Conditional Simulations With Directional Variograms

Jeffrey Yarus

08 juillet, 2025

Contents

1 Loading Packages

Next chunk tells you where you are.

```
getwd()
```

```
## [1] "/home/fors/Projets/gstlearn/gstlearn_studies/R/west_texas/gstlearn/scripts"
```

2 Demo Summary

In this demo we present Conditional Simulation. After preparing the data frame and removing outliers, we demonstrate the following:

- construcion of the Normal Score Transform
- Variogram construction in normal space
- Conditional simulation
- Some basic post processing methods.

2.1 Instructions:

1. Run the entire code to see the full workflow.

- The current code is using P_PHI, run this first

2. Select the following after you have run the code with P_PHI_pct and read the material:

- P_KH_md
- Note: In this Demo the code is NOT fully automated. Some functions will require specification of the variable names and/or modification of titles

In this Demo, we identify a variogram model-type that "best" fits the experimental variogram. We use a variogram model constructed with the **normalized data**

- Apply the anamorphosis (Normal Score Transform) to the database with the variables requested above
- Build the experimental semi-variogram from the transformed data
- Fit the variogram model to the experimental semivariogram

4. Perform Conditional Simulation for the Directional Models only

- Technically, we should run hundreds of realization to capture the full uncertainty, but…
- Run at least 11 realizations. THIS may TAKE A BIT OF TIME, **SO DON'T BE SURPRISED!**
  - Be sure to use the appropriate database with NO outliers!
  - Be sure to use the appropriate outlier code if you are evaluating a log-normally distributed variable (like P_KH_md)
- Observe the results of the 11 realization

5. Perform Post Processing

- Calculate the mean of the realizations for the directional model
  - Plot the results
  - Feel free to "paste" any additional images that will help you make your point.

6. Calculate the error variance map (standard deviation) of the conditional simulations (for the directional model).

- Plot the results
- Feel free to "paste" any additional images that will help you make your point.

7. Knit the final results for P_KH_md

3 Reading Data File and Creating a duplicate data frame for later use, but eliminate unnecessary variables

- Creates the **data frame*
- Converting L_ and N_ variables to factors (categorical data)
- Print the first 20 lines of the data frame

4 Record your Homework answers here

Questions to consider:

1. How does the variogram model on the transformed data compare to the variogram model you made on the non-transformed data?
2. What are your observations regarding the different realizations?
3. Why is it not advisable to plot the contours on the conditional simulation results or the error variance map? You can un-comment the code that draws the contours and see what happens!
4. How does the results of the mean of the directional conditional simulations compare with the previous Kriged solution

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

4.1 Inputs

4.1.1 Symbolic variables

```
xlon <- "C_X_ft"
ylat <- "C_Y_ft"
out_analysis <- "Raw (outlier analysis performed on raw data)"
property <- "P_PHI_pct"
```

4.2 Data Analytics - Managing outliers and adding the log of the selected property to the df

Use the code below for data that are approximately symmetrically distributed, like porosity.

```
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)))
```

Use this code for right-skewed distributions) In the previous paragraph, modify the property to P_KH_md or any right-skewed distribution.

4.3 Creating a Database for use with gstlearn

4.3.1 Creating databases with no outliers

The following chunk creates a database **with outliers** from a data frame and assigns the location of the coordinates and the selected mapping variable ("property").

```
df1 = df %>% dplyr::select(-F_Top)

# creating a database with the outlier from the Data Frame
db = Db_fromTL(df1)

# Define the different variables
err = db$setLocators(c(xlon, ylat), ELoc_X())
err = db$setLocator(property, ELoc_Z())
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      = 262
##
## 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
```

4.3.2 Creating databases with no outliers

The following chunk creates a database **without outliers** from a data frame and assigns the location of the coordinates and the selected mapping variable ("property").

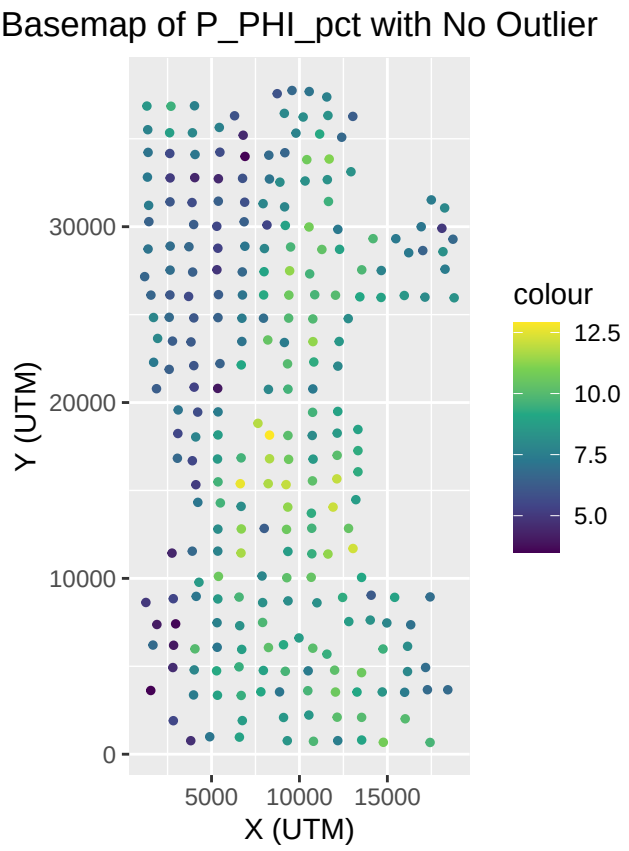
```
df1 = df %>% dplyr::select(-F_Top) %>% filter(outlier == FALSE)

# 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
```

4.4 Plotting a basemap from a database

4.4.1 Data Observations with no outlers

```
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 =
plot.end(p)
```



4.5 Neighborhood Design

4.5.1 Unique neighborhood

In the chunk below, we're going to create a unique neighborhood that uses all the data

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

4.5.2 Moving Neighborhood

At the same time, I would also want to experiment with creating a moving neighborhood for further purposes. Because there's not a need for moving neighborhood in this script, it's best to comment it out to decrease the running time of the program.

The inactive code below can be used to construct a moving neighborhood

```
#neigh.moving <- NeighMoving_create(nmaxi=10, nsect=8, nsmax=2, radius=10000)
```

4.6 Create the Grid in preparation for Kriging and Conditional Simulation

4.6.1 Determining the extents of the data for grid constructon

4.6.2 Creating the grid

```
dbgrid2 <- DbGrid_createCoveringDb(db_noout.db,dx=c(100,100))
dbgrid2
```

```
##
## Data Base Grid Characteristics
## ==============================
##
## Data Base Summary
## -----------------
## File is organized as a regular grid
## Space dimension              = 2
## Number of Columns            = 2
## Total number of samples      = 65844
##
## Grid characteristics:
## ---------------------
## Origin :   1198.000   670.000
## Mesh   :    100.000   100.000
## Number :        177       372
##
## Variables
## ---------
## Column = 0 - Name = x1 - Locator = x1
## Column = 1 - Name = x2 - Locator = x2
```

4.7 Performing the Anamorphosis (Normal Score Transform)

The Normal Score Transform (NST) converts the original variable being analyzed to an equivalent value in Gaussian Space. It does this through Q-Q plot of the quantiles for the input variable against the quantiles of the Gaussian Distribution. The Gaussian values are determined using the mean and standard deviation of the input variable which are subsequently plugged into the formula for the Gaussian Distribution. The quantiles are calculated from the resulting values.

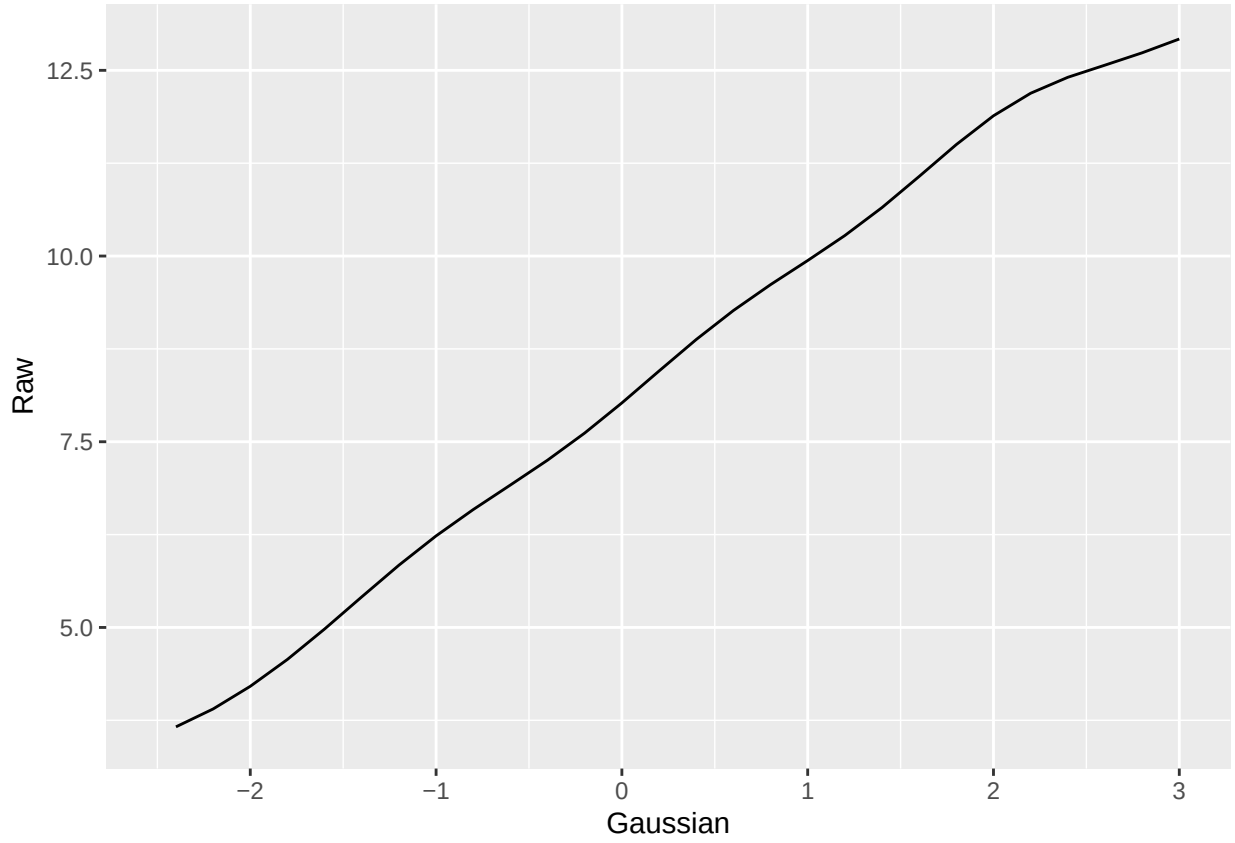
```
anam.db = AnamHermite(nbpoly=27)
err = anam.db$fitFromLocator(db_noout.db)
anam.db
```

```
##
## Hermitian Anamorphosis
## ----------------------
## Minimum absolute value for Y  = -2.5
## Maximum absolute value for Y  = 3
## Minimum absolute value for Z  = 3.56825
## Maximum absolute value for Z  = 12.9219
## Minimum practical value for Y = -2.5
## Maximum practical value for Y = 3
## Minimum practical value for Z = 3.56825
```

```
## Maximum practical value for Z = 12.9219
## Mean                           = 8.06092
## Variance                       = 3.48851
## Number of Hermite polynomials = 27
## Normalized coefficients for Hermite polynomials (punctual variable)
##                [,  0]    [,  1]    [,  2]    [,  3]    [,  4]    [,  5]    [,  6]
##      [  0,]     8.061    -1.865    -0.001     0.079     0.007     0.043     0.004
##      [  7,]    -0.018    -0.011    -0.009     0.012     0.021    -0.010    -0.023
##      [ 14,]     0.008     0.020    -0.006    -0.017     0.006     0.013    -0.007
##      [ 21,]    -0.011     0.009     0.009    -0.010    -0.007     0.011
```

Plot the Gaussian anamorphosis as a transform function. The rectilinear shape of the transform function indicates that the original variable was reasonably similar to a Gaussian variable: thus the transform is simply linear.

```
p = plot.init()
p = p + plot.anam(anam.db)
plot.end(p)
```



4.8 Calculating the Gaussian transform for processing the simulation maps

```
err = anam.db$rawToGaussian(db_noout.db, property)
dbfmt = DbStringFormat_createFromFlags(flag_resume = FALSE,
                                       flag_vars = FALSE,
                                       flag_extend = FALSE,
                                       flag_stats = TRUE,
                                       flag_array = FALSE,
```

```
                                        flag_locator = FALSE,
                                        names = "Y*")
db_noout.db$display(dbfmt)
```

```
##
## Data Base Characteristics
## =========================
##
## Data Base Statistics
## --------------------
## 17 - Name Y.P_PHI_pct - Locator z1
##  Nb of data          =        261
##  Nb of active values =        261
##  Minimum value       =     -2.500
##  Maximum value       =      2.977
##  Mean value          =      0.002
##  Standard Deviation  =      0.994
##  Variance            =      0.989
```

```
## NULL
```

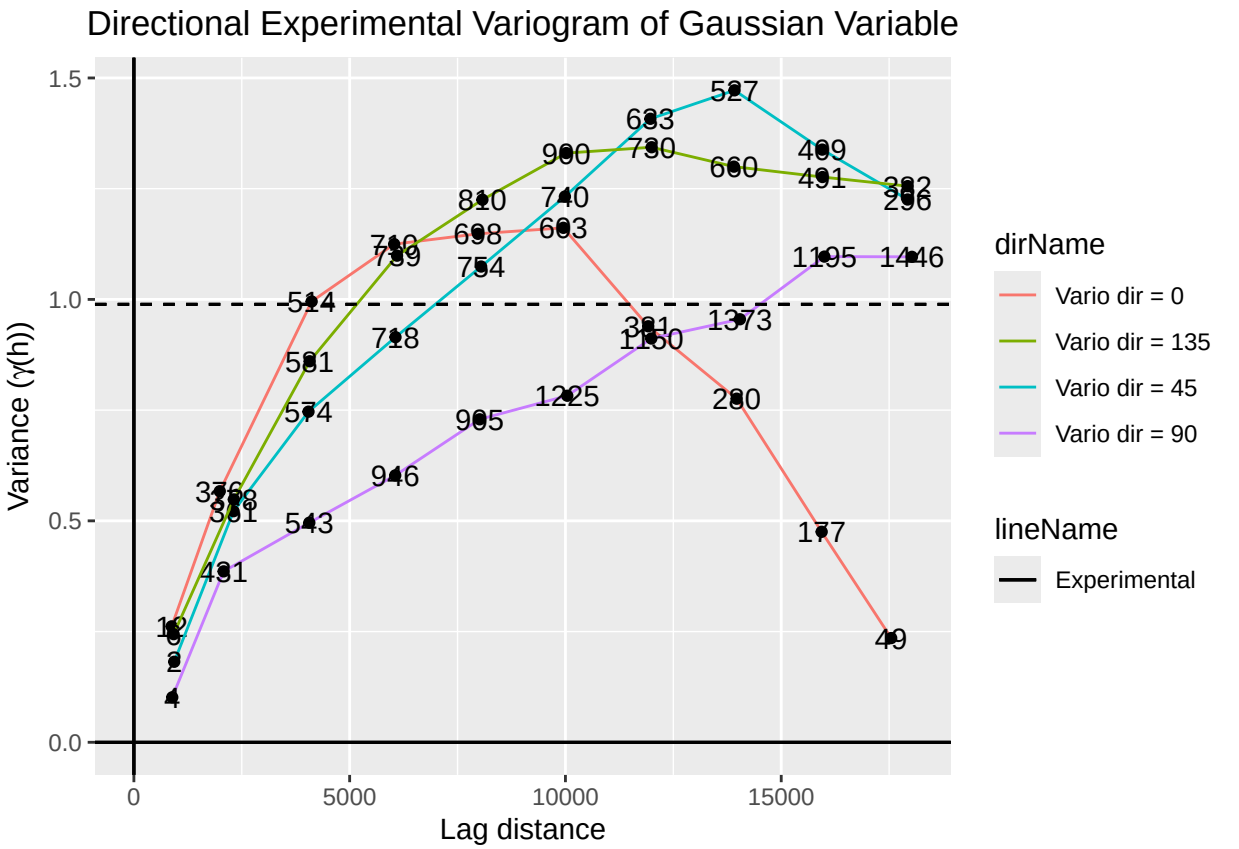
5 Selection of the Variogram Model types to be used

If you include many basic structures (variogram model types) into the auto-fitting algorithm of gstlearn, "fit" function will test each of the structures you specify and determine which structure or combination of structures are best used to model the experimental semivariogram. That said, keep in mind that selecting many structures may cause overfitting. The suggestion I propose is to limit the number of variogram model types to no more than 3 or 4.

Make your variogram model type selections in this code chunk. Be sure that if you specify the variogram model name that the name is spelled correctly and that it is in quotes.

```
varioparam <- VarioParam_createMultiple(ndir=4, nlag = 10, dlag=2000)
gaus.vario <- Vario_computeFromDb(varioparam, db_noout.db)
```

```
p = plot.init()
p = p + plot.vario(gaus.vario, drawPlabel = TRUE, flagLegend=TRUE)
p = p + plot.decoration(title = "Directional Experimental Variogram of Gaussian Variable",
                        xlab = "Lag distance",
                        ylab = expression(paste("Variance (", gamma, "(h))", sep="")))
plot.end(p)
```



Fitting the Model on the experimental variogram of the Gaussian variable

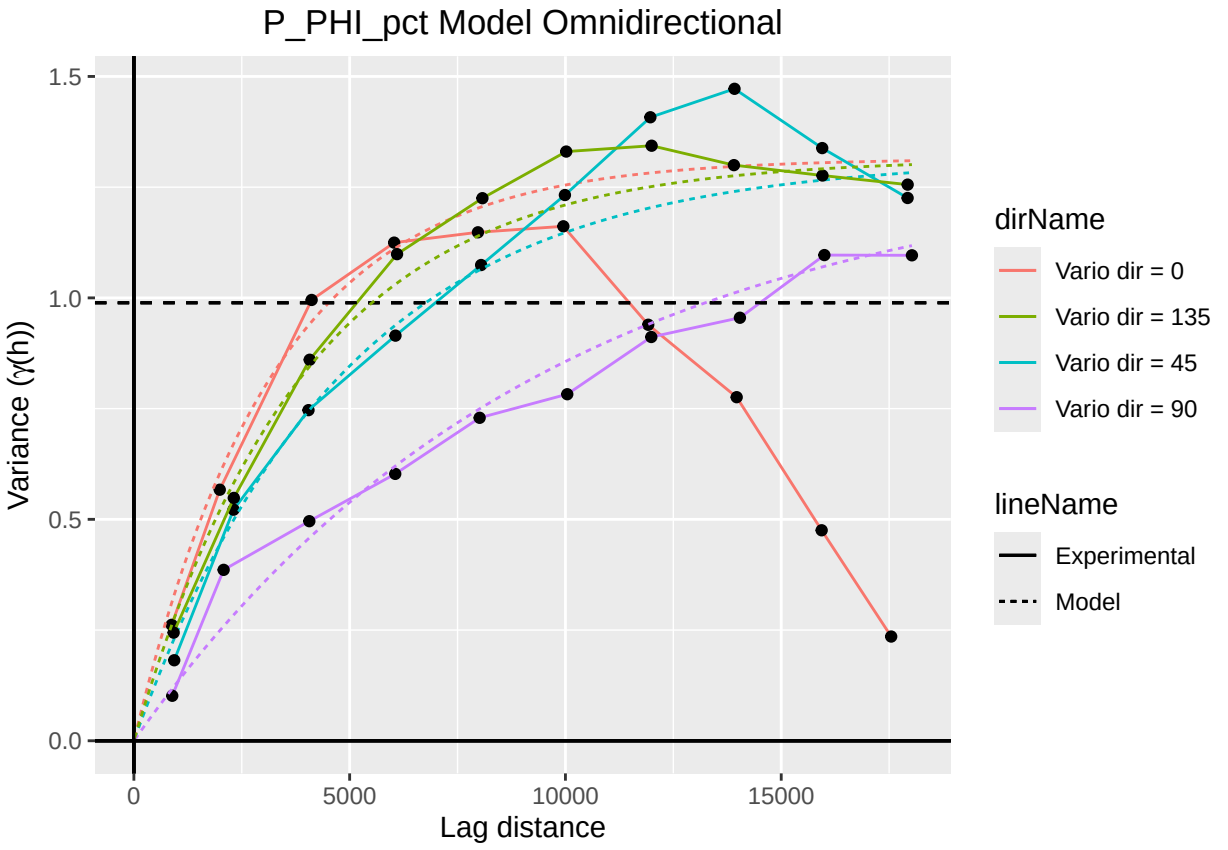
```
types = ECov_fromKeys(c("EXPONENTIAL"))

model.gaus.vario <- Model()
err = model.gaus.vario$fit(gaus.vario, types=types)
model.gaus.vario
```

```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 1
## Number of drift function(s)  = 0
## Number of drift equation(s)  = 0
##
## Covariance Part
## ---------------
## Exponential
## - Sill         =      1.315
## - Ranges       =   9629.270 30303.133
## - Theo. Ranges =   3214.330 10115.435
## - Angles       =    172.897     0.000
## - Rotation Matrix
##                [,  0]    [,  1]
##      [  0,]     0.992     0.124
```

```
##      [  1,]    -0.124     0.992
## Total Sill     =      1.315
## Known Mean(s)     0.000
```

Display the Model together with the experimental variogram

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



##CONDITIONAL SIMULATION

Having gone through all preparation steps, we can finally perform conditional simulation Inspired by 2D.html by D.Renard, we are using the Turning Bands algorithm with 1000 bands to perform a set of 11 simulations.

```
err = simtub(dbin = db_noout.db,
             dbout = dbgrid2,
             model = model.gaus.vario,
             neigh = neigh.unique,
             nbsimu = 11,
             nbtuba = 1000)
```

5.0.1 Backtransform

However, the simulations were produced in Gaussian space. Recall that the initial step was to transform the data using the Normal Score Transform (NST) into Gaussian space. Because of that, it's important to back transform the simulations into the original data space using function angaussianToRaw(). To help you

remember, the function name included Y2Z, so you are going from Y (transformed space) to Z (Original space).

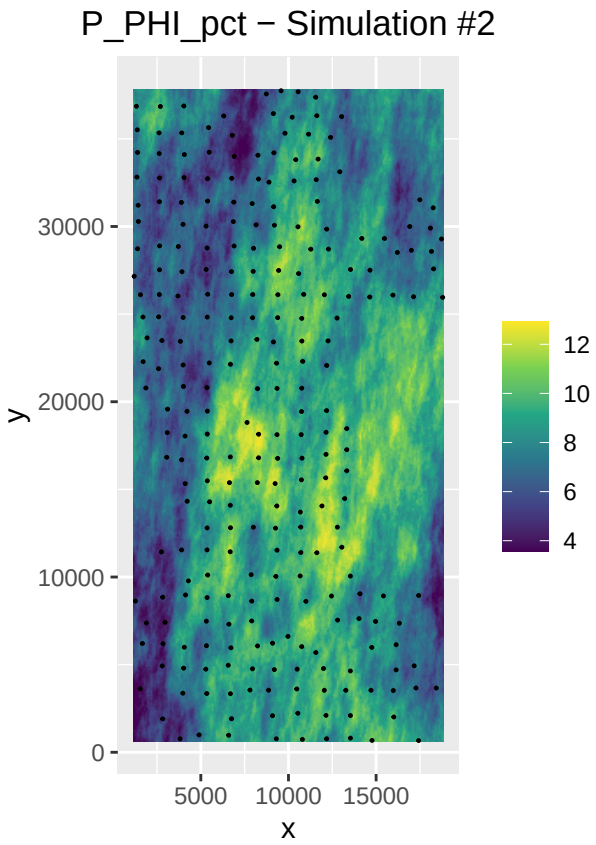
```
err = anam.db$gaussianToRaw(dbgrid2, name="Y.*")
```

5.1 Plotting the Conditional simulation results

Normally, we do not plot the conditional simulation results with contours. Check what happens when you un-comment the corresponding part of the code which plots the contours.

11 simulations were run. In the code chunk below, you can view the different simulations by changing the simulation number in the line beginning with, "name.image =".

```
p = plot.init(asp=1)
p = p + plot.raster(dbgrid2, name = "Z.*.2", flagLegend=TRUE, legendName="")
#p = p + plot.contour(dbgrid2, name = "Z.*.2", color = 'white')
p = p + plot.symbol(db_noout.db, nameSize = property, pch = 19, cex = 0.2)
p = p + plot.decoration(title = paste(property, "- Simulation #2"))
plot.end(p)
```



6 Calculating the mean of all realizations

Note what is inside SimGrid_mean_dir in the environment panel. It now has all 11 realizations and it has the mean of the realizations.

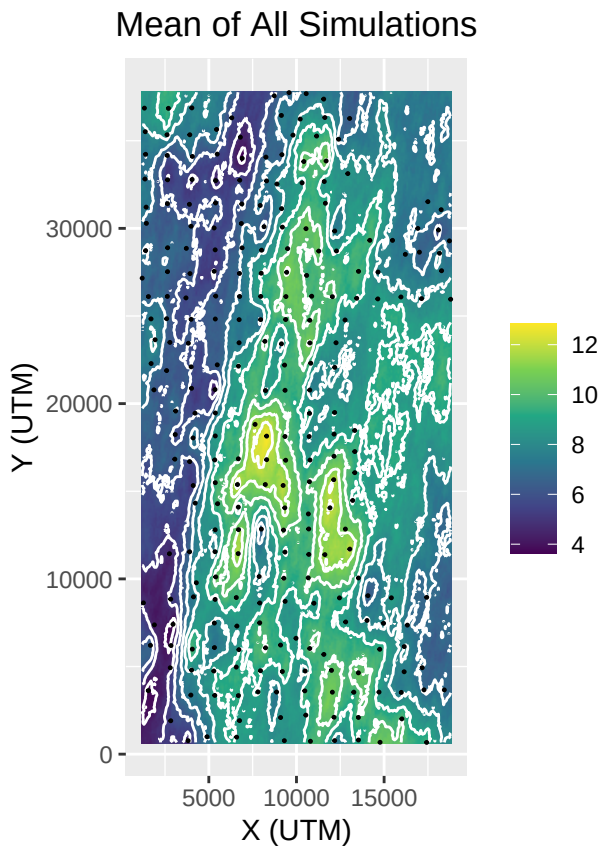
```
err = dbgrid2$statisticsBySample(dbgrid2$getNamesByLocator(ELoc_Z()),
                                 opers = EStatOption_fromKeys(c("MEAN", "STDV")))
```

7 Plotting the mean of realizations

7.1 Plot the Mean Map

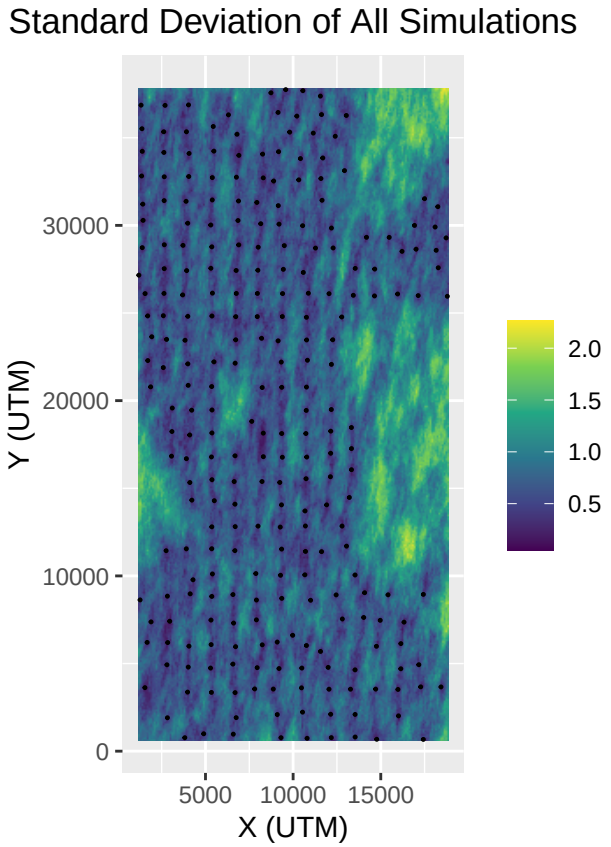
The following map represents the mean of 11 realizations. Theory states that if we ran an infinite number of realizations and calculated the mean, the result would be equivalent to the Kriged solution. 11 realizations are probably not enough to perfectly see this relationship, but you can get an idea. One thing to do would be to calculate the statistics of the mean map and check the mean, standard deviation, and variance. Another thing to do would be to plot the contours from the Kriged map created earlier on top of the mean map created from the 10 realizations.

```
p = plot.init(asp=1)
p = p + plot.raster(dbgrid2, name = "Stats.MEAN", flagLegend=TRUE, legendName="")
p = p + plot.contour(dbgrid2, name = "Stats.MEAN", color = 'white')
p = p + plot.symbol(db_noout.db, nameSize = property, pch = 19, cex = 0.2)
p = p + plot.decoration(title = "Mean of All Simulations", xlab = "X (UTM)",ylab = "Y (UTM)")
plot.end(p)
```



7.2 Plotting the standard deviation map

```
p = plot.init(asp=1)
p = p + plot.raster(dbgrid2, name = "Stats.STDV", flagLegend=TRUE, legendName="")
p = p + plot.symbol(db_noout.db, nameSize = property, pch = 19, cex = 0.2)
p = p + plot.decoration(title = "Standard Deviation of All Simulations", xlab = "X (UTM)",ylab = "Y (UTM
plot.end(p)
```



This concludes the Demo on Conditional Simulation and Post Processing.