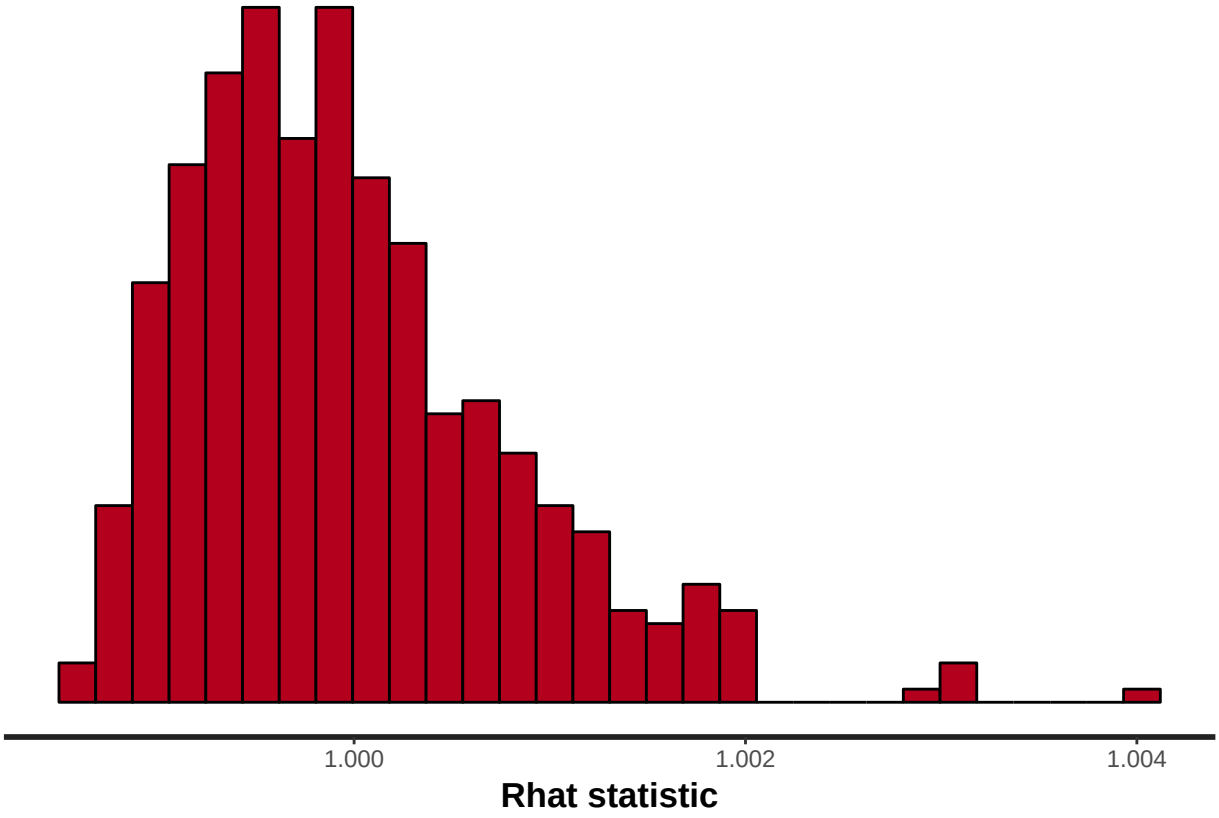

```
plot(TT,pars=c('theta'))
```

```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```




```
plot(TT, plotfun = "rhat")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

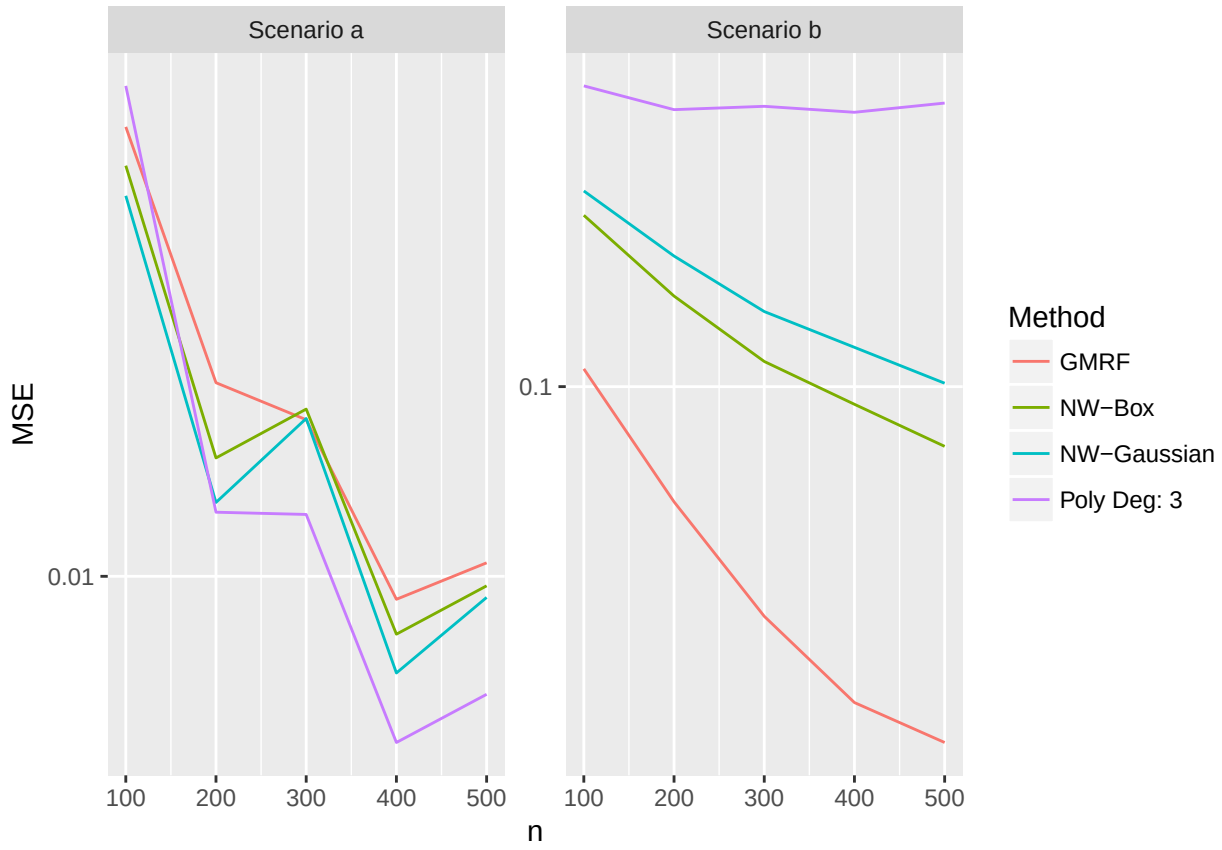


Simulation

```
oneSim <- function(n = 100, scen = "a") {
  if(scen == "a") {
    f <- function(x){2*x}
  } else if(scen == "b") {
    f <- function(x){sin(10*x)}
  }
  x <- sort(runif(n))
  y <- f(x) + rnorm(n)


  # Polynomial Models
  yh3 <- lm(y~poly(x,3))$fitted.values

  # Ndaraya-Watson Models
  h <- n^(-1/5)
  yh6 <- ksmooth(x, y, kernel = "box", bandwidth = h)$y
  yh7 <- ksmooth(x, y, kernel = "normal", bandwidth = h)$y

  # Markov Random Fields with Shrinkage Priors
  test<-list(J=n,y=y)
  test$N <- length(test$y)
  test$xvar1 <- 1:test$N
```

```
  uxv1 <- unique(test$xvar1)
  ruxv1 <- rank(uxv1)
  m.xv <- cbind(1:length(uxv1), uxv1, ruxv1)
  m.xv <- m.xv[order(m.xv[,2]),]
  duxv1 <- diff(m.xv[,2])
  suxv1 <- sort(uxv1)
  rnk.xv <- integer(test$N)
  for (ii in 1:test$N){
    rnk.xv[ii] <- ruxv1[which(uxv1==test$xvar1[ii])]
  }
  test$J <- length(uxv1)
  test$duxvar1 <- duxv1
  test$xrank1 <- rnk.xv
  TT<-stan('src/stan/Gaussian1.stan', data=test, chains=nchain, iter=niter, warmup=nburn, thin=nthin,
           control=list(adapt_delta=0.95, max_treedepth=12))
  yh8 <- matrix(unlist(extract(TT,pars=c("theta"))),nrow = 2500, byrow = FALSE)
  yh8 <- apply(yh8,2,median)

  mse <- colMeans( (cbind(yh3,yh6,yh7,yh8) - f(x))^2 )
  names(mse) <- c("Poly Deg: 3", "NW-Box","NW-Gaussian","GMRF")
  mse
}

multipleSim <- function(n = 100, scen = "a", nsim = 10) {
  set.seed(1)
  all.mse <- replicate(nsim, oneSim(n, scen))
  apply(all.mse, 1, mean)
}

n.seq <- seq(100, 500, by = 100)

# Simulation Results A
simA <- sapply(n.seq, FUN = multipleSim, scen = "a", nsim = 10)
resA <- data.frame("MSE" = as.numeric(simA),
                   "Method" = rep(rownames(simA), 10),
                   "n" = rep(n.seq, each = 4),
                   "Scenario" = rep("Scenario a", 40))

# Simulation Results B
simB <- sapply(n.seq, FUN = multipleSim, scen = "b", nsim = 10)
resB <- data.frame("MSE" = as.numeric(simB),
                   "Method" = rep(rownames(simB), 10),
                   "n" = rep(n.seq, each = 4),
                   "Scenario" = rep("Scenario b", 40))
```

```
library(ggplot2)
#dat <- rbind(resA, resB)
ggplot(dat, mapping = aes(x = n, y = MSE, color = Method)) +
  geom_line() + facet_wrap(~Scenario, scales = "free") + scale_y_log10()
```



As is shown, GMRF appeals to have some advantages as a non-parametric regression method.