Chapter 1

Introduction

Geometallurgy means optimising mineral processing adaptively to information on the microfabric, i.e. mineralogy, microstructure, chemistry, texture, etc. of the processed raw material (ore or secondary resources). The aim of this package is to add necessary statistical technologies to R.

The whole statistical analysis includes:

- Analysing microstructural information on raw and preprocessed material. This information typically comes e.g. from Mineral liberation analyser (MLA), or other imaging technologies. Stereological techniques are an important problem here.
- Analysing chemical compositional information. This information typically comes from laboratory essays.
- Mathematical modelling and statistical analysis of processing technologies with respect to their dependence and effect on geometallurgical parameters (i.e. microfabric and particle fabric). This data typically comes from planned processing experiments and consist of multiple datasets of the beforementioned types, typically structured in a complex way along a varying processing chaing.
- Geostatistical Analysis of geometallurgical parameters including prediction and simulation techniques for non-euclidean parameters. Corresponding datasets are multiple datasets of the first two kinds each marked with a spatial location and meta information like training images.
- Methods for the valuation of possible process chains based on given models of the processes and information on the feed material. These methods need various kinds typically inferred or modelled data.
- Stochastic optimisation of processes based on process models and geometallallurgical parameters.This approach integrates all of the informations inferred in the previous steps into a single stochastic optimisation algorithm, which typically needs some user intervention.

- Planning of geostatistical data collection. Typically we expect preexisting geostatistical geometallurgical datasets (one or more spatially distributed sets of datasets) to estimate the spatial continuity and predefined models for the understanding the effect of the information. This step works in a planning phase based on expected possible achievements by the stochastic optimisation algorithm.

The package has to be see as a prototype implementation and a quick and dirty way of providing test of ideas, and is now only in its starting phase.

1.1 Definition of Names

- **Mineral** A mineral is a type of crystals, defined by its apparent properties and physicochemical structure. The chemical formula is know, but individual elements can be substituted incompletely at different lattice positions. The MLA can typically identify the mineral, but the degree of substitution can only be quantified by other techniques, like e.g. the microprobe or ion beam analysis.

- **Particle**
  A particle is finite rigid body consisting of one or more grains of potentially different minerals.

-

- **(mineral) Grain**
  A (mineral) grain is a continues portion of a body consisting of the same mineral. It can consist of several (crystal) grains. This definition is add odds with the definition in texture analysis, where a grain is defined as a crystal grain.

- **(crystal) Grain**
  A (crystal) grain is a continues portion of a body consiting of the same mineral with a continues (non-ideal) lattice structure. Crystal Grains are separated by large angle grain boundaries. Crystal grains consist of defect zones and subgrains.

- **Subgrain**
  A subgrain is a part of a crystal grain with essentially constant crystallographic orientation.

- **Microstructure**
  The microstructure of a material is its geometric and mineralogical structure on the level of mineral grains.

- **Texture**
  The te

- **Microfabric**

- **Grade**
- **Composition**

1.2 The Database

Geometallurgy uses very large amounts of informations, each single measurement point, e.g. an MLA measurement containing several megabytes of data. The package does not work in main memory, but on databases. Thus, we first need to understand the database structure.

- **gmObject**
  Objects are the core entities in the database. They represent containers, which can be further described by additional information like a name or actual values.

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmObject` (
`gmID` BIGINT(19) UNSIGNED NOT NULL AUTO_INCREMENT,
`gmCreateTime` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`gmID`)
);
```

- **gmVarType**
  The variable type determines what kind of data the objects in the database represents. Furthermore the table containing the actual value is referenced.

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmVarType` (
`gmVarTypeID` VARCHAR(10) NOT NULL,
`gmVarTypeDescription` VARCHAR(255) NULL DEFAULT NULL,
`gmTable` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`gmVarTypeID`));

INSERT IGNORE INTO gmVarType
(gmVarTypeID,gmVarTypeDescription,gmTable)
VALUES
("set","a set of objects","gmRef"),
("real","a real number","gmNumeric"),
("int","an integer","gmInteger"),
("string","a string value","gmString"),
("text","a text value","gmText"),
("blob","a longblob","gmBlob");
```

- **gmVar**
  This table provides the variables, which can be used in the database. It also defines their variable type. Every variable will be referred by its gmVarID.

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmVar` (
`gmVarID` BIGINT(19) UNSIGNED NOT NULL AUTO_INCREMENT,
`gmVarName` VARCHAR(128) NOT NULL,
`gmVarTypeID` VARCHAR(10) NOT NULL,
`gmVarDescription` VARCHAR(255) NULL DEFAULT NULL,
PRIMARY KEY (`gmVarID`),
INDEX `fk_gmVar_gmVarType_idx` (`gmVarTypeID` ASC),
CONSTRAINT `fk_gmVar_gmVarType`
FOREIGN KEY (`gmVarTypeID`)
REFERENCES `gmDatabase`.`gmVarType` (`gmVarTypeID`)
ON DELETE CASCADE
ON UPDATE CASCADE);

INSERT IGNORE INTO gmVar
(gmVarID,gmVarName,gmVarTypeID,gmVarDescription)
VALUES
(1,"gmObject","set","a general object"),
(2,"gmUser","set","a person (as list in root) with access to db"),
(3,"gmName","string","the name of the object"),
(4,"gmUserName","string","the login name of the user"),
(5,"gmFullName","string","the full name of the user"),
(6,"gmPassword","string","the full name of the user"),
(7,"gmCreator","set","the person who created the object"),
(8,"gmAnnotation","set","annotation to an object"),
(9,"gmDescription","text","a description"),
(10,"gmDataUnit","set","a unit of data"),
(11,"project", "set", "a whole project"),
(12,"series", "set", "a set of several measurements for a given project"),
(13,"gmCreationTime","string","the creation time in 2013-09-18 10:34:33 format"),
(14,"gmTitle","string","the title of the object"),
(15,"gmRoot","set","the root object"),
(16,"phase","set","a mineral phase"),
(17,"phaseName","string","the unique name of the phase"),
(18,"formulaString","string","a string representation of the formula"),
(19,"element","set","a chemical element"),
(20,"symbol","string","the symbol of the element"),
(21,"atomicNumber","real","the average atomic weight of the element"),
(22,"density","set","the average density"),
(23,"unit","set","a unit"),
(24,"unitName","string","the name of the unit"),
(25,"conversion","set","a conversion rule for units"),
(26,"targetunit","set","the second unit of the conversion"),
(27,"conversionFactor","real","the factor to multiply with to get the target unit"
(28,"gmValue","set","a value with an error and a unit"),
(29,"sigma","real","the standard error"),
(30,"value","real","the real number value of something"),
```

```
(31,"MLAimage", "set", "a set of several frames representing a whole MLA image for a measure
(32,"frame", "set", "the container of the voxel image "),
(33,"particle", "set", "a coherent mineral object consisting of one or more mineral phases")
(34,"grain", "set", "a part of a particle consisting of one mineral phase"),
(35, "voxelImage", "text", "the MLA image represented by a voxel image, given as matrix"),
(36, "frameID", "int", "the ID of the frame in the series"),
(37, "grainID", "int", "the ID representing the grain in th voxel image"),
(38, "MLAresolution", "set", "the resolution of the MLA image"),
(39, "dx", "int", "the x dimension of a frame"),
(40, "dy", "int", "the y dimension of a frame")
```

- **gmRef**
  Data in a geomatallurgical database can be atomic (a single value) or non-atomic. The later does not contain values itself. Instead it is a container for other variables, which store actual values, or even other containers. For example, the container project may store a name variable, which stores the name of the project, and several series being containers themselves.

```
CREATE TABLE IF NOT EXISTS gmObject (
   gmID BIGINT UNSIGNED REFERECES gmObject(gmVarID),
   gmRefID BIGINT UNSIGNED REFERECES gmObject(gmVarID)
          ON DELETE NULL
          ON UPDATE CASCADE
);
```

  The following tables are constructed according to the same principle. gmInteger, gmNumeric, gmString, gmText and gmBlob will be used for storing an actual value. For every entry a corresponding gmID, connecting the value to a gmObject, a gmVarID, defining the meaning of the value, and the value itself, given by x, is needed. These should be used for storing atomic data.

- **gmInteger**

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmInteger` (
`gmID` BIGINT(19) UNSIGNED NOT NULL,
`gmVarID` BIGINT(19) UNSIGNED,
`x` INT(10) UNSIGNED NULL DEFAULT NULL,
PRIMARY KEY (`gmID`, `gmVarID`),
INDEX `fk_gmInteger_gmObject_idx` (`gmID` ASC),
CONSTRAINT `fk_gmInteger_gmObject`
FOREIGN KEY (`gmID`)
REFERENCES `gmDatabase`.`gmObject` (`gmID`)
ON DELETE CASCADE
```

```
ON UPDATE CASCADE,
CONSTRAINT `fk_gmInteger_gmVarID`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **gmNumeric**

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmNumeric` (
`gmID` BIGINT(19) UNSIGNED NOT NULL,
`gmVarID` BIGINT(19) UNSIGNED,
`x` DOUBLE NULL DEFAULT NULL,
PRIMARY KEY (`gmID`, `gmVarID`),
INDEX `fk_gmNumeric_gmObject_idx` (`gmID` ASC),
CONSTRAINT `fk_gmNumeric_gmObject`
FOREIGN KEY (`gmID`)
REFERENCES `gmDatabase`.`gmObject` (`gmID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmNumeric_gmVarID`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **gmString**

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmString` (
`gmID` BIGINT(19) UNSIGNED NOT NULL,
`gmVarID` BIGINT(19) UNSIGNED,
`x` VARCHAR(255) NULL DEFAULT NULL,
PRIMARY KEY (`gmID`, `gmVarID`),
INDEX `fk_gmString_gmObject_idx` (`gmID` ASC),
CONSTRAINT `fk_gmString_gmObject`
FOREIGN KEY (`gmID`)
REFERENCES `gmDatabase`.`gmObject` (`gmID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmString_gmVarID`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
```

```
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **gmText**

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmText` (
`gmID` BIGINT(19) UNSIGNED NOT NULL,
`gmVarID` BIGINT(19) UNSIGNED,
`x` LONGTEXT NULL DEFAULT NULL,
PRIMARY KEY (`gmID`, `gmVarID`),
INDEX `fk_gmText_gmObject_idx` (`gmID` ASC),
CONSTRAINT `fk_gmText_gmObject`
FOREIGN KEY (`gmID`)
REFERENCES `gmDatabase`.`gmObject` (`gmID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmText_gmVarID`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **gmBlob**

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmBlob` (
`gmID` BIGINT(19) UNSIGNED NOT NULL,
`gmVarID` BIGINT(19) UNSIGNED,
`x` LONGBLOB NULL DEFAULT NULL,
PRIMARY KEY (`gmID`, `gmVarID`),
INDEX `fk_gmBlob_gmObject_idx` (`gmID` ASC),
CONSTRAINT `fk_gmBlob_gmObject`
FOREIGN KEY (`gmID`)
REFERENCES `gmDatabase`.`gmObject` (`gmID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmBlob_gmVarID`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- **gmElements**
  This table provides an overview of the elements a container object may have.

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmElements` (
`gmVarID` BIGINT(19) UNSIGNED,
`member` BIGINT(19) UNSIGNED NOT NULL,
`required` BOOL NOT NULL,
`remark` VARCHAR(80),
INDEX `fk_gmElements_gmVar_idx` (`gmVarID` ASC),
INDEX `fk_gmElements_member_idx` (`member` ASC),
CONSTRAINT `fk_gmElements_gmVar`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmElements_member`
FOREIGN KEY (`member`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

- **gmInherits**
  A geometallalurgical database may need the abstract concept of inheritance. Therefore gmInherits describes the structure of inheritance between the different kinds of variables in the database.

```
CREATE TABLE IF NOT EXISTS `gmDatabase`.`gmInherits` (
`gmVarID` BIGINT(19) UNSIGNED,
`child` BIGINT(19) UNSIGNED NOT NULL,
`remark` VARCHAR(80),
INDEX `fk_gmInherits_gmVarID_idx` (`gmVarID` ASC),
INDEX `fk_gmInherits_child_idx` (`child` ASC),
CONSTRAINT `fk_gmInherits_gmVar`
FOREIGN KEY (`gmVarID`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT `fk_gmInherits_child`
FOREIGN KEY (`child`)
REFERENCES `gmDatabase`.`gmVar` (`gmVarID`)
ON DELETE CASCADE
ON UPDATE CASCADE)
```

1.3 Data preparation

- Connecting to the database

```
db <- dbConnect(drv="MySQL",
```

```
                    dbname="gmDatabase",
                    host="databaseserver",
                    user="username",
                    password="secret")
```

- Preparing the database (only necessary the first time)

```
gmInitialize(db)
gmCreateElement(db,"H",1,1)
gmCreateElement(db,"H2",1,1)
gmCreateElement(db,"O",8,15.9994)
gmCreateColor("blue",0.0,0.0,1.0)
NAvogadro <-6.02214129E23
gmCreatePhase(db,
                     Mineral="Water",
                     Chem="H2O",
                     Density=1,
                     Description="A well known fluid",
                     UnitCellVolume=1E-3*0.0182/NAvogadro,
                     formula=list(O=1,H=2),
                     substitute=list(
                      list(O=c(1,1,1)),
                      list(H=c(0.98,0.97,1),
                           H2=c(0.02,0.0,0.03)))
                     color="blue"
                    )
```

- Creating the dataset for multiple MLA datasets

```
gmCreateVariable(db,
                 name="ParticleMLA",
                 type="Frame")
gmCreateVariable(db,
                 name="MLA",
                 type="Frame")
gmCreateVariable(db,
                 name="MillingTime",
                 type="Numeric")
gmCreateVariable(db,
                    name="Particles",
                    type="Particle2D")
gmCreateVariable(db,
                    name="ParticleImages",
                    type="Image")
```

- Creating a dataset for multiple MLA datasets

```
gmCreateFrame(db,"MillingExperiment",
              c("ParticleMLA","MillingTime"))
```

- Loading a dataset from the MLA
  In which form to we get the dataset.

```
mla <- gmInsituMLALoad("C:/DIR/MLADir12mm")
mla.Particles <- gmSplitParticles(mla)
mla.ParticleImages <- gmSplitImages(mla,mla.Particles)
mla.Frame <- gmCreateFrame(
             data=list(Particles=mla.Particles,
                       ParticleImages=mla.ParticleImages
                       )
             )

gmAddToFrame(db,"MillingExperiment",data=
      list(ParticleMLA=gmCreateFrame(
      ParticleMLA=mla.Frame,
      MillingDiameter=0.012
     ))
```

1.4 Particle MLA Analysis

Particle based MLA is how Mineral liberation analysis is ment. There are various types of analysis thinkable.

1.4.1 Descriptive Analysis

In a pure descriptive analysis we analyse and compare apparent features of the 2D sections. For example a particle consisting of two phases will often show as a one phase particle in the 2D section analysed by the MLA. A particle of a certain diameter will show with a smaller diameter at almost any 2D section. According to the stereological principle some 2D averages have the same expectations as some 3D averages. This are essentially the 2D (area), 1D (line), 0D (point) portions, being equivalent to the 3D volume portions (modal mineralogy), the 2D line density and the 1D point density of different types of boundaries, beeing equivalent to surface area of the given type, and 1

-