

# Demo #5

## Cross Validation

Jeffrey Yarus

01 juillet, 2025

## Contents

<b>1 Demo Summary</b>	<b>1</b>
<b>2 Terminology and key functions from this Demo</b>	<b>2</b>
<b>3 Loading Packages</b>	<b>2</b>
<b>4 Initializing the Demo</b>	<b>2</b>
<b>5 Preparation for Geostatistical Analysis using gstlearn</b>	<b>4</b>
5.1 Database . . . . .	4
5.2 Creating databases with no outliers . . . . .	4
5.3 Plotting a basemap from a database . . . . .	5
5.4 Construction of the Experimental Variogram . . . . .	5
5.4.1 Omnidirectional Variogram . . . . .	5
5.4.2 Visualizing the Variogram . . . . .	6
5.4.3 Select basic structures for fitting this experimental variogram . . . . .	6
5.5 Fitting the Variogram Model . . . . .	7
5.5.1 Isotropic Variogram Model . . . . .	7
<b>6 Neighborhood design</b>	<b>9</b>
<b>7 Evaluation of the variogram using Cross Validation</b>	<b>9</b>
7.1 Graphical results of Cross Validation . . . . .	11

## 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 outlier!
- 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 automatic fitting function from *gstlearn* 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 automatic fitting procedure, selecting fewer model types to evaluate is generally better than selecting many.

**Reproduce the kriged map for isotropic Model** 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. **Standardized Error** The standardized error is the normalized error in units of standard deviation

## 3 Loading Packages

Next chunk tells you where you are.

```
getwd()

## [1] "/home/drenard/project_gstlearn/gstlearn_studies/R/west_texas/gstlearn/scripts"
```

## 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!

```
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>
```

```
# Identifying 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),
```

```

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 Preparation for Geostatistical Analysis using gstlearn

### 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)

```

# Creating databases with no outliers (removing the first variable)
df1 = df %>% dplyr::select(-F_Top) %>% filter(outlier == FALSE)

# creating a database from the Data Frame
db_noout.db = Db_fromTL(df1)

# Define the different variables
err = db_noout.db$setLocators(c(xlon, ylat), ELoc_X())
err = db_noout.db$setLocator(property, ELoc_Z())
db_noout.db

```

```

##
## Data Base Characteristics
## =====
##
## Data Base Summary
## -----
## File is organized as a set of isolated points
## Space dimension          = 2
## Number of Columns        = 16
## Total number of samples   = 261
##
## Variables
## -----
## Column = 0 - Name = N_Well - Locator = NA
## Column = 1 - Name = C_X_ft - Locator = x1
## Column = 2 - Name = C_Y_ft - Locator = x2
## Column = 3 - Name = L_3_FACIES - Locator = NA
## Column = 4 - Name = P_Delta_t - Locator = NA
## Column = 5 - Name = P_KH_md - Locator = NA
## Column = 6 - Name = P_PHI - Locator = NA
## Column = 7 - Name = P_PHI_pct - Locator = z1
## Column = 8 - Name = P_Thickness - Locator = NA
## Column = 9 - Name = P_Top_ft - Locator = NA
## Column = 10 - Name = iqr_val - Locator = NA
## Column = 11 - Name = iqr_val_adj - Locator = NA
## Column = 12 - Name = third_q - Locator = NA
## Column = 13 - Name = first_q - Locator = NA

```

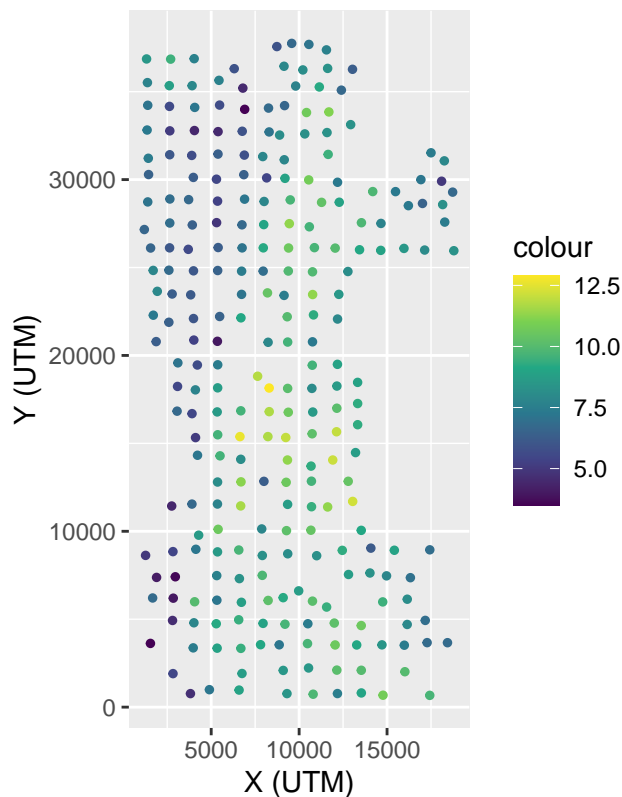
```
## Column = 14 - Name = outlier - Locator = NA
## Column = 15 - Name = Log_property - Locator = NA
```

### 5.3 Plotting a basemap from a database

We will plot the base-map to ensure we have identified and removed outliers

```
p = plot.init(asp=1)
p = p + plot.symbol(db_noout.db, nameColor = property, pch = 19, cex = 1, flagLegend = TRUE)
p = p + plot.decoration(title=paste("Basemap of", property, "with No Outlier"), xlab = "X (UTM)", ylab = "Y (UTM)", plot.end(p)
```

Basemap of P\_PHI\_pct with No Outlier



The base-map looks good, so we will continue.

### 5.4 Construction of the Experimental Variogram

#### 5.4.1 Omnidirectional Variogram

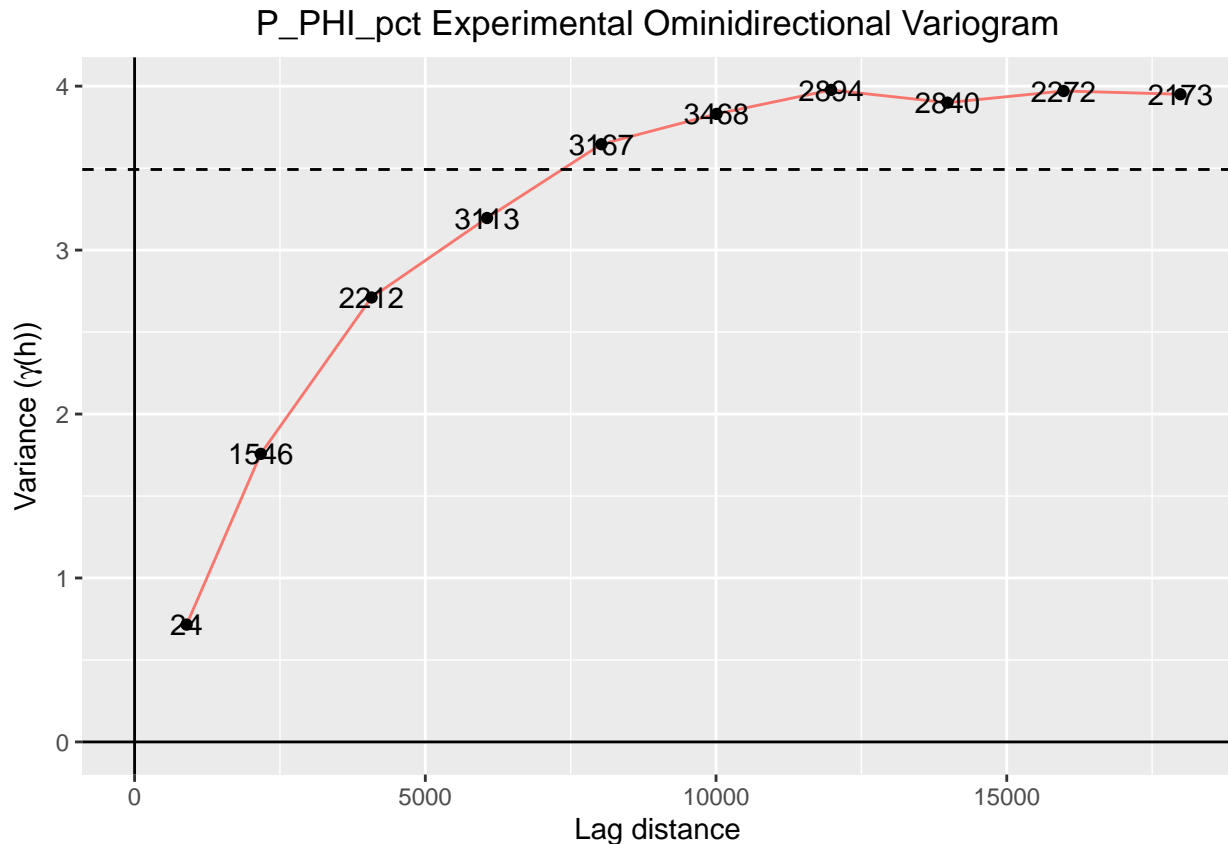
Be sure to look at the information printed out on the omnidirectional variogram. The code used here is similar to what would be used for directional variograms.

Run the chunk below...

```
varioparam <- VarioParam_createOmniDirection(nlag = 10, dlag=2000)
vario_omni <- Vario_computeFromDb(varioparam, db_noout.db)
```

### 5.4.2 Visualizing the Variogram

```
p = plot.init()
p = p + plot.vario(vario_omni, drawLabel = TRUE)
p = p + plot.decoration(title = paste(property, "Experimental Ominidirectional Variogram"),
                        xlab = "Lag distance",
                        ylab = expression(paste("Variance (", gamma, "(h))", sep="")))
plot.end(p)
```



### 5.4.3 Select basic structures for fitting this experimental variogram

Here I have selected the Exponential basic structure 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 basic structure. If you specify more than one, the fitting algorithm will decide which combination is 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 basic structure(s) can be done by explicitly naming the ones you want from the list below**

```
ECov_printAll()
```

```
## -2 - UNKNOWN : Unknown covariance
## -1 - FUNCTION : External covariance function
## 0 - NUGGET : Nugget effect
```

```
## 1 - EXPONENTIAL : Exponential
## 2 - SPHERICAL : Spherical
## 3 - GAUSSIAN : Gaussian
## 4 - CUBIC : Cubic
## 5 - SINCARD : Sine Cardinal
## 6 - BESSELJ : Bessel J
## 7 - MATERN : Matern
## 8 - GAMMA : Gamma
## 9 - CAUCHY : Cauchy
## 10 - STABLE : Stable
## 11 - LINEAR : Linear
## 12 - POWER : Power
## 13 - ORDER1_GC : First Order Generalized covariance
## 14 - SPLINE_GC : Spline Generalized covariance
## 15 - ORDER3_GC : Third Order Generalized covariance
## 16 - ORDER5_GC : Fifth Order Generalized covariance
## 17 - COSINUS : Cosine
## 18 - TRIANGLE : Triangle
## 19 - COSEXP : Cosine Exponential
## 20 - REG1D : 1-D Regular
## 21 - PENTA : Pentamodel
## 22 - SPLINE2_GC : Order-2 Spline
## 23 - STORKEY : Storkey covariance in 1-D
## 24 - WENDLAND0 : Wendland covariance (2,0)
## 25 - WENDLAND1 : Wendland covariance (3,1)
## 26 - WENDLAND2 : Wendland covariance (4,2)
## 27 - MARKOV : Markovian covariances
## 28 - GEOMETRIC : Geometric (Sphere only)
## 29 - POISSON : Poisson (Sphere only)
## 30 - LINEARSPH : Linear (Sphere only)
```

```
## NULL
```

Select the basic structure(s) you wish to use

```
types = ECov_fromKeys(c("NUGGET", "EXPONENTIAL", "SPHERICAL", "CUBIC"))
```

## 5.5 Fitting the Variogram Model

### 5.5.1 Isotropic Variogram Model

Having only calculated an omnidirectional variogram, it makes sense to fit it with a Model which is similar whatever the direction: an **isotropic** Model.

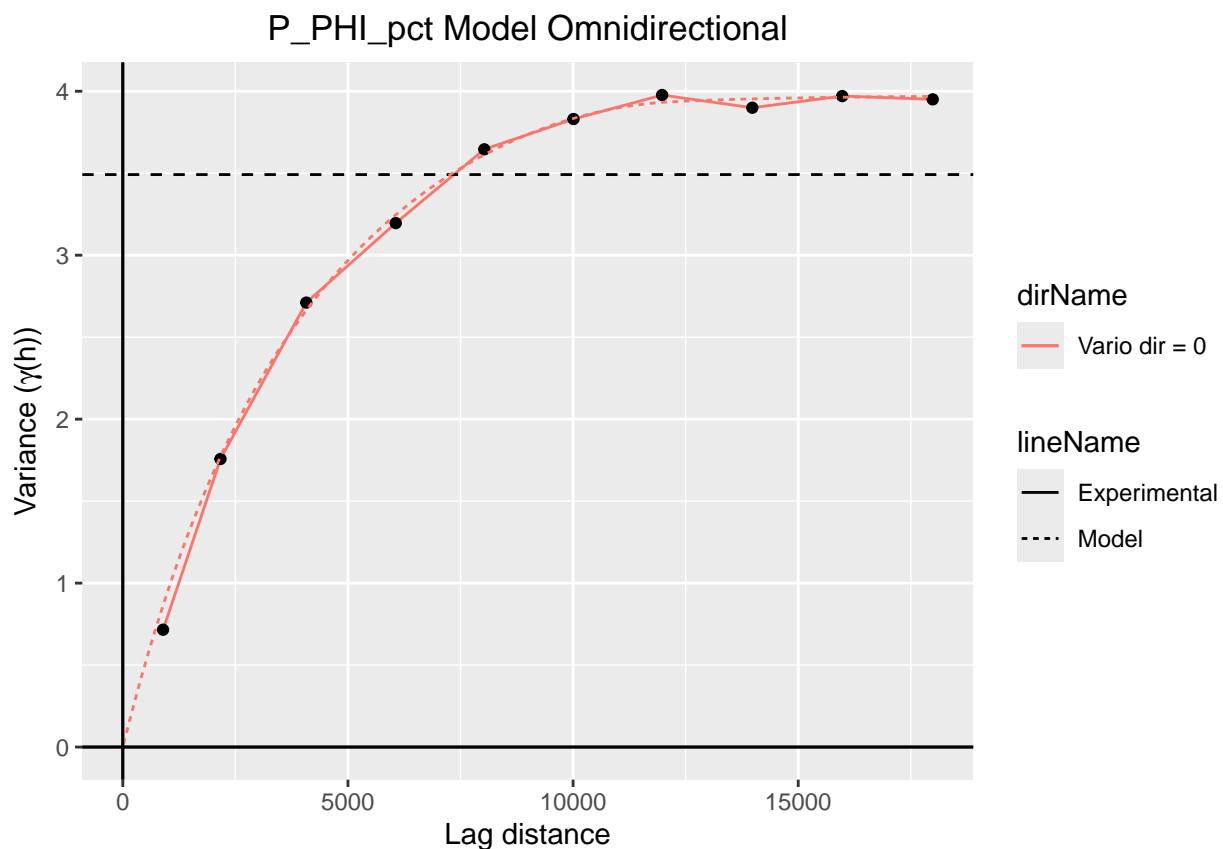
```
data.model.omni <- Model()
err = data.model.omni$fit(vario_omni, types=types)
data.model.omni
```

```
##
## Model characteristics
## =====
## Space dimension           = 2
## Number of variable(s)    = 1
## Number of basic structure(s) = 2
## Number of drift function(s) = 0
## Number of drift equation(s) = 0
##
```

```
## Covariance Part
## -----
## Exponential
## - Sill      =      2.688
## - Range     =    8543.036
## - Theo. Range = 2851.736
## Spherical
## - Sill      =      1.286
## - Range     = 12200.519
## Total Sill  =      3.973
## Known Mean(s)  0.000
```

In the next chunk, we visualize the Model together with the experimental variogram to appreciate the quality of the fit: the algorithm tries to minimize the (weighted) sum of the squared distance between the value of each variogram lag and the Model at the corresponding distance. The weight accounts for the distance (higher weight for smaller distance) and the number of pairs per lag (higher weight for larger number of pairs).

```
p = plot.init()
p = p + plot.varmod(vario_omni, data.model.omni, flagLegend = TRUE)
p = p + plot.decoration(title = paste(property, "Model Omnidirectional"), xlab = "Lag distance",
                          ylab = expression(paste("Variance (", gamma, "(h))", sep = "")))
plot.end(p)
```





## 6 Neighborhood design

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.

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.

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 this case, there are only 261 samples and this is considered quite small, so we will use the unique neighborhood option.

```
neigh.unique <- NeighUnique_create()
```

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 <- NeighMoving_create(nmaxi=50, radius=10000)
```

## 7 Evaluation of the variogram using Cross Validation

The function **xvalid** (for cross-validation in gstlearn) 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 Model 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 cross validation exercise 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. The new variables are as follows:

- estim: Estimated value ( $Z^*$ )
- stdev: Standard Deviation (S)
- esterr: Estimation Error ( $Z-Z^*$ ), 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.
- stderr: Standardized Error (sometimes referred to as the standardized residual), 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.

In this code chunk, we select the appropriate variable location. Note the specific arguments used in Naming-Convention. The first argument gives the **prefix** which will be systematically added to the variable name and the result type. The fourth one (set to FALSE) disables the capacity of setting automatically the newly calculated variables as the data variable (locator Z): thus, we do not have to set the appropriate variable between the two calls to **xvalid**.

```
err = db_noout.db$clearLocators(ELoc_Z())
err = db_noout.db$setLocator(property, ELoc_Z())
err = xvalid(db = db_noout.db,
  model = data.model.omni,
```

```

    neigh = neigh.unique,      # you can replace it by "myneigh"
    flag_xvalid_est = -1,      # If 1, (Z*-Z, esterr). If -1, (Z*, est)
    flag_xvalid_std = -1,      # If 1, (Z*-Z)/S. If -1, (S) S = Std Dev of est error
    namconv = NamingConvention("XvalUniq1", TRUE, TRUE, FALSE) # Adapt for Moving Neighborhood
)

err = xvalid(db = db_noout.db,
    model = data.model.omni,
    neigh = neigh.unique,
    flag_xvalid_est = 1,      # If 1, (Z*-Z, esterr). If -1, (Z*, est)
    flag_xvalid_std = 1,      # If 1, (Z*-Z)/S. If -1, (S) S = Std Dev of est error
    namconv = NamingConvention("XvalUniq1", TRUE, TRUE, FALSE)
)
db_noout.db

```

```

##
## Data Base Characteristics
## =====
##
## Data Base Summary
## -----
## File is organized as a set of isolated points
## Space dimension          = 2
## Number of Columns        = 20
## Total number of samples   = 261
##
## Variables
## -----
## Column = 0 - Name = N_Well - Locator = NA
## Column = 1 - Name = C_X_ft - Locator = x1
## Column = 2 - Name = C_Y_ft - Locator = x2
## Column = 3 - Name = L_3_FACIES - Locator = NA
## Column = 4 - Name = P_Delta_t - Locator = NA
## Column = 5 - Name = P_KH_md - Locator = NA
## Column = 6 - Name = P_PHI - Locator = NA
## Column = 7 - Name = P_PHI_pct - Locator = z1
## Column = 8 - Name = P_Thickness - Locator = NA
## Column = 9 - Name = P_Top_ft - Locator = NA
## Column = 10 - Name = iqr_val - Locator = NA
## Column = 11 - Name = iqr_val_adj - Locator = NA
## Column = 12 - Name = third_q - Locator = NA
## Column = 13 - Name = first_q - Locator = NA
## Column = 14 - Name = outlier - Locator = NA
## Column = 15 - Name = Log_property - Locator = NA
## Column = 16 - Name = XvalUniq1.P_PHI_pct.estim - Locator = NA
## Column = 17 - Name = XvalUniq1.P_PHI_pct.stdev - Locator = NA
## Column = 18 - Name = XvalUniq1.P_PHI_pct.esterr - Locator = NA
## Column = 19 - Name = XvalUniq1.P_PHI_pct.stderr - Locator = NA

```

### 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) for the variable being analyzed.

## 7.1 Graphical results of Cross Validation

It is possible to display the various cross validation results. However, sometimes it's easier to use S3 functions like those of ggplot2. So, the code below converts the database (db\_noout.db) into a data frame (df\_db\_noout).

```
df_db_noout <- db_noout.db$toTL()

P_property <-
  df_db_noout %>%
  ggplot(aes
    (!!sym(property))) +
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
  xlim(0,15) +
  labs(x = "Data")

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) +
  labs(x = "Estimation")

P_stdev <-
  df_db_noout %>%
  ggplot(
    aes(
      XvalUniq1.P_PHI_pct.stdev)) +
  geom_histogram(
    fill = "red",
    color = "white",
    bins = 21) +
  labs(x = "Standard Deviation")

P_esterr <-
  df_db_noout %>%
  ggplot(
    aes(
      XvalUniq1.P_PHI_pct.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") +
```

```

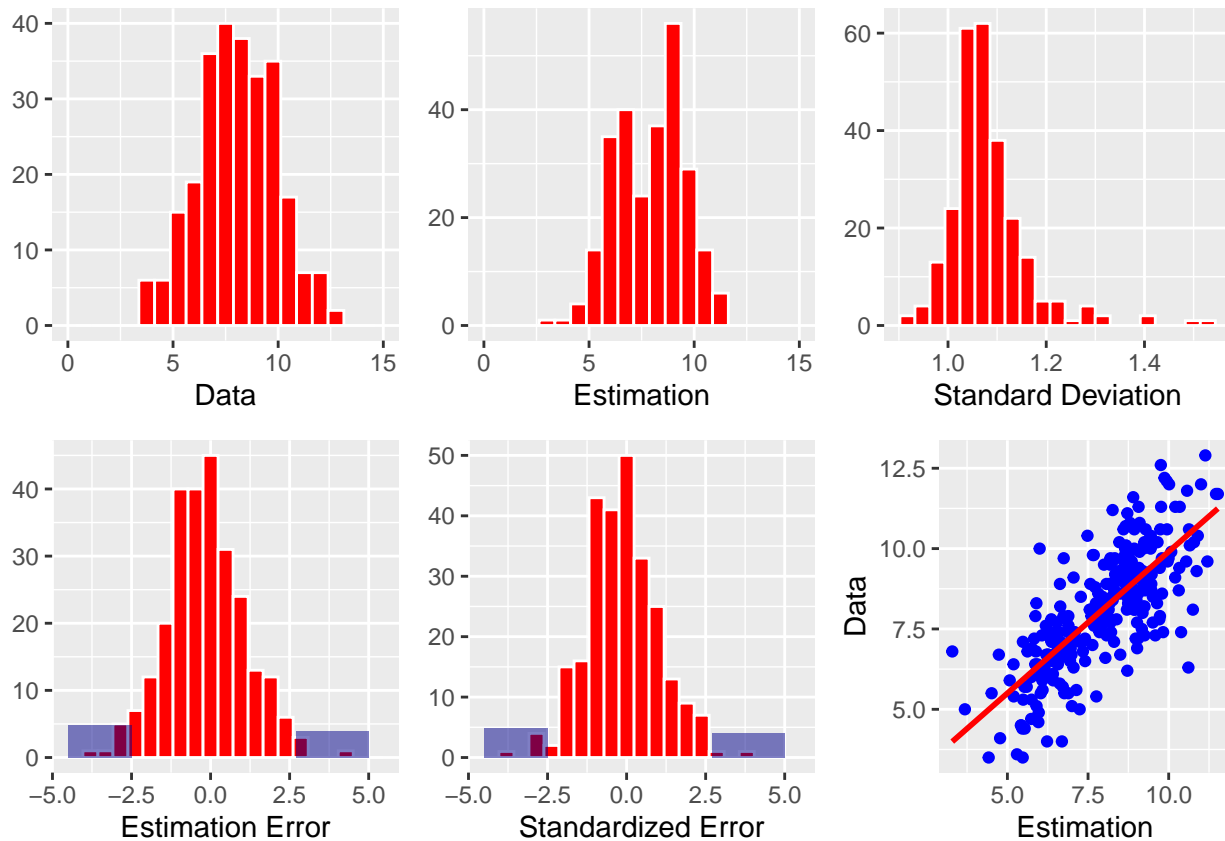
labs(x = "Estimation Error")

P_stderr <-
df_db_noout %>%
  ggplot(
    aes(
      XvalUniq1.P_PHI_pct.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") +
  labs(x = "Standardized Error")

P_cross <-
df_db_noout %>%
  ggplot(
    aes(
      y = !!sym(property),
      x = XvalUniq1.P_PHI_pct.estim)) +
  geom_point(
    col = "blue") +
  geom_smooth(
    method = 'lm',
    col = "red",
    formula = y ~ x,
    se = FALSE
  ) +
  labs(y = "Data") +
  labs(x = "Estimation")

ggarrange(
  P_property +
    theme(axis.title.y = element_blank()),
  P_estim +
    theme(axis.title.y = element_blank()),
  P_stddev +
    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()
)

```



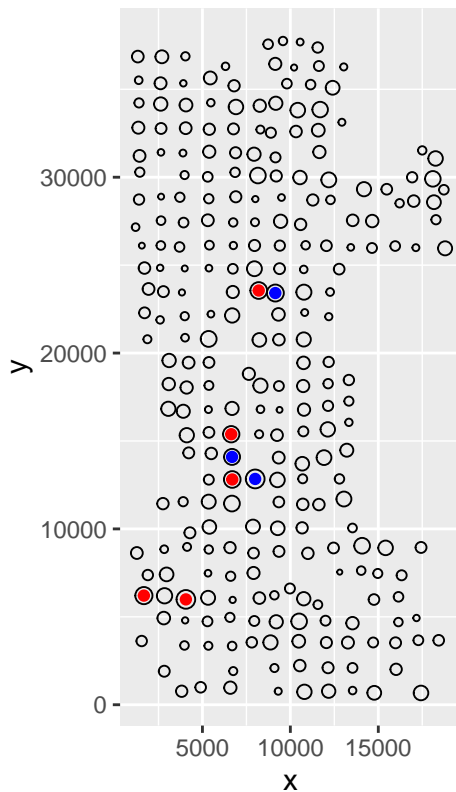
Using `gstlearn` functions, we can look more closely at a few statistical graphs to see the estimates that are beyond  $\pm 2.5$  standard deviations from mean. The graphics are:

- the base-map where the samples are represented with a size proportional to the absolute value of the standardized error. Note that the Data Base is duplicated to define a selection of the values where *stderr* is smaller than -2.5 (resp. larger than 2.5)

```
db_low.db = db_noout.db$clone()
iuid = db_low.db$addSelectionByVariable("*.stderr", upper=-2.5, name="sel_low")
db_sup.db = db_noout.db$clone()
iuid = db_sup.db$addSelectionByVariable("*.stderr", lower=2.5, name="sel_sup")

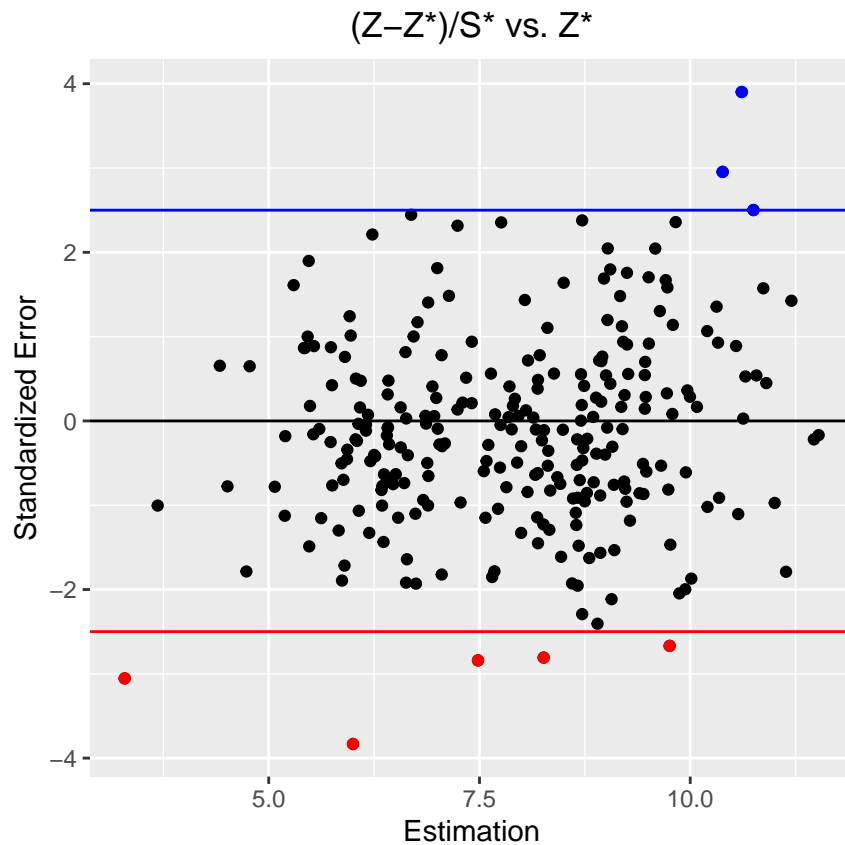
p = plot.init(asp=1)
p = p + plot.symbol(db_noout.db, nameSize = "*.stderr", flagAbsSize=TRUE, pch = 21, color='black')
p = p + plot.symbol(db_low.db, nameSize = "*.stderr", flagCst=TRUE, pch = 19, color='red')
p = p + plot.symbol(db_sup.db, nameSize = "*.stderr", flagCst=TRUE, pch = 19, color='blue')
p = p + plot.decoration(title="Standardized Error (abs) - Threshold = 2.5")
plot.end(p)
```

Standardized Error (abs) – Threshold = 2.5



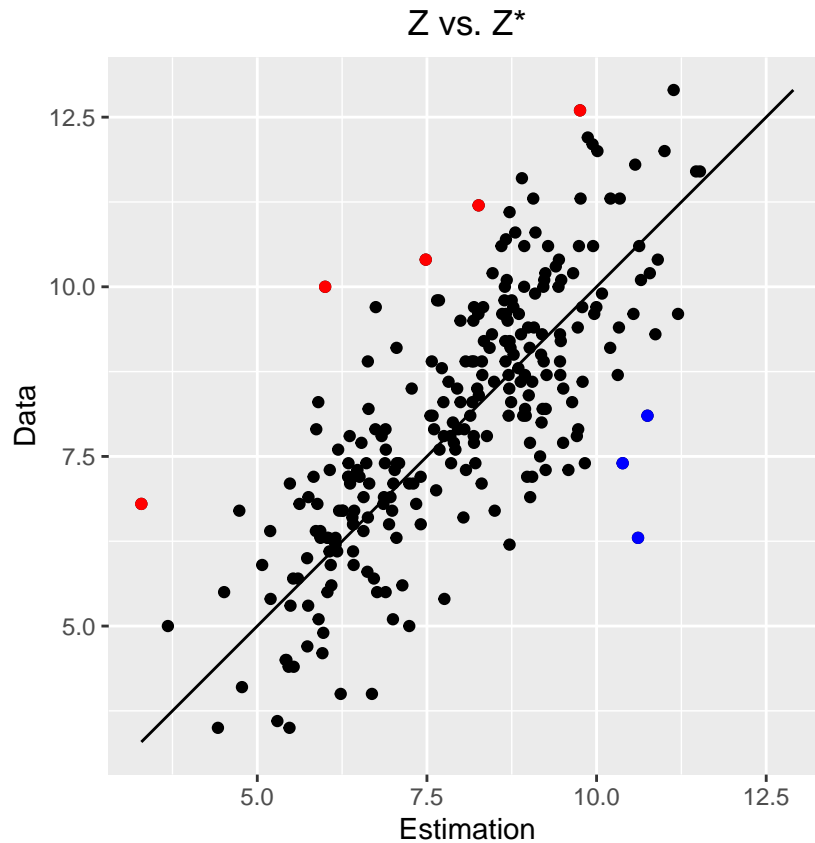
- the cross plot of the standardized error against the estimated values,

```
p = plot.init(asp=1)
p = p + plot.correlation(db_noout.db, namex = ".*estim", namey = ".*stderr", asPoint = TRUE)
p = p + plot.correlation(db_low.db, namex = ".*estim", namey = ".*stderr", asPoint = TRUE, color = 'red')
p = p + plot.correlation(db_sup.db, namex = ".*estim", namey = ".*stderr", asPoint = TRUE, color = 'blue')
p = p + plot.decoration(title="(Z-Z*)/S* vs. Z*", xlab = "Estimation", ylab = "Standardized Error")
p = p + geom_hline(yintercept = 0, lty = 1, color="black")
p = p + geom_hline(yintercept = -2.5, lty = 1, color="red")
p = p + geom_hline(yintercept = 2.5, lty = 1, color="blue")
plot.end(p)
```



- the cross plot of the estimated values against the actual values

```
p = plot.init(asp=1)
p = p + plot.correlation(db_noout.db, namex = "/*.estim", namey = property, asPoint = TRUE,
                        flagBiss=TRUE, bissColor='black')
p = p + plot.correlation(db_low.db, namex = "/*.estim", namey = property, asPoint = TRUE, col = 'red')
p = p + plot.correlation(db_sup.db, namex = "/*.estim", namey = property, asPoint = TRUE, col = 'blue')
p = p + plot.decoration(title="Z vs. Z*", xlab = "Estimation", ylab="Data")
plot.end(p)
```



Look at the various graphs just produced. The red values are those data samples where the estimates are less than -2.5 standard deviations from mean and the blue ones are for those where the estimates is more than 2.5 standard deviations from the mean.

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**