STDA Homework 1

Seokjun Choi

April 10, 2020

Note:
You can get full, run-able code files at my github page: Visit https://github.com/letsjdosth/SpaTempoDA
Because I provide the full code file separately, in this report I will show key-code blocks only instead of bringing the full, verbose code.

1 Problem 1

Suppose we want to simulate a random vector $Y \sim N(\mu, \Sigma)$. If Σ (Matern) is symmetric and positive definite, it can be represented using the Cholesky decomposition $\Sigma = LL'$, where L is a lower triangular matrix. Consider the following algorithm for simulating Y:

- **Calculate the matrix L.**
- **Sample $Z \sim N(0, I)$, where I is the $n \times n$ identity matrix.**
- **Let $Y = \mu + LZ$**

1.1 Problem 1-(a)

Show that Y generated in this way has the correct distribution. You may use the fact that a linear function of a multivariate normal random variable is again multivariate normal; just show the mean and variance are correct.

From the elementary observation from mathematical statistics, we have for $a, b \in \mathcal{R}$ and for a random variable X,

$$E[aX + b] = aE[X] + b$$

$$Var[aX + b] = a^2 Var[X]$$

And its multivariate version is, for proper $A \in \mathcal{R}^{n \times n}$, $b \in \mathcal{R}^n$, and for a random vector X,

$$E[Ax + b] = AE[X] + b$$

$$Var[AX + b] = AVar[X]A'$$

Then, using this observation and the fact at this problem, we can get

$$Y \sim N(\mu + 0, LIL')$$

Since $LL' = \Sigma$ by our setting, we get $Y \sim N(\mu, \Sigma)$, so we can confirm that the algorithm is valid.

1.2 Problem 1-(b)

Write a function or a few lines of code in R to implement this method for arguments mu and Sigma. You may use the built-in function chol for the Cholesky decomposition and rnorm to generate Z.

Here is implementation of the key part.

```
#R code

rnorm_chol_algorithm = function(mu, Sigma) {
    # mu is vector
    # Sigma should be symmetric, positive definite matrix
    dimension = dim(Sigma)
    vecZ = rnorm(dimension, 0, 1)
    chol_upperL = chol(Sigma)
    return(mu + t(chol_upperL) %*% vecZ)
}
```

1.3 Problem 1-(c)

For a mean and covariance function of your choosing, use your code from (b) and make a few plots illustrating realizations of a Gaussian process on [0;1], but changing the different parameters in the model. These differences will be easier to see if you keep the same Z sample but just change mu and Sigma.

For convenience, I use fields$Matern function to calculate covariance matrix from parameters of Matern. And for easy visualization, I will make only 2 dimensional chain. Here are 3 test cases. (In full code, there is one more case for ordinary(not Matern) GP.)

Fix ρ to 1. And,

- $\nu = 0.1$ (in code, test 3) The covariance matrix becomes

$$\begin{bmatrix} 1 & 0.98 \\ 0.98 & 1 \end{bmatrix}$$

- $\nu = 0.5$ (in code, test 2) : exponential covariance. The covariance matrix becomes

$$\begin{bmatrix} 1 & 0.90 \\ 0.90 & 1 \end{bmatrix}$$

- $\nu = 1$ (in code, test 4) The covariance matrix becomes

$$\begin{bmatrix} 1 & 0.81 \\ 0.81 & 1 \end{bmatrix}$$

I generate 5 (different) chains at each νs.

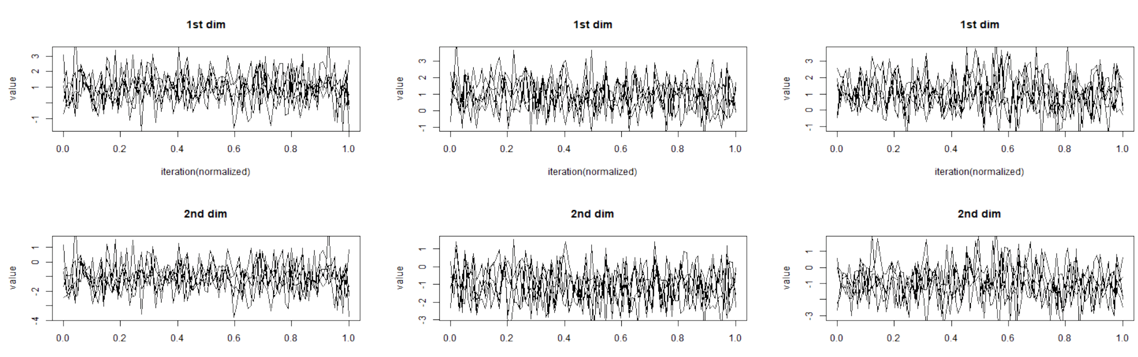

Figure 1: left: $\nu = 0.1$, middle: $\nu = 0.5$, right: $\nu = 1$

Nextly, I generate one chain longer, and get scatter-plot and trace-plot.

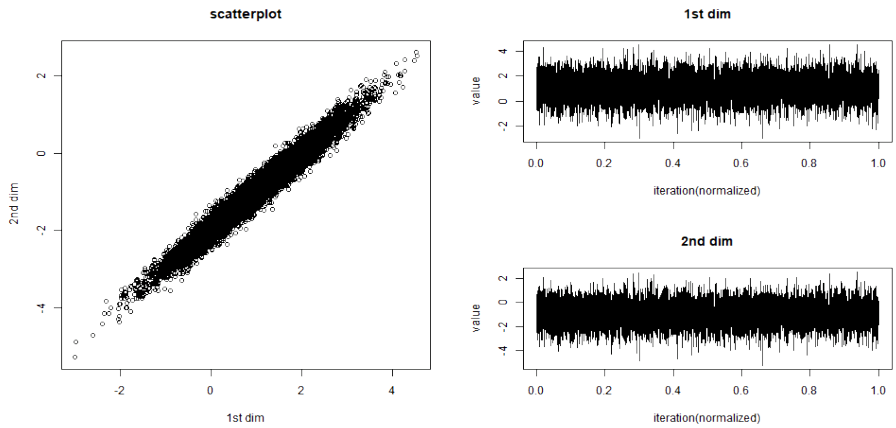

Figure 2: $\nu = 0.1$

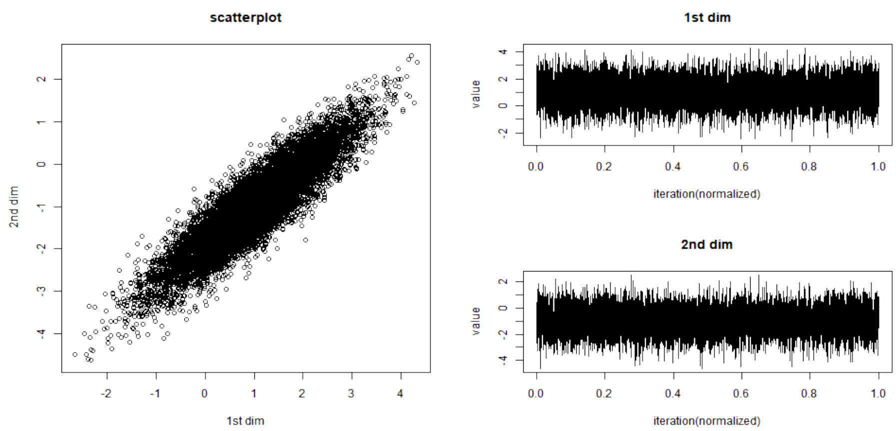

Figure 3: $\nu = 0.5$

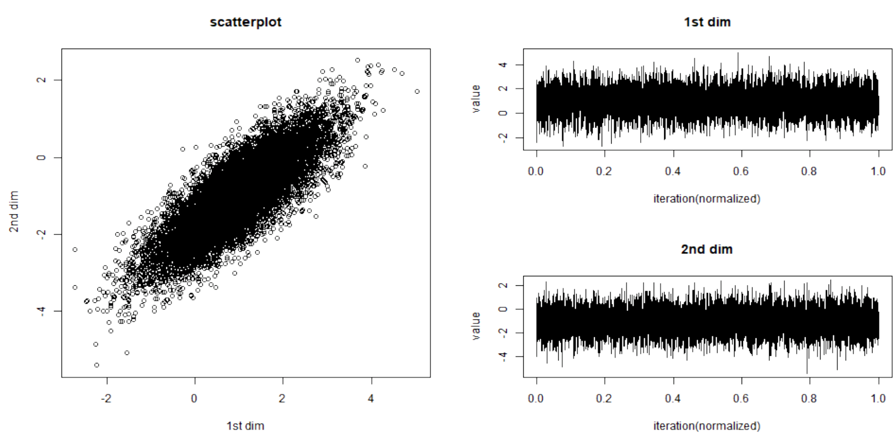

Figure 4: $\nu = 1$

By changing ν, the calculated covariance matrices are affected, and the effects are confirmed by the scatter-plot and trace-plot.

2 Problem 2

The file CAtemps.RData contains two R objects of class SpatialPointsDataFrame, called CAtemp and CAgrid. CAtemp contains a average temperatures from 1961-1990 at 200 locations (latitude and longitude) in California in degrees Fahrenheit, along with their elevation in meters. CAgrid contains elevations in meters over a grid of locations. I've given you some code to get started with this data in HW1.R Consider the following model for the temperature data.

$$Y_i = \mu(s_i; \beta) + e(s_i; \sigma^2, \rho, \tau)$$

where $\mu(s_i; \beta) = \beta_0 + \beta_1 Longitude(s) + \beta_2 Latitude(s) + \beta_3 Elevation(s)$ **and** $e(s_i; \sigma^2, \rho, \tau)$ **is a zero mean stationary Gaussian process with exponential covariance function. Another way of writing this is as**

$$Y_i = \mu(s_i; \beta) + Z(s_i; \sigma^2, \rho) + \epsilon_i$$

where now Z **is mean zero Gaussian process like** e **but without the nugget term, and the** ϵ **are iid** $N(0, \tau^2)$**, independent of Z. This is important because we want to predict** $\mu(s_i; \beta) + Z(s_i; \sigma^2, \rho)$ **without the measurement error.**

Before solving the problems, Let's see the data. Here is average temperature plot of CAtemp. We can

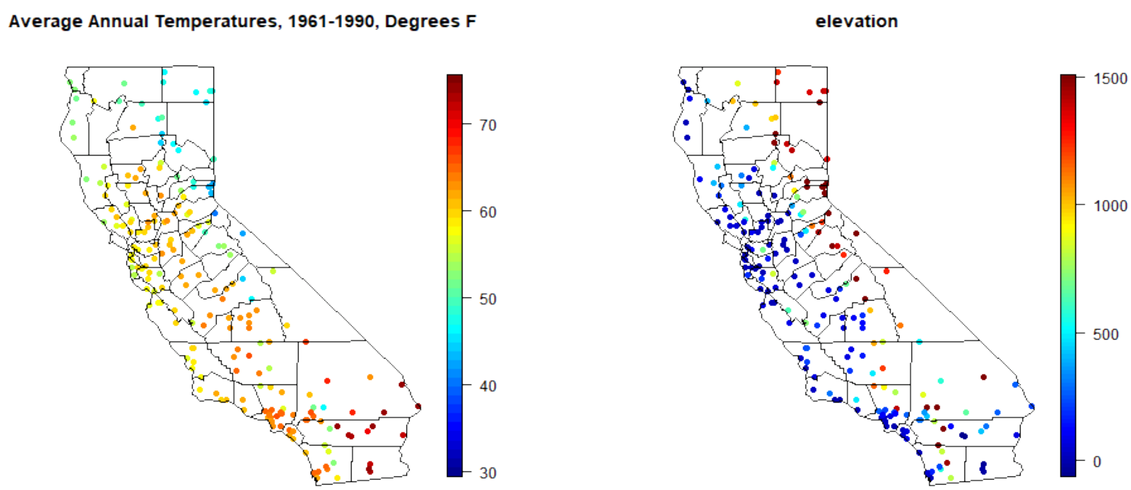

Figure 5: left: average temperature data in CAtemp, right: elevation data in CAtemp

find that data points are relatively sparse at upper-right, middle-right and lower-right region.

2.1 Problem 2-(a)

Using the CAtemp data, form a preliminary estimate of β using ordinary least squares and make a color plot of the residuals. Include your estimates and plot. It is easy work if we use built-in lm function of R. Load the CAtemps.Rdata and run this code to get what we want.

#R code

```
ols = lm(avgtemp~lon+lat+elevation, data=CAtemp)
print(ols$coeff) #estimates
CAtemp$ols.residual = ols$residual
```

The results of the ordinary least squares' coefficients calculated by R are

$$\hat{\beta}_{0:OLS} = 321.5114$$

$$\hat{\beta}_{1:OLS} = 2.324105$$

$$\hat{\beta}_{2:OLS} = 0.5646805$$

$$\hat{\beta}_{3:OLS} = -0.009648649$$

where $\mu(s_i;\beta) = \beta_0 + \beta_1 Longitude(s) + \beta_2 Latitude(s) + \beta_3 Elevation(s)$.

And, the residual at each data point is plotted.

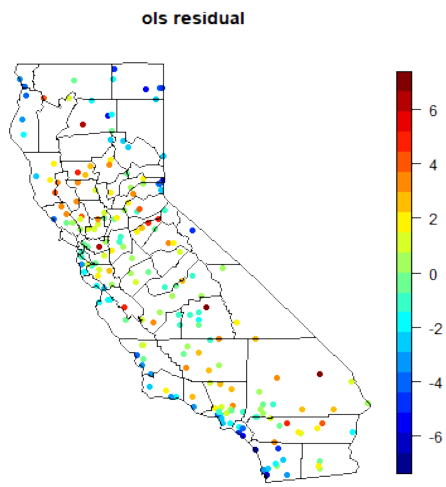

Figure 6: the residuals of OLS

Be careful, 0 is colored by light-green.

We can find that there are some patterns in residuals. So we have to doubt to exist some dependencies in this data.

2.2 Problem 2-(b)

Estimate the variogram non-parametrically and then fit the exponential variogram to it using weighted least squares. Make and include a plot of the nonparametric and parametric variogram functions. Also store your parameter estimates and report them.

For variogram, I use the 'variogram' function of gstat package in R. Then, the code become (magically) simple. Here is the key code to get non-parametric variogram. I choose the width to 50.

```
#R code

vg = variogram(ols.residual ~ 1, data = CAtemp, width=50)
print(plot(vg, xlab = "Distance", ylab = "Semi-variogram estimate"))

vgangle = variogram(ols.residual ~ 1, data = CAtemp, alpha = c(0, 45, 90, 135))
print(plot(vgangle, xlab = "Distance", ylab = "Semi-variogram estimate"))
```

(The print function call is needed to save the plot when I run this code in the console, not in the interactive interpreter. So, you may ignore it.)

The plots are here.

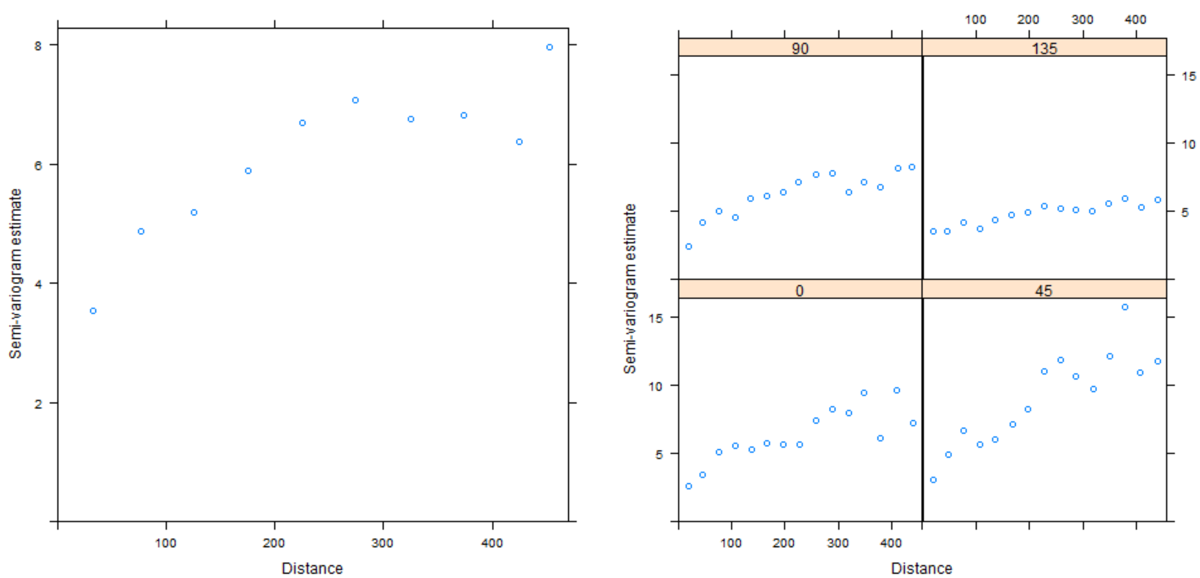

Figure 7: nonparametric variogram fit

The brief shape of variogram is not a flat line, so the doubt above is proper as we see here, too. And, since the shapes of the rotated variograms also have same shapes, the result doesn't have a big problem.

Next, I will fit the variogram to exponential variogram model. Again, the gstat package in R make our problem very simple. I will use 'fit.variogram' function in the package.

```
#R code

fitvg = fit.variogram(vg, vgm(1, "Exp", range=300, nugget=3))

# estimated parameters of variance term
s2.hat = fitvg$psill[2]
rho.hat = fitvg$range[2]
tau2.hat = fitvg$psill[1]
```

```
# plot
print(plot(vg, fitvg, xlab = "Distance", ylab = "Semi-variogram estimate"))
```

Below is the result plot.

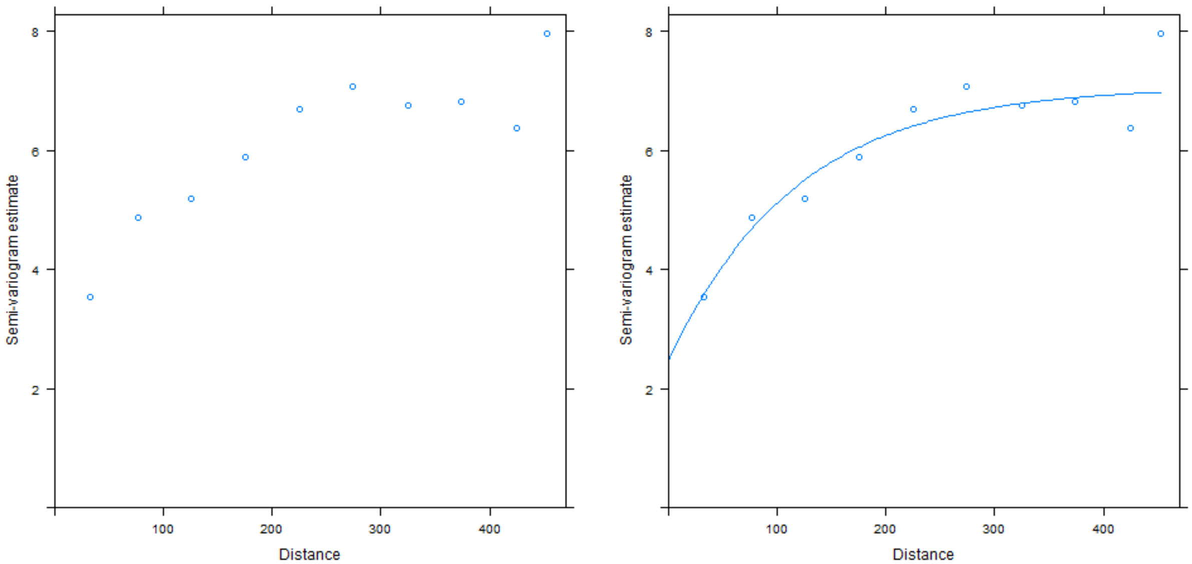

Figure 8: the variogram fitted to the exponential model

And, the estimated parameter values of covariance's using the least square method are,

$$\hat{\sigma}^2 = 4.635655$$

$$\hat{\rho} = 115.7653$$

$$\hat{\tau}^2 = 2.434879$$

2.3 Problem 2-(c)

We will now form the GLS estimate of β by hand, rather than using the gls function.

- **Use the rdist function in fields to create a matrix of distance (in miles) between pairs of locations in CAtemp.**
- **Create the covariance matrix, plugging in your estimates from the fitted variogram. (Hint: Sum tow matrix, one without a nugget and one using the diag function to create the matrix $\tau^2 I$)**
- **Invert the covariance matrix and store it for later reference.**
- **Create the X matrix (Hint: Use cbind.)**
- **Put all the pieces together to form $\hat{\beta}_{GLS}$.**

I implement the code of GLS by hand.

To get miles-distance matrix from (lat,lon) data, I use the chordal distance. And, following the direction of problem, I use exponential covariance function to make the covariance matrix.

#R code

```
make_chordaldist_u1u2_for_rdist = function(lat, lon){
    mean_earth_radius_in_miles = 3958.8 #by google
    chordaldist_x = mean_earth_radius_in_miles * cos(lat) * cos(lon)
    chordaldist_y = mean_earth_radius_in_miles * cos(lat) * sin(lon)
    chordaldist_z = mean_earth_radius_in_miles * sin(lat)
    chordaldist_mat = cbind(chordaldist_x, chordaldist_y, chordaldist_z)
    return(chordaldist_mat)
}

make_EXPcov_mat_without_nugget = function(dist_mat, rho, sigma_sqaure){
    #lecture note 1-2. page 40
    num_row = dim(dist_mat)[1]
    num_col = dim(dist_mat)[2]
    cov_gamma = matrix(0, num_row, num_col)
    for(i in 1:num_row){
        for(j in 1:num_col){
            cov_gamma[i,j] = exp(-dist_mat[i,j] * rho)
        }
    }
    cov_gamma = cov_gamma * sigma_sqaure
    return(cov_gamma)
}

# make distance matrix
CAtemp_data_u1u2= make_chordaldist_u1u2_for_rdist(CAtemp$lat, CAtemp$lon)
CAtemp_data_dist_mat = rdist(CAtemp_data_u1u2, CAtemp_data_u1u2)
# dim(dist_mat) # 200 200
# max(dist_mat)
# min(dist_mat)


# make covariance matrix
CAtemp_data_cov_spatial = make_EXPcov_mat_without_nugget(CAtemp_data_dist_mat, rho.hat, s2.hat)
CAtemp_data_cov_nugget = diag(200) * tau2.hat
CAtemp_data_cov_mat = CAtemp_data_cov_spatial + CAtemp_data_cov_nugget
```

```
# gls fit
b0 = rep(1,200)
gls.X = as.matrix(data.frame(b0, CAtemp$lon, CAtemp$lat, CAtemp$elevation))
gls.Y = as.matrix(CAtemp$avgtemp)
CAtemp_data_inv_cov_mat = solve(CAtemp_data_cov_mat)
gls.beta = solve(t(gls.X) %*% CAtemp_data_inv_cov_mat %*% gls.X) %*% t(gls.X) %*% CAtemp_data_inv_cov_mat %*% gls
print("gls coefficients")
cat('(Intercept)    lon       lat    elevation\n',gls.beta,'\n')
print('----------------')
```

The results of the GLS' coefficients calculated by R are

$$\hat{\beta}_{0:GLS} = 321.5114$$

$$\hat{\beta}_{1:GLS} = 2.324105$$

$$\hat{\beta}_{2:GLS} = 0.5646805$$

$$\hat{\beta}_{3:GLS} = -0.009647649$$

where $\mu(s_i;\beta) = \beta_0 + \beta_1 Longitude(s) + \beta_2 Latitude(s) + \beta_3 Elevation(s)$.

They are not differ drastically, but maybe OLS model affects more to the standard error of the estimates comparing to OLS model.

2.4 Problem 2-(d)

Calculate and plot the EBLUP of $\mu + Z$ at the location in CAgrid, plugging in your estimates from (b) and (c). Calculate and plot the (estimated) standard error of Z at each prediction location

I calculate MSE rather than standard error, because their roles simillar, but it shows more interesting things than estimated standard error in this case.

I will re-use the calculated value and R-functions to get EBLUP and MSE. Key parts of the code are here.

#R code

```
# make m0(=pred_mu) term(in lec2-2. page38)
pred_X = as.matrix(data.frame(rep(1,664), CAgrid$lon, CAgrid$lat, CAgrid$elevation))
pred_mu = pred_X %*% gls.beta
dim(pred_mu)
head(pred_mu)


# make covariance matrix
#inner
CAgrid_data_u1u2= make_chordaldist_u1u2_for_rdist(CAgrid$lat, CAgrid$lon)
pred_inner_dist_mat = rdist(CAgrid_data_u1u2, CAgrid_data_u1u2)

pred_cov_spatial = make_EXPcov_mat_without_nugget(pred_inner_dist_mat, rho.hat, s2.hat)
# pred_cov_nugget = diag(664) * tau2.hat #no nugget term!
pred_inner_cov_mat = pred_cov_spatial #+ pred_cov_nugget #no nugget term!


#cross
pred_cross_dist_mat = rdist(CAtemp_data_u1u2, CAgrid_data_u1u2)
dim(pred_cross_dist_mat) # 200, 664
pred_cross_cov_mat = make_EXPcov_mat_without_nugget(pred_cross_dist_mat, rho.hat, s2.hat)


## krigging mean
pred_Y = pred_mu +
    t(pred_cross_cov_mat) %*% CAtemp_data_inv_cov_mat %*% (gls.Y - gls.X%*%gls.beta)
dim(pred_Y)
CAgrid$pred.temp = pred_Y

# sd (I'll skip this. Instead, I'll find mse.)
pred_Y_var_mat = pred_inner_cov_mat -
    (pred_cross_cov_mat) %*% CAtemp_data_inv_cov_mat %*% pred_cross_cov_mat
# dim(pred_Y_var_mat) #664 664
# pred_Y_var = diag(pred_Y_var_mat)
# CAgrid$pred.sd.temp = sqrt(pred_Y_var)

# predicting mse
cal_b = t(pred_X) - t(gls.X) %*% CAtemp_data_inv_cov_mat %*% pred_cross_cov_mat
pred_mse_mat = pred_Y_var_mat +
    t(cal_b) %*% solve(t(gls.X) %*% CAtemp_data_inv_cov_mat %*% gls.X) %*% cal_b
pred_mse = diag(pred_mse_mat)
CAgrid$pred.mse.temp = pred_mse
```

The result plots of predicted temperature values and MSE values for each location in CAgrid are here. For convenience, I attach original data plot one more time.

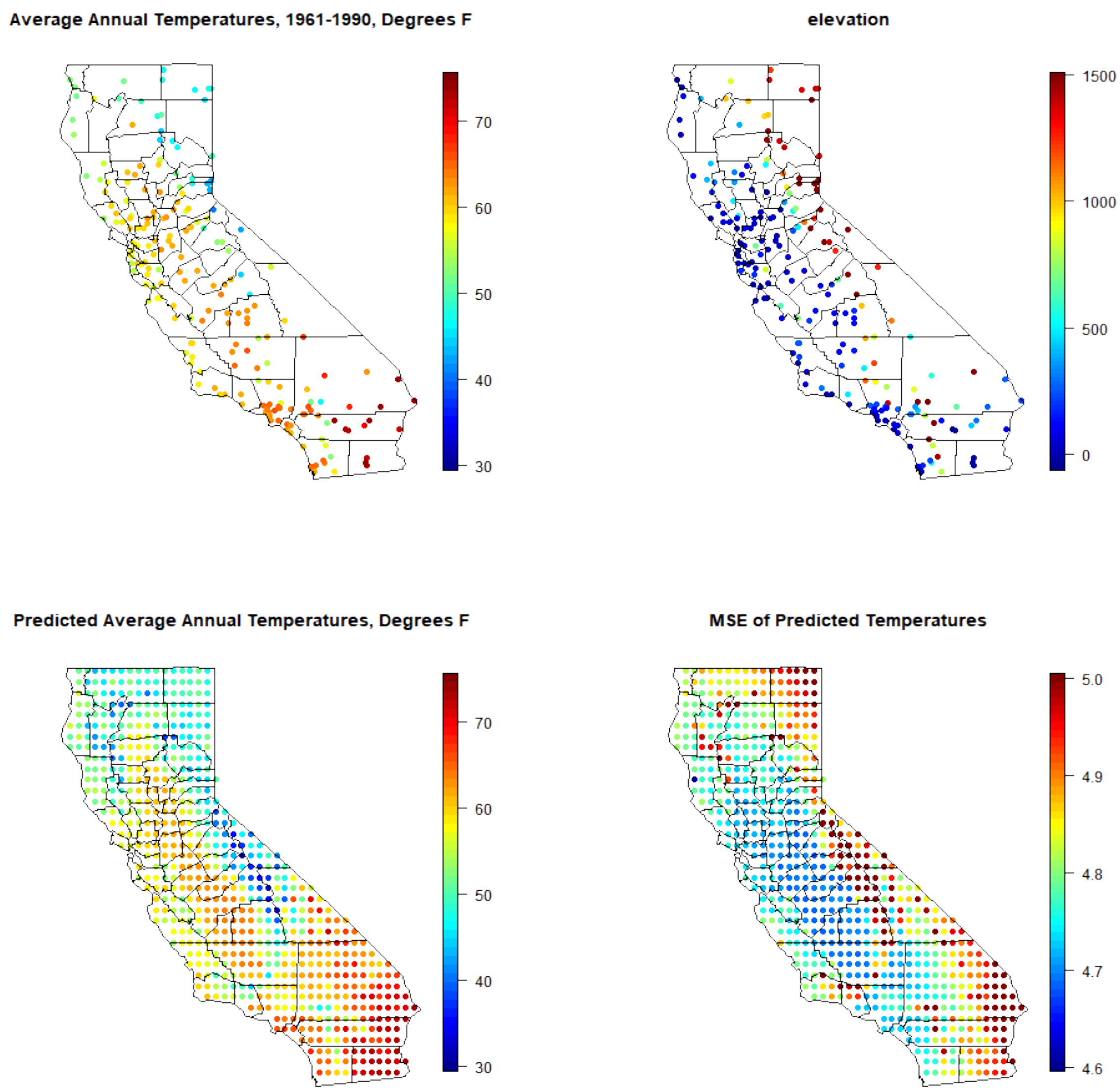

Figure 9: the temperatures of original data(up-left), the elevations of original data(up-right), the predicted temperatures by krigging(down-left), and MSEs by krigging(down-right)

We can find that the krigging is quite well enough to explain the original temperature values around the original data point's values. And, I should emphasize that the region of sparse data points has relatively higher MSE than the region of dense data points. It is reasonable.