Methods

Marshall Honaker

First, let's import the necessary packages:

```
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
```

```
library(RandomFields)
```

```
## Loading required package: sp
```

```
## Loading required package: RandomFieldsUtils
```

```
##
## Attaching package: 'RandomFields'
```

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

```
library(mvtnorm)
```

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.

```
get_score <- function(y, X, tau, beta){
  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.

```
get_spatial_covar_mat <- function(locs){
  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.*

```
get_likelihood <- function(y, X, tau, beta, locs, C){
  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)%*%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.

```
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)'.
```

```
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

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

```
##             [,1]
## [1,]  0.7718218
## [2,] -0.9441975
## [3,] -2.3581800
## [4,]  1.3074811
```

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

```
## [1] 0.01946928
```

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.

```
get_updated_params <- function(y, X, tau, beta, Sigma, locs, draw){
  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] <- mean((draw[,i] - mu_hat[i])*(draw[,j] - mu_hat[j]))
    }
  }
  temp <- list(mu_hat, S)
  names(temp) <- c("mu", "S")
  return(temp)
}

adaIS_singleQuantile <- function(y, X, tau, C, locs, M, num_reps){
  p <- ncol(X)
  n <- length(y)
  beta <- coef(lm(y ~ X))[-1]
  S <- cov(X)

  initial_draw <- rmvnorm(M, beta, S)

  params <- get_updated_params(y, X, tau, beta, S, locs, initial_draw)

  for(i in 1:num_reps){
    draws <- rmvnorm(M, params$mu, params$S)
    params <- get_updated_params(y, X, tau, params$mu, params$S, locs, draws)
  }
```

```
  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] 2.190507 1.482056 3.045611 5.253151
##
## $S
##             [,1]        [,2]        [,3]       [,4]
## [1,]  2.54840782 -0.61966840 -0.02504194  1.4124042
## [2,] -0.61966840  1.24333936 -0.05077152 -0.5390208
## [3,] -0.02504194 -0.05077152  0.89467426  0.1170340
## [4,]  1.41240417 -0.53902080  0.11703397  2.5330805
```

```
adaIS_singleQuantile(test_y, test_X, test_tau, 1, locs, m, 10)
```

```
## $mu
##        X1        X2        X3        X4
##  8.109002  8.682633 -1.355163  6.264647
##
## $S
##           [,1]      [,2]      [,3]      [,4]
## [1,] 573.69387 451.89317 -89.65611 374.67374
## [2,] 451.89317 385.66199 -68.53768 319.44452
## [3,] -89.65611 -68.53768  31.94909 -65.04507
## [4,] 374.67374 319.44452 -65.04507 274.41537
```

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

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).