Conditional Simulation With Directional Variograms

Jeffrey Yarus

27 May, 2025

Contents

1 Loading Packages

```
getwd()
```

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

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) Comment out the above chunk and use the code below when for the variable, P__KH__md or any right-skewed distribution. Of course, un-comment the code below if you intend using it.

```
# df <-
#   df %>%
#   mutate(
#     Log_property = log(!!sym(property)),
#     maxval_trans = max(Log_property),
#     # not necessary to calculate max
#     minval_trans = min(Log_property),
#     # not necessary to calculate min
#     iqr_val_trans = IQR(Log_property),
#     # calculates the IQR value
#     iqr_val_adj_trans = ((iqr_val_trans) * 1.5),
#     third_q_trans = quantile(Log_property, prob = 0.75, na.rm = TRUE),
#     first_q_trans = quantile(Log_property, prob = .25, na.rm = TRUE),
#     outlier =
#       (Log_property) > (third_q_trans + iqr_val_adj_trans) |
#       (Log_property) < (first_q_trans - iqr_val_adj_trans)
#   ) %>%
#   dplyr::select(-maxval_trans:-first_q_trans)
```

4.3 Creating A Database for use with RGeostats

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").

```
db <-
  # 262 wells with outlier
  df %>%
  dplyr::select(-F_Top) %>%
  db.create() %>%
  db.locate(c(xlon, ylat), "x") %>%
  #  db %>%
  db.locate(names = property, loctype = "z")
```

4.3.2 Creating databases with no outliers

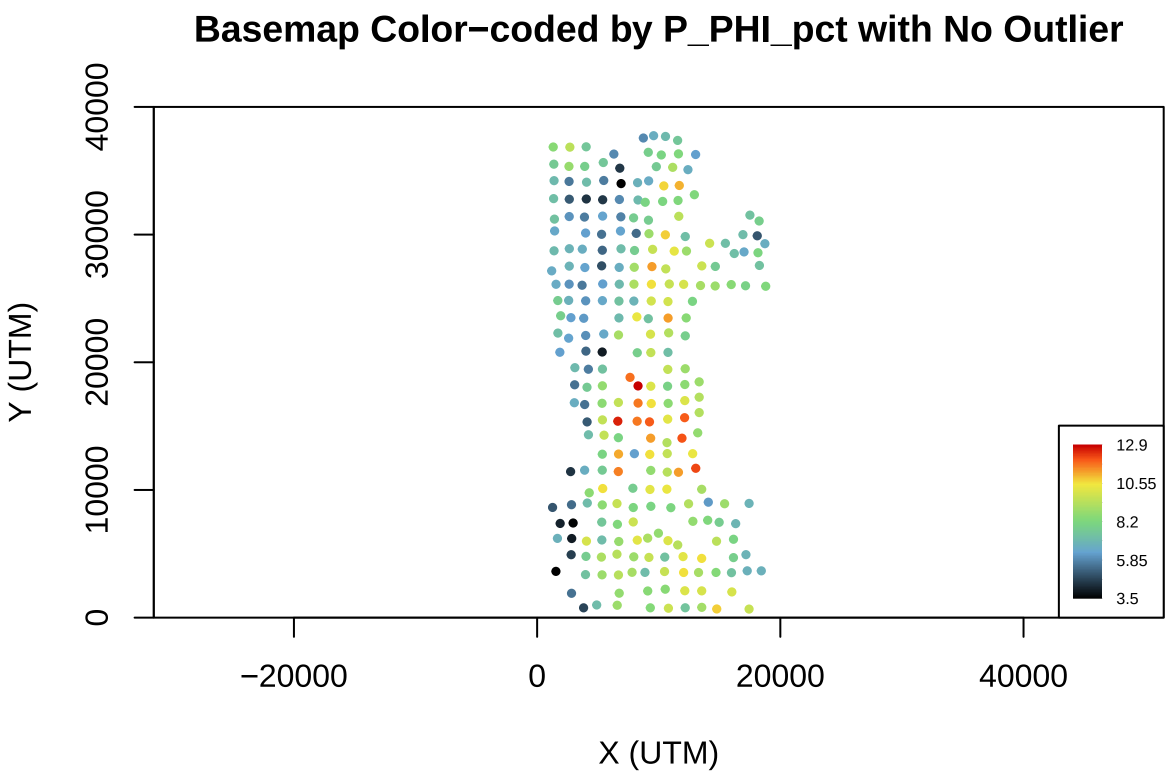
The following chunk creates a database **without outliers** from a data frame and assigns the location of the coordinates and the selected mapping variable ("property").

```
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_noout.db %>%
  db.locate(names = property, loctype = "z")
#db_noout.db@locators # The @ is addressor in S4 like the $ in S3
```

4.4 Plotting a basemap from a database

4.4.1 Data Observations with no outlers

```
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"))
```



4.5 Neighborhood Design

4.5.1 Unique neighborhood - Create a neiborhood that uses all the data

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)
```

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.

4.5.3 Moving Neighborhood

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

```
#neigh.moving <-
#neigh.create(
#ndim=2,
#nmaxi=10,
#adius=20,
#flag.sector = TRUE,nsect = 8,nsmax = 2)
```

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

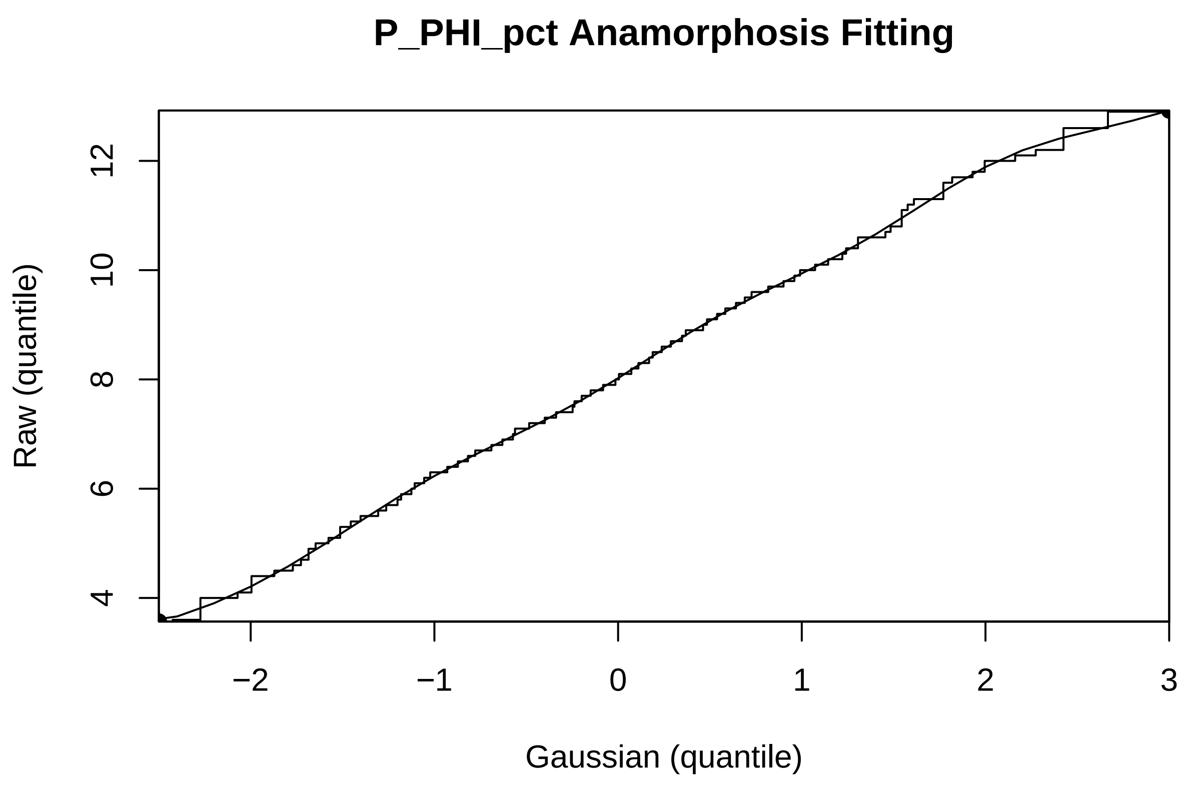
```
dbgrid2 <-
  db.grid.init(
    db_noout.db,
    dcell = c(100, 100),
  )
migrate(
  dbin = db_noout.db,
  dbout = dbgrid2,
  names = c("L*", "D*", "P*"),
  radix = ""
)
```

```
##
## Data Base Grid Characteristics
## ==============================
##
## Data Base Summary
## -----------------
## File is organized as a regular grid
## Space dimension              = 2
## Number of Columns            = 11
## Maximum Number of UIDs       = 11
## Total number of samples      = 65667
##
## Grid characteristics:
## ---------------------
## Origin :   1198.000   670.000
## Mesh   :    100.000   100.000
## Number :        177       371
##
## Variables
## ---------
## Column = 0 - Name = rank - Locator = NA
## Column = 1 - Name = x1 - Locator = x1
## Column = 2 - Name = x2 - Locator = x2
## Column = 3 - Name = L_3_FACIES - Locator = z1
## Column = 4 - Name = Log_property - Locator = z2
## Column = 5 - Name = P_Delta_t - Locator = z3
## Column = 6 - Name = P_KH_md - Locator = z4
## Column = 7 - Name = P_PHI - Locator = z5
## Column = 8 - Name = P_PHI_pct - Locator = z6
## Column = 9 - Name = P_Thickness - Locator = z7
## Column = 10 - Name = P_Top_ft - Locator = z8
```

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 qunatiles are calculated from the resulting values.

```r
anam.db <-
  anam.fit(
  db_noout.db,
  property,
  nbpoly = 27,
  title = paste(
    property,
    "Anamorphosis Fitting"),
  xlab = "Gaussian (quantile)",
  ylab = "Raw (quantile)"
)
```



4.8 Calculating the Backtransform for for final simulation maps

```r
db.anamor <-
  anam.z2y(
    db_noout.db,
    property,
    anam = anam.db)

print(
  db.anamor,
  flag.stats = TRUE,
  names = paste(
    "Gaussian.",
    property,
    sep = ""))
```

```
##
## Data Base Characteristics
## =========================
##
## Data Base Summary
## -----------------
## File is organized as a set of isolated points
## Space dimension              = 2
## Number of Columns            = 18
## Maximum Number of UIDs       = 18
## Total number of samples      = 261
##
## Data Base Statistics
## --------------------
## 18 - Name Gaussian.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
##
## Variables
## ---------
## Column = 0 - Name = rank - Locator = NA
## Column = 1 - Name = N_Well - Locator = NA
## Column = 2 - Name = C_X_ft - Locator = x1
## Column = 3 - Name = C_Y_ft - Locator = x2
## Column = 4 - Name = L_3_FACIES - Locator = NA
## Column = 5 - Name = P_Delta_t - Locator = p1
## Column = 6 - Name = P_KH_md - Locator = p2
## Column = 7 - Name = P_PHI - Locator = p3
## Column = 8 - Name = P_PHI_pct - Locator = NA
## Column = 9 - Name = P_Thickness - Locator = p4
## Column = 10 - Name = P_Top_ft - Locator = p5
## Column = 11 - Name = iqr_val - Locator = NA
## Column = 12 - Name = iqr_val_adj - Locator = NA
## Column = 13 - Name = third_q - Locator = NA
## Column = 14 - Name = first_q - Locator = f1
## Column = 15 - Name = outlier - Locator = NA
## Column = 16 - Name = Log_property - Locator = NA
## Column = 17 - Name = Gaussian.P_PHI_pct - Locator = z1
```

5 Selection of the Variogram Model types to be used

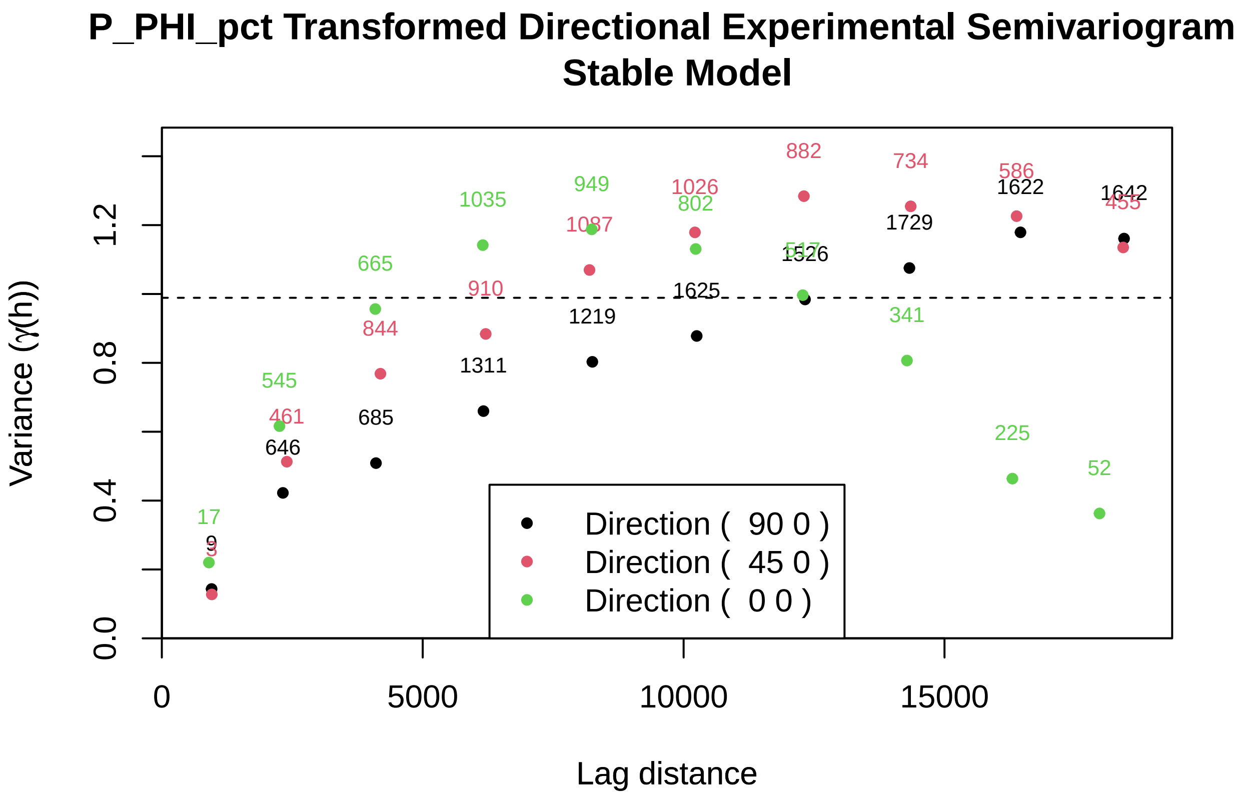
If you include many basic structures (variogram model types) into the auto-fitting algorithm of RGeostats, model.auto() will test each of the structures you specify and determine which structure or combination of structures are best used to model the experimental semivariogram. That said, keep in mind 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.

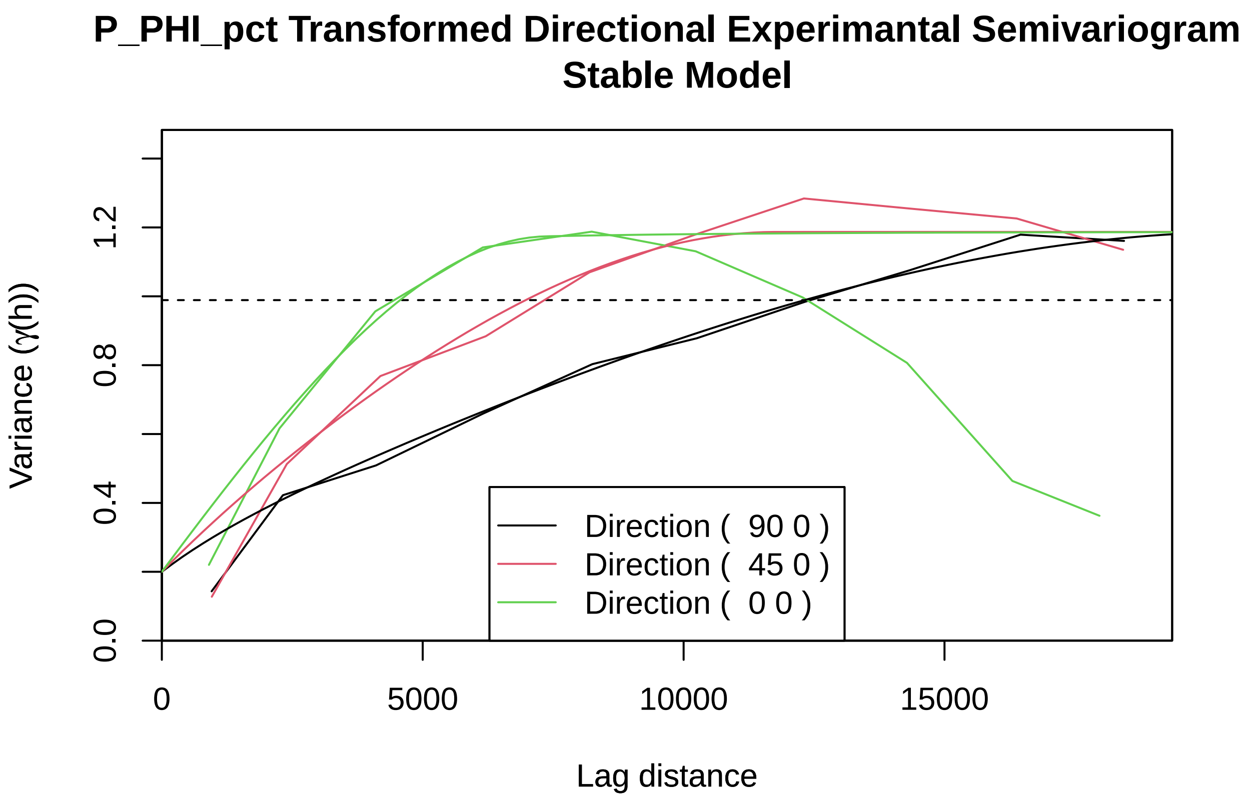
```
struct <- c(1, 2, 3)
```

```
gaus.vario = vario.calc(db.anamor, nlag = 10, dir = c(90, 45, 0))
#gaus.vario = vario.calc(db.anamor, nlag = 10)
```

```
plot(
  gaus.vario,
  type = "p",
  pch = 20,
  npairpt = TRUE,
  npairdw = FALSE,
  pos.legend = 2,
  cex = 0.7,
  title = paste(property, "Transformed Directional Experimental Semivariogram
  Stable Model"),
  xlab = "Lag distance",
  ylab = expression(paste("Variance (", gamma, "(h))",
                        sep = ""))
)
```



```
model.gaus.vario <- model.auto(
  gaus.vario,
  flag.noreduce = TRUE,      # following code for adjusting nugget
  equal = c("M1V1=0.2"),
  upper = c("M1V1=0.5"),
  struct = struct,
  draw = TRUE,
#  title = paste("Experimental Gaussian Variogram
#  with", struct, "fitting"),
  title = paste(property, "Transformed Directional Experimantal Semivariogram
  Stable Model"),
  pos.legend = 2,
  xlab = "Lag distance",
  ylab = expression(paste("Variance (", gamma,
                        "(h))",
                        sep = ""))
)
```



```r
rm(model,pos = 1)
print(model.gaus.vario)
```

```
##
## Model characteristics
## =====================
## Space dimension              = 2
## Number of variable(s)        = 1
## Number of basic structure(s) = 3
## Number of drift function(s)  = 1
## Number of drift equation(s)  = 1
##
## Covariance Part
## ---------------
## Nugget Effect
## - Sill         =      0.200
## Exponential
## - Sill         =      0.074
## - Ranges       =   4414.631 12570.085
## - Theo. Ranges =   1473.640  4195.998
## - Angles       =    -91.619     0.000
## - Rotation Matrix
##                [,  0]    [,  1]
##      [  0,]    -0.028     1.000
##      [  1,]    -1.000    -0.028
## Spherical
## - Sill         =      0.913
## - Ranges       =  24141.597  7256.269
## - Angles       =   -100.594     0.000
## - Rotation Matrix
##                [,  0]    [,  1]
##      [  0,]    -0.184     0.983
##      [  1,]    -0.983    -0.184
## Total Sill     =      1.187
##
## Drift Part
## ----------
## Universality Condition
```

##CONDITIONAL SIMULATION

Having gone through all preparation steps, we can finally perform conditional simulation Inspired by 2D.html by D.Renard, We're using the Turning Bands algorithm with 3000 bands to perform a set of 11 simulations.

```
db_sim <-
  simtub(
    db.anamor,
    dbgrid2,
    model.gaus.vario,
    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 anam.y2z(). To help you remember, the function name included Y2Z, so you are going from Y (transformed space) to Z (Original space).

```
db_sim2 <-
  anam.y2z(db_sim,
           names = paste("Simu.Gaussian.", property, "*",
           sep = ""),
           anam = anam.db)
```

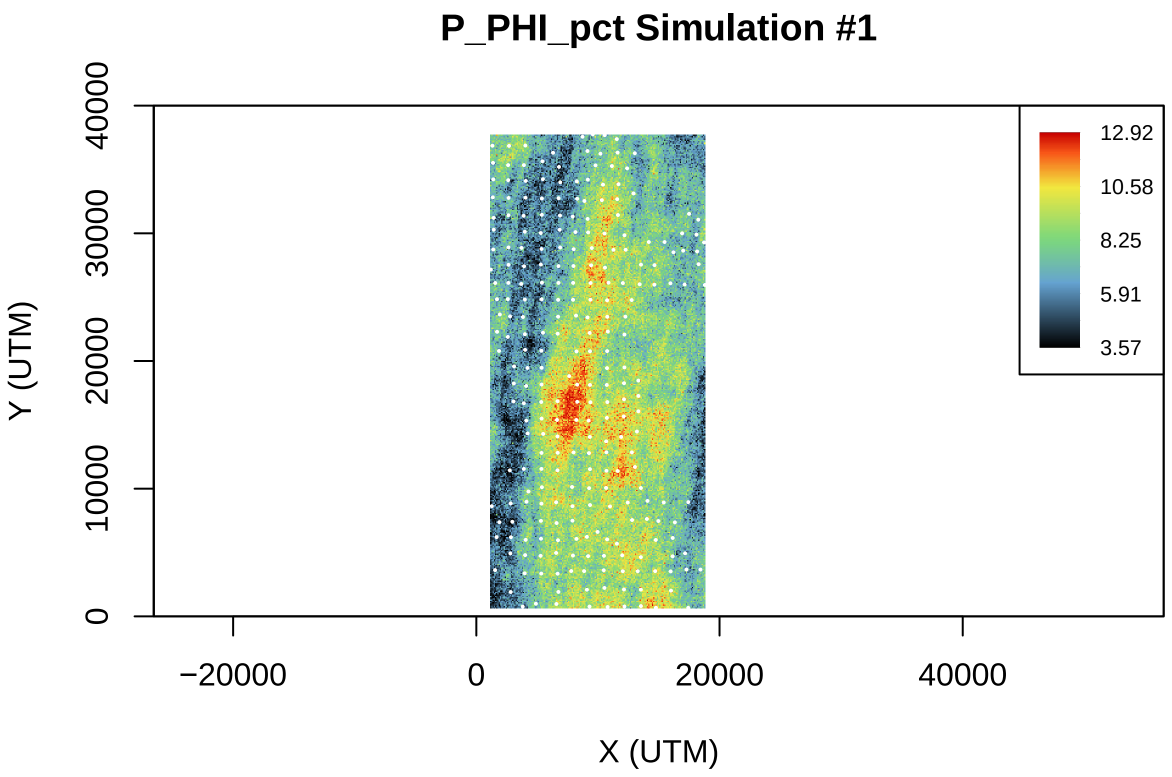
5.1 Plotting the Conditional simulation results

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

11 simulations were run; labeled S1 - S11. In the code chunk below, you can view the different simulations by changing the simulation number (".Sx") in the line beginning with, "name.image =".

```
plot(
  db_sim2,
  name.image = paste("Raw.Simu.Gaussian.", property, ".S1", sep = ""),
  title = paste(property, "Simulation #1"),
  pos.legend = 7,
  cex = .7,
  asp = 1.05,
  xlim = c(0, 30000),
  ylim = c(0, 40000),
  xlab = "X (UTM)",
  ylab = "Y (UTM)"
)

plot(
  db_noout.db,
  name.post = property,
  pch = 20,
  col = 'white',
  cex = 0.2,
  add = TRUE
)
```



```
# plot(
#   db_sim2,
#   name.contour = "Raw.Simu.Gaussian.P_PHI_pct.S1",
#   nlevels = 7,
#   col = ("black") ,
#   add = TRUE
# )
```

6 Calculting 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.

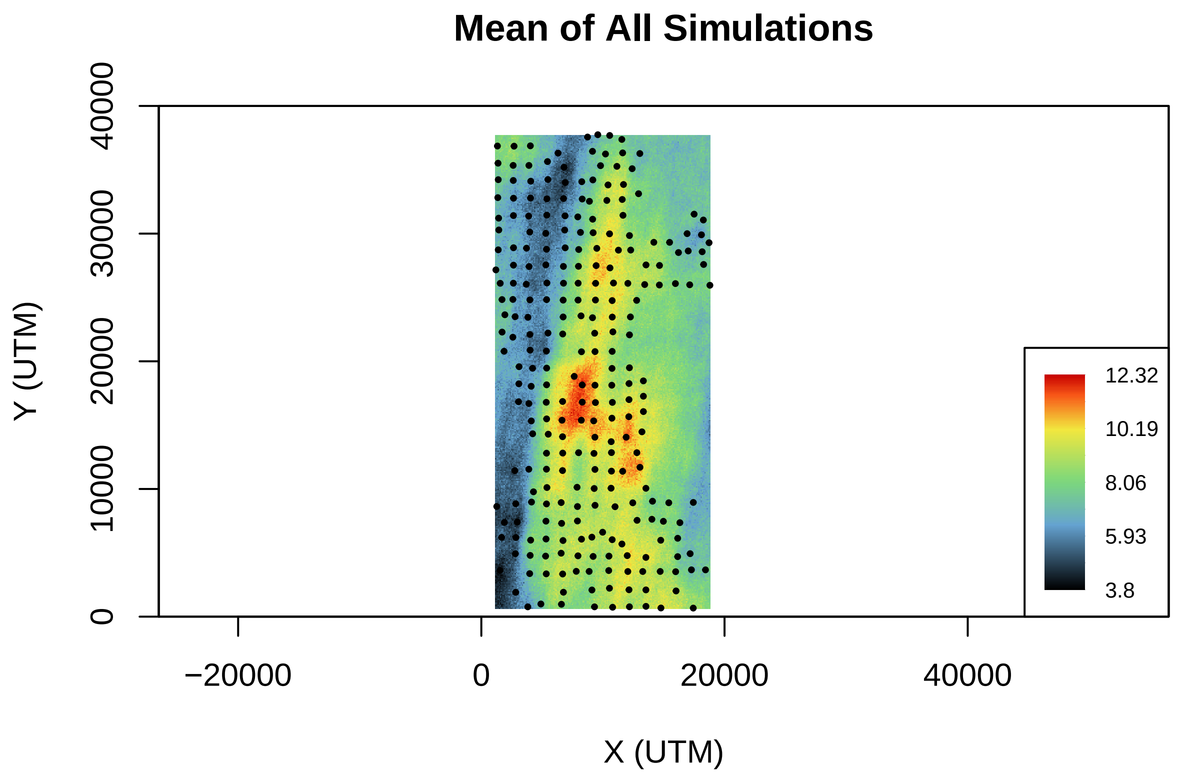
```
SimGrid_mean_dir <-
  db.stat.simu(db_sim2, names = "Raw.Simu.Gaussian.*", fun = "mean")
```

7 Plotting the mean of 10 realizations

7.1 Plot the Directinal 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. The last part of the code chunk which is currently commented out, will plot the earlier Kriged contours on top of the current mean map from 11 realizations. Uncomment the last part of the code and run the chunk to see what happens.

```r
plot(
  SimGrid_mean_dir,
  title = "Mean of All Simulations",
  pos.legend = 1,
  cex = .7,
  asp = 1.05,
  xlim = c(0, 30000),
  ylim = c(0, 40000),
  xlab = "X (UTM)",
  ylab = "Y (UTM)"
)
plot(
  db_noout.db,
  name.color = property,
  pch = 20,
  cex = 0.5,
  add = TRUE,
  col = 'black'
)
```

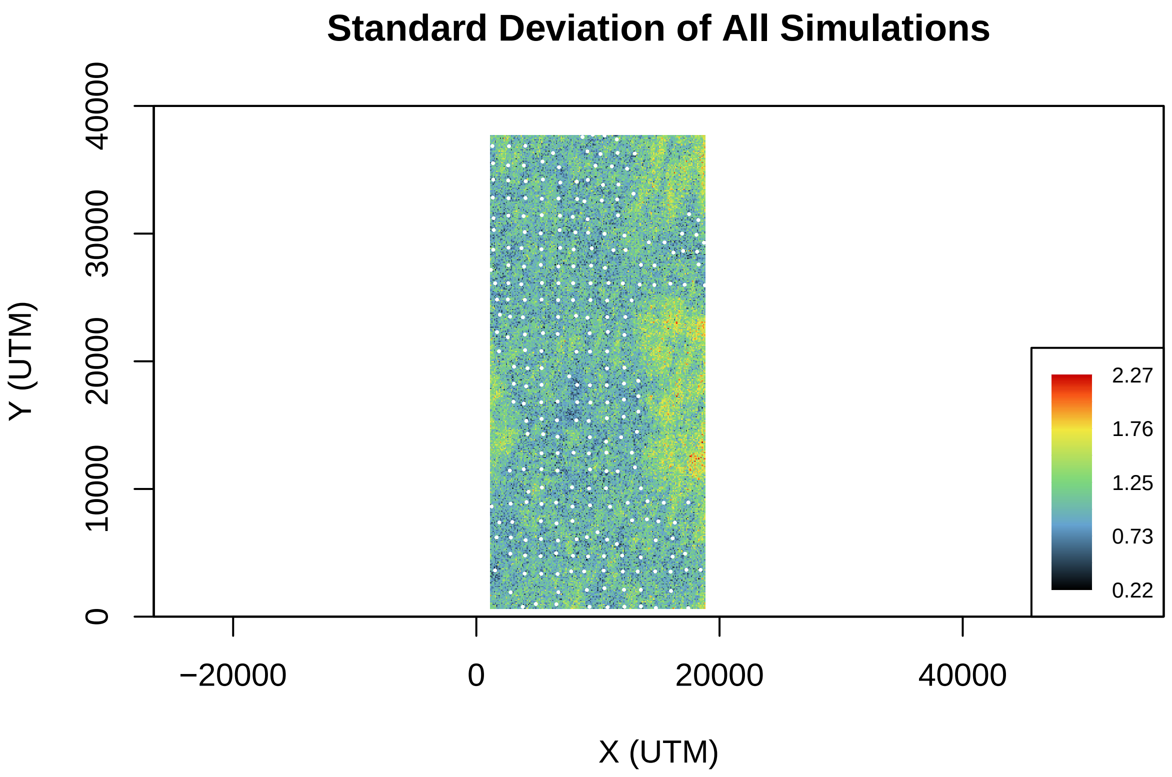


# Calculating the standard deviation of the 11 realizations

```
SimGrid_std_dir <-
  db.stat.simu(db_sim2, names = "Raw.Simu.Gaussian.*", fun = "stdv")
```

8 Plotting the standard deviation map

```
plot(
  SimGrid_std_dir,
  title = "Standard Deviation of All Simulations",
  pos.legend = 1,
  cex = .7,
  asp = 1.05,
  xlim = c(0, 30000),
  ylim = c(0, 40000),
  xlab = "X (UTM)",
  ylab = "Y (UTM)"
)
plot(
  db_noout.db,
  name.color =  property,
  pch = 20,
  col = 'white',
  cex = 0.2,
  add = TRUE
)
```



This concludes the Demo on Conditional Simulation and Post Processing