Methods

Marshall Honaker

First, let's import the necessary packages:

```r
library(fields)
```

```
## Loading required package: spam

## Loading required package: dotCall64

## Loading required package: grid

## Spam version 2.5-1 (2019-12-12) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code
```

```r
library(RandomFields)
```

```
## Loading required package: sp

## Loading required package: RandomFieldsUtils

##
## Attaching package: 'RandomFields'

## The following object is masked from 'package:RandomFieldsUtils':
##
##     RFoptions
```

```r
library(mvtnorm)
library(ggplot2)
```

Wu and Narisetty (2020) outline a score based likelihood approach to Bayesian multiple quantile regression. Here we provide R code for a modified version of this approach adapted for spatial data. Annotations will be made/provided to explain the ways in which the approach taken by Wu & Narisetty was modified to accomdodate spatially dependent data.

To begin, we will define so of the more basic, necessary functions. First, we will define the score function.

Basic Functions & Methods

In the aforementioned paper, the score function is given by: $s_\tau(\beta) = \sum_{i=1}^n x_i \psi_\tau(y_i - x_i^T \beta)$, where $\psi_\tau(u) = \tau - I_{\{u<0\}}(u)$. Note that in this case, $\psi_\tau(.)$ is essentially the check-loss function mentioned earlier in the paper applied top the residuals.

```r
get_score <- function(y, X, tau, beta){
  ## outputs px1 vector
  u <- y - X%*%beta
  psi <- ifelse(u < 0, tau - 1, tau)
  return(t(X)%*%psi)
}
```

I'm also going to define a function to obtain the spatial covaraince matrix. I'm going to set it to use the Matérn covariance function with $\nu = 1.5$ and $\phi = 1.5$ but we can change this later or make it more adaptive as need be.

```r
get_spatial_covar_mat <- function(locs){  ## outputs nxn spatial covariance matrix
  dist <- rdist(locs)
  return(Matern(dist, range = .2, nu = 1.5))
}
```

Next, we will write a method to obtain a spatially adapted rendition of the proposed working likelihood function. The working likelihood function proposed in the paper is given by $L(Y|X,\beta) = C \exp\left(-\frac{1}{2n} s_\tau(\beta)^T W s_\tau(\beta)\right)$ where W is a p x p weight matrix. Now, to account for the spatial variability/dependence within the data, we will instead use the sample Mahalanobis distance given by $(X - \hat{\mu})^T \Sigma^{-1} (X - \hat{\mu})$ where Σ is the spatial covariance matrix. *Note that while the paper requires W to be positive definite, the sample Mahalanobis matrix is only positive semi-definite. We will need to find a way to make sure that this this change will still lead to valid inference based on the posterior later on.*

```r
get_likelihood <- function(y, X, tau, beta, locs, C){
  ## Outputs 1x1 scalar that is the likelihood
  score <- get_score(y, X, tau, beta)
  X_centered <- X - colMeans(X)
  coef <- -1/(2*length(y))
  kernel <- exp(coef*t(score)%*%t(X_centered)%*%solve(get_spatial_covar_mat(locs))%*%X_centered%*%score
  return(C*kernel[1,1])
}
```

Testing basic methods

Now that we have defined the basic functions, let's test them to make sure they work.

```r
n <- 100
locs <- cbind(runif(n, 0, 10), runif(n, 0, 10))
m1 <- RMexp(var = 2, scale = 1.5) +
  RMnugget(var = .1) +
  RMtrend(mean = 1)
sim_vals <- RFsimulate(m1, x = locs[,1], y = locs[,2])
```

```
## New output format of RFsimulate: S4 object of class 'RFsp';
## for a bare, but faster array format use 'RFoptions(spConform=FALSE)'.
```

```r
test_X <- cbind(rnorm(n), rnorm(n), rnorm(n), rnorm(n))
test_beta <- c(1, 2, 3, 4)
test_y <- test_beta[2]*locs[,1] + test_beta[3]*locs[,2] + sim_vals$variable1
test_tau <- .5
```

```r
get_score(test_y, test_X, test_tau, test_beta)
```

```
##             [,1]
## [1,]   1.244454
## [2,] -10.492330
## [3,]  -2.805197
## [4,]   5.092152
```

```r
get_likelihood(test_y, test_X, test_tau, test_beta, locs, 1)
```

```
## [1] 4.040766e-36
```

So it seems like everything is working alright. Now let's move on to the importance sampling.

Ada Importance Sampling for a Single Quantile Level

Wu & Narisetty describe an Importance Sampling (IS) procedure on pgs. 13-14. I've done my best to code it here, but there are still some issues. I will discuss these issues later.

The `get_updated_params()` function simply returns a list whose elements are the parameter values (the mean vector and the covariance matrix) for a given set of simulated observations. This makes the `adaIS_singleQuantile()` function a little more readable when we have to update the parameter values in each iteration.

```r
get_updated_params <- function(y, X, tau, beta, Sigma, locs, draw){
  ## outputs list with mu (which is a p-length vector of estimated means) and S (which is the estimated
  n <- length(y)
  p <- ncol(X)

  w <- get_likelihood(y, X, tau, beta, locs, 1)*(1/(2*n)^p)/dmvnorm(draw, beta, Sigma)

  mu_hat <- apply(w*draw, 2, sum)/sum(w)

  S <- matrix(nrow = p, ncol = p)
  for(i in 1:p){
    for(j in 1:p){
      S[i,j] <- sum(w*(draw[,i] - mu_hat[i])*(draw[,j] - mu_hat[j]))/(sum(w)*(n - 1))
    }
  }
  temp <- list(mu_hat, S)
  names(temp) <- c("mu", "S")
  return(temp)
}

# get_effective_samp_size <- function(y, X, tau, beta, Sigma, locs, draw, M){
#   My attempt to write a function to obtain the effecitve sample size of our IS draws. I tried to foll
#   p <- ncol(X)
#   n <- length(y)
#   w <- get_likelihood(y, X, tau, beta, locs, 1)*(1/(2*n)^p)/dmvnorm(draw, beta, Sigma)
#   cv_sq <- ((M - 1)^(-1))*sum((w - mean(w))^2)/mean(w)^2
#   return(M/(1 + cv_sq))
# }

adaIS_singleQuantile <- function(y, X, tau, C, locs, M, num_reps){
  ## outputs list with $mu (a p-length vector of estimated means) and $S (the estimated pxp covariance
  p <- ncol(X) ## Step 1: Initialize starting values for IS algorithm
  n <- length(y) ## Step 1: Initialize starting values for IS algorithm
  beta <- coef(lm(y ~ X))[-1] ## Step 1: Initialize starting values for IS algorithm
  S <- cov(X) ## Step 1: Initialize starting values for IS algorithm

  params <- list(beta, S)  ## Put our initial parameter estimates/starting values into a list called pa
  names(params) <- c("mu", "S")

  for(i in 1:num_reps){ ## Step 4: Repeat steps 2 and 3 until we achieve the desired effective sample s
    draws <- rmvnorm(M, params$mu, params$S)  ## Step 2: Simulate M values from the proposal distributi
    params <- get_updated_params(y, X, tau, params$mu, params$S, locs, draws) ## Step 3: Update the par
  }
  return(params)
}
```

Now, to test our IS method.

```
m <- 10000
S_0 <- cov(test_X)
initial_draw <- rmvnorm(m, test_beta, S_0)

get_updated_params(test_y, test_X, test_tau, test_beta, cov(test_X), locs, initial_draw)
```

```
## $mu
## [1] -0.4089676  2.0827504  3.0511531  4.5454578
##
## $S
##              [,1]        [,2]          [,3]          [,4]
## [1,]  0.045519851 -0.01498001 -0.0071386549 -0.0101190765
## [2,] -0.014980011  0.04611052  0.0106236775  0.0115953527
## [3,] -0.007138655  0.01062368  0.0371977529  0.0008034202
## [4,] -0.010119077  0.01159535  0.0008034202  0.0370047360
```

```
adaIS_singleQuantile(test_y, test_X, test_tau, 1, locs, m, 22)
```

```
## $mu
##         X1         X2         X3         X4
## -0.1749607  2.2197797 -1.1171291  1.5834106
##
## $S
##               [,1]          [,2]          [,3]          [,4]
## [1,]  2.789442e-34  8.435732e-34  6.870978e-34 -2.399906e-34
## [2,]  8.435732e-34  3.417594e-33  1.695591e-33 -5.797144e-34
## [3,]  6.870978e-34  1.695591e-33  1.987767e-33 -7.873848e-34
## [4,] -2.399906e-34 -5.797144e-34 -7.873848e-34  4.381389e-34
```

So everything seems to be working, though it does need a little tuning/refinement.

Now, let's put everything together and conduct a more thorough simulation study.

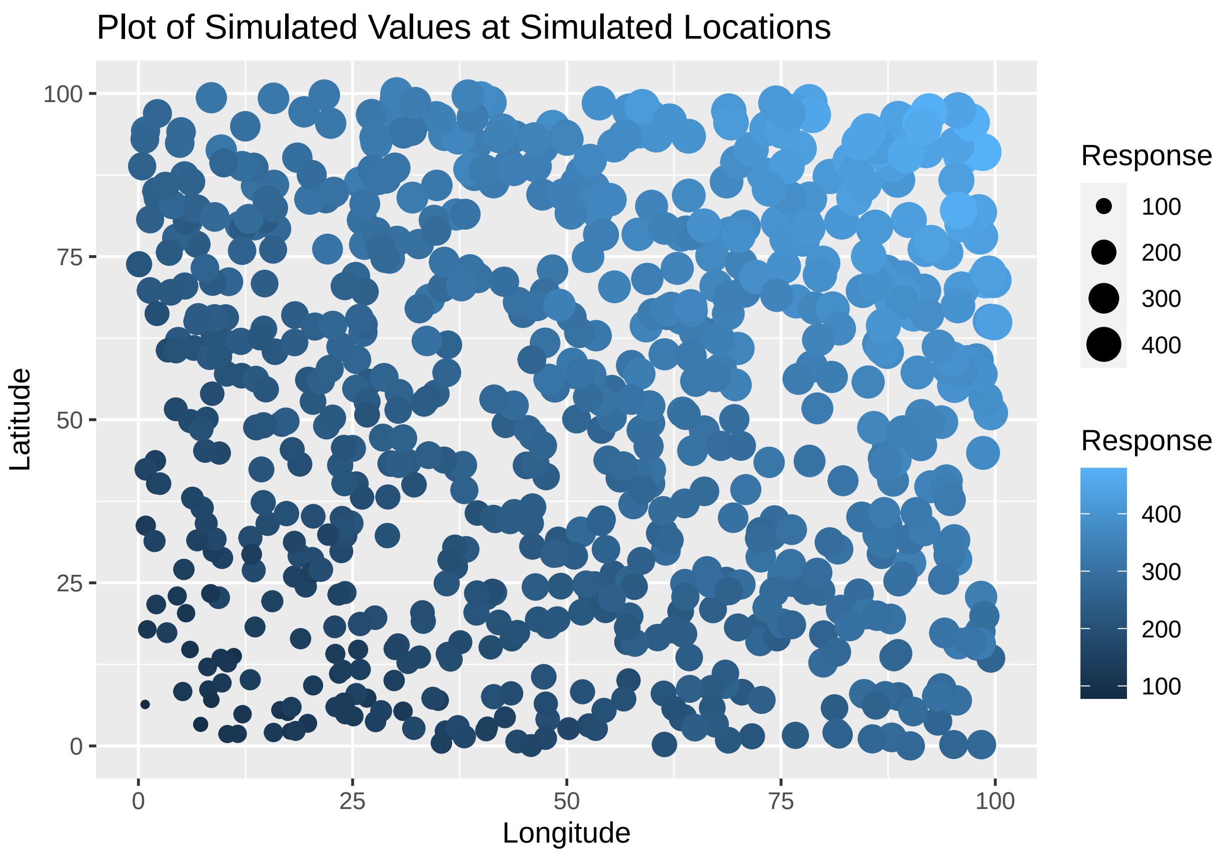
```
## First, let's simulate the locations
n <- 1000
locs <- as.data.frame(cbind(runif(n, 0, 100), runif(n, 0, 100)))
colnames(locs) <- c("long", "lat")

## Next, we will simulate the spatially dependent values
sim_mod <- RMexp(var = 4, scale = 3) +
  RMnugget(var = .75) +
  RMtrend(mean = 1.5)
sim_vals <- RFsimulate(sim_mod, x = locs$long, y = locs$lat)

## Now, we will simulate the values of the response and the covariates
beta <- c(2, 2, 5, 5, 5, 5)
X_sim <- cbind(rnorm(n, 10, 2), rexp(n, 5), rbeta(n, 5, 1), runif(n, 2.5, 7.5))
y_sim <- beta[1]*locs$long + beta[2]*locs$lat +
  beta[3]*X_sim[,1] + beta[4]*X_sim[,2] + beta[5]*X_sim[,3] + beta[6]*X_sim[,4]

## Next, we will split the data into training and test sets (75% of simulated values will be allocated
train_indices <- sample(1:n, round(.75*n), replace = FALSE)
X_train <- X_sim[train_indices,]
X_test <- X_sim[-train_indices,]
y_train <- y_sim[train_indices]
y_test <- y_sim[-train_indices]
locs_train <- locs[train_indices,]
locs_test <- locs[-train_indices,]

## Now, to plot the training data
sim_data <- as.data.frame(cbind(y_train, locs_train))
sim_plot <- ggplot(data = sim_data, aes(x = long, y = lat)) + geom_point(aes(col = y_train, size = y_tra
  labs(title = "Plot of Simulated Values at Simulated Locations",
       x = "Longitude",
       y = "Latitude",
       col = "Response",
       size = "Response")
plot(sim_plot)
```



```r
## Now that we have the simulated data, we can check out the performance of our functions for a variety
tau <- c(.01, .05, .1, .25, .5, .75, .9, .95)
b <- beta[3:length(beta)]

score_mat <- matrix(nrow = length(b), ncol = length(tau))
for(i in 1:ncol(score_mat)){
  score_mat[,i] <- get_score(y_train, X_train, tau[i], b)
}
score_mat
```

```
##            [,1]       [,2]      [,3]       [,4]       [,5]      [,6]      [,7]
## [1,] 74.572663 372.863316 745.72663 1864.31658 3728.63316 5592.9497 6711.5397
## [2,]  1.474357   7.371785  14.74357   36.85893   73.71785  110.5768  132.6921
## [3,]  6.219777  31.098887  62.19777  155.49443  310.98887  466.4833  559.7800
## [4,] 37.729617 188.648083 377.29617  943.24042 1886.48083 2829.7212 3395.6655
##           [,8]
## [1,] 7084.4030
## [2,]  140.0639
## [3,]  590.8789
## [4,] 3584.3136
```

```r
likelihood_vec <- rep(NA, length(tau)) ## THERE'S AN ISSUE HERE
for(i in 1:length(tau)){
  likelihood_vec[i] <- get_likelihood(y_train, X_train, tau[i], b, locs_train, 1)
}
likelihood_vec
```

```
## [1] 0 0 0 0 0 0 0 0
```

```r
test_draw <- rmvnorm(n, b, cov(X_train))
mu_mat <- matrix(nrow = length(b), ncol = length(tau))
S_list <- list()
param_list <- list(mu_mat, S_list)
names(param_list) <- c("mu", "S")
for(i in 1:length(tau)){
  x <- get_updated_params(y_train, X_train, tau[i], b, cov(X_train), locs_train, test_draw)
  param_list$mu[,i] <- x$mu
  param_list$S[[i]] <- x$S
}
param_list
```

```
## $mu
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
##
## $S
## $S[[1]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[2]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[3]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[4]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[5]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
```

```
##
## $S[[6]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[7]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
##
## $S[[8]]
##      [,1] [,2] [,3] [,4]
## [1,]  NaN  NaN  NaN  NaN
## [2,]  NaN  NaN  NaN  NaN
## [3,]  NaN  NaN  NaN  NaN
## [4,]  NaN  NaN  NaN  NaN
```

Issues so Far

So it seems like the issue with the `adaIS_singleQuantile()` method is with the number of iterations of the IS, not the number of draws on each iteration. I was able to run the IS with 1000000 draws per iteration for 10 iterations and it ran just fine. (Generally, we seem to be able to make the number of draws as large as we want as long as the number of iterations is relatively low.) However, when we increase the number of iterations anywhere above ~13, we get the following error:

```
Error in eigen(sigma, symmetric = TRUE) : infinite or missing values in 'x'
```

I'm not entirely sure what is causing this, but I have two theories:

One, it could have something to do with the calculations for the posterior covariance matrix. I mentioned earlier, when we swap the weight matrix W for the sample spatial Mahalanobis distance in the working likelihood, the formula for the posterior covariance given in the paper no longer yields a positive definite covariance matrix. I found a work around to get the IS to work in the short term, but it doesn't take into account the importance weights.

Two, this could actually be because we swapped the weight matrix for the sample spatial Mahalanobis distance. The paper specifies that W must be positive definite, but the sample Mahalanobis distance is only positive *semi*-definite. That may explain why it doesn't work for larger iterations (eventually, the IS just randomly encounters the "semi-definite" case in which one of the Eigenvalues is effectively zero and it can't perform the necessary computations).

Because the likelihood function $L(Y|X,\beta)$ is defined with respect to the score function $s_\tau(\beta)$ (that is, $L(Y|X,\beta) = C\exp\left(-\frac{1}{2n}s_\tau(\beta)^T(X-\hat{\mu})^T\Sigma^{-1}(X-\hat{\mu})s_\tau(\beta)\right)$ and the score function is the gradient of the log-likelihood with respect to the parameters, we will need to solve a differential equation to get a functional form for the score function and likelihood.

UPDATE:

I think the use of the term "score function" here may be a little misleading. Though I've never seen it defined in this way, I found this slide deck developed by Koenker himself which sheds a little light on the subject. After going through this (especially slide 15), I don't think $s_\tau(\beta)$ is the traditional "score" function (i.e; the gradient of the log-likelihood). It seems like when they said "score function" they meant "*rank*score function", which is another way of defining quantiles. It seems that the rankscore function actually comes from the quantile regression objective function $\hat{\beta}(\tau) = \arg\min_\beta \sum_{i=1}^n \rho_\tau(y_i - x_i^T\beta)$, there's no need to solve any sort of more complicated differential equation since the likelihood is no longer a function of its own derivative.

Udates & Suggestions: 22 November

On slide 18 of Lec3_ContinuousVariation_IntroGP.pdf, Dr. Sang states that the pairwise Mahalanobis distance between two points s and s' is given by $d = \sqrt{(s-s')^T U^T D^{-2} U (s-s')}$ when we have an anisotropic covariance structure that we want to stabilize by rotating and scaling the axes. The Mahalanobis structure we have right now is a little different from this, because we use the column centered feature matrix instead of the matrix of pairwise distances between observations. (That is, we use $(X-\mu)$ instead of $(s-s')$.) If we were to instead use the matrix of pairwise distances (I'd say Euclidean, but Manhattan could be fitting since we're working with data from New York), we would then have an entirely spatial Mahalanobis distance. However, to accomodate this we would probably also need to change the score function but we could make that reflect the percentiles of the locations as opposed to the features (so instead of return a px1 vecotr whose entries correspond to the each feature, the score would return a 2x1 vector whose entries coorespond to the percentile of the location[s]).

Also, did I do that simulation study correctly? I know the spatial variation and nugget effect are right, but do I need to do anything else to make sure the dependence oif the features on the locations is correct?

We should also add `predict` function that will make predictions for observations made at specified locations. I'll need to come up with something for the simulation study, but we can try to improve on this later.

We should also probably find a way to select an appropriate value of C. I've been using 1 so far because it's a nice value that won't throw errors, but there might be a constant that gives better results.

There's a problem with the likelihood function. It just keeps returning zero. I know the likelihood should be small, but should it be this small?