pm Example

Qi Wang

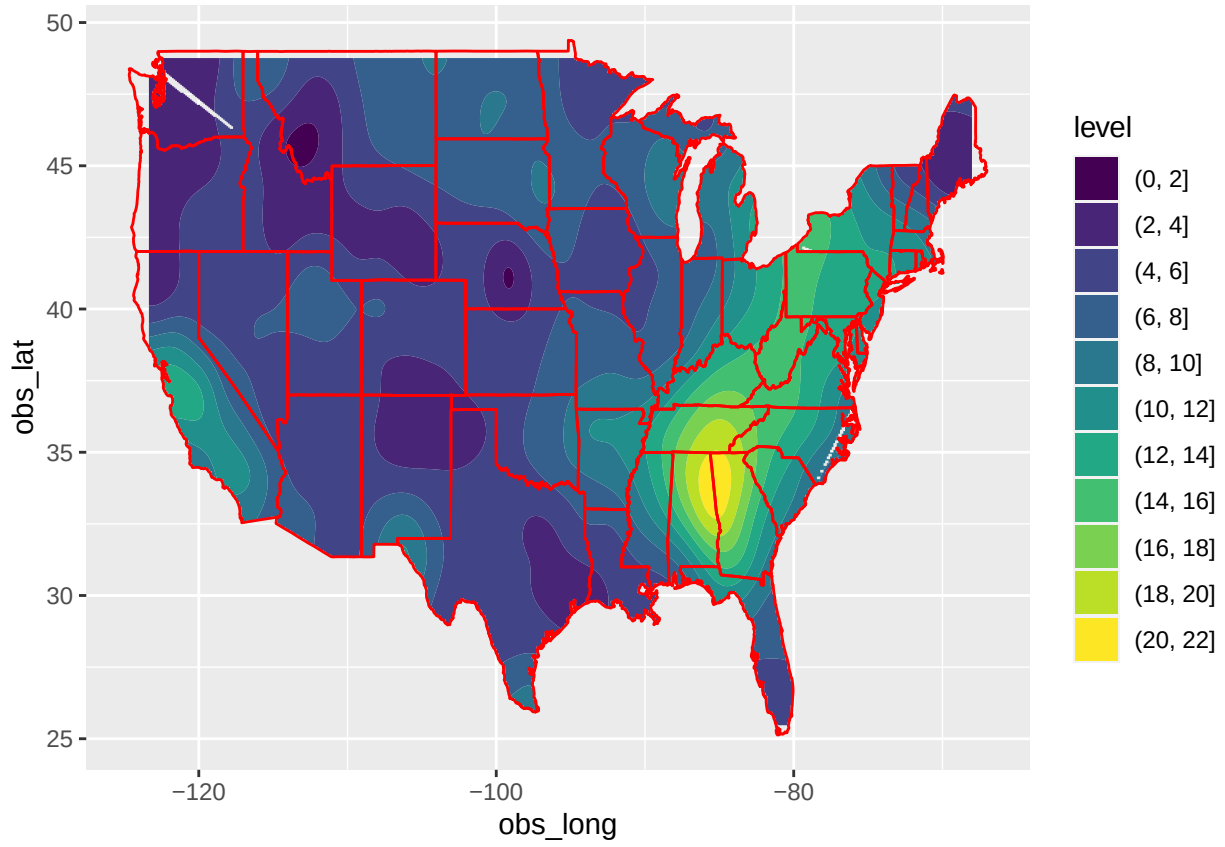
2023/4/3

Read the data and visualization

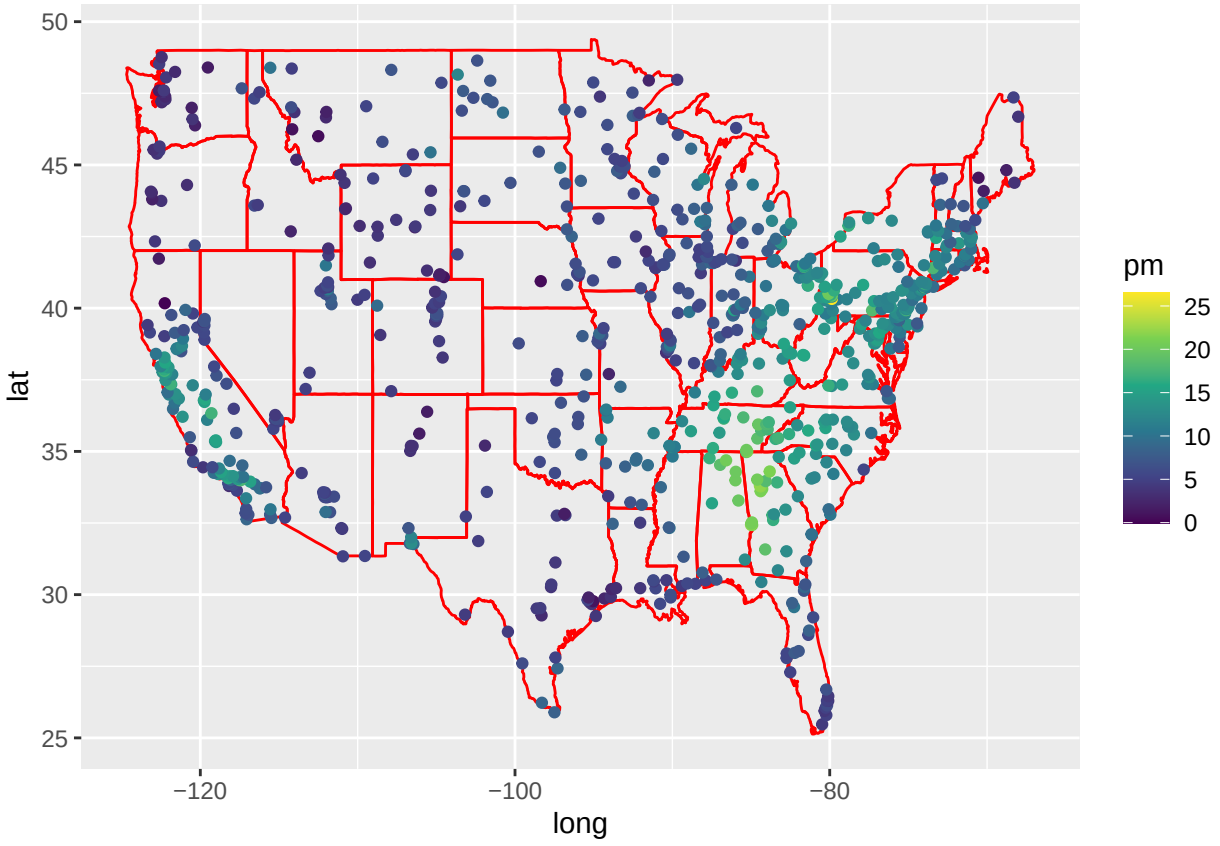
```
pm_dat <- read.csv(here::here("pm25_0605.csv"), header = T)
long <- pm_dat$Longitude
lat <- pm_dat$Latitude
pm <- pm_dat$PM25
dat_pm <- as.data.frame(cbind(long, lat, pm))
coordinates(dat_pm) <- ~ long + lat
```

```
us_map <- map_data("state", region=c("alabama", "arizona", "arkansas", "california", "colorado", "conne
                                     "delaware", "florida", "georgia", "idaho", "illinois", "indiana",
                                     "iowa", "kansas", "kentucky", "louisiana", "maine", "maryland",
                                     "massachusetts", "michigan", "minnesota", "mississippi", "missouri
                                     "montana", "nebraska", "nevada", "new hampshire", "new jersey",
                                     "new mexico", "new york", "north carolina", "north dakota", "ohio"
                                     "oklahoma", "oregon", "pennsylvania", "rhode island", "south caroli
                                     "south dakota", "tennessee", "texas", "utah", "vermont", "virginia
                                     "washington", "west virginia", "wisconsin", "wyoming"))

states = c("Alabama", "Arizona", "Arkansas", "California", "Colorado", "Connecticut",
           "Delaware", "Florida", "Georgia", "Idaho", "Illinois", "Indiana",
           "Iowa", "Kansas", "Kentucky", "Louisiana", "Maine", "Maryland",
           "Massachusetts", "Michigan", "Minnesota", "Mississippi", "Missouri",
           "Montana", "Nebraska", "Nevada", "New Hampshire", "New Jersey",
           "New Mexico", "New York", "North Carolina", "North Dakota", "Ohio",
           "Oklahoma", "Oregon", "Pennsylvania", "Rhode Island", "South Carolina",
           "South Dakota", "Tennessee", "Texas", "Utah", "Vermont", "Virginia",
            "West Virginia", "Wisconsin", "Wyoming", "Washington")


coords <- cbind(long, lat)
x.res <- 500
y.res <- 500
surf <- mba.surf(cbind(coords, pm), no.X = x.res, no.Y = y.res, h = 5, m = 2, extend = TRUE)$xyz.est

all_inside <- NULL
exp_grid <- expand.grid(surf$x, surf$y)
for (i in 1:length(states)) {
  tem = spBayes::pointsInPoly(as.matrix(map_data("state", region = states[i])[,1:2]),as.matrix(exp_grid
```

```
  all_inside <- unique(c(all_inside, tem))

}
obs_pm <- surf$z[all_inside]
obs_long <-  as.matrix(exp_grid)[all_inside,1]
obs_lat <-  as.matrix(exp_grid)[all_inside,2]
p1 <-
ggplot() +
  geom_contour_filled(aes(x = obs_long, y = obs_lat, z = obs_pm))+
   geom_path(data = us_map, aes(x = long, y = lat, group = group), color = "red")
p2 <-
ggplot() +
  geom_path(data = us_map, aes(x = long, y = lat, group = group), color = "red") +
  geom_point(aes(x = long, y = lat, color = pm)) +
  scale_color_viridis_c()

p1
```



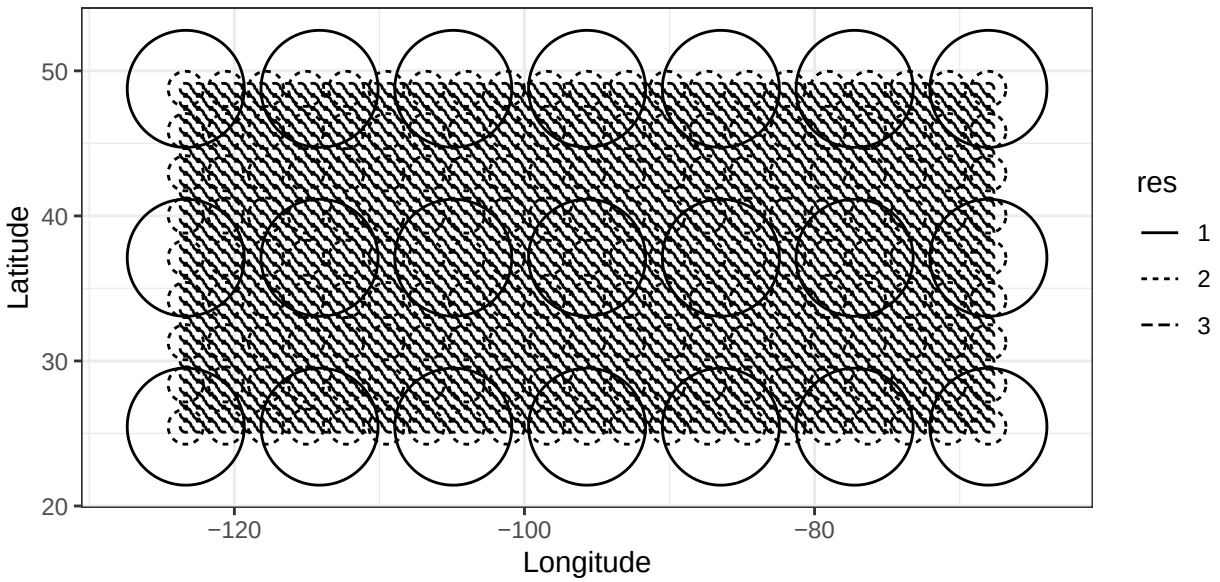

```
p2
```



Basis Function Generation

```
gridbasis1 <- auto_basis(mainfold = plane(), data = dat_pm, nres = 1, type = "Gaussian", regular = 1)
gridbasis2 <- auto_basis(mainfold = plane(), data = dat_pm, nres = 2, type = "Gaussian", regular = 1)
gridbasis3 <- auto_basis(mainfold = plane(), data = dat_pm, nres = 3, type = "Gaussian", regular = 1)

show_basis(gridbasis3) +
  coord_fixed() +
  xlab("Longitude") +
  ylab("Latitude")
```

```
## Note: show_basis assumes spherical distance functions when plotting
```




```
basis_1 <- matrix(NA, nrow = nrow(dat_pm), ncol = length(gridbasis1@fn))
for (i in 1:length(gridbasis1@fn)) {
  basis_1[,i] <- gridbasis1@fn[[i]](coordinates(dat_pm))
}

basis_2 <- matrix(NA, nrow = nrow(dat_pm), ncol = length(gridbasis2@fn))
for (i in 1:length(gridbasis2@fn)) {
  basis_2[,i] <- gridbasis2@fn[[i]](coordinates(dat_pm))
}

basis_3 <- matrix(NA, nrow = nrow(dat_pm), ncol = length(gridbasis3@fn))
for (i in 1:length(gridbasis3@fn)) {
  basis_3[,i] <- gridbasis3@fn[[i]](coordinates(dat_pm))
}


basis_use <- basis_3[,-(1:ncol(basis_2))]

depth <- 3
shape_row <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 2 ]))
shape_col <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 1 ]))
basis_arr <- array(NA, dim = c(nrow(dat_pm), shape_row, shape_col))
for (i in 1:nrow(dat_pm)) {
  basis_arr[i,,] <- matrix(basis_use[i,], nrow = shape_row, ncol = shape_col, byrow = T)
}
```

Deep Kriging Model

```
set.seed(0)
train_index <- sample(1:nrow(pm_dat), 600, replace = FALSE)
```

```
basis_tr <- basis_arr[train_index,,]
basis_te <- basis_arr[-train_index,,]


x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), shape_row*shape_col))  # So we want to reshape each o
x_te <- array_reshape(basis_te, c(nrow(basis_te), shape_row*shape_col))  # Same as prervious step

pm_tr <- pm[train_index]
pm_te <- pm[-train_index]


model_dk <- keras_model_sequential()

model_dk %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(ncol(x_tr))) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')

# Compile the model

model_dk %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)

mod_train_dk <- model_dk %>%
  fit(x = x_tr, y = pm_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_dk <- model_dk %>%
          evaluate(x_te, pm_te)
```

Convolutional Kriging Model

```
# Define a few parameters to be used in the CNN model
batch_size <- 128
epochs <- 50

# Input image dimensions
img_rows <- shape_row
img_cols <- shape_col
```

```
x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), img_rows, img_cols, 1))
x_te <- array_reshape(basis_te, c(nrow(basis_te), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)
```

```
model_ck <- keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu', input_shape = input_shape) %>%
  #layer_conv_2d(filters = 32, kernel_size = c(2,2), activation = 'relu') %>%
  #layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')


model_ck %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)


mod_train_ck <- model_ck %>%
  fit(x = x_tr, y = pm_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_ck <- model_ck %>%
          evaluate(x_te, pm_te)
```

```
rbind(loss_dk, loss_ck)
```

```
##             loss      mse
## loss_dk 5.160487 5.160487
## loss_ck 4.896048 4.896048
```