Chen Simulation 2d

Qi

2023-04-15

In this file, we simulate the data and evaluate the performance of the 2-D non-stationary data. The data follows a process like follows:

$$Y(s) = sin\{30(\bar{s} - 0.9)^4\}cos\{2(\bar{s} - 0.9)\} + (\bar{s} - 0.9)/2$$

where $s = (s_x, s_y)^T \in R^2$ and $\bar{s} = \frac{s_x + s_y}{2}$. And the range of the coordinates are $[0, 1]$ and there are 900 samples from the grid 30 by 30 in surface $[0, 1]^2$. And here a three level multi-resolution model is used to generate the basis function, the basis are generated in the grid of 10 by 10, 19 by 19 and 37 by 37.

```
sim_coords <- expand.grid(seq(0,1,length.out = 30),seq(0,1,length.out = 30))
sim_sbar <- (sim_coords[,1] + sim_coords[,2])/2
sim_y <- sin(30*(sim_sbar-0.9)^4) * cos(2*(sim_sbar-0.9)) + (sim_sbar-0.9)/2
p_obs <-
ggplot() +
  geom_raster(aes(x = sim_coords[,1], y = sim_coords[,2], fill = sim_y)) +
  scale_fill_viridis_c(name = "") +
  labs(x = "Longitude",  y = "Latitude")

  basis_dist_1 <- spDists(as.matrix(sim_coords), as.matrix(basis_1))
  basis_dist_2 <- spDists(as.matrix(sim_coords), as.matrix(basis_2))
  basis_dist_3 <- spDists(as.matrix(sim_coords), as.matrix(basis_3))

  theta_1 <- 2.5* diff(seq(from = 0, to = 1, length.out = 10))[1]
  theta_2 <- 2.5* diff(seq(from = 0, to = 1, length.out = 19))[1]
  theta_3 <- 2.5* diff(seq(from = 0, to = 1, length.out = 37))[1]

  basis_fun_1 <- nychka_fun(basis_dist_1, theta = theta_1)
  basis_fun_2 <- nychka_fun(basis_dist_2, theta = theta_2)
  basis_fun_3 <- nychka_fun(basis_dist_3, theta = theta_3)
```

By applying the likfit function, we can get the MLE of the estimation of the Matern kernel parameters, and the MLE is based on each training set since we are using MLE from the each training set to approximate the covariance matrix.

```
pair_dist_2d <- spDists( as.matrix(sim_coords) )
set.seed(0)
train_all_index <- sample(1:10, 900, replace = TRUE)
krig_mean_all <- rep(NA, 900)
dkrig_mean_all <- rep(NA, 900)
ckrig_mean_all <- rep(NA, 900)
nn_mean_all <- rep(NA, 900)
```

```r
mse_vec_krig <- rep(NA, 10)
mse_vec_nn <- rep(NA, 10)
mse_vec_dkrig <- rep(NA, 10)
mse_vec_ckrig <- rep(NA,10)
```

```r
for (curr_index in 1:10) {

  print(paste("Now doing index ", curr_index))

  train_index <- which(train_all_index != curr_index)
  train_coords <- sim_coords[train_index,]
  train_y <- sim_y[train_index]
  test_coords <- sim_coords[-train_index,]
  test_y <- sim_y[-train_index]

  # Change the population
  curr_res <- likfit(coords = train_coords, data = train_y, ini.cov.pars = c(1,0.1), fix.kappa = FALSE,

  curr_beta <- curr_res$beta
  curr_sig <- curr_res$sigmasq
  curr_phi <- curr_res$phi
  curr_nu <- curr_res$kappa

  cov_mat <-  curr_sig * matern(pair_dist_2d, kappa = curr_nu, phi = curr_phi)

  # Classical Kriging
  exp_sig_11 <- cov_mat[train_index, train_index]
  exp_sig_12 <- cov_mat[train_index, -train_index]
  exp_sig_21 <- t(exp_sig_12)
  exp_sig_22 <- cov_mat[-train_index, -train_index]
  krig_mean_all[-train_index] <- curr_beta + exp_sig_21 %*% solve(exp_sig_11) %*% matrix( as.numeric(tra
  mse_vec_krig[curr_index] <- mean((sim_y[-train_index] -
                                      krig_mean_all[-train_index])^2)
}
```

```
## [1] "Now doing index  1"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  2"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
```

```
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  3"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  4"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  5"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  6"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  7"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
```

```
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  8"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  9"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
## [1] "Now doing index  10"
## ---------------------------------------------------------------
## likfit: likelihood maximisation using the function optim.
## likfit: Use control() to pass additional
##          arguments for the maximisation function.
##         For further details see documentation for optim.
## likfit: It is highly advisable to run this function several
##         times with different initial values for the parameters.
## likfit: WARNING: This step can be time demanding!
## ---------------------------------------------------------------
## likfit: end of numerical maximisation.
```

```
for (curr_index in 1:10) {
  train_index <- which(train_all_index != curr_index)
  train_coords <- sim_coords[train_index,]
  train_y <- sim_y[train_index]
  test_coords <- sim_coords[-train_index,]
  test_y <- sim_y[-train_index]
# dnn_mean_all

  x_tr <- array_reshape( as.matrix(train_coords), c(length(train_y), 2))
  x_te <- array_reshape( as.matrix(test_coords), c(length(test_y), 2))

  y_tr <- train_y
  y_te <- test_y
  # mse_epoch <- rep(NA, 200)
  # epoch_pred <- matrix(NA, nrow = 200, ncol = length(test_y))
```

```
  model_dnn <- keras_model_sequential()
  model_dnn %>%
  layer_dense(units = 100, activation = 'relu', input_shape = c(ncol(x_tr)), kernel_initializer = 'he_u
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'linear')

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

  model_checkpoint <- callback_model_checkpoint(
  filepath = "C:/Users/10616/Desktop/temp/best_weights.h5",
  save_best_only = TRUE,
  monitor = "val_loss",
  mode = "min",
  verbose = 1
)
  dnn_history <- model_dnn %>%
  fit(x = x_tr, y = y_tr, epochs = 200, batch_size = 64, callbacks = list(model_checkpoint), validation_
  model_dnn %>% load_model_weights_hdf5("C:/Users/10616/Desktop/temp/best_weights.h5")


  nn_mean_all[-train_index] <- predict(model_dnn, x_te)
  mse_vec_nn[curr_index] <- evaluate(model_dnn, x_te, y_te)[2]

}
```

```
for (curr_index in 1:10) {
  train_index <- which(train_all_index != curr_index)
  train_coords <- sim_coords[train_index,]
  train_y <- sim_y[train_index]
  test_coords <- sim_coords[-train_index,]
  test_y <- sim_y[-train_index]

  x_tr <- cbind(train_coords, basis_fun_1[train_index,],
                basis_fun_2[train_index,],basis_fun_3[train_index,])

  x_te <- cbind(test_coords, basis_fun_1[-train_index,],
                basis_fun_2[-train_index,],basis_fun_3[-train_index,])


  x_tr <- array_reshape( as.matrix(x_tr), c(length(train_y), ncol(x_tr)))
  x_te <- array_reshape( as.matrix(x_te), c(length(test_y), ncol(x_tr)))



model_dk <- keras_model_sequential()
model_dk %>%
  layer_dense(units = 100, activation = 'relu', input_shape = c(ncol(x_tr)), kernel_initializer = 'he_u
  layer_dense(units = 100, activation = 'relu') %>%
```

```
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 100, activation = 'relu') %>%
  layer_dense(units = 1, activation = 'linear')

model_dk %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)
 model_checkpoint <- callback_model_checkpoint(
  filepath = "C:/Users/10616/Desktop/temp/best_weights.h5",
  save_best_only = TRUE,
  monitor = "val_loss",
  mode = "min",
  verbose = 1
)

mod_train_dk <- model_dk %>%
  fit(x = x_tr, y = train_y, epochs = 200, batch_size = 64, callbacks = list(model_checkpoint), validati
  model_dk %>% load_model_weights_hdf5("C:/Users/10616/Desktop/temp/best_weights.h5")

  dkrig_mean_all[-train_index] <- predict(model_dk, x_te)
  mse_vec_dkrig[curr_index] <- evaluate(model_dk, x_te, test_y)[2]


}
```

```
for (curr_index in 1:10) {
  train_index <- which(train_all_index != curr_index)
  train_coords <- sim_coords[train_index,]
  train_y <- sim_y[train_index]
  test_coords <- sim_coords[-train_index,]
  test_y <- sim_y[-train_index]

  basis_tr <- basis_fun_3[train_index,]
  basis_te <- basis_fun_3[-train_index,]

  x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), 37, 37, 1))
  x_te <- array_reshape(basis_te, c(nrow(basis_te), 37, 37, 1))

  input_shape <- c(37, 37, 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_flatten() %>%
  layer_dense(units = 100, activation = 'relu') %>%

  layer_dense(units = 100, activation = 'relu') %>%

  layer_dense(units = 100, activation = 'relu') %>%
```

```
  layer_dense(units = 100, activation = 'relu') %>%

  layer_dense(units = 1, activation = 'linear')


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

 model_checkpoint <- callback_model_checkpoint(
  filepath = "C:/Users/10616/Desktop/temp/best_weights.h5",
  save_best_only = TRUE,
  monitor = "val_loss",
  mode = "min",
  verbose = 1
)
mod_train_ck <- model_ck %>%
  fit(x = x_tr, y = train_y, epochs = 200, batch_size = 64, callbacks = list(model_checkpoint), validati
  model_ck %>% load_model_weights_hdf5("C:/Users/10616/Desktop/temp/best_weights.h5")
ckrig_mean_all[-train_index] <- predict(model_ck, x_te)
  mse_vec_ckrig[curr_index] <- evaluate(model_ck, x_te, test_y)[2]

}
```

```
p_krig <-
  ggplot() +
  geom_raster(aes(x = sim_coords[,1], y = sim_coords[,2], fill = krig_mean_all)) +
  scale_fill_viridis_c(name = "") +
  labs(x = "Longitude",  y = "Latitude", color = "")

p_dnn <-
  ggplot() +
  geom_raster(aes(x = sim_coords[,1], y = sim_coords[,2], fill = nn_mean_all)) +
  scale_fill_viridis_c(name = "") +
  labs(x = "Longitude",  y = "Latitude")

p_dk <-
  ggplot() +
  geom_raster(aes(x = sim_coords[,1], y = sim_coords[,2], fill = dkrig_mean_all)) +
  scale_fill_viridis_c(name = "") +
  labs(x = "Longitude",  y = "Latitude")

p_ck <-
    ggplot() +
  geom_raster(aes(x = sim_coords[,1], y = sim_coords[,2], fill = ckrig_mean_all)) +
  scale_fill_viridis_c(name = "") +
  labs(x = "Longitude",  y = "Latitude")
```

```
sqrt(mean((krig_mean_all - sim_y)^2))
```

```
## [1] 0.01698922
```

```
sqrt(mean((nn_mean_all - sim_y)^2))
```

```
## [1] 0.01644038
```

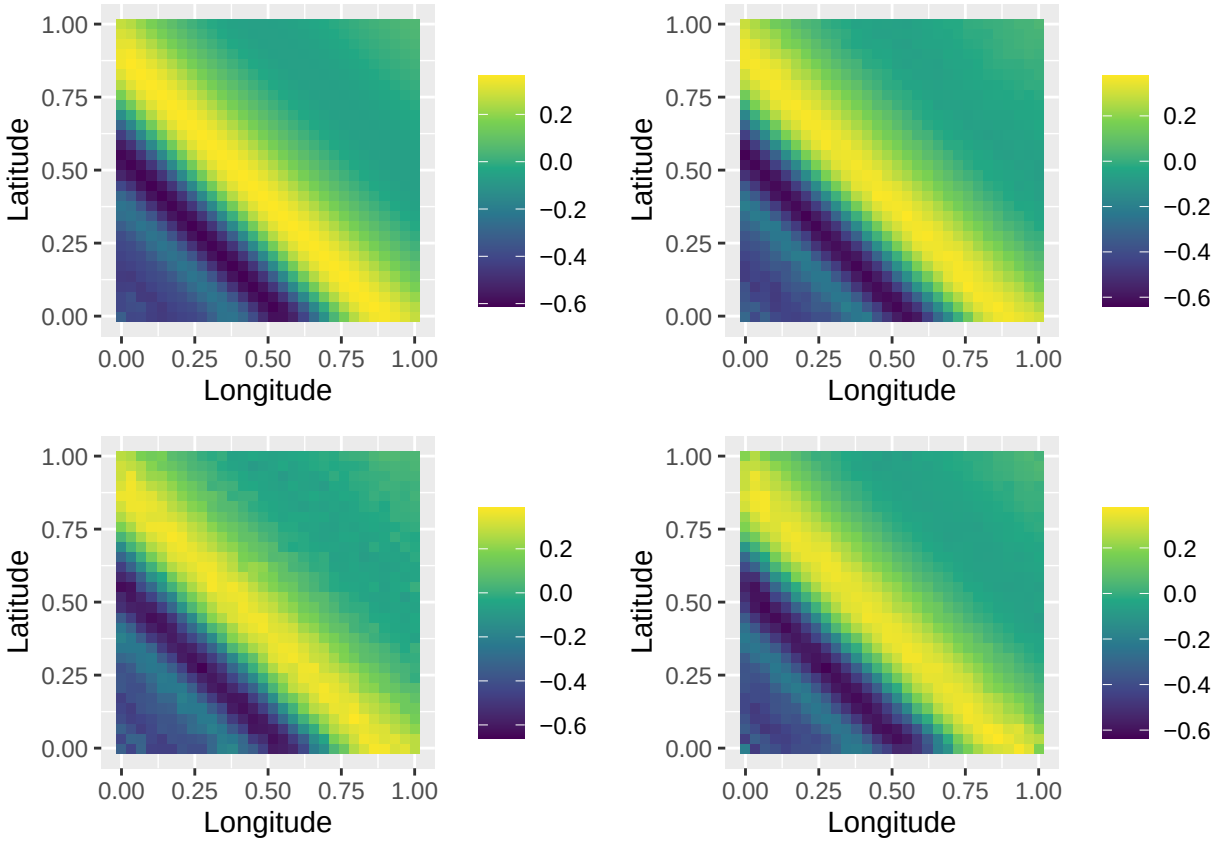
```
sqrt(mean((dkrig_mean_all - sim_y)^2))
```

```
## [1] 0.02078761
```

```
sqrt(mean((ckrig_mean_all - sim_y)^2))
```

```
## [1] 0.02420818
```

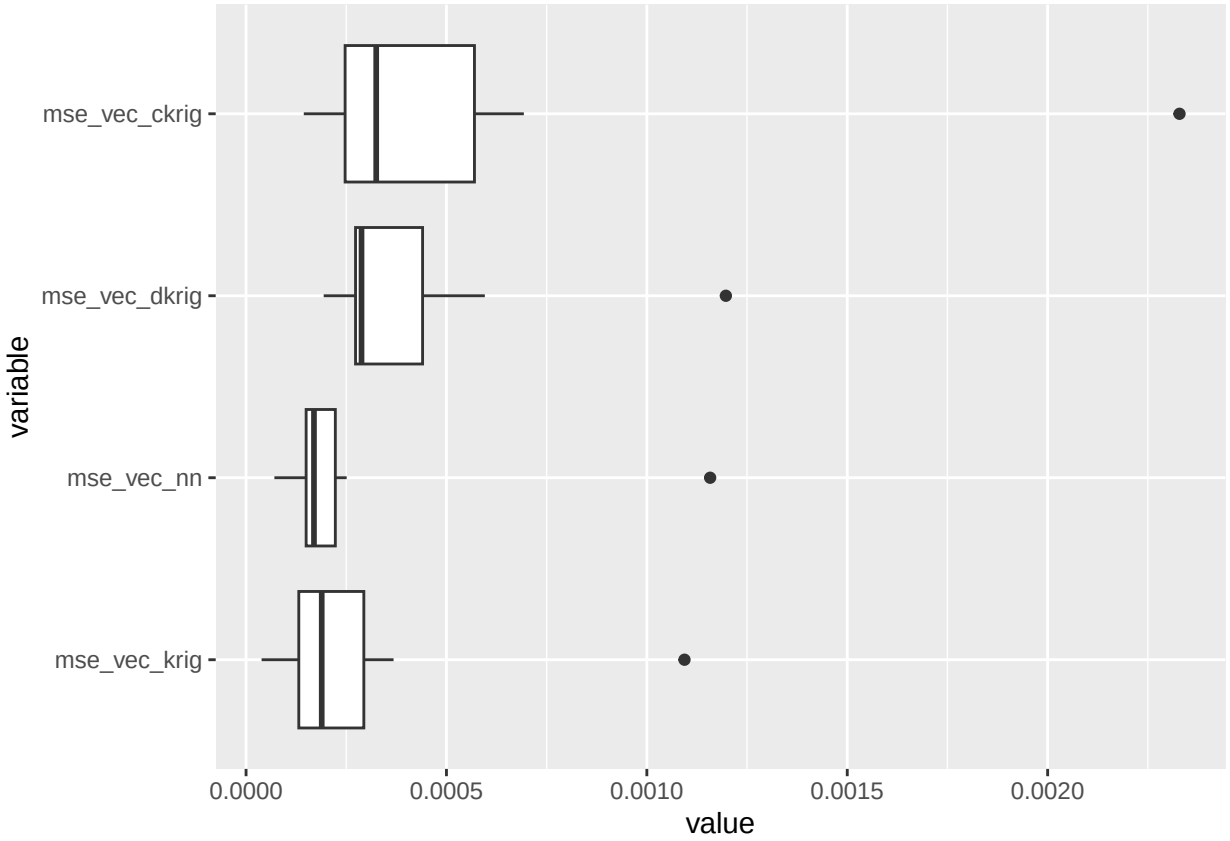
```
cowplot::plot_grid(p_krig, p_dnn, p_dk, p_ck)
```




```
mse_mat <- as.data.frame(cbind(mse_vec_krig, mse_vec_nn, mse_vec_dkrig, mse_vec_ckrig))
write.csv(mse_mat, here::here("chen_simulation/mse_all_10_cv_2d.csv"), row.names = FALSE)
```

```
ggplot() +
  geom_boxplot(data = reshape2::melt(mse_mat), aes(x = variable, y = value)) +
   coord_flip()
```

```
## No id variables; using all as measure variables
```


mse_vec_ckrig
mse_vec_dkrig
mse_vec_nn
mse_vec_krig
variable
0.0000
0.0005
0.0010
0.0015
0.0020
value