Demo #5

Cross Validation

Jeffrey Yarus

27 May, 2025

Contents

1 Demo Summary

In this demo, the objective is to cross validate the variogram model we created previously.

Please be sure to read through the text I have included before each code chunk.

If you wish to run the demo, be sure to**run the entire code chunk by chunk.** The variable P_PHI_pct is already in place and ready to run. The variogram model and kriging parameters are already set. **Erase your Global Environment before beginning**

We will recreate the experimental semivariogram and variogram model from the earlier demo using ther variable, P_PHI_pct

- Be sure you are using the appropriate database with NO outliers!
- The objective is to evaluate the performance of your variogram model on the kriging or conditional simulation results and provide a method for selecting the “best” variogram model.

- Run the cross validation code chunks to help us determine which of several variogram model(s) work best (i.e., which generated the fewest number of over and under estimates). The idea is to run cross validation on the variogram models you think are best. Then, see which one generates the fewest over and under estimates.

Be careful not to overfit the variogram models.

- Keep in mind that selecting a bunch of variogram models and allowing the auto.fit() function from RGeostats to generate a really nice fit to the experimental variogram, does not mean that the resulting kriged map will be reasonable! There is a risk of overfitting. As a rule of thumb, when using auto.fit(), selecting fewer model-types to evaluate is generally better than selecting many.

reproduce the omnidirectional kriged map for P__PHI and its Error Variance (Standard Deviation) Maps using the "best" variogram model you constructed. Be sure you update (change) the various titles, axes names, input names, as appropriate within the body of each chunk where appropriate.

2 Terminology and key functions from this Demo

1. **Cross Validation** or, "leave one out"

- A bootstrapping method that will assess the value of each known sample one at a time and compare the estimated value with the actual value. The estimated value should be within 2.5 standard deviations of the known value. At the end of the analysis, a number of graphical statistics are presented. The preferred variogram model is the one with the least number of over or under estimates

2. **Estimation Error**

- The estimation error is the difference between the actual value and the estimated value

3. **Standard Error-Estimation Error**

- The standardized error is the normalized error in units of standard deviation

3 Loading Packages

```r
getwd()
```

```
## [1] "/mnt/vstor/CSE_MSE_RXF131/cradle-members/sdle/jmy41/GIT/jmy-research/topics/RPS_N58"
```

4 Initializing the Demo

As in the previous demos, we:

- Read in our data
- Create the data frame (tibble)
- Ensure our categorical variables (if present) are designated as factors
- Create a backup data frame
- Remove the unnecessary variables from the tibble
- Create our assigned symbolic names
- Identify the outliers.

Run the chunk below…BE SURE TO SET THE PATH CORRECTLY!

```r
file.name <- "../../data/WT-2D-all-outlier.csv"
df <- read_csv(file.name)
```

```
## Rows: 262 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): F_Top
## dbl (10): N_Well, C_X_ft, C_Y_ft, L_3_FACIES, P_Delta_t, P_KH_md, P_PHI, P_P...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
df %<>%
   mutate(across(matches("N_|L_"), factor))

# Creating a duplicate data frame for later use, but eliminate unnecessary variables

df2 <- df %>%
  dplyr::select(L_3_FACIES:P_Top_ft)

# Creating Symbolic Inputs

xlon <- "C_X_ft"
ylat <- "C_Y_ft"
property <- "P_PHI_pct"
head(df, n = 20)
```

```
## # A tibble: 20 x 11
##    F_Top       N_Well C_X_ft C_Y_ft L_3_FACIES P_Delta_t P_KH_md P_PHI P_PHI_pct
##    <chr>       <fct>   <dbl>  <dbl> <fct>          <dbl>   <dbl> <dbl>     <dbl>
##  1 S_Siltstone 5001    11564   5691 2               60.8    3.51 0.094       9.4
##  2 S_Siltstone 5002    10679  13706 2               59.1    5.81 0.093       9.3
##  3 S_Siltstone 5003     6311  36307 1               55.9    0.77 0.059       5.9
##  4 S_Siltstone 5004     2754  11437 1               52.5    0.55 0.044       4.4
##  5 S_Siltstone 5005     9386  24799 3               63.3    6.29 0.099       9.9
##  6 S_Siltstone 5006    10761  24755 3               63.1    2.9  0.099       9.9
##  7 S_Siltstone 5007    10769  23463 3               65.9   12.3  0.113      11.3
##  8 S_Siltstone 5008     9145  23413 1               57.9    0.79 0.074       7.4
##  9 S_Siltstone 5009     8201  23557 3               64.4    8.19 0.104      10.4
## 10 S_Siltstone 5010     6727  23472 1               55.8    1.45 0.071       7.1
## 11 S_Siltstone 5011     3834  23446 1               53.9    1.6  0.062       6.2
## 12 S_Siltstone 5012     2780  23494 1               54.8    1.57 0.063       6.3
## 13 S_Siltstone 5013     3994  22098 1               53.5    1.51 0.06        6
## 14 S_Siltstone 5014     5482  22214 1               53.9    1.79 0.065       6.5
## 15 S_Siltstone 5015     6692  22138 2               60.8    8.21 0.091       9.1
## 16 S_Siltstone 5016     9324  22199 3               63.5    4.48 0.1        10
## 17 S_Siltstone 5017    10817  22305 2               61.7    2.73 0.093       9.3
## 18 S_Siltstone 5018    10761  20774 1               57.4    2.68 0.073       7.3
## 19 S_Siltstone 5019     9350  20761 3               62.5    3.42 0.096       9.6
## 20 S_Siltstone 5020     8237  20748 1               57.2    3.82 0.079       7.9
## # i 2 more variables: P_Thickness <dbl>, P_Top_ft <dbl>
```

```
# Idnetifying the outliers

df <-
  df %>%
  mutate(
    iqr_val = IQR(!!sym(property)),
    iqr_val_adj = ((iqr_val) * 1.5),
    third_q = quantile(!!sym(property), prob = 0.75, na.rm = TRUE),
```

```r
    first_q = quantile(!!sym(property), prob = .25, na.rm = TRUE),
    outlier =
      (!!sym(property)) > (third_q + iqr_val_adj) |
      (!!sym(property) < (first_q - iqr_val_adj))) %>%
    mutate(Log_property = log(!!sym(property)))
```

5 Working in R S4 - Preparation for Geostatistical Analysis using RGeostats

5.1 Database

The following chunk creates a database **that does not include outliers**

5.2 Creating databases with no outliers

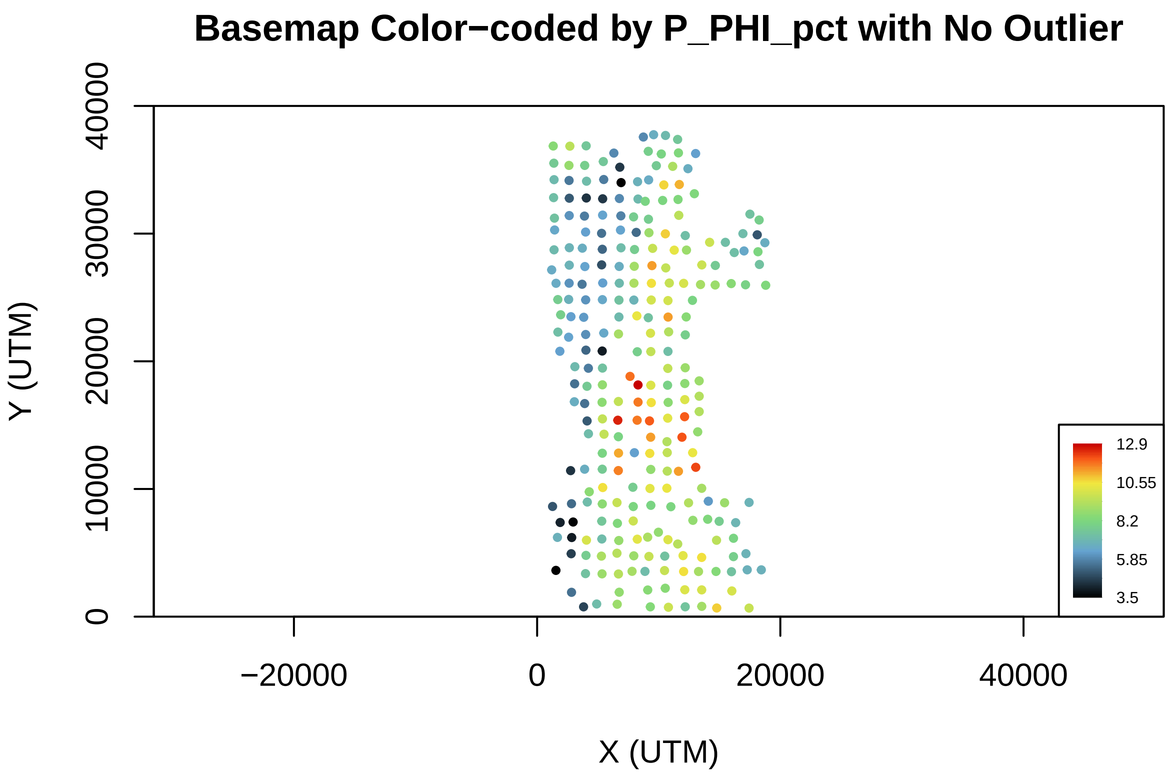
Here, we "filter out" the outliers (when they are present)

```r
db_noout.db <-
  df %>%
  dplyr::select(-F_Top) %>%
  filter(outlier == FALSE) %>%  # When FALSE, outliers will be "filtered out."
  db.create() %>%
  db.locate(c(xlon, ylat), "x") %>%
  db.locate(names = property, loctype = "z")
```

5.3 Plotting a basemap from a database

We will plot the basemap to ensure we have identified and removed outliers

```r
db.plot(
  db_noout.db,
  name.color = property,
  asp = 1.05,
  pos.legend = 1,
  cex = 0.5,
  xlim = c(0, 20000),
  ylim = c(0, 40000),
  pch = 19,
  xlab = "X (UTM)",
  ylab = "Y (UTM)",
  title = paste("Basemap Color-coded by", property, "with No Outlier"))
```



The basemap looks good, so we will continue and recreate To determine the ceiling of the grid (or the coordinates of the upper-right corner), I wrote a roundUp() function that takes the largest value in both selected x-coordinate (C_X) and y-coordinate (C_Y) column and find the nearest 10, 100, 1000, etc. that's larger than the largest value

5.4 Determining the extents of the data for grid constructon

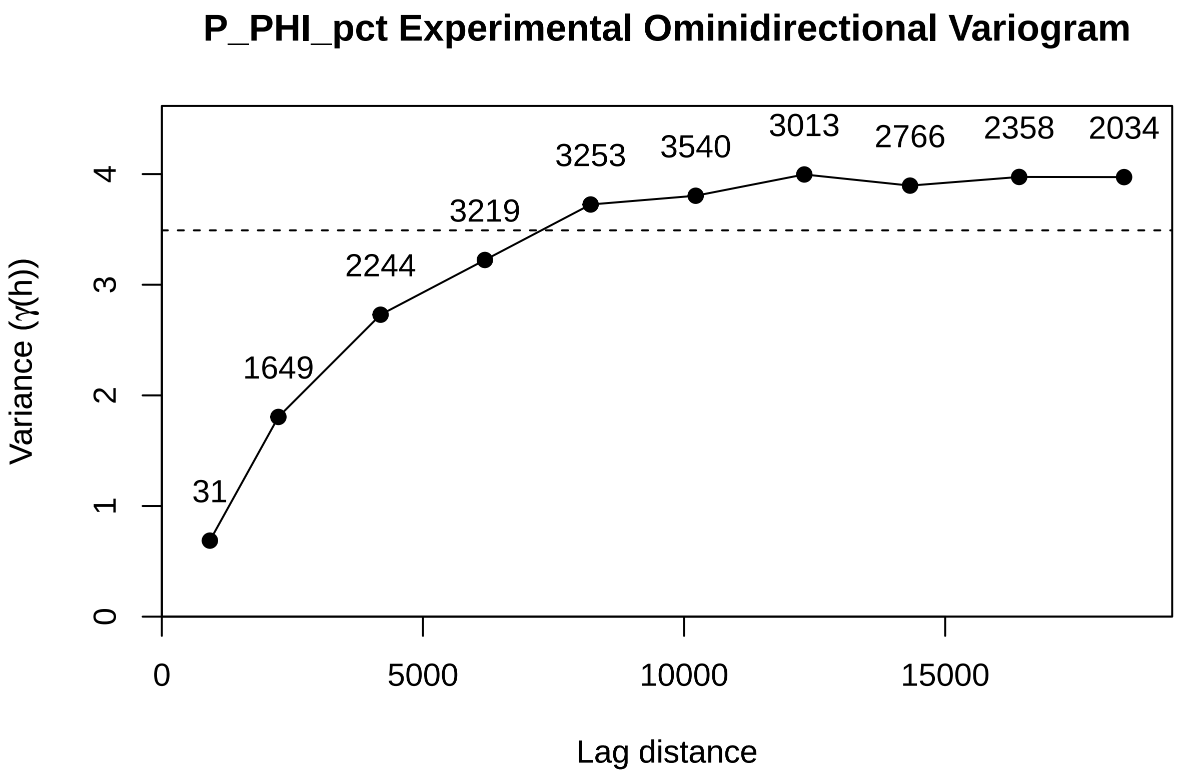
5.5 Construction of the Variograms

5.5.1 Omnidirectional Variogram Model

Be sure to look at the information printed out on the omnidirectional variogram The code used here is similar to what would be used for directional variograms

Run the chunk below…

```
vario.omni <- vario.calc(db_noout.db, nlag = 10)
plot(
  vario.omni,
  #  lag = 15,
  type = "o",
  pch = 19,
  npairpt = TRUE,
  npairdw = FALSE,
  title = paste(property, "Experimental Ominidirectional Variogram"),
  xlab = "Lag distance",
  ylab = expression(paste("Variance (", gamma, "(h))", sep = ""))
)
```



# Visualizing the variogram model

5.6 Select model type for this data set:

Here I have selected a the Exponential (#2) model for the demo. However, that may not be the correct variogram model. You will need to select from the variogram list what you believe to be the "best" variogram model that you will then validate through cross validation.

Make your variogram selection in this code chunk. It's possible to include more than 1 basic structure to fit the model with, the program will calculate and pick out the one that fits best. In the chunk below, I have specified only one variogram model type. If you specify more than one, the fitting algorithm will decide which one or which one or combinations are best. Be careful, the algorithm will be inclined to build nested structures to precisely fit the experimental variogram, so overfitting is very possible! Less is generally better. **Also note that the specification of the variogram(s) can be done by explicitly naming the ones you want (in quotes), or by listing the numbers from the variogram list above. In the code chunk below, I am selecting only the Stable variogram model. That may not necessarily be the correct model for other variables!**

```
melem.name()
```

```
##  [1] "Nugget Effect"   "Exponential"     "Spherical"       "Gaussian"
##  [5] "Cubic"           "Cardinal Sine"   "J-Bessel"        "K-Bessel"
##  [9] "Gamma"           "Cauchy"          "Stable"          "Linear"
## [13] "Power"           "Order-1 G.C."    "Cosinus"         "Triangle"
## [17] "Cosexp"          "1-D Regularized" "Penta"           "Storkey"
## [21] "Wendland-2,0"    "Wendland-3,1"    "Wendland-4,2"    "Markov"
```

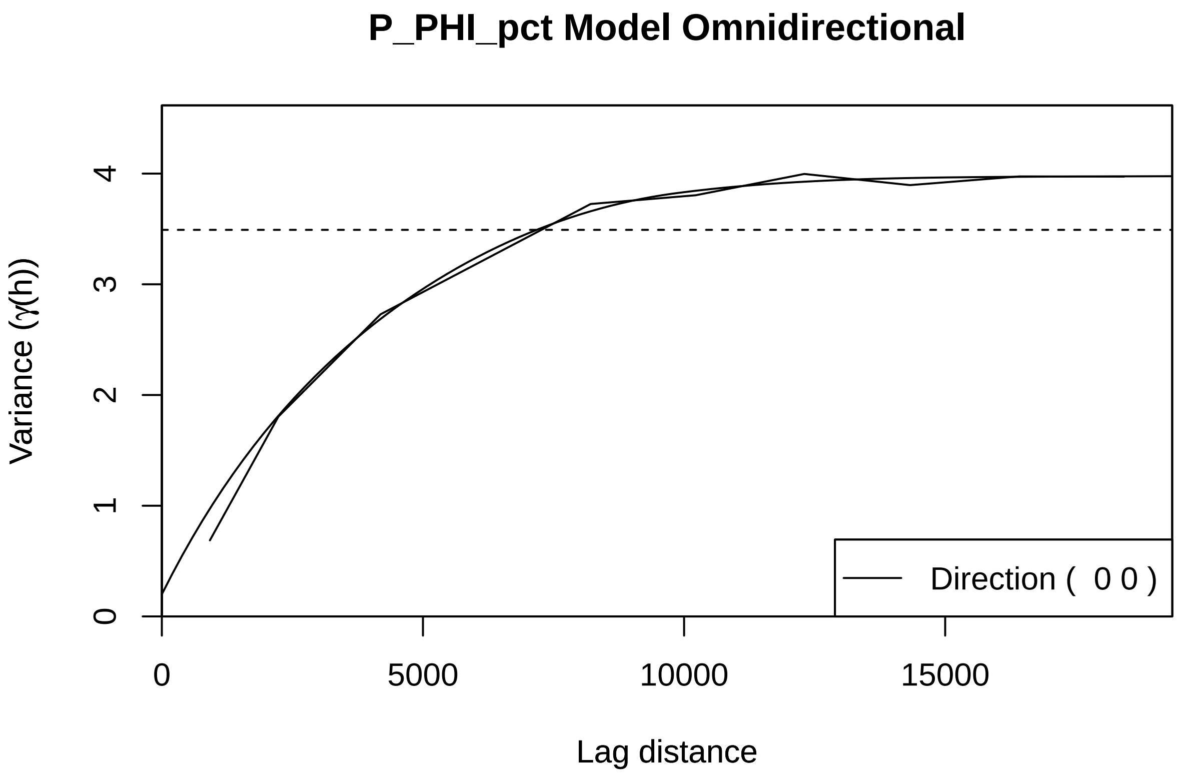
Select the variogram model(s) you wish to use

```
struct <- c(1,2,3,5)
```

5.7 Understanding the Variogram Model

5.7.1 Omnidirectional Variogram Model

```
data.model.omni <-
  model.auto(
    vario.omni,
    struct = struct,
    flag.noreduce = TRUE,      # following code for adjusting nugget
    lower = c("M1V1=0.2"),
    upper = c("M1V1=0.5"),
    title = paste(property, "Model Omnidirectional"),
    pos.legend = 1,
    xlab = "Lag distance",
    ylab = expression(paste("Variance (", gamma, "(h))", sep = ""))
  )
```



```
data.model.omni
```

```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 4
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill          =      0.200
## Exponential
## - Sill          =      2.352
## - Range         =   8670.299
## - Theo. Range   =   2894.217
## Spherical
## - Sill          =      0.880
## - Range         =   9859.490
## Cubic
## - Sill          =      0.548
## - Range         =  17055.174
## Total Sill      =      3.979
##
## Drift Part
## ----------
## Universality Condition
```

When performing cross validation, a neighborhood must be defined. Recall, the neighborhood defines the region within which samples are selected for use in the kriging estimation algorithm. In this case, the data set is considered small, so we can simply use a "unique" or "global" neighborhood, meaning use all the samples in the data set.

6 Neighborhood design

The choice of neighborhood and the neighborhood design is important, particularly when there is a lot of data. The basic choices are either a **Unique** neighborhood or a **Moving** neighborhood. A unique, or global neighborhood requires essentially no parameterization and uses all the sample data. This could be problematic if there were a lot of data as it would greatly slow the computational time. In this case, there are only 261 samples and this is considered quite small, so we will use the unique neighborhood option. A moving neighborhood is more efficient when dealing with large data sets. However, it requires more parameterization like; the size of the search circle or ellipse, the number of sectors that divide the neighborhood in to equal size compartments, the optimal number of samples to gather in each sector, and the optimal number of samples to gather overall.

In the chunk below, we're going to create a unique neighborhood using function neigh.create() with type "0" for Unique Neighborhood and nidm = 2 for 2-Space Dimensions

```
neigh.unique <- neigh.create(type = 0, ndim = 2)
```

We can also create a moving neighborhood that we can use for comparison. Recall that a moving neighborhood restricts the selected neighboring points to those closer to the estimation node. In this case, selecting up to the closest 50 neighbors within 10,000m of the estimation node.

```
myneigh <- neigh.create(
  ndim = 2,
  nmaxi = 50,
  radius = 10000)
```

Here we create a second database (db_noout.db2) so we preserve the original

```
db_noout.db2 <-
  db_noout.db
```

7 Evaluation of the variogram using Cross Validation

The function xvalid() in RGeostats produces estimates of the known sample data. It does this by dropping one sample value at a time and using the neighboring data along with the kriging algorithm to re-estimate it. This way we can compare the actual values to the kriging estimates. Recall that kriging requires a variogram which we have modeled using the code above. If we change the variogram model, the estimates will change. We can construct different variogram models and run the xvalid() (cross validation) exercize to help us determine which variogram model works best.

The code produces a set of new variables that are listed in the database identified for the cross validation, in this case db_noout.db2. The new variables are as follows:

- estim: Estimated value
- stderr: Standardized Error (sometimes referred to as the standardized residual), the deviation of an observed value from its expected value, scaled by the standard deviation of the error term (the difference between an observed value and its expected value, divided by the standard deviation of the error term: (Z*-Z)/S). Don't confuse this with, "Standard Error," which refers to the variability or dispersion of a sample statistic relative to the true population parameter.

- esterr: Estimation Error, the difference between the predicted value generated by a model and the true value of the target variable accounting for factors such as bias, variance, and irreducible error that may influence the model's performance on unseen data.
- stdev: Standard Deviation (S)

In this code chunk, we select the appropriate variable location for the db.locate() function. **For P_PHI_pct, the location is, "9".** If you select another variable, like P_KH_md, **the location is, "8".** To see a list of the variables and their positions, you can open the twisty for db_noout.db2 in the Global Environment and count the variables listed under @items

```
db_noout.db2 <- db.locerase(db_noout.db2, "z")

db_noout.db2 <-
  db.locate(db_noout.db2, "9", "z")

db_noout.db2 <-
  xvalid(
    db_noout.db2,
    data.model.omni,
    neigh.unique,              # comment-out to check Moving Neighborhood
#    myneigh,                  # Uncomment to check Moving Neighborhood
    flag.est = -1,             # If 1, (Z*-Z, esterr). If -1, (Z*, est)
    radix = "XvalUniq1")       # comment-out to check Moving Neighborhood
#    radix = "XvalMov1")       # Uncomment to check Moving Neighborhood

db_noout.db2 <- db.locerase(db_noout.db2, "z")

db_noout.db2 <-
  db.locate(db_noout.db2, "9", "z")

db_noout.db2 <-
  xvalid(
    db_noout.db2,
    data.model.omni,
    neigh.unique,           # comment-out to check Moving Neighborhood
#    myneigh,               # Uncomment to check Moving Neighborhood
    flag.std = -1,          # If 1, (Z*-Z)/S. If -1, (S) S = Std Dev of est error
    radix = "XvalUniq2")  # comment-out to check Moving Neighborhood
#    radix = "XvalMov2")  # Uncomment to check Moving Neighborhood

db_noout.db3 <-
  db_noout.db2

#db_noout.db3
```

If you select a different variable to analyze

You will need to change the name of the estimation variable. The code will generate the following variables: The estimate (estim), the standard deviation value (stdev), the estimation error value (esterr), and the standard error value (stderr) to match the variable being analyzed.

The radix argument simply provides a file identifier. Here, I used the radix = Uniq1 to signify that the analysis was run using a unique neighborhood. If you use the moving neighborhood, then I used radix = XvalMov2. You can create any radix you wish, but be consistent.

For example, if you wish to analyze P_Kh_md, you would make the following changes to the code chunk below

-XvalUniq1.P_PHI_pct.estim –> XvalUniq1.P_KH_md.estim

-XvalUniq1.P_PHI_pct.stdev –> XvalUniq1.P_KH_md.stdev

-XvalUniq1.P_PHI_pct.esterr –> XvalUniq1.P_KH_md.esterr

-XvalUniq1.P_PHI_pct.stderr –> XvalUniq1.P_KH_md.stderr

7.1 Graphical results of Cross Validation

It is possible to display the various cross validation results using the S4 functions in RGeostat. However, sometimes it's easier to use S3 functions like those of ggplot2. So, the code below converts the database (db) into a data frame (df). To do that, I am simply extracting the values we created and placed them in db_noout.db3 under **items@** and assigning them to a data frame called, **df_db_noout

```
df_db_noout <-
  data.frame(db_noout.db3@items)

P_property <-
  df_db_noout %>%
  ggplot(aes
  (!!sym(property))) +
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
  xlim(0,15)
P_estim <-
  df_db_noout %>%
    ggplot(
      aes(
       XvalUniq1.P_PHI_pct.estim)) +  #change to Xvalid.P_KH_md.estim
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
  xlim(0,15)
```

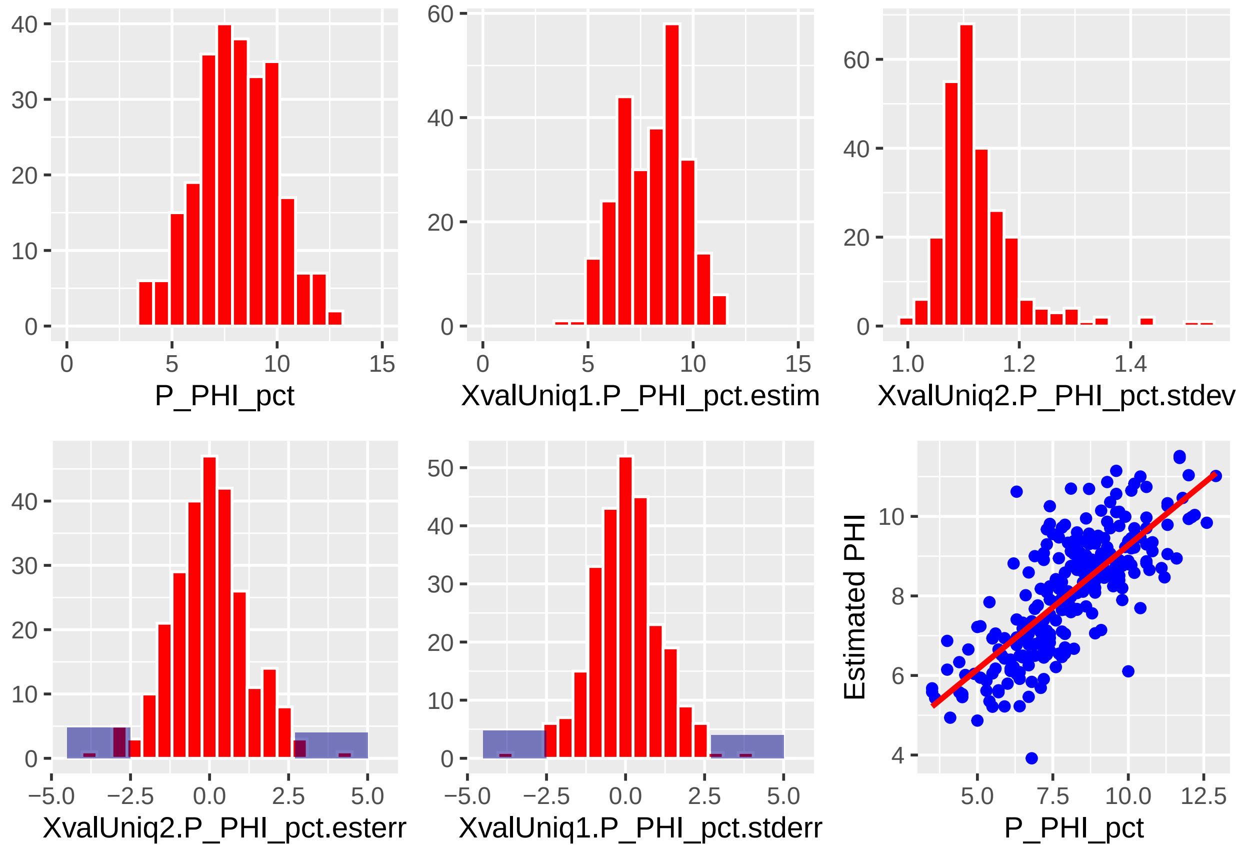
```r
P_stdev <-
  df_db_noout %>%
  ggplot(
    aes(
      XvalUniq2.P_PHI_pct.stdev)) + #change to Xvalid.P_KH_md.stdev
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21)

P_esterr <-
  df_db_noout %>%
  ggplot(
    aes(
      XvalUniq2.P_PHI_pct.esterr)) + #change to Xvalid.P_KH_md.esterr
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
    annotate("rect", xmin = -4.5, xmax = -2.5, ymin = 0, ymax = 4.8,
           alpha = .5,fill = "darkblue") +
    annotate("rect", xmin = 2.7, xmax = 5.0, ymin = 0, ymax = 4,
           alpha = .5,fill = "darkblue")

P_stderr <-
  df_db_noout %>%
  ggplot(
    aes(
      XvalUniq1.P_PHI_pct.stderr)) + #change to Xvalid.P_KH_md.stderr
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
    annotate("rect", xmin = -4.5, xmax = -2.5, ymin = 0, ymax = 4.8,
           alpha = .5,fill = "darkblue") +
    annotate("rect", xmin = 2.7, xmax = 5.0, ymin = 0, ymax = 4,
           alpha = .5,fill = "darkblue")

P_cross <-
df_db_noout %>%
  ggplot(
    aes(
      x = !!sym(property),
      y = XvalUniq1.P_PHI_pct.estim)) + #change to Xvalid.P_KH_md.estim
  geom_point(
    col = "blue") +
  geom_smooth(
    method = 'lm',
    col = "red",
    formula = y ~ x,
    se = FALSE
  ) +
  labs(y = "Estimated PHI") #change to "Estimated KH"
```

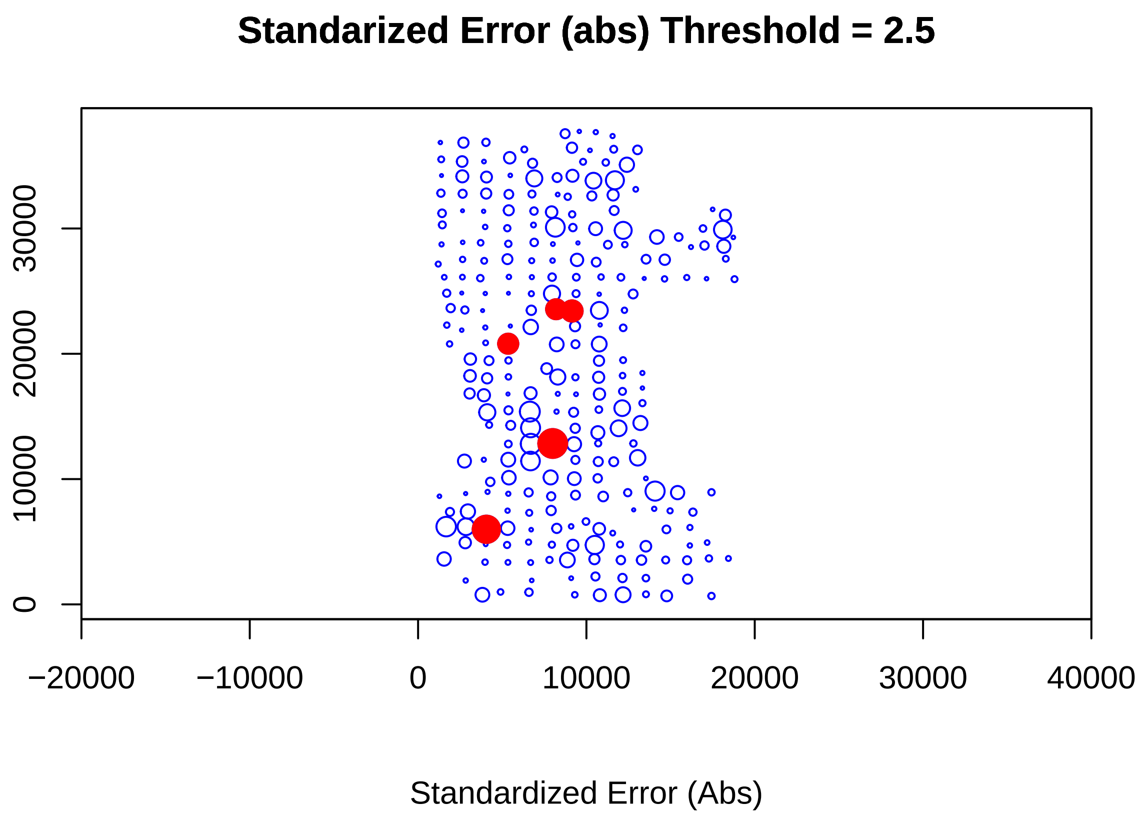
```
ggarrange(
  P_property +
    theme(
      axis.title.y = element_blank()),
  P_estim +
    theme(axis.title.y = element_blank()),
  P_stdev +
    theme(axis.title.y = element_blank()),
  P_esterr +
    theme(axis.title.y = element_blank()),
  P_stderr +
    theme(axis.title.y = element_blank()),
  P_cross +
    theme()
)
```



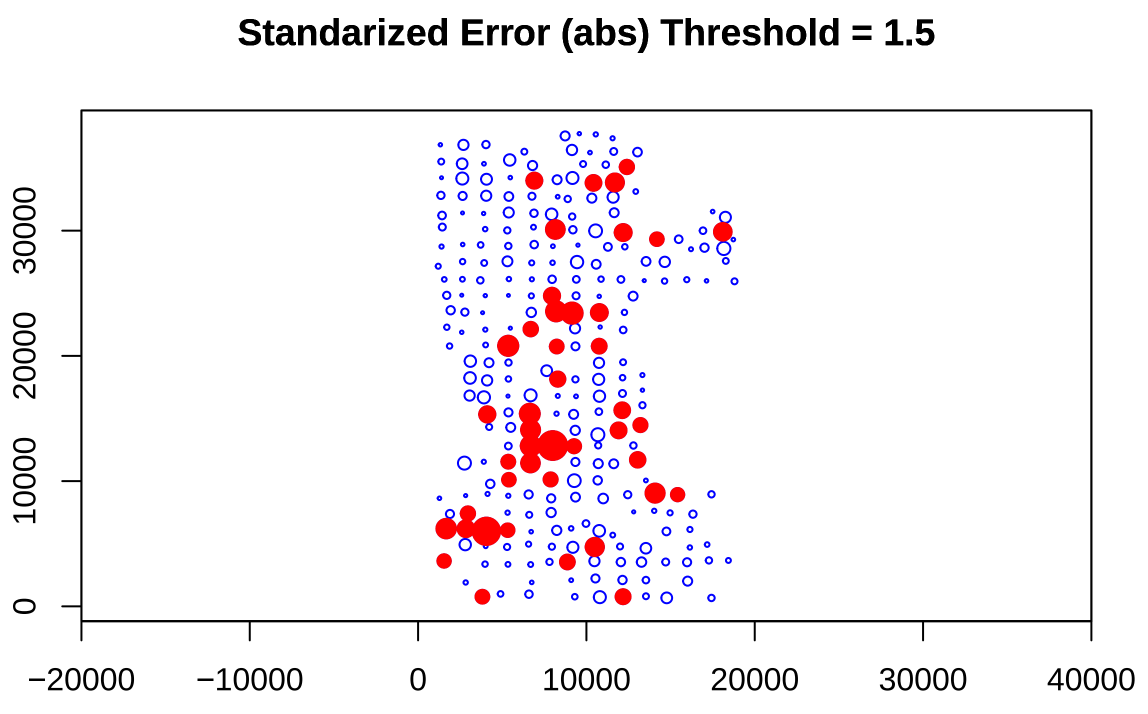
We cam also use the S4 objects in RGeostats. We use the draw.xvalid function to produce the basemap below which annotates in red the over and under estimates

Using S4, we can look more closely at a few statistical graphs to see the estimates that are beyond +/- 2.5 standard deviations from mean. First, the standardized error graph, then the cross plot of the standardized error against the estimated values, and finally, the cross plot of the estimated values against the actual values

```
draw.xvalid(
  mode = 1,
  db_noout.db3,
  thresh = 2.5,
  main="Standarized Error (abs) Threshold = 2.5",
  xlim = c(-20000, 40000))
```

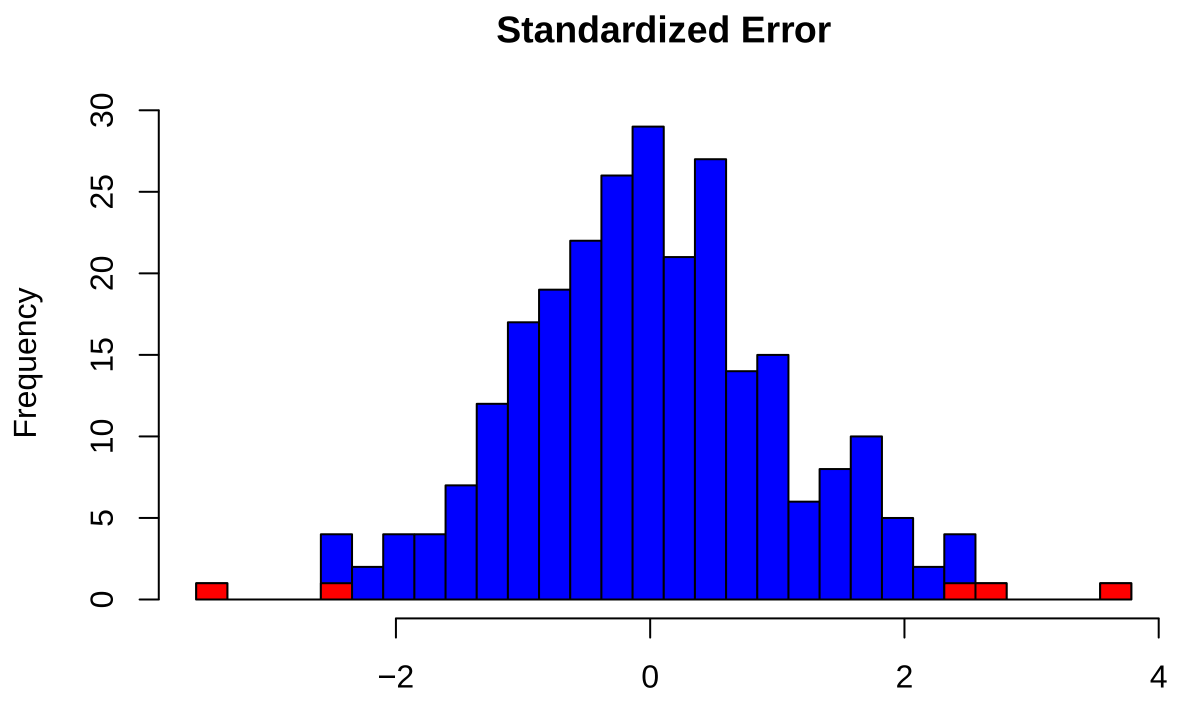


```
draw.xvalid(
  mode = 1,
  thresh = 1.5,
  main="Standarized Error (abs) Threshold = 1.5",
  xlim = c(-20000, 40000),
  db_noout.db3)
```

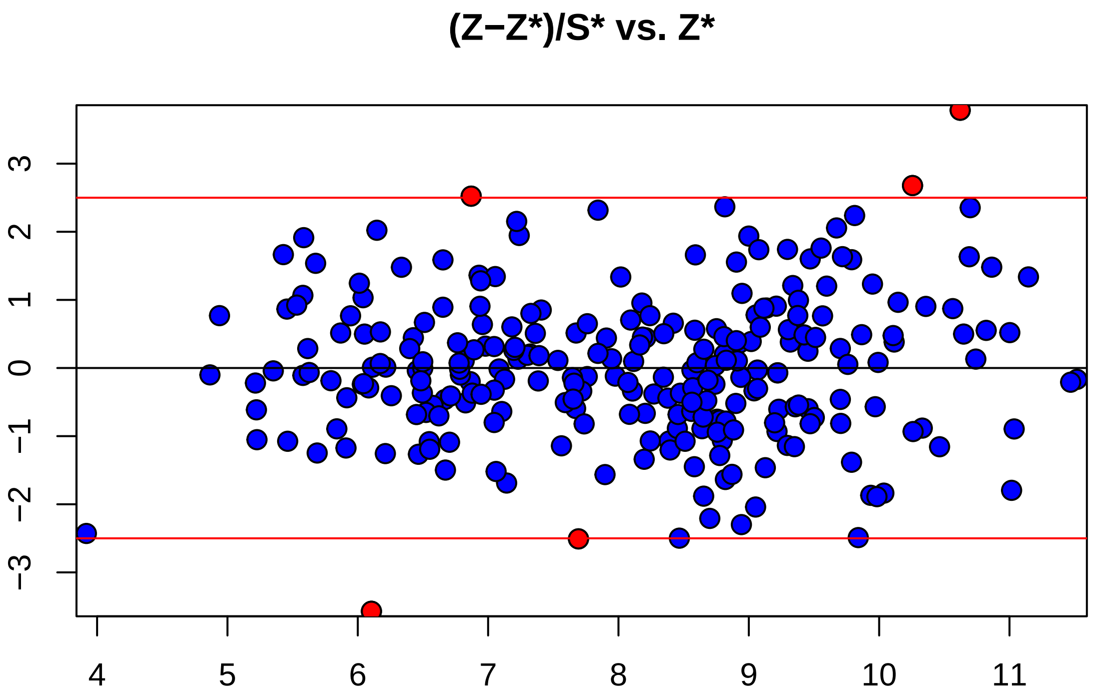


Standardized Error (Abs)

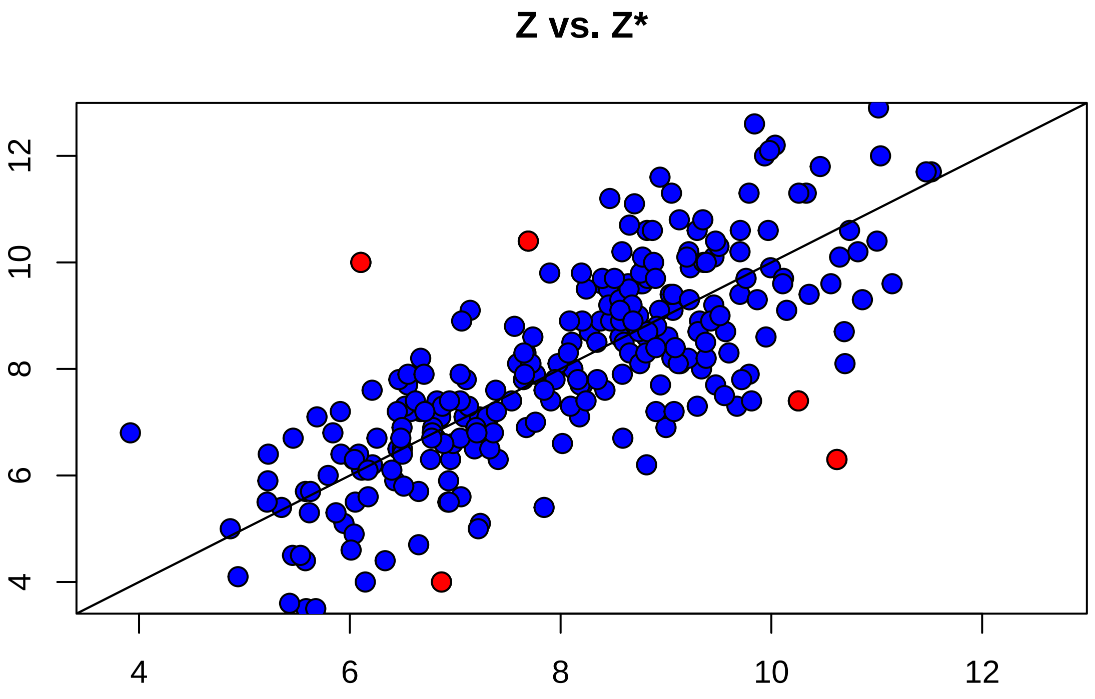
```
draw.xvalid(
  mode = 2,
  thresh = 2.5,
  nbins = 31,
  db_noout.db3)
```



```
draw.xvalid(
  mode = 3,
  thresh = 2.5,
  db_noout.db3)
```



```
draw.xvalid(
  db_noout.db3,
  property,
  thresh = 2.5,
  mode = 4)
```



Look at the various graphs just produced. The red values are those data samples where the estimates are more than 2.5 standard deviations from the mean except for the second image where I set the threshold to 1.5

From here, we can try different variogram model types to see which generated the fewest over and under estimates. NOTE: Fewer over and under estimates is a very mechanical way of determining which variogram model is best. In practice, you should use your best judgement as a domain expert to make a final decision
End of Demo #5