PluriGaussian Simulations

gstlearn Team

January 2023

Preamble

```
rm(list=ls())
library(gstlearn)
library(ggplot2)
library(ggpubr)
```

Defining some essential parameters:

```
ndim = 2
defineDefaultSpace(ESpaceType_RN(), ndim);
```

```
## NULL
```

```
nbsimu = 20
nbcc   = 4
cmap   = c('red', 'blue', 'yellow')
```

Downloading the data base (from the distribution **Exdemo_PGS.db**) and creating the output Grid, the Model (Cubic) and the Neighborhood (Unique):

```
fileNF = file.path(Sys.getenv('GSTLEARN_DATA'),"PluriGaussian","Data.NF")
data = Db_createFromNF(fileNF)

grid = DbGrid_create(nx=c(110,110))

model = Model_createFromParam(type=ECov_CUBIC(), ranges=c(50,30))

neigh = NeighUnique()
```

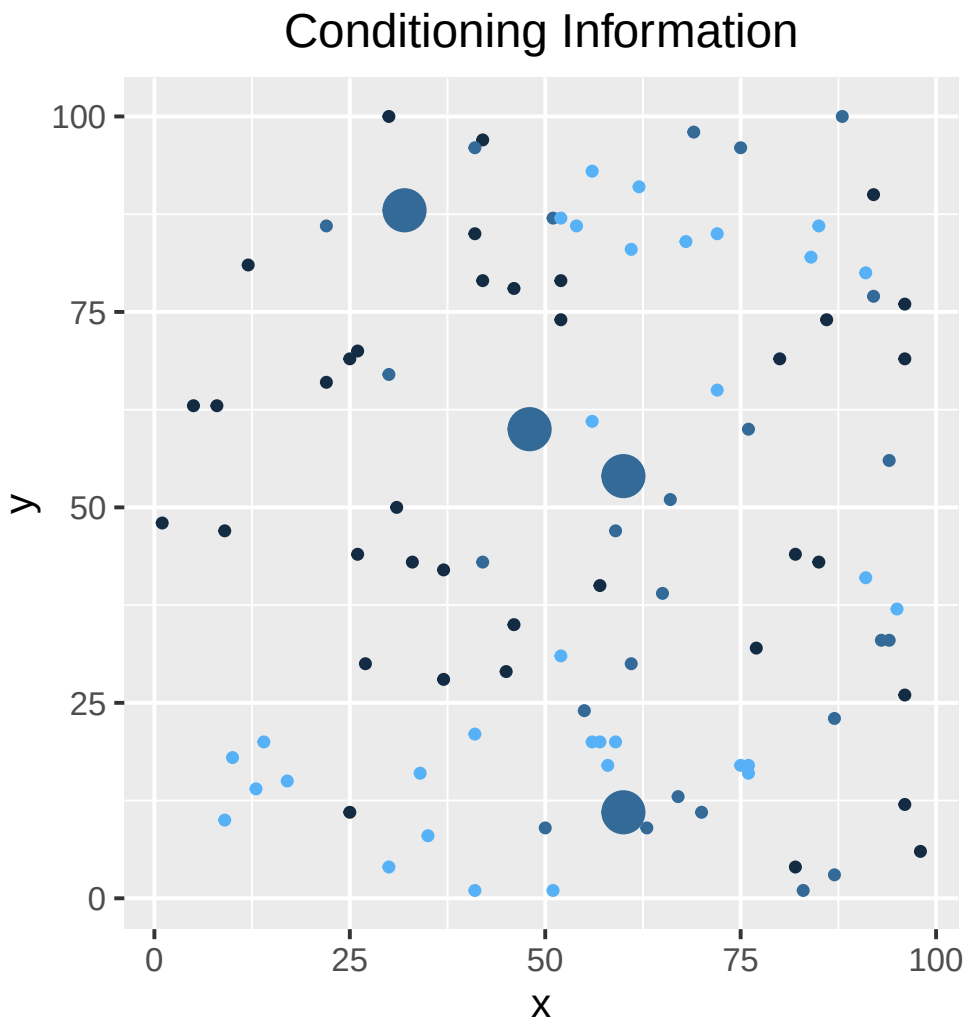
Defining Internal Display Function

Defining the internal function **plot.dat** used to visualize the data set with convenient color representation:

- first facies in *red*
- second facies in *blue*
- third facies in *yellow*
- samples which must belong to the same connected component in *black*

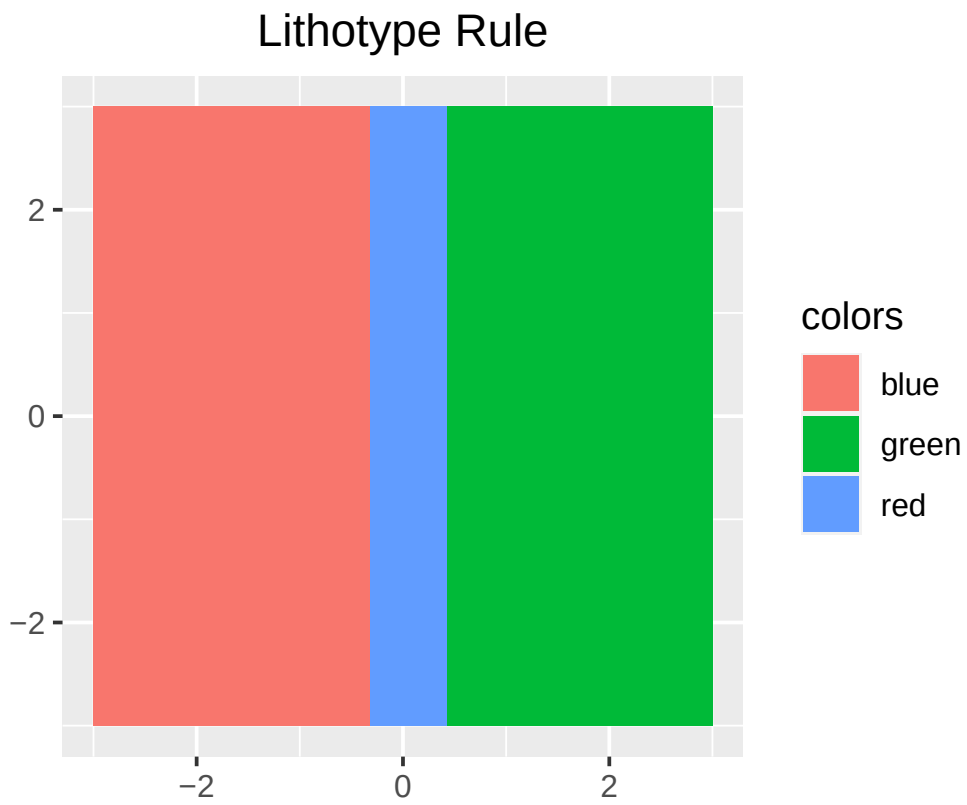
Display Data

```
# TODO: Utiliser CMAP
plot.setDefault(1, dims=c(6,6))
p = plot(data, name_color="facies", name_size="connect")
#         edgecolors='black', sizmin=40, cmap=cmap)
plot.decoration(p, title="Conditioning Information")
```



Proportions and Lithotype Rule

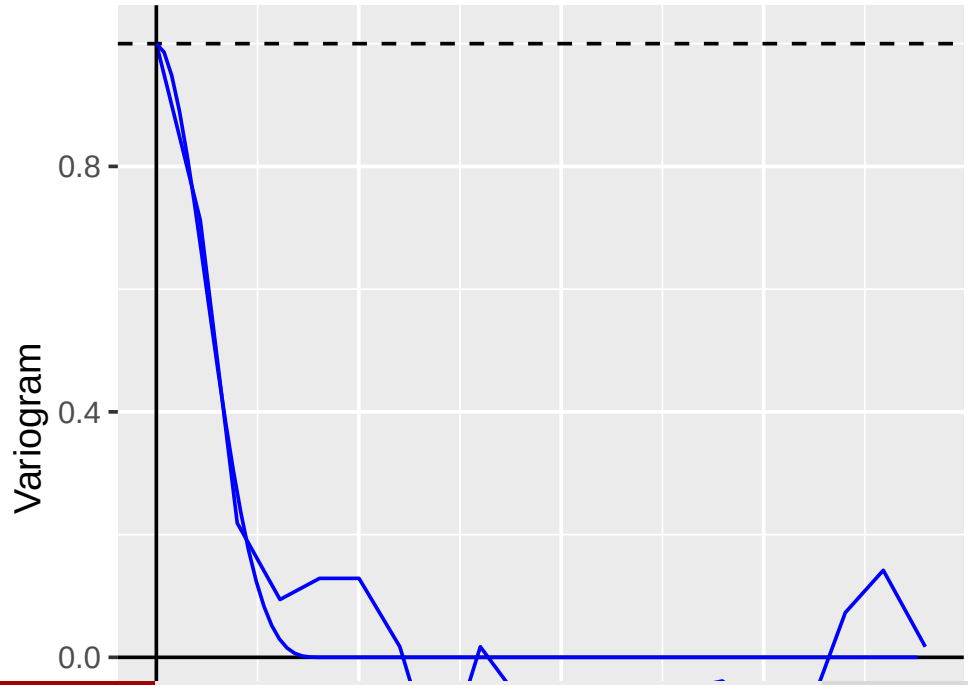
```
props = dbStatisticsFacies(data)
rule = Rule_createFromNames(c("S","S","F1","F2","F3"))

# TODO: utiliser cmap
p = plot(rule, proportions = props, maxG=3)
plot.decoration(p, title="Lithotype Rule")
```



Model of Underlying GRF

Calculate the Experimental Variogram of the Underlying Gaussian Random Function and fit the Model (used in PGS).

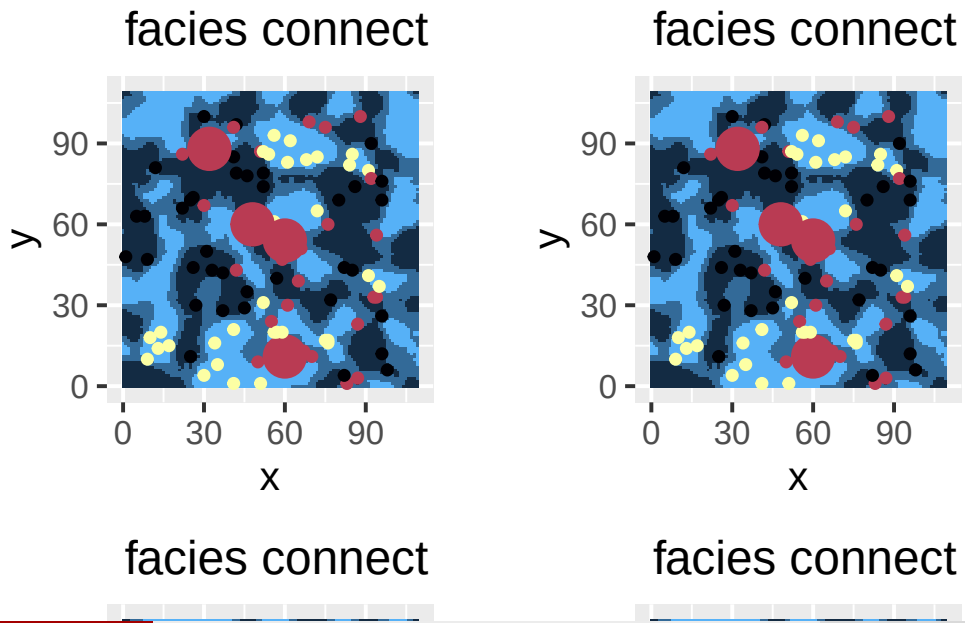
```
varioparam = VarioParam_createOmniDirection(dpas=5, npas=20)
ruleprop = RuleProp_createFromRule(rule, props)
vario = variogram_pgs(data, varioparam, ruleprop)

model_gaus = Model()
err = model_gaus$fit(vario, types=ECov_fromKeys(c("CUBIC")),
                     constraints=Constraints(1.))

plot.varmod(vario, model_gaus, asCov=TRUE)
```



PluriGaussian Simulation

```
err = simpgs(dbin=data, dbout=grid, ruleprop=ruleprop, model1=model_gaus,
             neighparam=neigh, nbsimu=nbsimu,
             namconv = NamingConvention("SimuPGS"))
```

```
p1 = plot(grid, "SimuPGS.1")
p1 = plot(data, name_color="facies", name_size="connect", padd=p1)
p2 = plot(grid, "SimuPGS.2")
p2 = plot(data, name_color="facies", name_size="connect", padd=p2)
p3 = plot(grid, "SimuPGS.3")
p3 = plot(data, name_color="facies", name_size="connect", padd=p3)
p4 = plot(grid, "SimuPGS.4")
p4 = plot(data, name_color="facies", name_size="connect", padd=p4)
ggarrange(p1,p2,p3,p4, nrow=2, ncol=2)
```



Acceptation Function

Acceptation internal function: Select a **Target Facies** and build its Connected Components. For each simulation outcome, check the ranks of the connected component(s) at constraining wells and **accept** the simulation if all ranks are similar.

```
accept <- function(data, grid, name, verbose=FALSE, transBinary=TRUE, faccc=2)
{
  # Get the indices of samples which should be connected (starting from 0)
  rankData = which(data["connect"] == 1) - 1
  rankGrid = grid$locateDataInGrid(grid, data, rankData)
    if (verbose)
    {
      cat("Number of conditioning data (connected) =",length(rankData),"\n")
    cat("Their ranks in the input Data Base =",rankData,"\n")
    cat("Their ranks in the output Data Base =",rankGrid,"\n")
    }

  # Perform the labelling into connected components
  err = grid$setLocator(name, ELoc_Z(), cleanSameLocator=TRUE)
  err = dbMorpho(grid, EMorpho_CC(), vmin=faccc-0.5, vmax=faccc+0.5)
  cc_list = grid[rankGrid,"Morpho*"]
  if (verbose)
    cat("List of their connected components indices =",cc_list,"\n")

  # Check that the data points belong to the same connected component
  number = length(unique(cc_list))
  retval = (number == 1)
```

Experiment the Acceptation Function on one Simulation outcome

```
isValid = accept(data, grid, "SimuPGS.1", TRUE)
```

```
## Number of conditioning data (connected) = 4
## Their ranks in the input Data Base = 4 12 15 16
## Their ranks in the output Data Base = 6000 1270 6648 9712
## List of their connected components indices = 1 1 1 1
## Acceptation score = TRUE
```

```
cat("Connectivity for Simulation #1 :",isValid,"\n")
```

```
## Connectivity for Simulation #1 : TRUE
```

Probability Map

- For each simulation, convert a pixel into 1 if it matches the Target Facies and 0 otherwise
- Calculate the mean per pixel over all simulation outcomes

This operation provides the probability that each pixel belongs to the Target Facies, calculated over all simulations that fulfill the Connectivity Constraint.

```
nb.valid = 0
for (i in 1:nbsimu)
{
  name = paste("SimuPGS.",i, sep="")
  cat("simu ",i," name=",name,"\n")
  isValid = accept(data, grid, name)
  if (isValid)
  {
    cat("Simulation ",name,"is valid\n")
    nb.valid = nb.valid + 1
  }
  else
    grid$deleteColumn(name)
}
```

```
## simu  1  name= SimuPGS.1
## Simulation  SimuPGS.1 is valid
## simu  2  name= SimuPGS.2
## simu  3  name= SimuPGS.3
## simu  4  name= SimuPGS.4
## simu  5  name= SimuPGS.5
```