Cluster Characterization in Atom Probe Tomography: Machine Learning using Multiple Summary Functions- Additional Supplementary Information

Roland A Bennett, Andrew P. Proudian, Jeramy D. Zimmerman

March 2022

This is a companion to the main paper which will help the user make user of our model. The full model used in the paper is done on a high performance computing system, so here we will be showing a scaled down version.

1 Setup

We utilize a variety of *R* packages, mostly *spatstat* and our own *rapt*. The original package is available on Andrew's github, https://github.com/aproudian2/rapt, but since I (Roland) am currently the most active rapt developer, mine may be more up to date. Therefore, the most recent version of *rapt* is my master branch, available my github https://github.com/rolandrolandroland/rapt. Intallation from github requires the *devtools* package and instructions are given below

Contact: Roland Bennett (graduate student): rolandbennett@mines.edu, Jeramy Zimmerman (Professor): jdzimmer@mines.edu

If you would like to use the data that we have already generated (training size = 1458, testing sizes = 365), then include the `load("walkthrough_data.RData")` command below and keep the `evaluate` variable set to `FALSE`. If you would like to run it yourself, comment out the `load` and set `evaluate` to `TRUE`. This will take a long time on a standard desktop/laptop, unless you scale the simulation down (reduce number of relabelings `n_relabs` or training data `nrand`)

Load the required packages

2 Simulation Parameters

Next, we will define the parameters for our simulated point patterns. The point pattern will be 60 units^3 with an average intensity of 1.005 pts/unit^3.

```
# simulation parameters
maxGr <- 4 # max radius at which to calculate G and F
maxKr <- 30 # max radius at which to calculate K
nGr <- 2000 # number of radius values at which to calculate G
maxGXGHr <- 3 # max radius at which to calculate GXGH
nGXr <- 2000 # number of radius values at which to calculate GXGH
vside <- 0.3
# maxGXHDr = 10
nKr <- 100 # number of radius values at whih to calculate GT
pcp <- 0.05114235 # percentage of points to assign as dopant
tol <- 0.005 # percent by which dopant points can deviate
```

```r
# from expected before cluster simulation fails
intensity <- 1.005 # total intensity of point pattern points/unit^3
size <- c(0, 60) # dimensions for pattern

set.seed(100)

# A type points are dopants, C type points are host
dopant_formula <- "A"
host_formula <- "C"
```

3 Random Relabelings

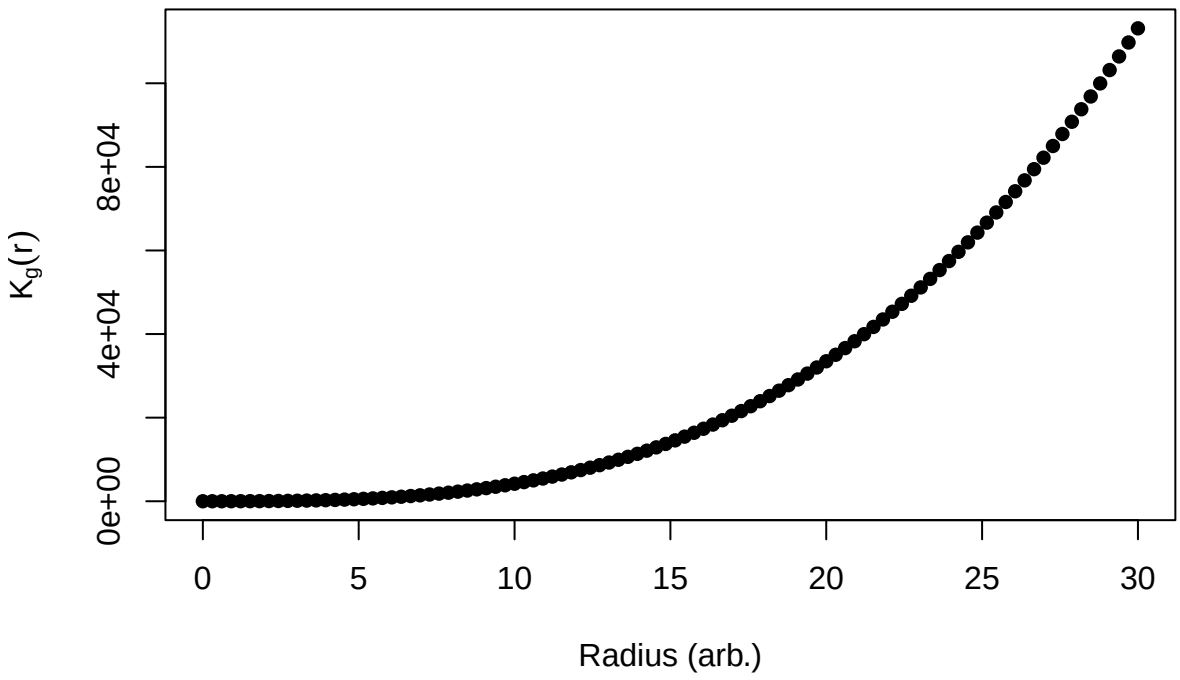
In order to calculate our features, we must first get the expected values of each summary function. The expected value is correlated to the overall intensity of the point pattern and concentration of the dopant, but not the distribution of the dopant. To get this expected value, we perform 10,000 random relabelings. For more details on this, see the Methods section of the main body of the paper. Note: for this example, we are using only 100 relabelings in order to reduce the compuational expense.

```r
# Create random poisson pattern to get expected values for summary functions
box_size <- box3(xrange = size, yrange = size, zrange = size)
pp3_box <- rpoispp3(lambda = intensity, domain = box_size, nsim = 1)
pp3_box <- rlabel(pp3_box,
  labels = as.factor(c(rep("A", pcp * 10000), rep("C", 10000 * (1 - pcp)))),
  permute = FALSE
)
pp3_dopant_box <- pp3_box[marks(pp3_box) == "A"]
pp3_box

# Total number of host and dopant type points
host_total <- sum(pp3_box$data$marks == host_formula)
dopant_total <- sum(pp3_box$data$marks == dopant_formula)


all_funcs <- c("K", "G", "F", "GXGH")
ncores <- detectCores()
start <- as.numeric(Sys.time())
cl <- makePSOCKcluster(ncores)
n_relabs <- 1000
invisible(clusterEvalQ(cl, c(library(rapt), library(spatstat))))
clusterExport(cl, c( "pp3", "host_formula",
                     "dopant_formula", "all_funcs",
                      "vside", "maxGr", "maxKr",
                     "nGr", "nKr", "maxGXGHr",
                     "nGXr", "n_relabs"))


## Number of relabelings to perform. Change to 10000 for full version
# compute n_relabs relabelings
print("Begin relabeling")
relabel_pattern <- parLapply(cl, 1:n_relabs, relabel_summarize,
  funcs = all_funcs,
  pattern = pp3_box,
  maxKr = maxKr, nKr = nKr,
```

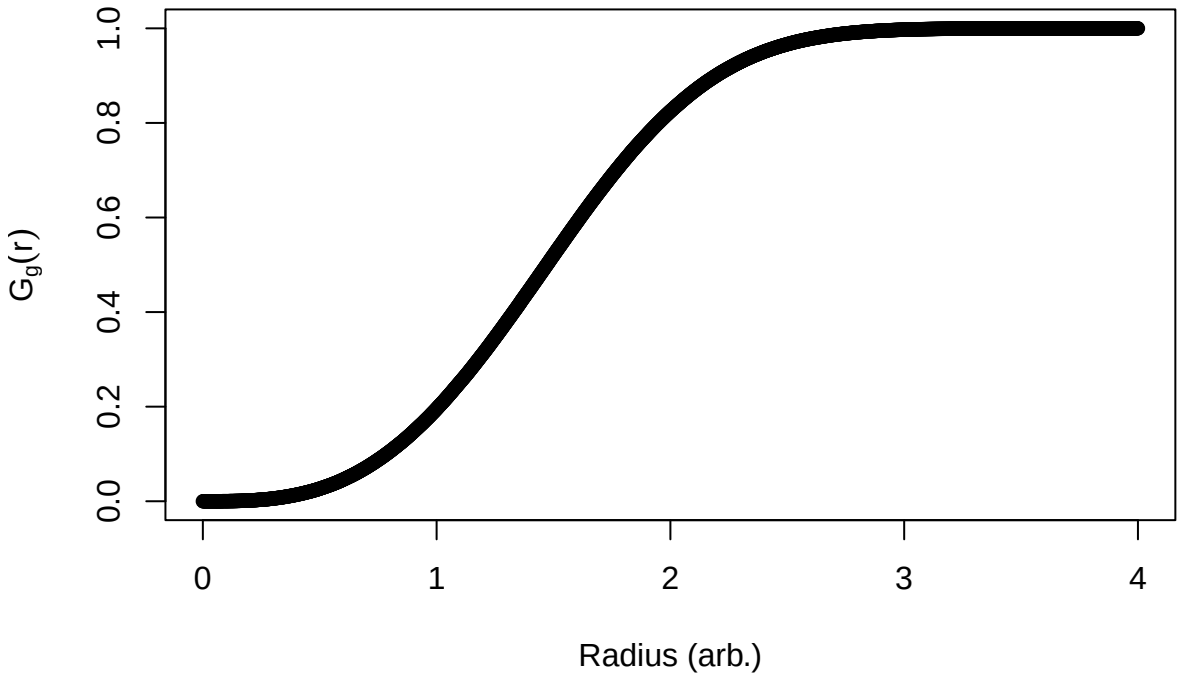
```
  maxGr = maxGr, nGr = nGr,
  maxGXGHr = maxGXGHr, nGXr = nGXr, vside = vside,
  host_formula = host_formula, dopant_formula = dopant_formula,
  K_cor = "trans", G_cor = "km", F_cor = "km",
  GXGH_cor = "km", GXHG_cor = "km"
)

# Take averages to get expected values
rrl_box <- average_relabelings(relabel_pattern,
  envelope.value = .95,
  funcs = all_funcs,
  K_cor = "trans", G_cor = "km", F_cor = "km",
  GXGH_cor = "km", GXHG_cor = "km"
)

# remove relabel_pattern object, since it is large and the info we need is in rrl_full
rm(relabel_pattern)

print(as.numeric(Sys.time()) - start)
stopCluster(cl)
```

Now, we will plot the results of the random relabelings. These are the medians of n_relabs relabelings.

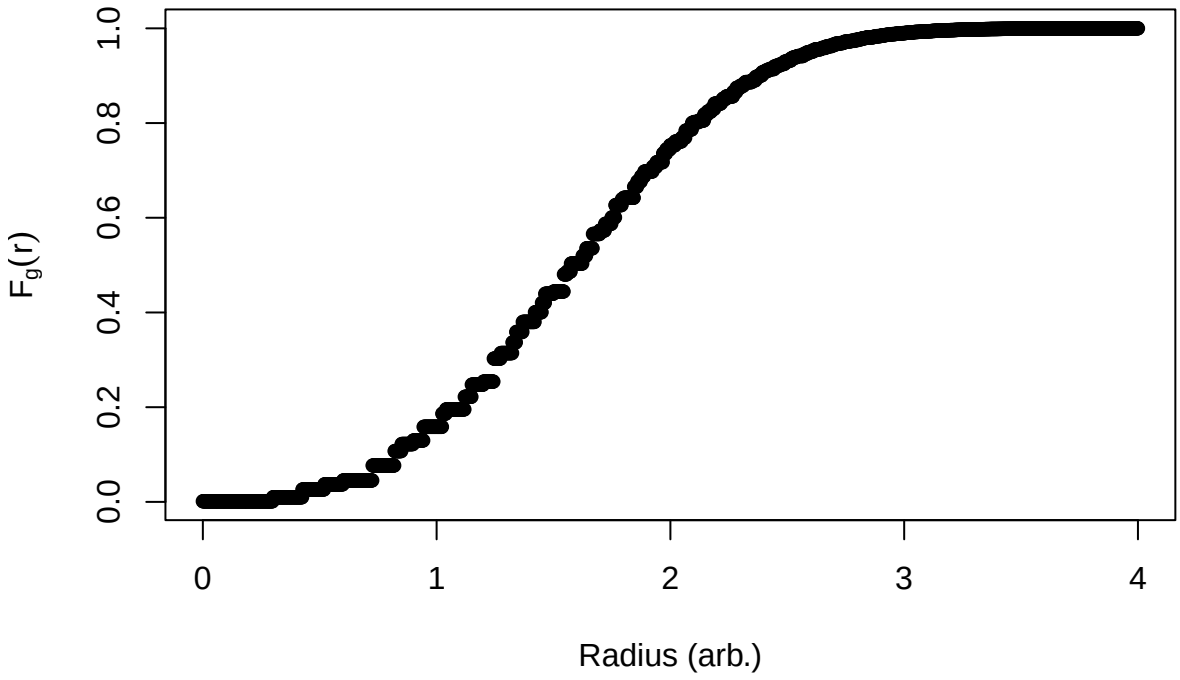
```
# plot relabeled objects
plot(rrl_box$rrl_K$r, rrl_box$rrl_K$mmean,
  xlim = c(0, maxKr),
  xlab = "Radius (arb.)", ylab = expression(K[g](r)), pch = 16
)
```



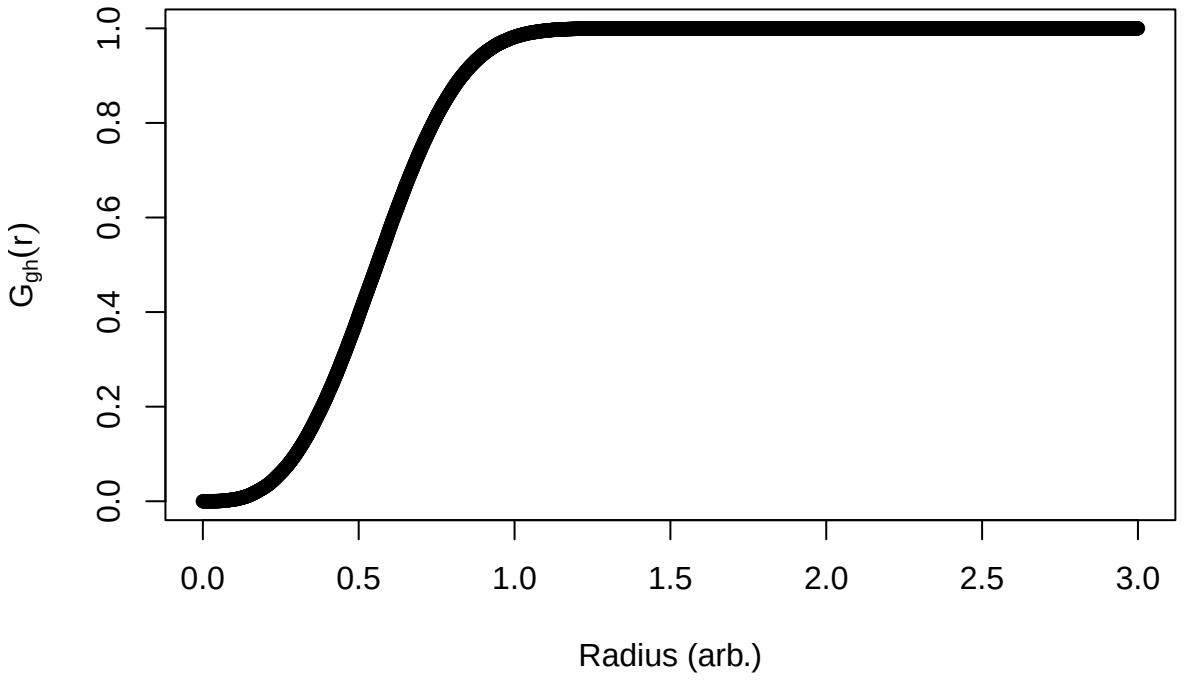

```
plot(rrl_box$rrl_G$r, rrl_box$rrl_G$mmean,
  xlim = c(0, maxGr),
  xlab = "Radius (arb.)", ylab = expression(G[g](r)), pch = 16
)
```




```
plot(rrl_box$rrl_F$r, rrl_box$rrl_F$mmean,
  xlim = c(0, maxGr),
  xlab = "Radius (arb.)", ylab = expression(F[g](r)), pch = 16
)
```




```
plot(rrl_box$rrl_GXGH$r, rrl_box$rrl_GXGH$mmean,
  xlim = c(0, maxGXGHr), ylim = c(0, 1),
  xlab = "Radius (arb.)", ylab = expression(G[gh](r)), pch = 16
)
```



4 Training Data

To train our model, we will use a ratio of 4:1 training:testing data. The full model us trained using 100,000 data sets and tested on 25,000 data sets, but as described in the paper, only little accuracy is lost with only 5,000 training data sets. First, we will name the features and output. For full discussion of the features, see the supplemental information.

```
# Name features
G_feats <- c("G_max_diff", "G_max_diff_r", "G_min_diff", "G_zero_diff_r")
F_feats <- c("F_min_diff", "F_min_diff_F")
K_feats <- c("Tm", "Rm", "Rdm", "Rddm", "Tdm")
GXGH_feats <- c("GXGH_min_diff", "GXGH_95diff_r", "GXGH_FWHM")

# Name output metrics
output <- c("rhoc", "rhob", "cr", "rb", "rw")
feats <- c(G_feats, F_feats, K_feats, GXGH_feats)
param_names <- c("cr", "rhoc", "rhob", "rb", "pb")
```

The `create_training_data()` function is used to generate our training and testing data. It takes a point pattern (pp3 object), assigns labels so that it exhibits clustering based upon the input parameters ρ_c (intra cluster density), ρ_b (background density), μ_r (mean cluster radius), δr (radius dispersity), and pb (position blur). Summary function are calculated on this newly labeled pattern and then compared to the expected values (from the relabeling done previously) to extract our features_

Now we will actually perform the training. First, we will generate input parameters using a random uniform distribution across our chosen intervals.

```
set.seed(100)


# create parameters that will be used in cluster objects
nrand <- 2000 # number of different combinations of parameters
# set nrand = 100,000 for results used in paper
nclust <- 1 # number of clusters per combination
cr <- runif(nrand, min = 2, max = 6.5) # mu_R values
rhoc <- runif(nrand, min = 0.20, max = 1) # rhoc values
rhob <- runif(nrand, min = 0, max = .035) # rhob values
rb <- runif(nrand, min = 0, max = 0.5) # radius blur (aka beta) values
pb <- runif(nrand, min = 0, max = 0.2) # position blur (aka xi) values
cr = unlist(lapply(cr, rep, nclust))
rhoc = unlist(lapply(rhoc, rep, nclust))
rhob = unlist(lapply(rhob, rep, nclust))
rb = unlist(lapply(rb, rep, nclust))
pb= unlist(lapply(pb, rep, nclust))
params = as.data.frame(cbind(cr, rhoc, rhob, rb, pb))
seeds = 1:nrand

tol <- 0.005
pattern <- pp3_box
train_gen_start <- as.numeric(Sys.time())
# Generate training data using parameters and relabled objects above
ncores <- detectCores() -1
# Create a PSOCK cluster containing the number of cores earlier specified
cl <- makePSOCKcluster(ncores, setup_strategy = "sequential", outfile = "output_file")
invisible(clusterEvalQ(cl, c(library(rapt), library(zoo),
                             library(dplyr), library(spatstat.utils))))
# Load the rapt and zoo libraries into the parallel computing environment

clusterExport(cl, c("params", "dopant_formula", "host_formula",
                    "pattern", "rrl_box", "pcp", "tol",
                    "maxKr", "nKr", "maxGr", "nGr", "feats",
                    "maxGXGHr", "nGXr",
                    "vside"))
# create training data
print("Create Training Data")

train_data_list <- parLapply(cl, 1:nrow(params), function(i) {
  create_training_data(i = i, pattern = pattern,
                    rrl = rrl_box, nper = 1,
                    params = params[i,],
                    pcp = pcp,  tol = tol,
                    maxKr = maxKr, nKr = nKr,
                    maxGr = maxGr, nGr = nGr,
                    maxGXGHr = maxGXGHr, nGXr = nGXr,
                    vside = vside, feats = feats,
                    total_time_start = as.numeric(Sys.time()))
})

# get rid of entries with missing values
train_data_list <- lapply(train_data_list, function(set) {
```

```
  if (length(set) == 27) {
    set
  }
})
# transform from list to matrix
train_data <- matrix(unlist(train_data_list), byrow = TRUE,
                     ncol = length(train_data_list[[1]]))

# give column names
# give column names
col_names <- c(param_names, "jitter", feats, "G_time", "F_time",
               "K_time", "GXGH_time", "GXHG_time",
               "cycle_time", "train_gen_time")


colnames(train_data) <- col_names
train_data <- as.data.frame(train_data)
train_data <- train_data[complete.cases(train_data), ]
stopCluster(cl)
print(dim(train_data))
```

If you want to predict weighted radius, then calculate it for each cluster

```
# add rw
sigma <- train_data[, "cr"] * train_data[, "rb"]
train_data$rw <- (train_data[, "cr"]^4 + 6 * train_data[, "cr"]^2 *
                    sigma^2 + 3 * sigma^4) /
  (train_data[, "cr"]^3 + 3 * train_data[, "cr"] * sigma^2)

ind <- round(nrow(train_data) * 4 / 5, 0)
test_data <- train_data[(ind + 1):nrow(train_data), ]
# colnames(test_data) = col.names
train_data <- train_data[1:ind, ]
# colnames(test_data) = c(colnames(test_data[1:29]), "rw")
# sizes of training data
dim(train_data)
dim(test_data)
```

Now we will actually apply our freshly calculated features to a Bayesian regularized neural network! Below we define the function to do so. It takes a metric of interest, training and testing data, and a vector with the names of features to be used as input. The output is a list with few different interesting entries. The first is simply the predictions. Then we have a few different ways of quantifying error: the root mean squared error (RMSE), root mean squared percent error (RMSPE) and mean absolute error (MAE), and mean percent error (MPE). Sixth and finally is the time that it took to run the model.

```
trainer <- function(to_predict, train_data, test_data, features) {
  cycle_start <- as.numeric(Sys.time())
  pp_train <- preProcess(train_data[, features],
    methods = c("scale", "center", "pca", "BoxCox"),
    thresh = 1
  )
  data_train_pca <- predict(pp_train, train_data[, features])
  test_pca <- predict(pp_train, test_data[, features])
  model <- train(data_train_pca, train_data[, to_predict],
    method = "brnn",
```

```
    trControl = trainControl("cv", number = 10),
    tuneGrid = data.frame("neurons" = c(5, 7, 9))
  )
  toRound <- 6
  a <- predict(model, newdata = test_pca)
  # RMSEP = sqrt(mean((a-test_data[,to_predict])^2))
  RMSPE <- sqrt(mean(((test_data[, to_predict] - a) / test_data[, to_predict])^2)) * 100
  RMSE <- sqrt(mean(((test_data[, to_predict] - a))^2))
  MAE <- mean(abs(test_data[, to_predict] - a))
  MPE <- mean(abs((test_data[, to_predict] - a) / test_data[, to_predict])) * 100
  cycle_time <- as.numeric(Sys.time()) - cycle_start
  list(preds = a, RMSE = RMSE, RMSPE = RMSPE,
       MAE = MAE, MPE = MPE, cycle_time = cycle_time)
}
```

5 Predictions

Now to apply the *trainer*() function above. We generate the predictions for our metrics in parallel using *parLapply* in order to save time.

```
time.check.4 <- as.numeric(Sys.time()) ## just before performing the training
output
ncores <- detectCores()
cl <- makePSOCKcluster(ncores)
invisible(clusterEvalQ(cl, c(library(caret), library(brnn))))
clusterExport(cl, c("trainer", "train_data", "test_data", "feats"))

set.seed(100)
preds <- parLapply(cl, output, trainer,
  train_data = train_data, test_data = test_data,
  features = feats
)
time.check.5 <- as.numeric(Sys.time())
stopCluster(cl)
```

Now that we have our predictions, lets look at how accurate they are! First, simply the error metrics we have stored in the results

```
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
results <- sapply(preds, function(x) {
  c(RMSE = x$RMSE, RMSPE = x$RMSPE, MAE = x$MAE, MPE = x$MPE)
})
colnames(results) <- output
results <- signif(results, 3) # round to 3 significant figures
results
```

```
##         rhoc     rhob     cr       rb    rw
## RMSE  0.0117 2.79e-04  0.440 8.61e-02 0.479
```

```
## RMSPE 2.8700 2.03e+01 10.500 3.97e+03 7.080
## MAE   0.0085 2.10e-04  0.325 6.63e-02 0.286
## MPE   1.8200 4.96e+00  7.840 3.18e+02 4.950
```

Now, we will remake the plots shown in Figure 1 of the main paper. The "error_plot" function displays what percent difference of the predictions are within the The "est_plot" function compares each prediction to the true value.

```
library(RColorBrewer)
error_plot <- function(diff.perc, percs, cols, ...) {
  n <- length(diff.perc)
  diff.perc.sorted <- sort(abs(diff.perc))
  xs <- (1:n) * 100 / n

  plot(xs, diff.perc.sorted, xlab = "", type = "n", ...)
  vals <- list("all" = data.frame("perc" = xs, "error" = diff.perc.sorted))

  ind <- rep(0, length(percs) + 1)
  for (i in 1:length(percs)) {
    ind[i + 1] <- round(n * percs[i] / 100)
    vals[[i + 1]] <- vals$all[(ind[i] + 1):ind[i + 1], ]
    n.i <- nrow(vals[[i + 1]])
    points(vals[[i + 1]]$perc, vals[[i + 1]]$error, pch = 16, col = cols[i], cex = 1)
    segments(-10, tail(vals[[i + 1]]$error, n = 1), tail(vals[[i + 1]]$perc, n = 1),
             tail(vals[[i + 1]]$error, n = 1),
      col = cols[i], lwd = 2, lty = 2
    )
    segments(tail(vals[[i + 1]]$perc, n = 1), -10, tail(vals[[i + 1]]$perc, n = 1),
             tail(vals[[i + 1]]$error, n = 1),
      col = cols[i], lwd = 2, lty = 2
    )
  }
  ind.last <- (tail(ind, n = 1) + 1):n
  points(vals$all$perc[ind.last], vals$all$error[ind.last], col = "black", pch = 16, cex = 1)
}


est_plot <- function(diff.perc, percs, cols, pred, obs, ...) {
  diff <- obs - pred
  sorted <- data.frame("obs" = obs[order(abs(diff.perc))], "pred" = pred[order(abs(diff.perc))])
  n <- nrow(sorted)

  plot(sorted$obs, sorted$pred, col = "black", pch = 16, type = "n", cex = 0.5, ...)

  vals.subed <- list("all" = sorted)
  ind <- rep(0, length(percs) + 1)
  ind.last <- (tail(ind, n = 1) + 1):n
  points(vals.subed$all$obs[ind.last], vals.subed$all$pred[ind.last],
         col = "black", pch = 16, cex = 0.3)
  for (i in length(percs):1) {
    ind[i + 1] <- round(n * percs[i] / 100)
    vals.subed[[i + 1]] <- sorted[(ind[i] + 1):ind[i + 1], ]
    n.i <- nrow(vals.subed[[i + 1]])
    points(vals.subed[[i + 1]]$obs, vals.subed[[i + 1]]$pred, pch = 16,
           cex = 0.3, col = cols[i])
```

```
  }

  abline(0, 1, col = "black", lty = 2, lwd = 1.5)
}
figShow = "asis"
figWidth = "75%"
margins = c(5, 5, 1, 0.5)
```

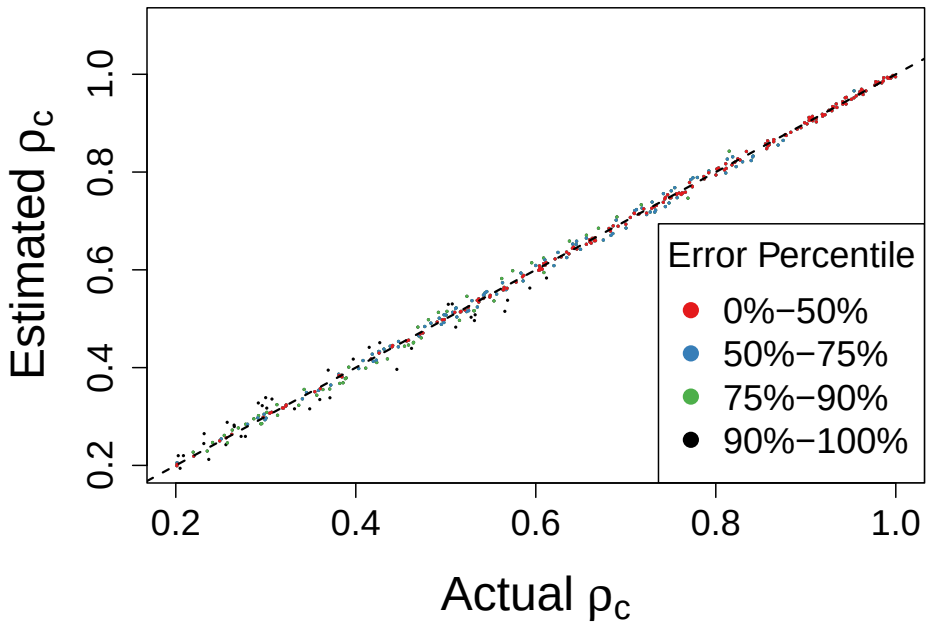
```
par(mar = margins)

obs <- test_data[, "rhoc"]
pred <- preds[[1]][[1]]
print(length(obs))
```

```
## [1] 365
```

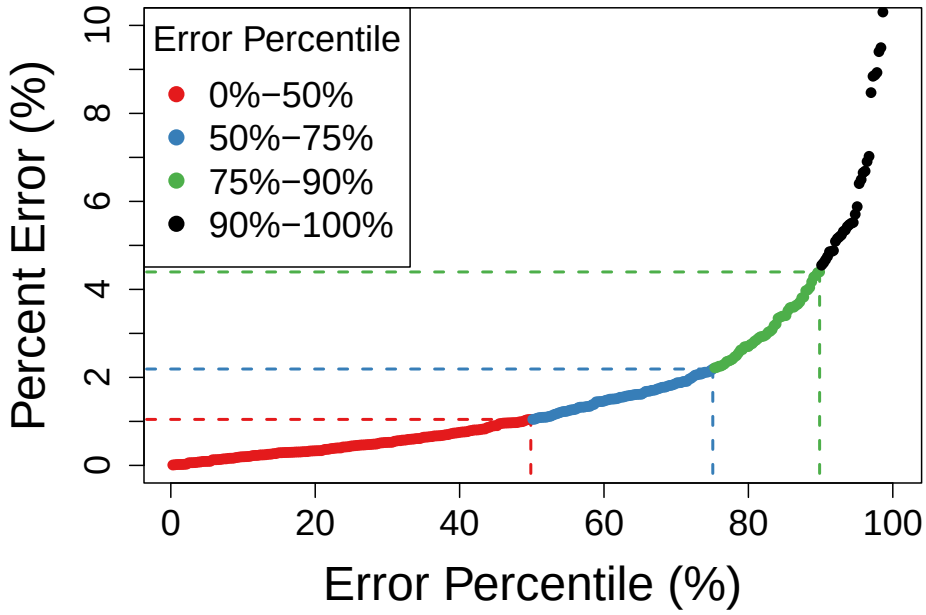
```
print(length(pred))
```

```
## [1] 365
```

```
diff <- pred - obs
diff.perc <- diff * 100 / obs
est_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  pred = pred, obs = obs,
  ylab = " ",
  xlab = " ",
  ylim = c(0.2, 1.1), xlim = c(0.2, 1),
  cex.axis = 1.5
)
legend("bottomright",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
mtext(
  side = 2, font = 2, expression(paste("Estimated ", rho[c])),
  cex = 2, line = 3
)
mtext(
  side = 1, font = 2, expression(paste("Actual ", rho[c])),
  cex = 2, line = 4
)
```




```
error_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  yaxt = "n",
  ylab = "", ylim = c(0, 10),
  cex.lab = 2, cex.main = 3, cex.axis = 1.5
)
mtext(side = 1, font = 1, "Error Percentile (%)", cex = 2, line = 3.25)
mtext(side = 2, font = 1, "Percent Error (%)", cex = 2, line = 3.25)
axis(side = 2, at = seq(0, 10, 1), labels = T, cex.axis = 1.5)
legend("topleft",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```




```
par(mar = margins)

# rhob est plot for combo 26, G, K, GXDH, F
obs <- test_data[, "rhob"]
pred <- preds[[2]][[1]]
diff <- pred - obs
print(length(obs))
```

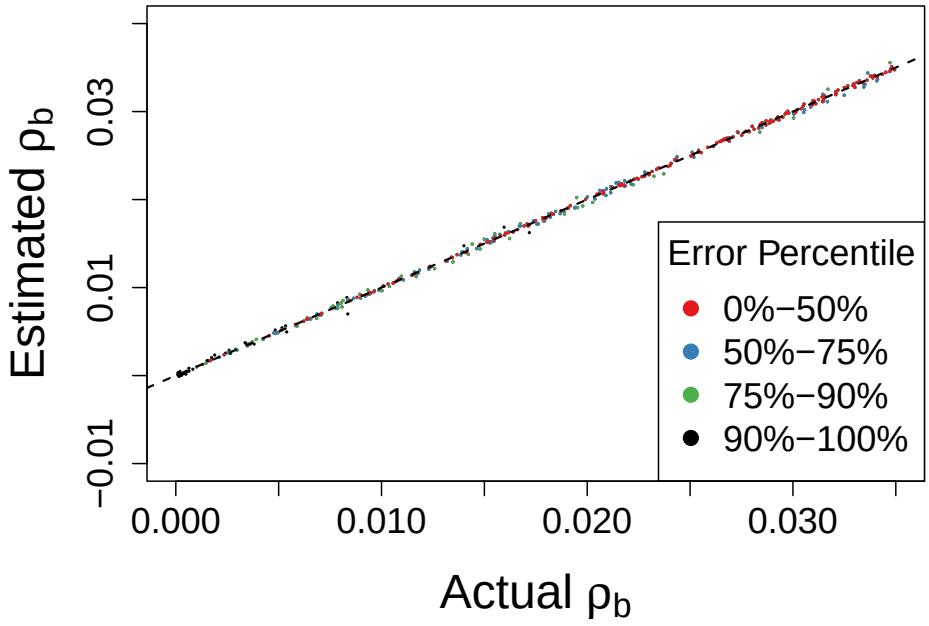
```
## [1] 365
```

```
print(length(pred))
```

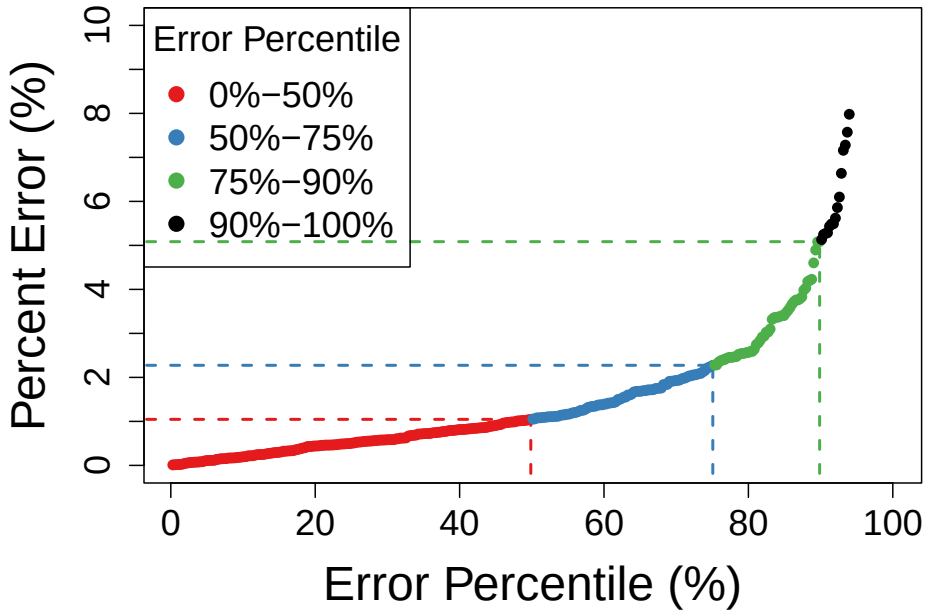
```
## [1] 365
```

```
diff.perc <- diff * 100 / obs

est_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  pred = pred, obs = obs,
  xlab = "",
  ylab = "",
  ylim = c(-0.01, 0.040), xlim = c(0, 0.035),
  cex.lab = 2, cex.main = 2.5, cex.axis = 1.5
)
mtext(
  side = 2, font = 2, expression(paste("Estimated ", rho[b])),
  cex = 2, line = 3
)
mtext(
  side = 1, font = 2, expression(paste("Actual ", rho[b])),
  cex = 2, line = 4
)
legend("bottomright",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
```

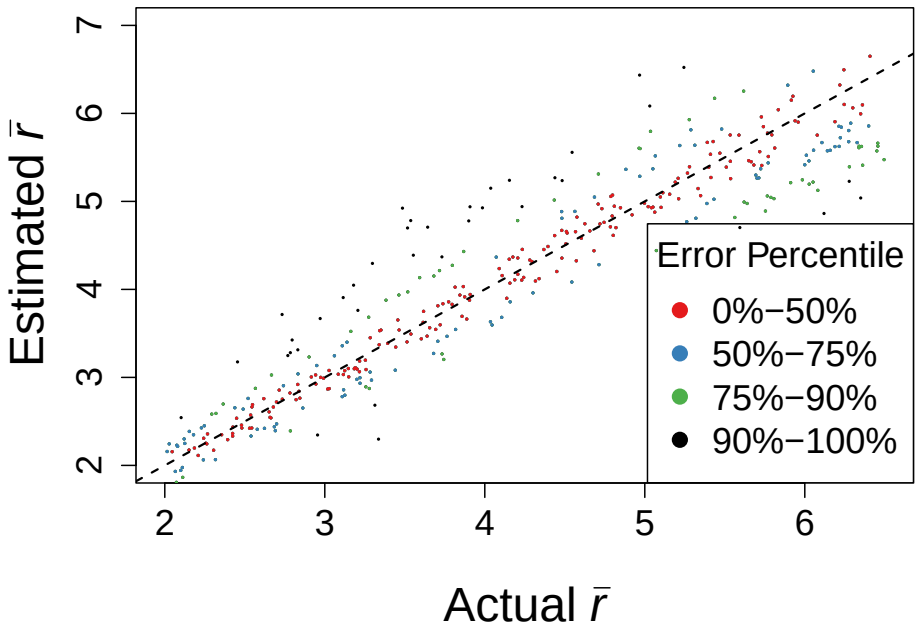
```
  title = "Error Percentile", cex = 1.5
)
```



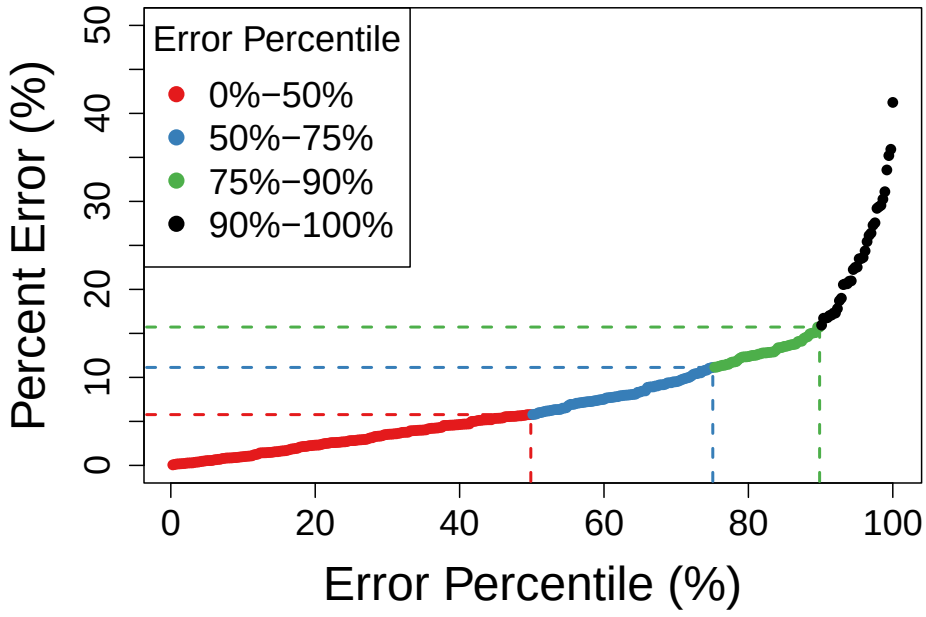

```
error_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  yaxt = "n",
  ylab = "", ylim = c(0, 10),
  cex.lab = 2, cex.main = 3, cex.axis = 1.5
)
mtext(side = 1, font = 1, "Error Percentile (%)", cex = 2, line = 3.25)
mtext(side = 2, font = 1, "Percent Error (%)", cex = 2, line = 3.25)
axis(side = 2, at = seq(0, 10, 1), labels = T, cex.axis = 1.5)
legend("topleft",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```



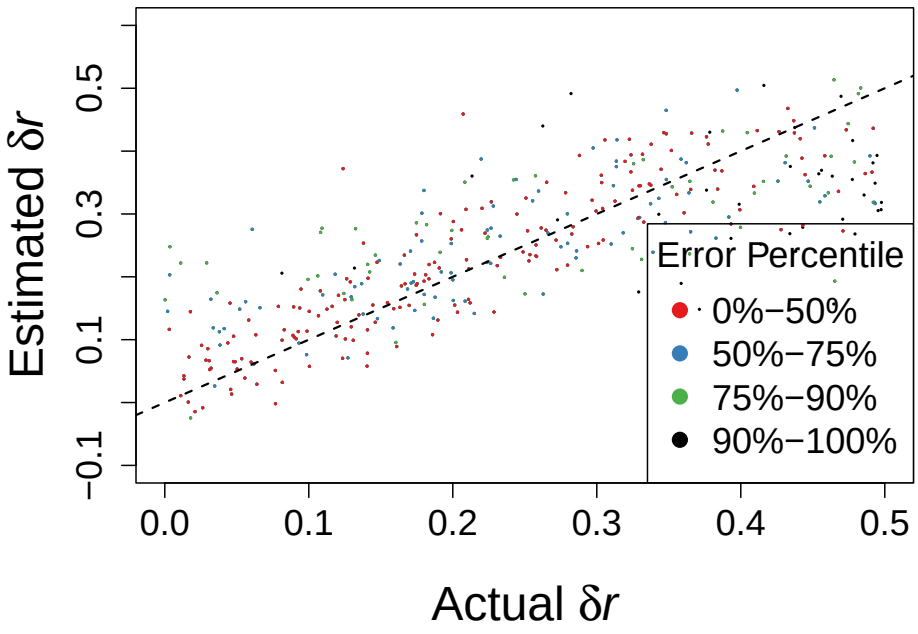

```
par(mar = margins)

obs <- test_data[, "cr"]
pred <- preds[[3]][[1]]
diff <- pred - obs
diff.perc <- diff * 100 / obs

est_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  pred = pred, obs = obs,
  xlab = " ",
  ylab = " ",
  ylim = c(2, 7), xlim = c(2, 6.5),
  cex.lab = 2, cex.main = 3, cex.axis = 1.5
)
mtext(
  side = 2, font = 2, expression(paste("Estimated ", italic(bar(r)))),
  cex = 2, line = 3
)
mtext(
  side = 1, font = 2, expression(paste("Actual ", italic(bar(r)))),
  cex = 2, line = 4
)
legend("bottomright",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```



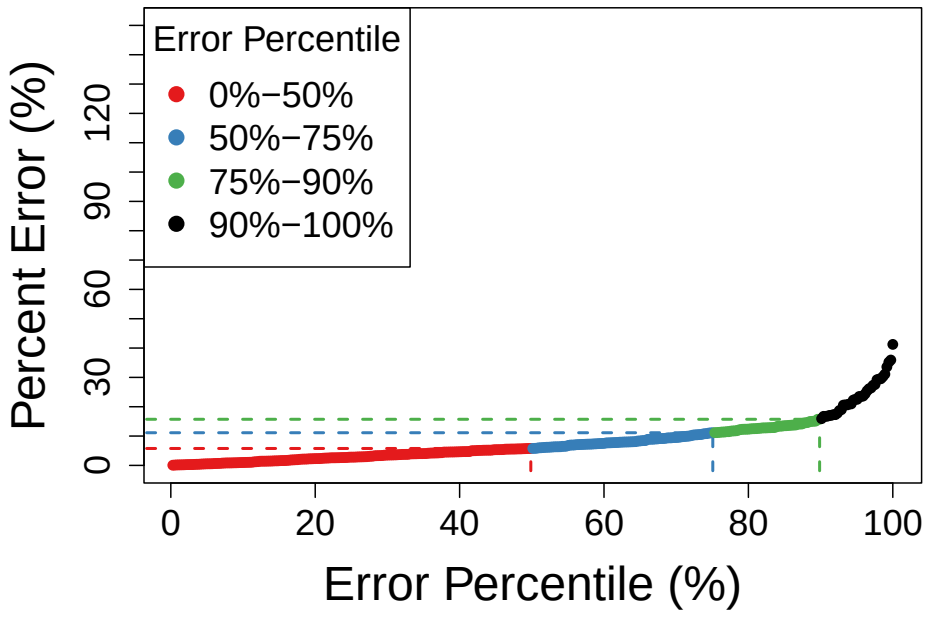

```
error_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  yaxt = "n",
  ylab = "", ylim = c(0, 50),
  cex.lab = 2, cex.main = 3, cex.axis = 1.5
)
mtext(side = 1, font = 1, "Error Percentile (%)", cex = 2, line = 3.25)
mtext(side = 2, font = 1, "Percent Error (%)", cex = 2, line = 3.25)
axis(side = 2, at = seq(0, 50, 5), labels = T, cex.axis = 1.5)
legend("topleft",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```



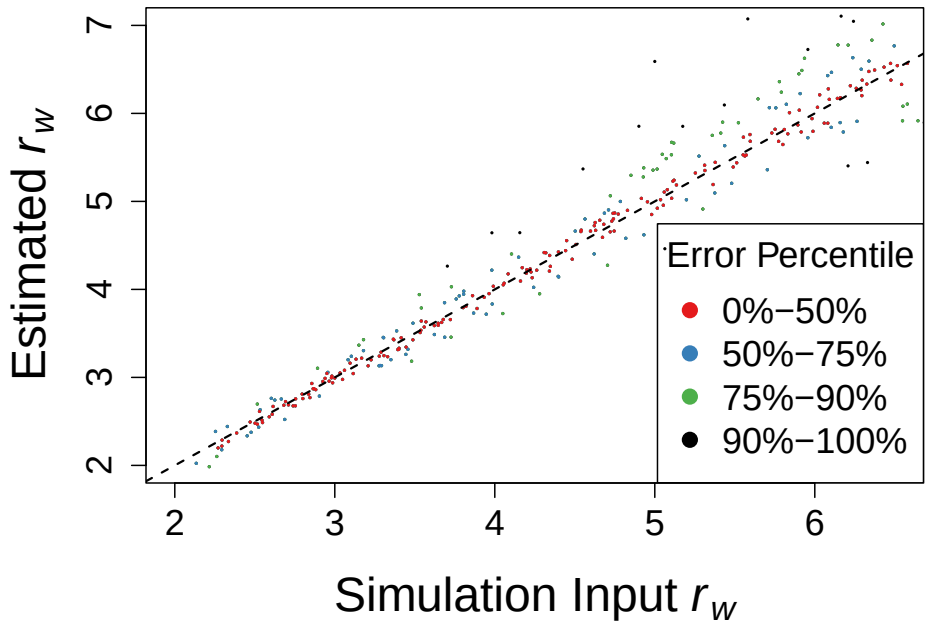

```
par(mar = margins)

obs <- test_data[, "rb"]
pred <- preds[[4]][[1]]
diff <- pred - obs

est_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  pred = pred, obs = obs,
  xlab = "",
  ylab = "",
  ylim = c(-0.1, 0.6), xlim = c(0, 0.5),
  cex.lab = 2, cex.main = 2, cex.axis = 1.5
)
mtext(
  side = 2, font = 2, expression(paste("Estimated ", italic(delta), italic(r))),
  cex = 2, line = 3
)
mtext(
  side = 1, font = 2, expression(paste("Actual ", italic(delta), italic(r))),
  cex = 2, line = 4
)
legend("bottomright",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```



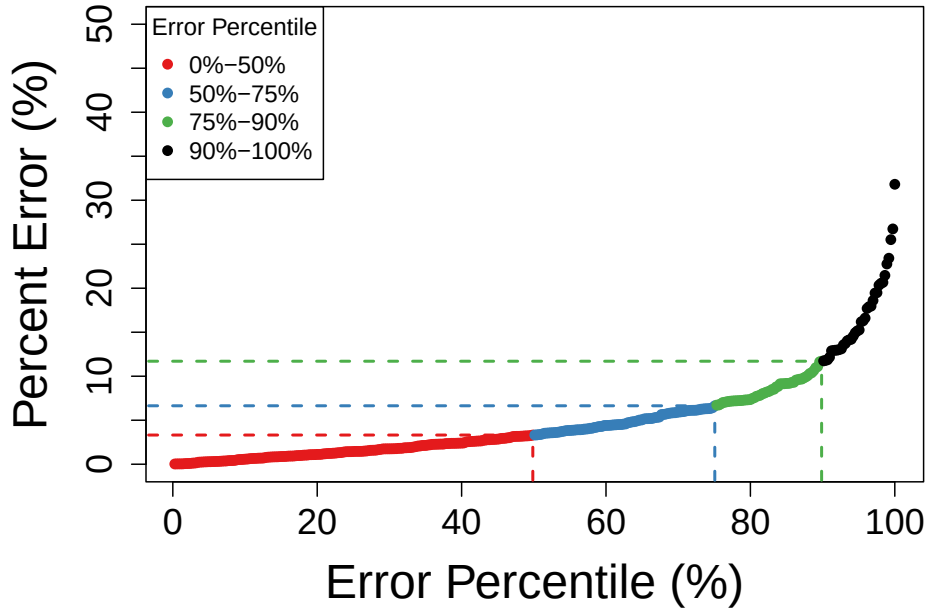

```
error_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  yaxt = "n",
  ylab = "", ylim = c(0, 150),
  cex.lab = 2, cex.main = 3, cex.axis = 1.5
)
mtext(side = 1, font = 1, "Error Percentile (%)", cex = 2, line = 3.25)
mtext(side = 2, font = 1, "Percent Error (%)", cex = 2, line = 3.25)

axis(side = 2, at = seq(0, 150, 10), labels = T, cex.axis = 1.5)
legend("topleft",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"),
  title = "Error Percentile", cex = 1.5
)
```




```
par(mar = margins)

# rw
obs <- test_data[, "rw"]
pred <- preds[[5]][[1]]
diff <- pred - obs
diff.perc <- diff * 100 / obs

est_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  pred = pred, obs = obs,
  xlab = " ",
  ylab = " ",
  ylim = c(2, 7), xlim = c(2, 6.5),
  cex.lab = 2, cex.main = 2, cex.axis = 1.5
)
mtext(
  side = 2, font = 2, expression(paste("Estimated ", italic(r[w]))),
  cex = 2, line = 3
)
mtext(
  side = 1, font = 2, expression(paste("Simulation Input ", italic(r[w]))),
  cex = 2, line = 4
)
legend("bottomright",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"), title = "Error Percentile", cex = 1.5
)
```




```
error_plot(diff.perc, c(50, 75, 90), brewer.pal(3, "Set1"),
  yaxt = "n",
  ylab = "", ylim = c(0, 50),
  cex.lab = 2, cex.main = 2, cex.axis = 1.5
)
mtext(side = 1, font = 1, "Error Percentile (%)", cex = 2, line = 3.25)
mtext(side = 2, font = 1, "Percent Error (%)", cex = 2, line = 3.25)
axis(side = 2, at = seq(0, 50, 5), labels = T, cex.axis = 1.5)
legend("topleft",
  legend = c("0%-50%", "50%-75%", "75%-90%", "90%-100%"),
  pch = 16, col = c(brewer.pal(3, "Set1"), "black"), title = "Error Percentile"
)
```



```
save.image("walkthrough_data.RData")
```