R e análise espacial para Geologia - Volume 2

André Luiz Lima Costa

“A second chance doesn’t mean you’re in the clear. In many ways, it is the more difficult thing. Because a second chance means that you have to try harder. You must rise to the challenge without the blind optimism of ignorance.”

Ling Ma

Para Regiane, Gabriela e George. Pilares abençoados de minha recuperação pessoal, sou eternamente agradecido a vocês...

Índice

Introdução....5
Dados....5
PARTE 4 – R, PostgreSQL e Postgis....6
4.1 – Instalando PostgreSQL e Postgis....6
4.1.1 – Windows....6
4.1.2 – Linux – Centos ou RedHat (PostgreSQL e Postgis)....11
4.1.3 – Linux – Ubuntu ou debian (PostgreSQL e Postgis)....11
4.1.4 – OSX....12
4.1.5 – Ajustes finais....12
4.2 – O PostgreSQL e SQL....13
4.2.1 – Testando instalação e Criando o Super Usuário....13
4.2.2 – Criando o superusuário....13
4.2.3 – Escopo de acesso ao banco de dados....13
4.2.4 – SQL....14
4.3 – Postgis básico....25
PARTE 5 – Criando o Banco de Dados....29
5.1 – Criando banco de dados geoespacial....29
5.1.1 – O banco de dados....29
5.1.2 – Sistema de Coordenadas do banco de dados....29
5.2 – Carregando dados geoespaciais no geobanco de dados....29
5.2.1 – Dados vector....29
5.2.2 – Dados raster....34
5.3 – Dados de Campo em geral....38
5.3.1 – Dados de sondagem....38
Inserindo dados....40
PARTE 6....43
6.1 – Integração de dados geológicos e análise espacial....43
6.1.1 - Dados na superfície....43
6.1.2 - Estruturando informação em espaço tridimensional....45
6.1.3 Recursos Minerais usando poligonais....49
6.2 – Modelagem de recursos usando R e GSLIB....55
6.2.1 – Instalando o GSLIB....55
6.2.2 – Preparando os dados....56
6.2.3 – Gerando Histogramas e CDF (Frequência de distribuição acumulada)....58
6.2.4 – Desagrupamento do dados (declustering)....62
6.2.5 – Histograma e CDF dos dados desagrupados....64
6.2.6 – Semivariograma....67
6.2.7 – Krigagem 3D....70
6.2.8 – Estatísticas do recurso....71
6.2.9 – Visualizando os Resultados....74
Visualizando os furos de sonda (criando arquivo vtk)....75
Visualizando as amostras (criando arquivo vtk)....77
Visualizando modelo de bloco (criando arquivo vtk)....79
6.3 – Curvas LAS e exemplo de análise petrofísica básica....84
6.3.1 – Lendo o arquivo LAS e carregando em formato data.frame....84
6.3.2 – Carregando os dados no banco de dados....84
6.3.3 – Extraindo parâmetros petrofísicos....88
Carregando as curvas....88
Calculando o VSh....88

Calculando a Porosidade.....89
Calculando o Sw.....91
Calculando Net Reservoir e Net Pay.....92
6.4 - Dados SEG-Y.....98
6.5 – Dados de Geofísica aérea (Oasis Montaj).....104

Introdução

Neste volume dois daremos continuidade ao uso de R em geologia. O objetivo agora é a integração de R com outras tecnologias e ferramentas no contexto da geologia.

Vamos cobrir na **PARTE 4** o básico de um banco de dados PostgreSQL e a extensão Postgis,

Na **PARTE 5** vamos cobrir como colocar os dados de exploração geológica no banco de dados para fácil acesso usando R

Na **PARTE 6** vamos usar R e outros softwares modernos que integram de maneira direta com banco de dados para visualização ou para a criação de novas interpretações.

Dados

Os dados usados nesta apostila se encontram em:

http://amazeone.com.br/barebra/apostilaRgeo/index.php

VISITE:

http://amazeone.com.br

PARTE 4 – R, PostgreSQL e Postgis

Lidando com grande quantidade de dados geoespaciais e de diversas fontes se faz crucial usar um banco de dados para armazenar, organizar e acessar toda essa informação. O PostgreSQL é o melhor banco de dados relacional existente com suporte a dados geoespaciais usando a extensão Postgis. Vamos aqui cobrir alguns conceitos bem básicos deste banco de dados.

4.1 – Instalando PostgreSQL e Postgis

O PostgreSQL pode ser instalado em vários sistema operacionais, veremos abaixo como instalar nos sistemas mais populares.

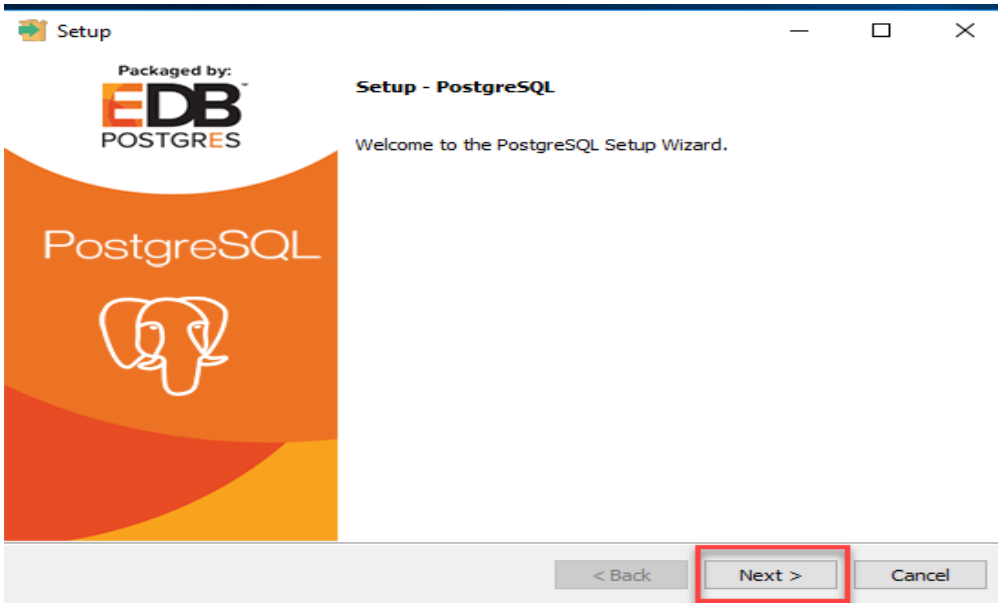
4.1.1 – Windows

Do site www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows selecione a versão do Postgresql e o sistema operacional windows (32 ou 64 bits).

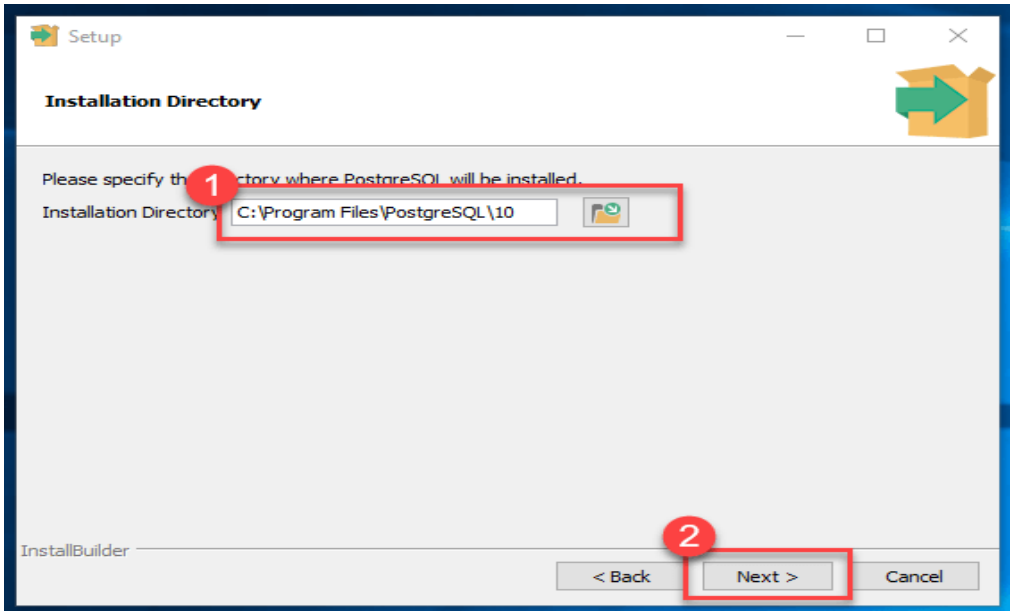
PostgreSQL Database Download

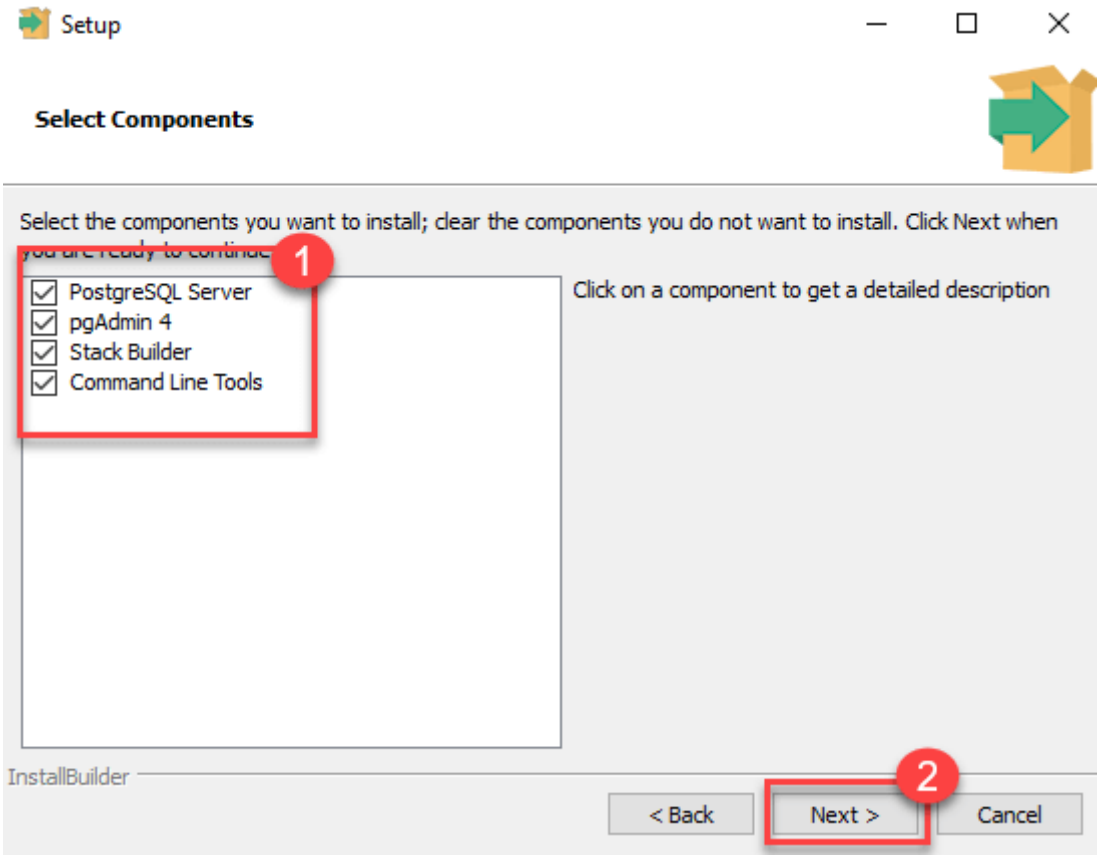
Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.3	N/A	N/A	Download	Download	N/A
11.8	N/A	N/A	Download	Download	N/A
10.13	Download	Download	Download	Download	Download
9.6.18	Download	Download	Download	Download	Download

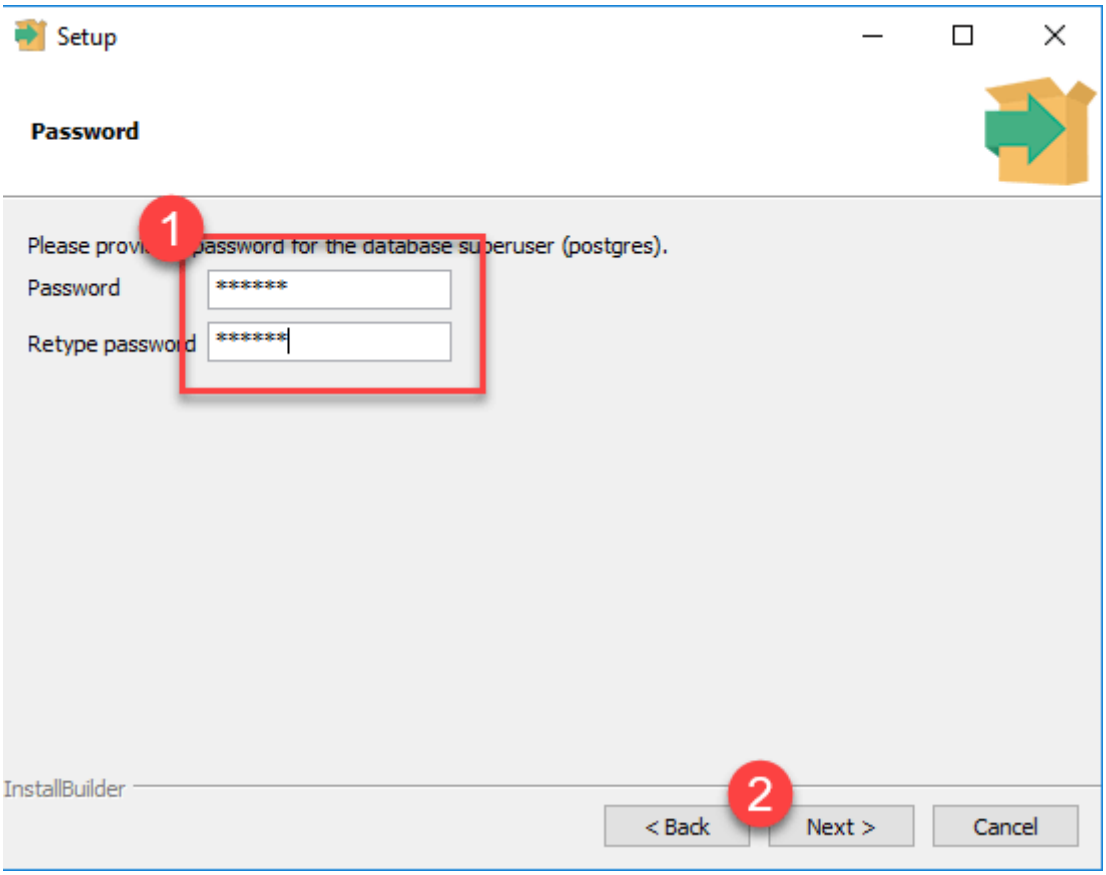
Faça o download e execute o **.exe** a seguinte tela deverá aparecer:

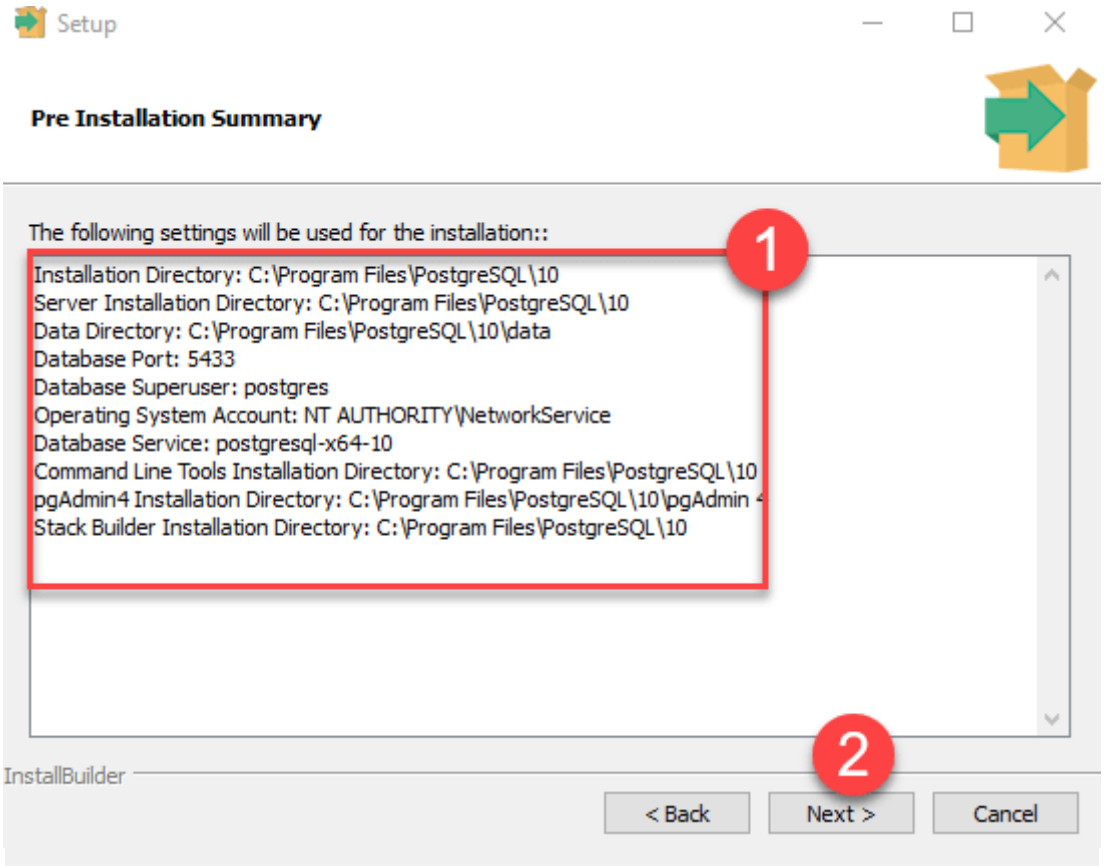


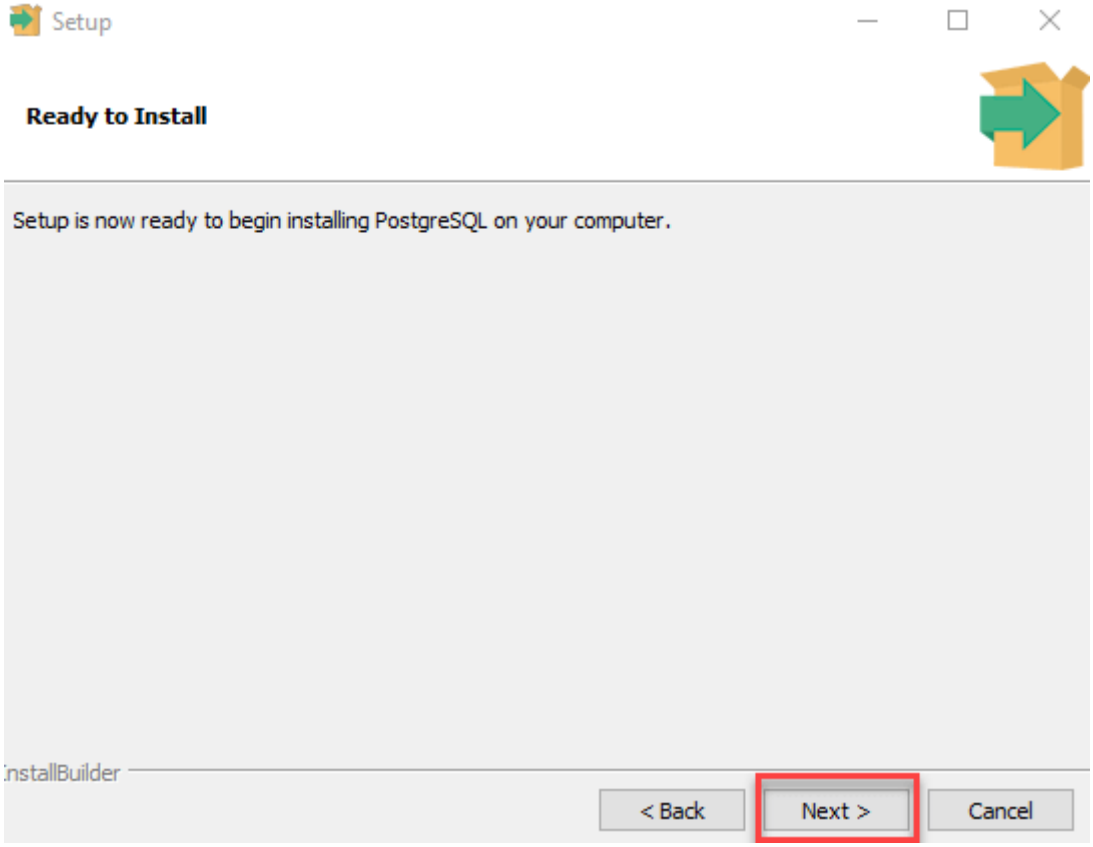
Siga as telas de instruções, selecionando as opções padrões, crie uma senha para o usuário postgres quando solicitado (guarde esta senha). Quando chegar na telo do StackBuilder selecione executar para podermos instalar o Postgis adiante.

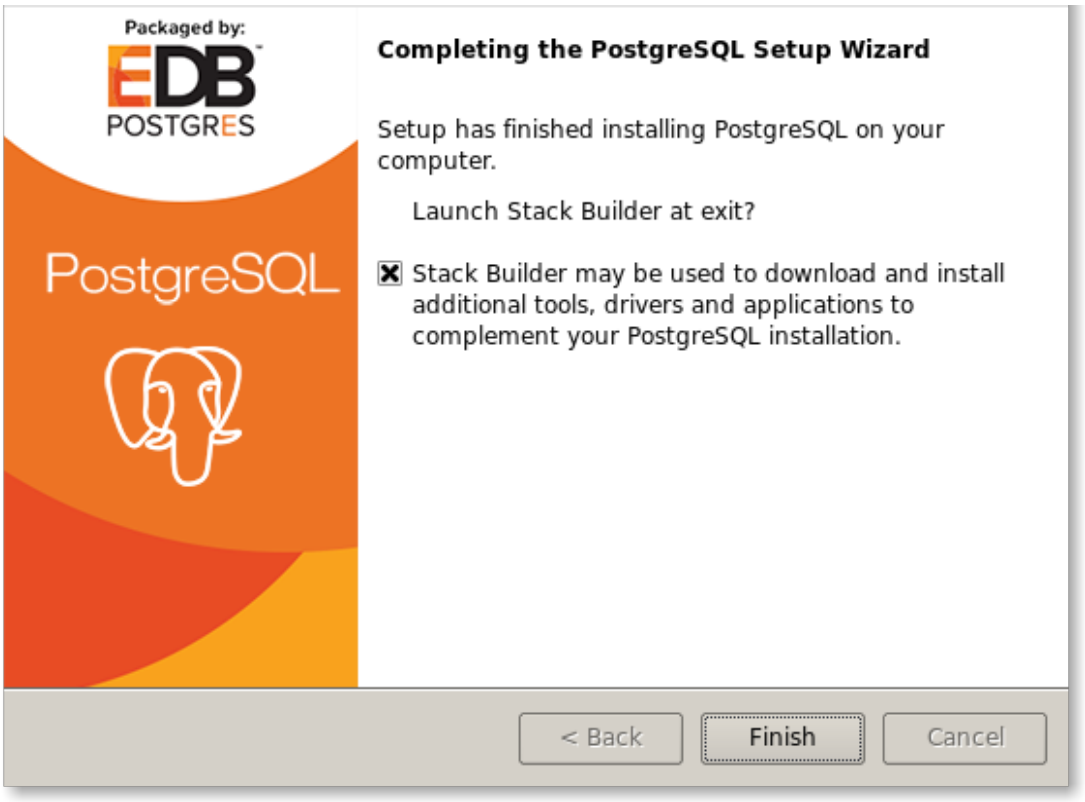



Setup
Select Components
Select the components you want to install; clear the components you do not want to install. Click Next when
1
PostgreSQL Server
pgAdmin 4
Stack Builder
Command Line Tools
Click on a component to get a detailed description
InstallBuilder
2
< Back
Next >
Cancel

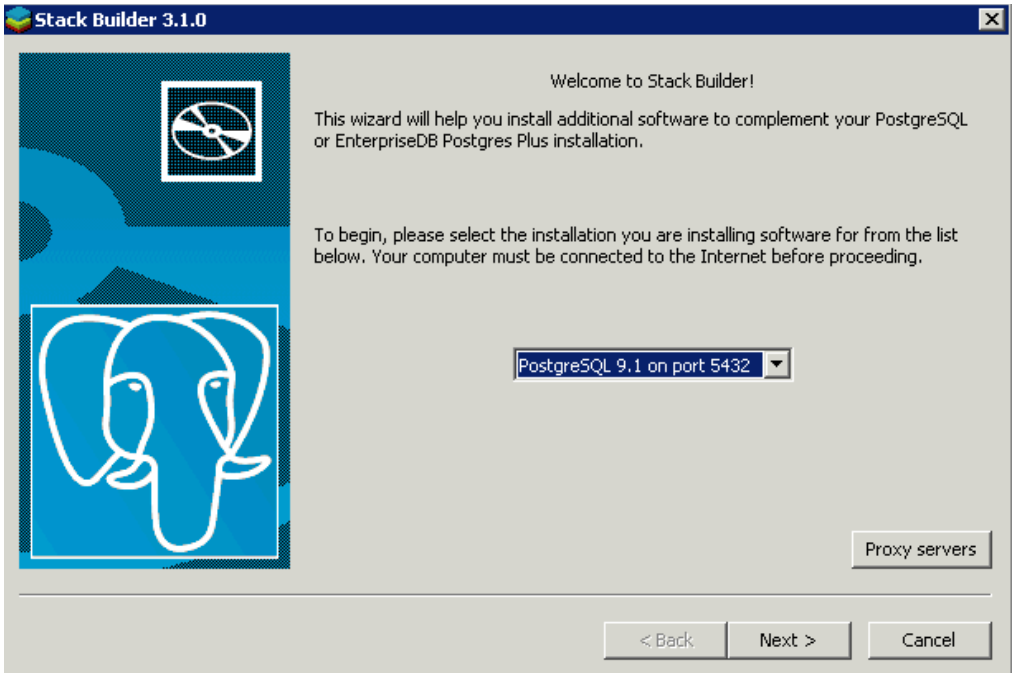

Setup
Password
1
Password
******
Retype password
******
InstallBuilder
2
< Back
Next >
Cancel


Setup
Pre Installation Summary
The following settings will be used for the installation::
1
Installation Directory: C:\Program Files\PostgreSQL\10
Server Installation Directory: C:\Program Files\PostgreSQL\10
Data Directory: C:\Program Files\PostgreSQL\10\data
Database Port: 5433
Database Superuser: postgres
Operating System Account: NT AUTHORITY\NetworkService
Database Service: postgresql-x64-10
Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\10
pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\10\pgAdmin
Stack Builder Installation Directory: C:\Program Files\PostgreSQL\10
2
InstallBuilder
< Back
Next >
Cancel

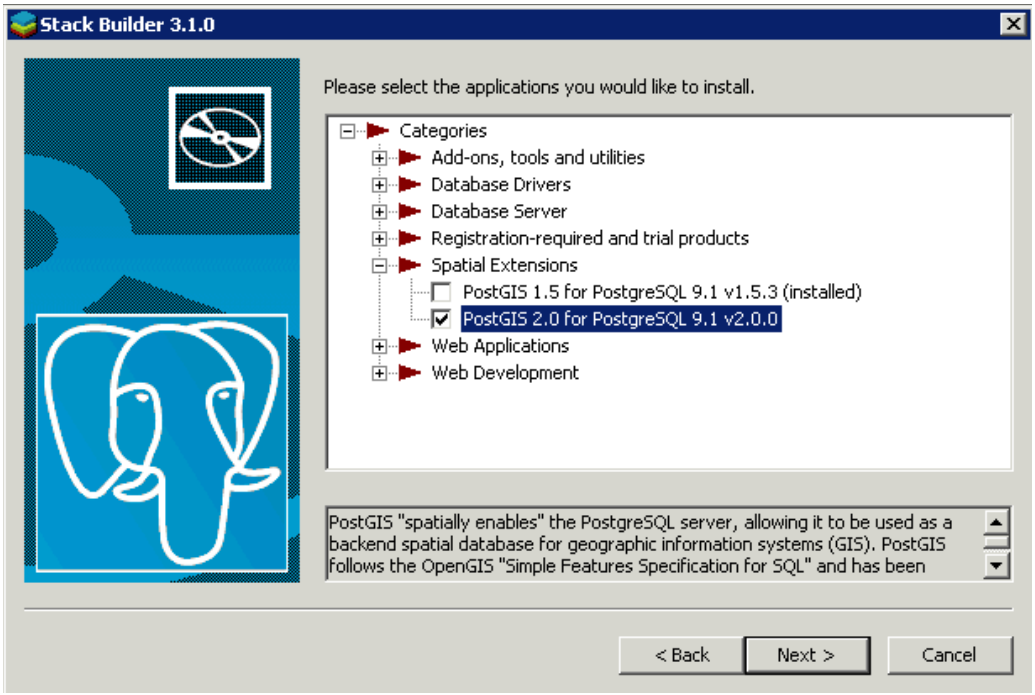

Setup
Ready to Install
Setup is now ready to begin installing PostgreSQL on your computer.
InstallBuilder
< Back
Next >
Cancel



A seguinte tela deverá aparecer (Versão poderá ser mais recente).



Escolha o componente PostGIS (a versão pode ser diferente da mostrada)



PostgreSQL e Postgis devem estar agora instalados no seu sistema windows.

4.1.2 – Linux – Centos ou RedHat (PostgreSQL e Postgis)

Abaixo seguem os comandos para instalar PostgreSQL e Postgis num sistema RedHat ou Centos

```
sudo yum install postgresql12-server
sudo /usr/pgsql-11/bin/postgresql-12-setup initdb
sudo systemctl start postgresql-12.service
sudo systemctl enable postgresql-12
sudo yum install postgresql12-devel
sudo yum -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
sudo yum install postgis30_12
sudo yum install geos-devel
```

4.1.3 – Linux – Ubuntu ou debian (PostgreSQL e Postgis)

Abaixo segue um script de referência (copie e grave como pgR_script.sh) para instalação num sistema RedHat ou Centos e execute usando `sudo ./pgR_script.sh`

```
sudo apt-get install postgresql-12
sudo apt-get install postgresql-server-dev-12
sudo apt install postgis
sudo apt install libgeos-dev
```

4.1.4 – OSX

A forma mais simples de instalar é usando homebrew. Instale homebrew conforme abaixo usando terminal:

```
$ xcode-select –install
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ echo "export LC_ALL=en_US.UTF-8" >> ~/.bash_profile
$ echo "export LANG=en_US.UTF-8" >> ~/.bash_profile
$ echo "export PATH=/usr/local/bin:$PATH" >> ~/.bash_profile && source
~/.bash_profile
$
```

No terminal execute:

```
$ brew install postgres
$ brew install postgis
$ pg_ctl -D usrlocal/var/postgres start
$ initdb usrlocal/var/postgres
```

4.1.5 – Ajustes finais

Abaixo segue uma lista de pacotes que devem ser instalados no seu ambiente R:

```
> install.packages('RPostgreSQL')
> install.packages('rpostgis')
```

Vamos agora dar uma pincelada nos conceitos de PastgreSQL, SQL básico e Postgis básico. Para se aprofundar nesses tópicos existe vasta informação na internet.

4.2 – O PostgreSQL e SQL

Após a instalação do PostgreSQL, o primeiro passo é criarmos um usuário que ira ser o gerenciador ou administrador do banco de dados e em seguida vamos determinar o acesso do banco de dados. Depois iremos cobrir os principais aspectos de SQL de forma bem superficial por não se tratar do tópico principal deste volume.

4.2.1 – Testando instalação e Criando o Super Usuário

Em uma tela monitor teste a instalação do postgresql como o usuário padrão ‘postgres’ que é criado durante a instalação usando `sudo –i –u postgres`
Após entrar digite `psql` e o prompt do psql deverá aparecer.

```
$ sudo –i –u postgres
[sudo] password for usuário:
postgres@ $ psql
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

postgres=# \q
postgres@ $ exit
logout
$
```

Sua instalação deverá apresentar o resultado acima, saia do prompt do psql usando `\q` e saia do usuário postgres usando `exit`.

4.2.2 – Criando o superusuário

Efetue novamente o login como ‘postgres’ e execute o comando `createuser` conforme mostrado abaixo. O ideal é que o `nomeDoUsuário` seja já o seu usuário no seu sistema operacional para efeitos de praticidade.

```
$ sudo –i –u postgres
[sudo] password for usuário:
postgres@ $ createuser --interactive
Enter name of role to add: nomeDoUsuário
Shall the new role be a superuser? (y/n) y
```

Pronto, já temos um usuário com todos os poderes no postgresql.

4.2.3 – Escopo de acesso ao banco de dados

Agora vamos configurar o postgreSQL de acordo com o acesso que este dará aos usuários, somente localhost ou visível em intra / internet. Também podemos configurar grupos ou usuários com alguns tipos de restrições.

Acesso Local
Se o acesso ao banco de dados for efetuado somente na máquina local (localhost) nada é preciso ser feito no que diz respeito a configurações, pode pular para o próximo tópico.

Acesso Remoto

Caso o acesso ao banco de dados for a partir de máquinas remotas se faz necessária a alteração dos seguintes arquivos de configuração e o reinício do postgreSQL para aplicar as mudanças,:
Modificamos como o usuário raiz o arquivo postgresql.conf que pode ser localizado usando:

```
$ sudo -i
root@~# find / -name "postgresql.conf"
/etc/postgresql/12/main/postgresql.conf
```

No meu sistema ele está em.: `/etc/postgresql/12/main/postgresql.conf`

Edite o arquivo e acrescente `listen_addresses = '*'` na última linha do arquivo usando :

```
root@:~# echo "listen_addresses='*'" >> /etc/postgresql/12/main/postgresql.conf
```

Edite o arquivo `pg_hba.conf` e acrescente `host all all 0.0.0.0/32 md5` na última linha do arquivo usando :

```
root@~# echo "host all all 0.0.0.0/32 md5">>/etc/postgresql/12/main/pg_hba.conf
```

Agora reiniciamos o PostgreSQL usando:

```
root@:~# systemctl restart postgresql.service
```

Desta forma usuários do banco de dados podem usar máquina remotas para acessar o banco de dados após entrarem com as credenciais de **nome de usuário** e **senha**.

Modificando os valores que entramos no pg_hba.conf podemos restringir mais o acesso caso seja necessário. Podemos também refinar o acesso usando o comando SQL `ROLE` que veremos e seguir.
Pode agora sair do usuário root usando `exit`.

4.2.4 – SQL

SQL ou Structured Querry Language é a forma que criamos tabelas e extraímos informações destas em um banco de dados.

Vamos criar o nosso banco de dados com caracteres UTF8 e nome geobanco. Existem duas maneiras de fazermos isso: Usando o comando createbd

```
$ createdb geobanco --encoding=utf8
```

Ou usando SQL de dentro do banco de dados template1 que é instalado junto com postgresql.

```
$ psql template1
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

template1=# CREATE DATABASE geobanco ENCODING UTF8;
CREATE DATABASE
template1=# \q
$
```

Uma vez criado nosso banco de dados entramos nele usando:

```
$ psql geobanco
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

geobanco=#
```

Vamos agora ver rapidamente os comandos principais do PostgreSQL

Já vimos acima que acessamos o banco de dados via linha de comando usando:

psql [nomeDoBancoDeDados]
Por exemplo:

```
$ psql geobanco
```

Agora veremos os comando de visualização de informações de dentro do ambiente Postgresql que é indicado pelo prompt "`geobanco=#`". Estes comandos se iniciam por '\':

O comando \c conecta a outro banco de dados;
Por exemplo, o comando seguinte se conecta ao banco de dados template1 e de volta ao banco de dados geobanco:

```
geobanco=# \c template1
You are now connected to database "template1" as user "username".
template1=# \c geobanco
You are now connected to database "geobanco" as user "username".
geobanco=#
```

Lista todos os bancos de dados no servidor PostgreSQL:

```
geobanco=# \l
```

Lista todos os esquemas:

```
geobanco=# \dn
```

Lista todos procedimentos armazenados e funções:

```
geobanco=# \df
```

Lista todas as visões (views):

```
geobanco=# \dv
```

Lista todas as tabelas no banco de dados atual:

```
geobanco=# \dt
```

Lista todas as tabelas no banco de dados atual com mais detalhes:

```
geobanco=# \dt+
```

Lista o detalhe de uma tabela do banco de dados individualmente:

```
geobanco=# \d+ nome_tabela
```

Mostra o procedimento armazenado ou função individualmente:

```
geobanco=# \df+ nome_função
```

Mostra o resultado em formato mais apresentável:
geobanco=# \x

Lista todos os usuários:
geobanco=# \du

Para sair do psql use:
geobanco=# \q

Veremos agora como criar um usuário. Isso é feito através de uma regra (ROLE). É interessante, e seguro criarmos um usuário com somente o privilégio de leitura mas não de alteração de tabelas em um banco de dados. Vamos ver abaixo como isso é feito:

Primeiro criamos uma regra com o nome de 'leitor', lembre que todo comando SQL no ambiente psql deve ser terminado com '**;**' exceto os comandos que se iniciam com '/'.

geobanco=# CREATE ROLE leitor;

E definimos que essa regra somente lê tabelas do esquema PUBLIC e que não pode fazer login no banco de dados.

geobanco=# GRANT usage on SCHEMA public to leitor;

Agora criamos um usuário chamado 'droid' com senha de acesso 'devcor'

geobanco=# CREATE USER droid WITH PASSWORD 'devcor';

E adicionamos a 'leitor' que transfere as permissões de leitor para 'droid'.

geobanco=# GRANT leitor TO droid;

Usamos \du pra checar os usuários.

```
geobanco=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 droid     |                                                            | {leitor}
 leitor    | Cannot login                                               | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 xxxx      | Superuser, Create role, Create DB                          | {}
```

Para garantir o acesso a uma tabela para o usuário 'droid' teremos que usar:

geobanco=# GRANT SELECT ON **nome_da_tabela** TO leitor;

Para garantir o acesso a todas as tabelas para o usuário 'droid' usamos:

geobanco=# GRANT SELECT ON ALL TABLES IN SCHEMA public to leitor;

Para retirar o acesso a uma tabela para o usuário 'droid' teremos que usar:

geobanco=# REVOKE SELECT ON **nome_da_tabela** FROM leitor;

Para retirar acesso a todas as tabelas para o usuário 'droid' usamos:

geobanco=# REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM leitor;

Agora que vimos como criar o banco de dados, navegar sobre informações do banco de dados com os comandos \ e criar e garantir acesso ao banco de dados vamos ver superficialmente como criar as tabelas com os dados e as tarefas básicas relacionadas a seleção, inserção, atualização e deleção de informações.

Criando uma nova tabela:

```
CREATE [TEMP] TABLE [IF NOT EXISTS] nome_tabela(
   chavepri SERIAL PRIMARY KEY,
   col1 type(size) NOT NULL,
   col2 type(size) NULL,
   ...
);
geobanco=# CREATE TABLE collar(
   holeid varchar(12) PRIMARY KEY,
   x numeric(12,2) NOT NULL,
   y numeric(12,2) NOT NULL,
   sonda int DEFAULT 1
);
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 holeid | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)
```

Adicionando uma nova coluna na tabela:

```
ALTER TABLE nome_tabela ADD COLUMN nova_coluna TYPE;
geobanco=# ALTER TABLE collar ADD COLUMN z numeric(6,2);
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 holeid | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
 z      | numeric(6,2)          |           |          |
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)
```

Removendo uma coluna da tabela:

```
ALTER TABLE nome_tabela DROP COLUMN nome_coluna;
geobanco=# ALTER TABLE collar DROP COLUMN z;
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 holeid | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)
```

Renomeando uma coluna da tabela:

```
ALTER TABLE nome_tabela RENAME nome_coluna TO novo_nome_coluna;
geobanco=# ALTER TABLE collar RENAME holeid TO idfuro;
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
```

```
--------+----------------------+-----------+----------+---------
 idfuro | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
Indexes:
    "collar_pkey" PRIMARY KEY, btree (idfuro)
```

Definindo ou removendo um valor padrão para uma coluna:

```
ALTER TABLE nome_tabela ALTER COLUMN [SET DEFAULT valor | DROP
DEFAULT]
geobanco=# ALTER TABLE collar ALTER COLUMN sonda SET DEFAULT 2;
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
 idfuro | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 2
Indexes:
    "collar_pkey" PRIMARY KEY, btree (idfuro)
```

Removendo uma chave primaria de uma tabela:

```
ALTER TABLE nome_tabela DROP CONSTRAINT nome_chave_primaria;
geobanco=# ALTER TABLE collar DROP CONSTRAINT collar_pkey;
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
 idfuro | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 2
```

Adicionando uma chave primária a uma tabela:

```
ALTER TABLE nome_tabela ADD PRIMARY KEY (coluna,...);
geobanco=# ALTER TABLE collar ADD PRIMARY KEY (idfuro,x,y);
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
 idfuro | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 2
Indexes:
    "collar_pkey" PRIMARY KEY, btree (idfuro, x, y)
```

Renomeando uma tabela.

```
ALTER TABLE nome_tabela RENAME TO novo_nome_tabela;
geobanco=# ALTER TABLE collar RENAME TO bocafuro;
geobanco=# \d collar
Did not find any relation named "collar".
geobanco=# \d bocafuro
                    Table "public.bocafuro"
 Column |         Type          | Collation | Nullable | Default
--------+----------------------+-----------+----------+---------
```

```
 idfuro | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 2
Indexes:
    "collar_pkey" PRIMARY KEY, btree (idfuro, x, y)
```

Removendo uma tabela e suas dependências:

```
DROP TABLE [IF EXISTS] nome_tabela CASCADE;
geobanco=# DROP TABLE bocafuro CASCADE;
geobanco=# \d bocafuro
Did not find any relation named "bocafuro".
```

Índices facilitam as seleções e a velocidade de seleção de dados, vamos ver como criar e remover índices de uma tabela.
Vamos novamente criar a nossa tabela:

```
geobanco=# CREATE TABLE collar(
   holeid varchar(12) PRIMARY KEY,
   x numeric(12,2) NOT NULL,
   y numeric(12,2) NOT NULL,
   sonda int DEFAULT 1
);
```

Crinado um índice com nome espeficado em uma tabela:

```
CREATE [UNIQUE] INDEX index_name ON table (column,...)
geobanco=# CREATE INDEX indice ON collar (holeid);
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 holeid | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)
    "indice" btree (holeid)
```

Removendo um índice específico de uma tabela:

```
DROP INDEX index_name;
geobanco=# DROP INDEX indice;
geobanco=# \d collar
                    Table "public.collar"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 holeid | character varying(12) |           | not null |
 x      | numeric(12,2)         |           | not null |
 y      | numeric(12,2)         |           | not null |
 sonda  | integer               |           |          | 1
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)
```

Criando e acessando dados em uma tabela. Vamos criar duas tabelas e demonstrar como inserir, extrair e modificar os dados nelas.
Criando as tabelas:

```
geobanco=# CREATE TABLE collar (
holeid VARCHAR(20) PRIMARY KEY,
projeto VARCHAR(40) NOT NULL,
area VARCHAR(40) NOT NULL,
```

```
target VARCHAR NOT NULL,
x NUMERIC(12,2) NOT NULL,
y NUMERIC(12,2) NOT NULL,
z NUMERIC(6,2) NOT NULL,
srid INTEGER NOT NULL,
az NUMERIC(5,2) NOT NULL,
dip NUMERIC(4,2) NOT NULL,
td NUMERIC(7,2) NOT NULL,
rig VARCHAR(20) NOT NULL,
drillco VARCHAR(50),
inicio DATE,
fim DATE);
geobanco=# \d collar
                    Table "public.collar"
 Column  |         Type          | Collation | Nullable | Default
---------+-----------------------+-----------+----------+---------
 holeid  | character varying(20) |           | not null |
 projeto | character varying(40) |           | not null |
 area    | character varying(40) |           | not null |
 target  | character varying     |           | not null |
 x       | numeric(12,2)         |           | not null |
 y       | numeric(12,2)         |           | not null |
 z       | numeric(6,2)          |           | not null |
 srid    | integer               |           | not null |
 az      | numeric(5,2)          |           | not null |
 dip     | numeric(4,2)          |           | not null |
 td      | numeric(7,2)          |           | not null |
 rig     | character varying(20) |           | not null |
 drillco | character varying(50) |           |          |
 inicio  | date                  |           |          |
 fim     | date                  |           |          |
Indexes:
    "collar_pkey" PRIMARY KEY, btree (holeid)

geobanco=# CREATE TABLE survey (
holeid VARCHAR(20) NOT NULL,
estacao VARCHAR(10) NOT NULL,
x NUMERIC(12,2) NOT NULL,
y NUMERIC(12,2) NOT NULL,
depth NUMERIC(6,2) NOT NULL,
az NUMERIC(5,2) NOT NULL,
dip NUMERIC(4,2) NOT NULL,
diametro VARCHAR(20) NOT NULL,
comentario VARCHAR(50));
geobanco=# \d survey
                      Table "public.survey"
   Column   |         Type          | Collation | Nullable |Default
------------+-----------------------+-----------+----------+--------
 holeid     | character varying(20) |           | not null |
 estacao    | character varying(10) |           | not null |
 x          | numeric(12,2)         |           | not null |
 y          | numeric(12,2)         |           | not null |
 depth      | numeric(6,2)          |           | not null |
 az         | numeric(5,2)          |           | not null |
 dip        | numeric(4,2)          |           | not null |
 diametro   | character varying(20) |           | not null |
 comentario | character varying(50) |           |          |
```

Inserindo uma nova linha em tabelas:

```
INSERT INTO nome_tabela(coluna1,coluna2,…) VALUES(valor_1,
valor_2,...);
geobanco=# INSERT INTO collar(holeid,projeto,area,target, x,y,z,
srid,az,dip,td,rig,drillco,inicio,fim) VALUES('PBAT-10-
01','Potássio AM','AUTAZES','AUTAZES',271624,9614442,22,29191,0,-
90,781.1,'LY-50','BoartLongyear','28/01/2010','05/05/2010');
geobanco=# INSERT INTO survey(holeid,
estacao,x,y,depth,az,dip,diametro,comentario) VALUES('PBAT-10-01',
'1',271624,9614442,0,-90,0,'HQ','');
```

Inserindo múltiplas linhas em tabelas:

```
INSERT INTO nome_tabela(coluna1, coluna2, …)
VALUES(valor_1,valor_2,...),(valor_1,valor_2,...),(valor_1,
valor_2,  ...) ...
geobanco=# INSERT INTO collar(holeid,projeto,area,target, x,y,z,
srid,az,dip,td,rig,drillco,inicio,fim) VALUES('PBAT-10-
02','Potássio AM', 'AUTAZES', 'AUTAZES',279340, 9612680, 27,
29191,0,-90,889.25,'LY-50', 'BoartLongyear','05/05/2010',
'11/08/2010'), ('PBAT-10-05','Potássio AM', 'AUTAZES', 'AUTAZES',
280941, 9613595,31,29191,0,-90,883.25,'CS-4002', 'Geosol',
'19/11/2010', '02/03/2011');
geobanco=# INSERT INTO survey(holeid, estacao,x,y,depth, az,dip,
diametro,comentario) VALUES('PBAT-10-02',1,279340,9612680,0,-
90,0,'HQ',''),('PBAT-10-05',1,280941,9613595,0,-90,0,'HQ','');
```

Observe que em SQL valores não numéricos são sempre envolvidos por aspas simples ' '.

Selecionando todos os dados de uma tabela:

```
SELECT * FROM nome_tabela;
geobanco=# SELECT * FROM survey;
   holeid   | estacao |     x     |     y      | depth |   az   | dip  | diametro |
comentario
------------+---------+-----------+------------+-------+--------+------+----------
+------------
 PBAT-10-01 | 1       | 271624.00 | 9614442.00 |  0.00 | -90.00 | 0.00 | HQ       |
 PBAT-10-02 | 1       | 279340.00 | 9612680.00 |  0.00 | -90.00 | 0.00 | HQ       |
 PBAT-10-05 | 1       | 280941.00 | 9613595.00 |  0.00 | -90.00 | 0.00 | HQ       |
(3 rows)
```

Selecionando dados de uma coluna específica de todas as linhas da tabela:

```
SELECT lista_coluna FROM nome_tabela;
geobanco=# SELECT holeid,drillco FROM collar;
   holeid   |    drillco
------------+---------------
 PBAT-10-01 | BoartLongyear
 PBAT-10-02 | BoartLongyear
 PBAT-10-05 | Geosol
(3 rows)
```

Selecionando valores únicos de uma coluna na tabela:

```
SELECT DISTINCT (coluna) FROM nome_tabela;
geobanco=# SELECT DISTINCT (rig) FROM collar;
   rig
---------
 CS-4002
 LY-50
(2 rows)
```

Selecionando dados de uma tabela usando um filtro:

```
SELECT * FROM nome_tabela WHERE condição;
geobanco=# SELECT holeid, x, y, z FROM collar WHERE
drillco='BoartLongyear';
   holeid   |     x     |     y      |   z
------------+-----------+------------+-------
 PBAT-10-01 | 271624.00 | 9614442.00 | 22.00
 PBAT-10-02 | 279340.00 | 9612680.00 | 27.00
(2 rows)
```

Assinalando um apelido à coluna selecionada:

```
SELECT coluna_1 AS uma_coluna, ...FROM nome_tabela;
geobanco=# SELECT holeid AS furo FROM survey;
    furo
------------
 PBAT-10-01
 PBAT-10-02
 PBAT-10-05
(3 rows)
```

Selecionando dados usando operador LIKE :

```
SELECT * FROM nome_tabela WHERE columa LIKE '%valor%';
geobanco=# SELECT holeid,x,y,z FROM collar WHERE rig LIKE '%CS%';
   holeid   |     x     |     y      |   z
------------+-----------+------------+-------
 PBAT-10-05 | 280941.00 | 9613595.00 | 31.00
(1 row)
```

Selecionando dados usando o operador BETWEEN (entre):

```
SELECT * FROM nome_tabela WHERE coluna BETWEEN menor AND maior;
geobanco=#  SELECT  holeid,dip,az,td  FROM  collar  WHERE  td  BETWEEN
880 AND 885;
   holeid   |  dip   |  az  |   td
------------+--------+------+--------
 PBAT-10-05 | -90.00 | 0.00 | 883.25
(1 row)
```

Retorna o número de linhas de uma tabela:

```
SELECT COUNT (*) FROM nome_tabela;
geobanco=# SELECT COUNT (*) FROM survey;
 count
-------
     3
(1 row)
```

Ordena as linhas de resultado de forma ascendente ou descendente:

```
SELECT lista_colunas FROM nome_tabela ORDER BY coluna ASC [DESC],
coluna2 ASC [DESC],...;
geobanco=# SELECT holeid,td FROM collar ORDER BY td ;
   holeid   |   td
------------+--------
 PBAT-10-01 | 781.10
 PBAT-10-05 | 883.25
 PBAT-10-02 | 889.25
```

Agrupa linhas de resultado usando a instrução GROUP BY coluna.

```
SELECT lista_coluna FROM nome_tabela GROUP BY coluna_1,
coluna_2, ...;
geobanco=# SELECT sum(td),drillco FROM collar GROUP BY drillco;
   sum   |    drillco
---------+---------------
 1670.35 | BoartLongyear
  883.25 | Geosol
(2 rows)
```

Filtrando grupos usando a instrução HAVING.

```
geobanco=# SELECT holeid,rig,td FROM collar GROUP BY holeid HAVING
td>885;
   holeid   |  rig  |   td
------------+-------+--------
 PBAT-10-02 | LY-50 | 889.25
(1 row)
```

Modificando dados.

Atualizando os dados para todas linhas:

```
UPDATE nome_tabela SET coluna_1 = valor_1,    ...;
geobanco=# UPDATE collar SET dip = 90;
UPDATE 3
geobanco=# SELECT holeid,dip FROM collar;
   holeid   |  dip
------------+-------
 PBAT-10-01 | 90.00
 PBAT-10-02 | 90.00
 PBAT-10-05 | 90.00
(3 rows)
```

Atualizando os dados para as linhas que satisfaçam a condição especificada pela instrução WHERE.

```
UPDATE nome_tabela SET coluna_1 = valor_1,    … WHERE condicao;
geobanco=# UPDATE survey SET dip = 90 WHERE holeid='PBAT-10-01';
geobanco=# select holeid,estacao,dip from survey;
   holeid   | estacao |  dip
------------+---------+-------
 PBAT-10-02 | 1       |  0.00
 PBAT-10-05 | 1       |  0.00
 PBAT-10-01 | 1       | 90.00
(3 rows)
```

Deletando todas a linhas de uma tabela:

```
DELETE FROM nome_tabela;
geobanco=# DELETE FROM survey;
geobanco=# select * from survey;
  holeid  |  estacao  |  x  |  y  |  depth  |  az  |  dip  |  diametro  |
comentario
--------+---------+---+---+-------+----+-----+----------+---------
(0 rows)
```

Deletando linhas específicas baseado em uma condição:

```
DELETE FROM nome_tabela WHERE condicao;
geobanco=# DELETE FROM collar WHERE td<885;
geobanco=# SELECT * FROM collar;
  holeid   |   projeto   |  area   | target  |      x      |      y
|   z    | srid  |  az  |  dip  |   td   |  rig  |    drillco    |
inicio   |    fim
-----------+-------------+---------+---------+-----------
+------------+-------+-------+------+-------+--------+-------
+---------------+------------+------------
  PBAT-10-02  |  Potássio  AM  |  AUTAZES  |  AUTAZES  |  279340.00  |
9612680.00 | 27.00 | 29191 | 0.00 | 90.00 | 889.25 | LY-50 |
BoartLongyear | 2010-05-05 | 2010-08-11
(1 row)
```

Para remover um banco de dados usamos, fora do ambiente psql:

```
$ dropdb geobanco
```

Cobrimos aqui o básico do SQL, muito mais é possível ser feito. Existem inúmeras fontes na internet para aprofundar o conhecimento em SQL.

4.3 – Postgis básico

Postgis é uma extensão do PostgreSQL para lidar com dados espaciais dentro de um sistema de banco de dados relacionais (RDBMS). Dentro de uma tabela que representa uma entidade espacial temos a coluna geometria que é geralmente chamada de ‘geom’ e as demais colunas com os atributos associados a essa geometria.

Vamos criar novamente o nosso banco de dados com caracteres UTF8 e nome geobanco.

```
$ createdb geobanco --encoding=utf8
```

Agora, de dentro don banco de dados iremos criar a extensão postgis.

```
$ psql geobanco
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

geobanco=# CREATE EXTENSION postgis;
CREATE EXTENSION
```

Quando criamos a extensão na versão 3 do postgis uma tabela e duas visões (views) são criadas. Esses elementos servem de suporte para as tabelas com elementos ‘geom’ que serão criadas. A tabela ‘spatial_ref_sys’, como o próprio nome já diz é uma lista com todos os sistemas de referência geográficos existentes

Postgis te dá a escolha de duas formas diferentes para armazenar a sua informação espacial:
Geometry, que assume que todo o seu dado espacial é representado em um plano cartesiano (como uma projeção de mapa);
Geography, que assume que seus dados espaciais são representados por pontos na superfície terrestre definidos por latitudes e longitudes.

Visualizando as tabelas e suas colunas:

```
geobanco=# \d
              List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+---------
 public | geography_columns | view  | regiane
 public | geometry_columns  | view  | regiane
 public | spatial_ref_sys   | table | regiane
(3 rows)

geobanco=# \d geography_columns;
                View "public.geography_columns"
       Column        |  Type   | Collation | Nullable | Default
---------------------+---------+-----------+----------+---------
 f_table_catalog     | name    |           |          |
 f_table_schema      | name    |           |          |
 f_table_name        | name    |           |          |
 f_geography_column  | name    |           |          |
 coord_dimension     | integer |           |          |
 srid                | integer |           |          |
 type                | text    |           |          |
```

```
geobanco=# \d geometry_columns;
                    View "public.geometry_columns"
      Column   |       Type        | Collation | Nullable | Default
---------------+-------------------+-----------+----------+--------
 f_table_catalog   |character varying(256)| C         |          |
 f_table_schema    | name                 |           |          |
 f_table_name      | name                 |           |          |
 f_geometry_column | name                 |           |          |
 coord_dimension   | integer              |           |          |
 srid              | integer              |           |          |
 type              | character varying(30)|           |          |

geobanco=# \d spatial_ref_sys;
                  Table "public.spatial_ref_sys"
  Column   |          Type           | Collation | Nullable | Default
-----------+-------------------------+-----------+----------+--------
 srid      | integer                 |           | not null |
 auth_name | character varying(256)  |           |          |
 auth_srid | integer                 |           |          |
 srtext    | character varying(2048) |           |          |
 proj4text | character varying(2048) |           |          |
Indexes:
  "spatial_ref_sys_pkey" PRIMARY KEY, btree (srid)
Check constraints:
  "spatial_ref_sys_srid_check" CHECK (srid > 0 AND srid <= 998999)
```

Se vamos trabalhar com dados do tipo raster teremos que adicionar a extensão ‘postgis_raster’ (versão Postgis >= 3) e duas novas visões serão criadas para auxiliar os elementos espaciais do tipo raster.

Visualizando o que é adicionado pela extensão postgis_raster:

```
geobanco=# CREATE EXTENSION postgis_raster;
CREATE EXTENSION
geobanco=# \d
              List of relations
 Schema |       Name         | Type  |  Owner
--------+--------------------+-------+---------
 public | geography_columns  | view  | regiane
 public | geometry_columns   | view  | regiane
 public | raster_columns     | view  | regiane
 public | raster_overviews   | view  | regiane
 public | spatial_ref_sys    | table | regiane
(5 rows)

geobanco=# \d raster_columns
                    View "public.raster_columns"
      Column   |       Type        | Collation | Nullable | Default
---------------+-------------------+-----------+----------+--------
 r_table_catalog  | name              |           |          |
 r_table_schema   | name              |           |          |
 r_table_name     | name              |           |          |
 r_raster_column  | name              |           |          |
 srid             | integer           |           |          |
 scale_x          | double precision  |           |          |
 scale_y          | double precision  |           |          |
 blocksize_x      | integer           |           |          |
 blocksize_y      | integer           |           |          |
 same_alignment   | boolean           |           |          |
```

```
 regular_blocking | boolean            |           |          |
 num_bands        | integer            |           |          |
 pixel_types      | text[]             | C         |          |
 nodata_values    | double precision[] |           |          |
 out_db           | boolean[]          |           |          |
 extent           | geometry           |           |          |
 spatial_index    | boolean            |           |          |
```

```
geobanco=# \d raster_overviews
               View "public.raster_overviews"
     Column      |  Type   | Collation | Nullable | Default
-----------------+---------+-----------+----------+---------
 o_table_catalog | name    |           |          |
 o_table_schema  | name    |           |          |
 o_table_name    | name    |           |          |
 o_raster_column | name    |           |          |
 r_table_catalog | name    |           |          |
 r_table_schema  | name    | default   |          |
 r_table_name    | name    | default   |          |
 r_raster_column | name    | default   |          |
 overview_factor | integer |           |          |
```

Como não criamos nenhuma tabela com dados espaciais, as visões estão vazias, a medida que cada tabela com dado espacial é criada, uma entrada é também criada nas respectivas visões seja os elementos do tipo geometry, geography ou raster.

A tabela spatial_ref_sys apresenta 8500 linhas, cada uma representando um diferente SRID e seus textos descritivos em formato WKT SRS e proj4.

Postgis possui geometrias do tipo ponto, linha, área, multiponto, multilinha, multiareas e coleções. Estes entes geométricos podem estar associados a uma componente Z (elevação) e uma componente M (medida), assim as formas possíveis de geometrias são:
POINT → ‘POINT(0 0)’
MULTIPOINT → ‘MULTIPOINT(0 0, 1 1)’
POINT Z → ‘POINTZ(0 0 100)’
MULTIPOINT Z → ‘MULTIPOINTZ(0 0 100, 1 1 123)’
POINT M→ ‘POINTM(0 0 1)’
MULTIPOINT M→ ‘MULTIPOINTM(0 0 1, 1 1 2)’
POINT ZM → ‘POINTZM(0 0 100 1)’
MULTIPOINT ZM → ‘MULTIPOINTZM(0 0 100 1, 1 1 123 2)’
LINESTRING → ‘LINESTRING(0 0, 1 0, 1 1 , 0 1)’
MULTILINESTRING → ‘MULTILINESTRING((0 0, 1 0), (1 1 , 0 1))’
LINESTRING Z → ‘LINESTRINGZ(0 0 100, 1 0 100, 1 1 100 , 0 1 100)’
MULTILINESTRING Z → ‘MULTILINESTRINGZ((0 0 100, 1 0 100), (1 1 100, 0 1 100))’
LINESTRING M → ‘LINESTRINGM(0 0 1, 1 0 2, 1 1 2 , 0 1 1)’
MULTILINESTRING M → ‘MULTILINESTRINGM((0 0 1, 1 0 2), (1 1 2, 0 1 1))’
LINESTRING ZM → ‘LINESTRINGZM(0 0 100 1, 1 0 100 2, 1 1 100 2, 0 1 100 1)’
MULTILINESTRING ZM → ‘MULTILINESTRINGZM((0 0 100 1, 1 0 100 2), (1 1 100 2, 0 1 100 1))’
POLYGON → ‘POLYGON((0 0,1 0,1 1,0 1,0 0))’
MULTIPOLYGON → ‘MULTYPOLYGON(((0 0,1 0,1 1,0 1,0 0)),((-1 -1,2 -1,2 2, -1 2, -1 -1)))’
POLYGON Z → ‘POLYGONZ((0 0 100,1 0 100,1 1 100,0 1 100,0 0 100))’
MULTIPOLYGON Z → ‘MULTYPOLYGONZ(((0 0 100,1 0 100,1 1 100,0 1 100,0 0 100)),((-1 -1 100,2 -1 100,2 2 100, -1 2 100, -1 -1 100)))’

POLYGON M → ‘POLYGONM((0 0 1,1 0 2,1 1 2,0 1 2,0 0 1))’
MULTIPOLYGON M → ‘MULTYPOLYGONZM(((0 0 100 1,1 0 100 2,1 1 100 2,0 1 100 1,0 0 100 1)),((-1 -1 100 1,2 -1 100 1,2 2 100 1, -1 2 100 1, -1 -1 100 1)))’
POLYGON ZM → ‘POLYGONZM((0 0 100 1,1 0 100 2,1 1 100 2,0 1 100 2,0 0 100 1))’
MULTIPOLYGON ZM → ‘MULTYPOLYGONZ(((0 0 100,1 0 100,1 1 100,0 1 100,0 0 100)),((-1 -1 100,2 -1 100,2 2 100, -1 2 100, -1 -1 100)))’
GEOMETRY COLLECTION → GEOMETRYCOLLECTION(POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)),POINT(0 0))

Vamos criar uma tabela chamada pontos e adicionar uma geometria ponto nela;

```
geobanco=# CREATE TABLE pontos(gid serial PRIMARY KEY, geom geometry(POINTZ,4326), atributo varchar(30));
geobanco=# INSERT INTO pontos VALUES(1,ST_GeomFromText('POINTZ(0 51.49 57)', 4326),'Greenwich');
geobanco=# select ST_asText(geom),atributo from pontos;
      st_astext       | atributo
----------------------+-----------
 POINT Z (0 51.49 57) | Greenwich
(1 row)
```

A visão geometry_columns agora apresenta a seguinte entrada:

```
geobanco=# select * from geometry_columns;
 f_table_catalog | f_table_schema | f_table_name | f_geometry_column | coord_dimension | srid | type
-----------------+----------------+--------------+-------------------+-----------------+------+-------
 geobanco        | public         | pontos       | geom              |               3 | 4326 | POINT
(1 row)
```

Finalizamos saindo do ambiente psql e removendo o banco de dados usando:

```
$ dropdb geobanco
```

No escopo deste volume essa é a introdução de SQL e postgis. Postgis é extremamento completo para lidar com dados espaciais e aconselho a leitura de http://postgis.net/stuff/postgis-3.0.0.pdf para aqueles que queiram se aprofundar no assunto.

Neste volume faremos a transações com PostgreSQL/postgis usando R e, a medida que vamos avançando, explicações sobre o que é adicionado no banco de dados serão feitas.

PARTE 5 – Criando o Banco de Dados

5.1 – Criando banco de dados geoespacial

5.1.1 – O banco de dados

A primeira coisa que vamos fazer é montar nosso banco de dados com elementos mínimos para o funcionamento, ou seja, o banco de e extensões.

Tabelas serão implementadas e nos permitirão bastante flexibilidade ao se tornarem um banco base para análises espaciais de dados geológicos.

As extensões postgis_raster e postgis criarão algumas tabelas adicionais para dar suporte a elementos do tipo raster e vector. Falaremos mais delas no decorrer deste guia. Execute os comandos abaixo no terminal monitor criar o banco de dados e sua extensões. Note que fazemos isso sem entrar no ambiente psql usando as diretivas `-d` para o nome do banco de dados e `- c` para o comando que é passado entre aspas.

```
$ createdb geobanco --encoding=utf-8
$ psql -d geobanco -c "CREATE EXTENSION postgis;"
$ psql -d geobanco -c "CREATE EXTENSION postgis_raster;"
$ psql -d geobanco -c "ALTER DATABASE geobanco SET
postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';"
```

Nosso banco de dados geoespacial acaba de ser criado. Nas seções seguintes veremos como interagir e colocar dados que serão usados no decorrer deste volume.

5.1.2 – Sistema de Coordenadas do banco de dados

Um aspecto importante em se usar um banco de dados geoespacial é que sempre temos que estarmos atentos a qual sistema de referência de coordenadas (CRS) usaremos. Sempre utilize um único sistema de referência e de preferência use o WGS-84 sempre que possível.

5.2 – Carregando dados geoespaciais no geobanco de dados

5.2.1 – Dados vector

Usando o pacote rgdal podemos abrir diversos formatos do tipo vector e raster e com o pacote rpostgis vamos inserir ou extrair esses objetos no geobanco.

Importando arquivos GIS para o geobanco usando rgdal.

Importando um objeto vector do tipo ponto:

```
> library(rgdal)
> library(raster)
> poços.petróleo<-readOGR(dsn='.',layer='monitor_20190917')
> poços.petróleo
class       : SpatialPointsDataFrame
features    : 29836
extent      : -73.37677, -34.82621, -32.92657, 4.528003  (xmin, xmax, ymin,
ymax)
crs         : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

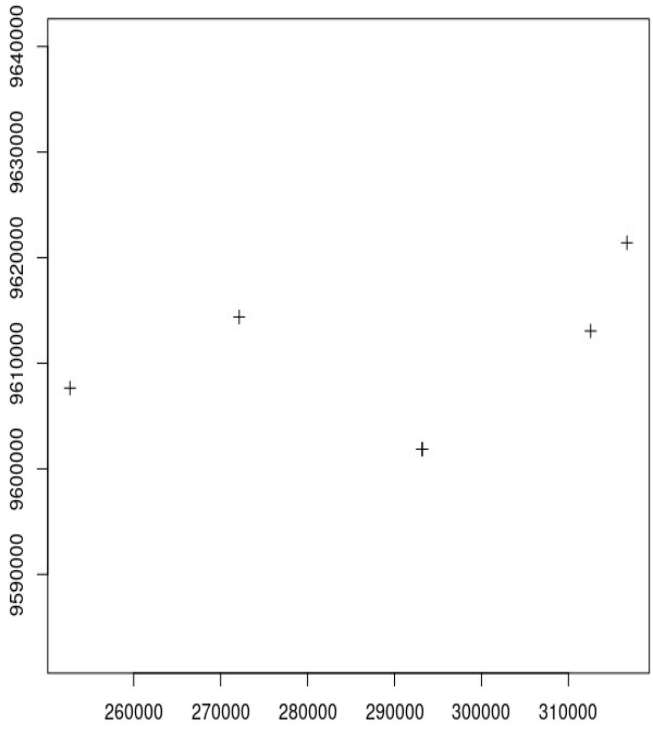
```
variables   : 23
names       : po_o_opera,      bacia, bloco,    campo,    ativo,       fase,
ambiente, lda,           operador,  po_o_anp, cartodb_id, objetivo,
tipo, repeti_es,     in_cio, ...
min values  :  1163IIIBA,       Acre,     A,  ABALONE,  ABALONE, Des./Prod.,
MAR, 0.0,              Alcom,     1A1BA,          1,    ERRO!, Direcional,
Repetido, 1922/01/01, ...
max values  :  WWFRB001D, Tucano Sul,    XA, XERELETE, XERELETE, Exploração,
TERRA, nan, Wintershall BM-S-14, 9XRL1DRJS,      29877, Produção,   Vertical,
Único, 2019/09/12, ...
```

Reprojetando para UTM WGS84 (SRID 32721):

```
> poços.repro <- spTransform(poços.petróleo, CRS("+init=epsg:32721"))
> poços.repro
class       : SpatialPointsDataFrame
features    : 29836
extent      : -1328101, 3006395, 6345429, 10504165  (xmin, xmax, ymin, ymax)
crs         : +init=epsg:32721 +proj=utm +zone=21 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 23
names       : po_o_opera,      bacia, bloco,    campo,    ativo,       fase,
ambiente, lda,           operador,  po_o_anp, cartodb_id, objetivo,
tipo, repeti_es,     in_cio, ...
min values  :  1163IIIBA,       Acre,     A,  ABALONE,  ABALONE, Des./Prod.,
MAR, 0.0,              Alcom,     1A1BA,          1,    ERRO!, Direcional,
Repetido, 1922/01/01, ...
max values  :  WWFRB001D, Tucano Sul,    XA, XERELETE, XERELETE, Exploração,
TERRA, nan, Wintershall BM-S-14, 9XRL1DRJS,      29877, Produção,   Vertical,
Único, 2019/09/12, ...
```

Recortando para uma extensão pré definida e plotando:

```
> caixa<-extent(250000,320000,9601000,9625000)
> poços.final <- crop(poços.repro,caixa)
> plot(poços.final, axes=TRUE)
```



Agora temos nosso objeto reprojetado e recortado para uma extensão qualquer. Vamos dar prosseguimento carregando este objeto vector do tipo ponto no geobanco.

Carregamos agora o objeto vector (poços.final) no banco de dados como uma tabela (pocos) da seguinte maneira (substitua ‘você’ e ‘segredo’ pelo seu nome de usuário e senha):

```
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> pgInsert(con, name = c("public", "pocos"), data.obj = poços.final)
```

Se entrarmos no banco de dados com psql veremos:

```
geobanco=# \d
             List of relations
 Schema |       Name        | Type  | Owner
--------+-------------------+-------+-------
 public | geography_columns | view  | andre
 public | geometry_columns  | view  | andre
 public | pocos             | table | andre
 public | raster_columns    | view  | andre
 public | raster_overviews  | view  | andre
 public | spatial_ref_sys   | table | andre
(6 rows)
geobanco=# \d pocos
                    Table "public.pocos"
   Column   |         Type          | Collation | Nullable | Default
------------+-----------------------+-----------+----------+------
 po_o_opera | text                  |           |          |
 bacia      | text                  |           |          |
 bloco      | text                  |           |          |
 campo      | text                  |           |          |
 ativo      | text                  |           |          |
 fase       | text                  |           |          |
 ambiente   | text                  |           |          |
 lda        | text                  |           |          |
 operador   | text                  |           |          |
 po_o_anp   | text                  |           |          |
 cartodb_id | integer               |           |          |
 objetivo   | text                  |           |          |
 tipo       | text                  |           |          |
 repeti_es  | text                  |           |          |
 in_cio     | text                  |           |          |
 t_rmino    | text                  |           |          |
 conclus_o  | text                  |           |          |
 profundida | double precision      |           |          |
 profundi_1 | integer               |           |          |
 profundi_2 | double precision      |           |          |
 sonda      | text                  |           |          |
 latitude   | double precision      |           |          |
 longitude  | double precision      |           |          |
 geom       | geometry(Point,32721) |           |          |
geobanco=# select po_o_anp,fase,objetivo,tipo from pocos;
  po_o_anp  |    fase    | objetivo |   tipo
------------+------------+----------+----------
 9FZ5AAM    | Exploração | Especial | Vertical
 9FZ5AM     | Exploração | Especial | Vertical
 9PA1AAM    | Exploração | Especial | Vertical
 9PA2AM     | Exploração | Especial | Vertical
 1LIT1AM    | Exploração | Pioneiro | Vertical
 1BRSA112AM | Exploração | Pioneiro | Vertical
(6 rows)
```

Agora vamos Importar um objeto vector do tipo linha:

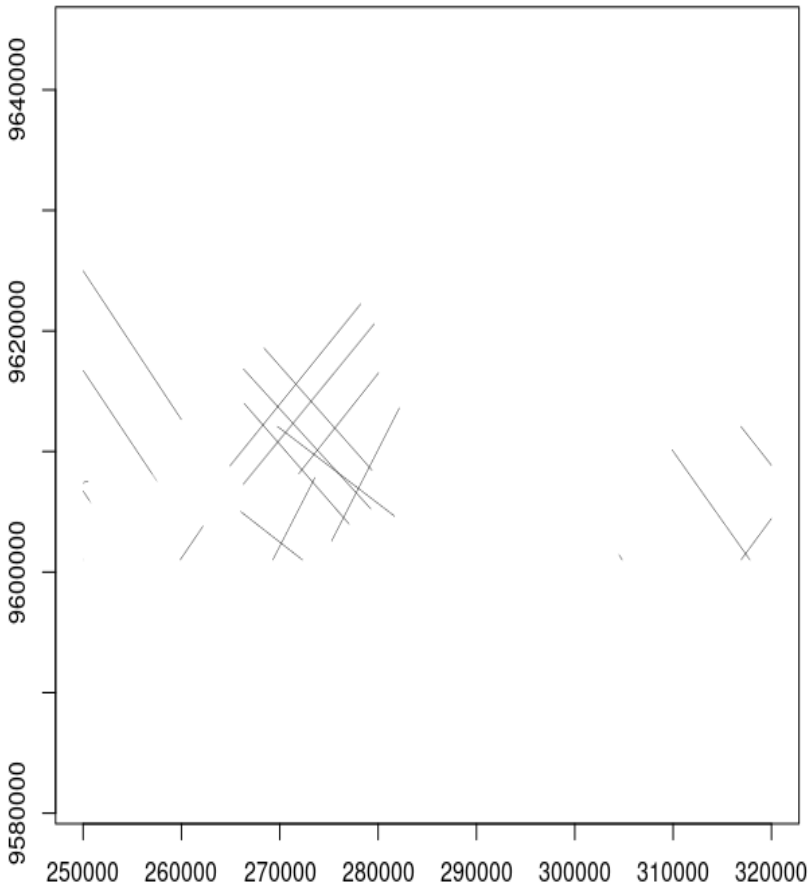
```
> library(rgdal)
> library(raster)
> sísmica2dpos<-readOGR(dsn='.',layer='2dposc')
> sísmica2dpos
class       : SpatialLinesDataFrame
features    : 140
extent      : -72, -55, -10, 0.7248806  (xmin, xmax, ymin, ymax)
crs         : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 27
names       :             SURV_NAME,             PJOB_NAME, PROC_TYPE,
PROC_DATE, SOURCE, COMPANY,         JOBCOMP_S,    PJOB_S, SURV_TYPE,    GEOM_S,
GEOM_NAME,   KIND, COUNTRY, NUMLINES, LOADDATE, ...
min values  : 0026_2D_ITANHAUA_JUMA, 0026_2D_ITANHAUA_JUMA,   MIG FIN, 1989-04-
15,     NA,  PUBLIC, 15715521418523971, 157155214,        2D, 157152905,
0026_ITANHAUA_JUMA,    CMP,  BRASIL,        1, 20180122, ...
max values  :   R0212_IGARAPE_MARIA,   R0212_IGARAPE_MARIA,   STK FIN, 2015-06-
15,     NA,  PUBLIC,  9440840218523971,  94408402, 2D Detail,  63607362,
R0212_IGARAPE_MARIA, Source,  BRASIL,        9, 20180122, ...
```

Reprojetando para WGS84 Zona 21S (SRID 32721):

```
> sis2d.repro <- spTransform(sísmica2dpos, CRS("+init=epsg:32721"))
> sis2d.repro
class       : SpatialLinesDataFrame
features    : 140
extent      : -1182576, 722496.8, 8870054, 10080423  (xmin, xmax, ymin, ymax)
crs         : +init=epsg:32721 +proj=utm +zone=21 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 27
names       :             SURV_NAME,             PJOB_NAME, PROC_TYPE,
PROC_DATE, SOURCE, COMPANY,         JOBCOMP_S,    PJOB_S, SURV_TYPE,    GEOM_S,
GEOM_NAME,   KIND, COUNTRY, NUMLINES, LOADDATE, ...
min values  : 0026_2D_ITANHAUA_JUMA, 0026_2D_ITANHAUA_JUMA,   MIG FIN, 1989-04-
15,     NA,  PUBLIC, 15715521418523971, 157155214,        2D, 157152905,
0026_ITANHAUA_JUMA,    CMP,  BRASIL,        1, 20180122, ...
max values  :   R0212_IGARAPE_MARIA,   R0212_IGARAPE_MARIA,   STK FIN, 2015-06-
15,     NA,  PUBLIC,  9440840218523971,  94408402, 2D Detail,  63607362,
R0212_IGARAPE_MARIA, Source,  BRASIL,        9, 20180122, ...
```

Recortando para uma extensão pré definida e plotando:

```
> caixa<-extent(250000,320000,9601000,9625000)
> sis2d.final <- crop(sis2d.repro,caixa)
> plot(sis2d.final, axes=TRUE,lwd=0.2)
```

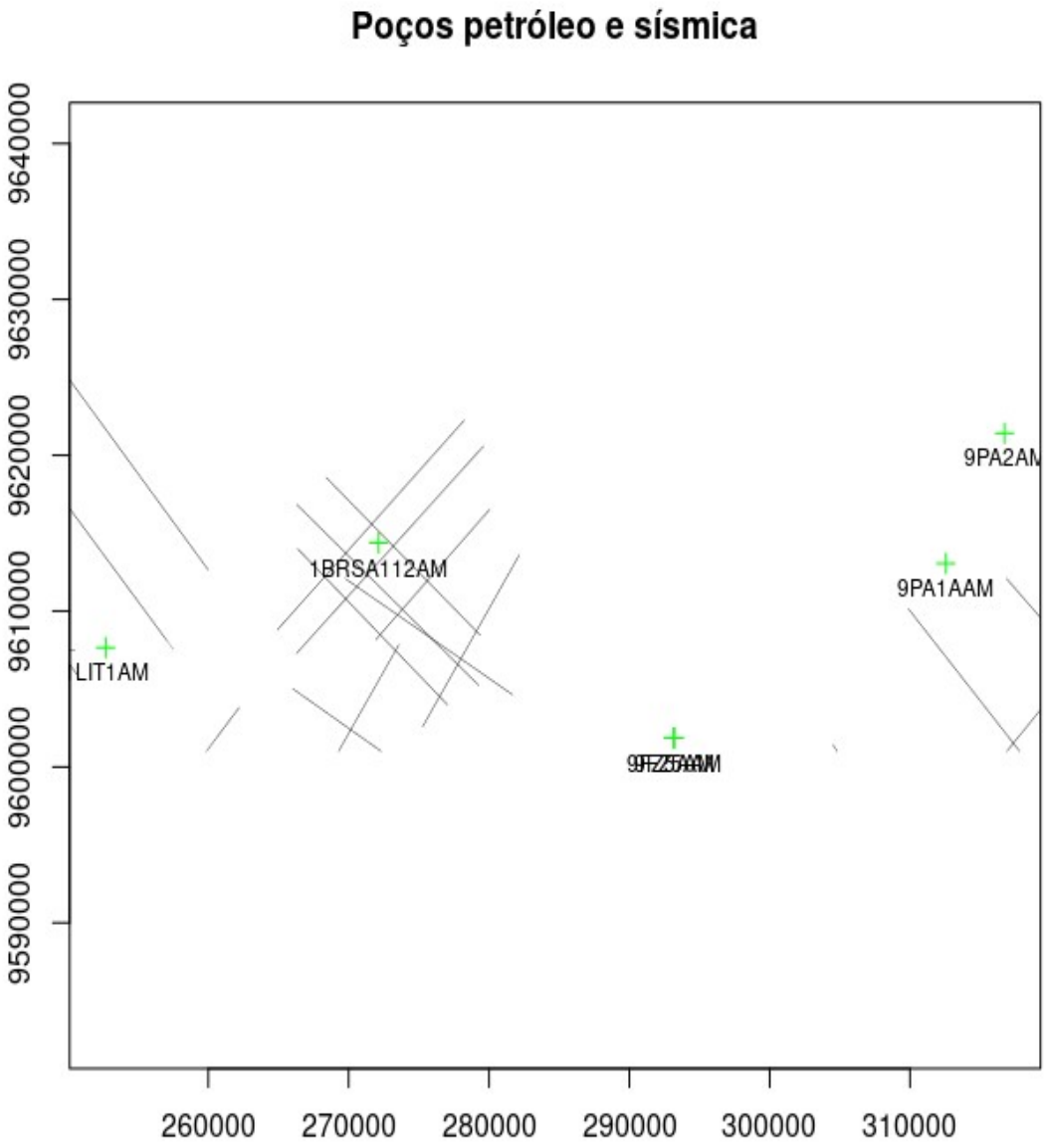


Carregamos agora o objeto vector (sis2d.final) no banco de dados como uma tabela (sis2d) da seguinte maneira:

```
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> pgInsert(con, name = c("public", "sis2d"), data.obj = sis2d.final)
```

Agora que temos todos os objetos no geobanco vamos mostrar como extrair eles e plotar em um mapa:

```
> library(rpostgis)
> library(raster)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> sis2d<-pgGetGeom(con,c('public','sis2d'))
> poços<-pgGetGeom(con,c('public','pocos'))
> plot(poços, axes=TRUE,col='green', main='Poços petróleo e sísmica')
> plot(sis2d,lwd=0.2,add=T)
> text(poços,poços$po_o_anp,pos=1,cex=0.75)
```



Bem simples a manipulação de dados vector e como usar eles a partir do geobanco. Vamos agora ver como utilizar objetos do tipo raster.

5.2.2 – Dados raster

Da mesma forma que fizemos com objetos vector acima vamos fazer com imagem raster. Vamos carregar imagens raster, recortar e reprojetar para depois carregar no geobanco. Depois vamos ver como extraímos estas imagens do banco de dados de volta para o ambiente R.

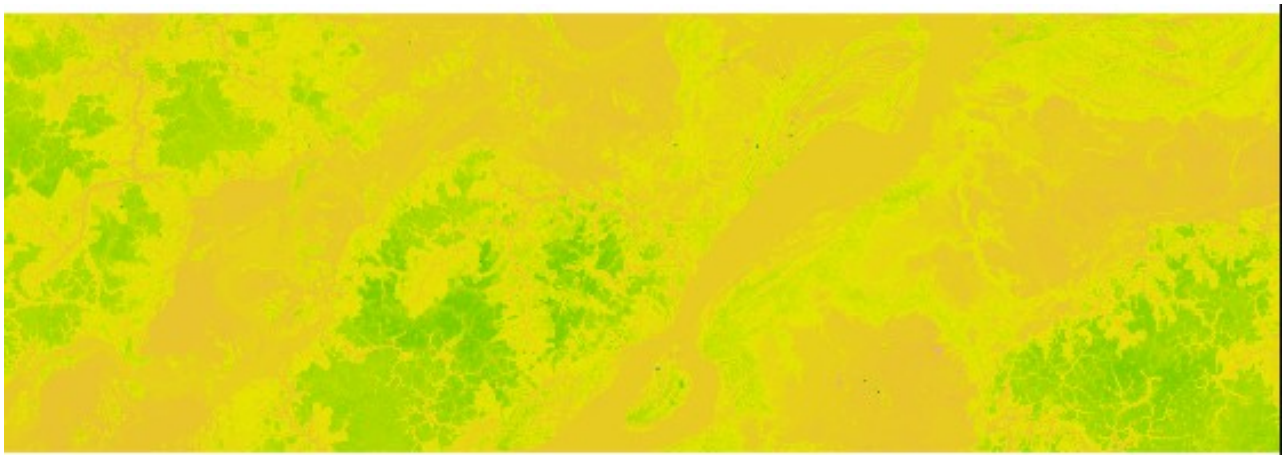
Carregando a primeira imagem raster (Imagem de satélite Sentinel2):

```
> library(raster)
Carregando pacotes exigidos: sp
> img<-brick('TCI.tif')
> img
class      : RasterBrick
dimensions : 2400, 7000, 16800000, 3  (nrow, ncol, ncell, nlayers)
resolution : 10, 10  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
```

```
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : /home/andre/livroR/TCI.tif
names      : TCI.1, TCI.2, TCI.3
min values :     0,     0,     0
max values :   255,   255,   255
> plotRGB(img,stretch='lin')
```

Carregando a segunda imagem raster (DEM ALOS alta resolução):

```
> library(raster)
> dem<-raster('DEM.tif')
> dem
class      : RasterLayer
dimensions : 778, 2270, 1766060  (nrow, ncol, ncell)
resolution : 30.837, 30.84833  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : /home/andre/livroR/DEM.tif
names      : DEM
values     : -32768, 32767  (min, max)
> plot(dem,axes=F)
```



Carregando as bandas 8 e 4 do Sentinel2:

```
> library(raster)
> b4<-raster('B4.tif')
> b8<-raster('B8.tif')
```

Vamos agora carregar estes objetos (img, dem, b4 e b8) no nosso geobanco em tabelas de nome tci,b4, b8 e dem (demora um pouco, 8 minutos, devido a área e resolução das imagens):

```
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> pgWriteRast(con, c("public", "tci"), img)
> pgWriteRast(con, c("public", "dem"), dem)
> pgWriteRast(con, c("public", "b4"), b4)
> pgWriteRast(con, c("public", "b8"), b8)
```

E agora lendo os rasters do geobanco para objetos no ambiente R:

```
> library(RPostgreSQL)
> library(rpostgis)
> library(raster)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> imgdb<-pgGetRast(con,"tci",bands=TRUE)
> imgdb
class      : RasterStack
dimensions : 2400, 7000, 16800000, 3  (nrow, ncol, ncell, nlayers)
resolution : 10, 10  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
names      : TCI.1, TCI.2, TCI.3
min values :    19,    35,    50
max values :   255,   255,   255
> demdb<-pgGetRast(con,"dem",bands=TRUE)
> demdb
class      : RasterLayer
dimensions : 778, 2270, 1766060  (nrow, ncol, ncell)
resolution : 30.837, 30.84833  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : memory
names      : DEM
values     : -78, 115  (min, max)
> b4db<-pgGetRast(con,"b4",bands=TRUE)
> b4db
class      : RasterLayer
dimensions : 2400, 7000, 16800000  (nrow, ncol, ncell)
resolution : 10, 10  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : memory
names      : B4
values     : 267, 13697  (min, max)
> b8db<-pgGetRast(con,"b8",bands=TRUE)
> b8db
class      : RasterLayer
dimensions : 2400, 7000, 16800000  (nrow, ncol, ncell)
resolution : 10, 10  (x, y)
extent     : 250000, 320000, 9601000, 9625000  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=21 +south +datum=WGS84 +units=m +no_defs
```

```
+ellps=WGS84 +towgs84=0,0,0
source     : memory
names      : B8
values     : 1, 13623  (min, max)
```

Checando agora o banco de dados e as visões (views) temos:

```
geobanco=# \d
                List of relations
 Schema |        Name          |   Type   | Owner
--------+----------------------+----------+-------
 public | b4                   | table    | andre
 public | b4_rid_seq           | sequence | andre
 public | b8                   | table    | andre
 public | b8_rid_seq           | sequence | andre
 public | dem                  | table    | andre
 public | dem_rid_seq          | sequence | andre
 public | geography_columns    | view     | andre
 public | geometry_columns     | view     | andre
 public | pocos                | table    | andre
 public | raster_columns       | view     | andre
 public | raster_overviews     | view     | andre
 public | sis2d                | table    | andre
 public | spatial_ref_sys      | table    | andre
 public | tci                  | table    | andre
 public | tci_rid_seq          | sequence | andre
(15 rows)
```

```
geobanco=# select * from geometry_columns
 f_table_catalog | f_table_schema | f_table_name | f_geometry_column | coord_dimension |
srid  |      type
-----------------+----------------+--------------+-------------------+-----------------
+-------+-----------------
 geobanco        | public         | pocos        | geom              |               2 |
32721 | POINT
 geobanco        | public         | sis2d        | geom              |               2 |
32721 | MULTILINESTRING
(2 rows)
geobanco=# select * from raster_columns;
 r_table_catalog | r_table_schema | r_table_name | r_raster_column | srid  |    scale_x
|    scale_y     | blocksize_x | blocksize_y | same_alignment | regular_blocking |
num_bands |     pixel_types     | nodata_values | out_db  |
extent
| spatial_index
-----------------+----------------+--------------+-----------------+-------
+--------------+----------------+------------+-------------+----------------
+-----------------+-----------+--------------------+----------------+---------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
------------------+---------------
 geobanco        | public         | tci          | rast            | 32721 |
10 |           -10 |         100 |         100 | t               | f               |
3 | {32BUI,32BUI,32BUI} | {0,0,0}       | {f,f,f} |
0103000020D17F0000010000000500000000000000080840E4100000000FD4F62410000000080840E410000000
0B55B624100000000008813410000000B55B624100000000008813410000000FD4F62410000000080840E41
00000000FD4F6241 | t
 geobanco        | public         | dem          | rast            | 32721 |
30.8370044053 | -30.8483290488 |         100 |         100 | t               | f
|         1 | {32BSI}             | {-99999}      | {f}     |
0103000020D17F0000010000000500000000000000080840E4112000000FD4F62410000000080840E410000000
0B55B624114020000008813410000000B55B624114020000008813411200000FD4F62410000000080840E41
12000000FD4F6241 | t
 geobanco        | public         | b4           | rast            | 32721 |
10 |           -10 |         100 |         100 | t               | f               |
1 | {32BUI}             | {0}           | {f}     |
0103000020D17F0000010000000500000000000000080840E4100000000FD4F62410000000080840E410000000
```

```
0B55B624100000000008813410000000B55B624100000000008813410000000FD4F624100000000080840E41
00000000FD4F6241 | t
 geobanco        | public         | b8           | rast            | 32721 |
10 |           -10 |         100 |         100 | t              | f               |
1 | {32BUI}             | {0}           | {f}     |
0103000020D17F000001000000050000000000000080840E4100000000FD4F624100000000080840E410000000
0B55B624100000000008813410000000B55B624100000000008813410000000FD4F624100000000080840E41
00000000FD4F6241 | t
(4 rows)
```

Como podem ver, os elementos espaciais adicionados estão presentes também nas ‘views’ de informação do geobanco.

Vamos agora ver como carregar outros formatos de dados espaciais no geobanco de dados.

5.3 – Dados de Campo em geral

Vamos agora falar sobre como armazenar no geobanco informações espaciais obtidas através de coleta de dados de subsuperfície (sondagens em geral), de geoquímica de amostras e geofísicas de superfície /poços.

O objetivo e ilustrar como o processo pode ser feito. Num geobanco real, o design do banco de dados , estruturação das tabelas e intercorrelação entre a informação é mais minuciosa.

Ao criarmos campos de geometrias nesta seção, estamos definindo um SRID. Usando 32721 que corresponde ao WGS-84 UTM Zona 21S.

5.3.1 – Dados de sondagem

Na indústria mineral e também na área de exploração de petróleo e gás temos a presença constante de dados oriundos de perfurações e sondagens.

Vamos aqui criar um modelo de tabelas para o armazenamento de informações oriundas de programas de sondagem em exploração mineral. A primeira tabela a ser criada será a dos cabeçalhos ou **collar** dos poços:

holeid	varchar(30)
c_projeto	varchar(30)
c_area	varchar(30)
c_target	varchar(30)
c_x	numeric(12,2)
c_y	numeric(12,2)
c_z	numeric(6,2)
c_srid	integer
c_az	numeric(5,2)
c_dip	numeric(5,2)
c_td	numeric(6,2)
c_rig	varchar(30)
c_drillco	varchar(30)
c_inicio	date
c_fim	date

Agora vamos criar uma tabela com a informação da perfilagem espacial (**survey**) destes poços:

holeid	varchar(30)
s_estacao	integer

s_x	numeric(12,2)
s_y	numeric(12,2)
s_depth	numeric(6,2)
s_az	numeric(5,2)
s_dip	numeric(5,2)
s_diam	varchar(30)
s_coment	text

Tabela dos dados de descrição (**log**) dos testemunhos destes poços:

holeid	varchar(30)
l_from	numeric(6,2)
l_to	numeric(6,2)
l_rcode	varchar(10)
l_desc	text
l_dia	varchar(30)
l_sample	varchar(30)
l_batch	varchar(30)

Uma tabela da descrição geotécnica (**geotec**) dos testemunhos:

holeid	varchar(30)
g_from	numeric(6,2)
g_to	numeric(6,2)
g_avc	numeric(6,2)
g_rec	numeric(4,2)
g_box	varchar(30)
g_sum10	numeric(4,2)
g_frac	integer

Uma tabela de resultados de geoquímica (**assay**):

holeid	varchar(30)
a_from	numeric(6,2)
a_to	numeric(6,2)
a_sample	varchar(30)
a_batch	varchar(30)
a_dispatch	varchar(30)
a_elem	varchar(10)
a_valor	numeric(9,3)
a_unidade	varchar(30)

Uma tabela com os resultados das amostras de checagem (**qaqc**):

holeid	varchar(30)
q_aliq	varchar(30)
q_sample	varchar(30)
q_batch	varchar(30)
q_dispatch	varchar(30)
q_elem	varchar(10)
q_valor	numeric(9,3)
q_unidade	varchar(30)

Uma tabela para os dados de geofísica de poço (**dhd**):

holeid	varchar(30)
d_curve	varchar(30)

d_depth	numeric(6,2)
d_value	numeric(12,2)

Criando essas tabelas no banco de dados geobanco:

```
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1",user="voce",password="segredo",
dbname="geobanco")
> sql<-"CREATE TABLE collar (holeid varchar(30) PRIMARY KEY,c_projeto
varchar(30) NOT NULL, c_area varchar(30) NOT NULL,c_target varchar(30) NOT
NULL,geom GEOMETRY(POINTZ,32721),c_srid integer NOT NULL,c_az numeric(5,2) NOT
NULL,c_dip numeric(5,2) NOT NULL,c_td numeric(6,2), c_rig varchar(30),
c_drillco varchar(30),c_inicio date,c_fim date);"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE survey (holeid varchar(30) NOT NULL,s_estacao integer NOT
NULL,s_x numeric(12,2) NOT NULL,s_y numeric(12,2) NOT NULL,s_depth numeric(6,2)
NOT NULL, s_az numeric(5,2) NOT NULL,s_dip numeric(5,2) NOT NULL,s_diam
varchar(30),s_coment text);"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE log (holeid varchar(30) NOT NULL, l_from numeric(6,2) NOT
NULL,l_to numeric(6,2) NOT NULL, l_rcode varchar(10), l_desc text, l_dia
varchar(30),l_sample varchar(30),l_batch varchar(30));"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE geotec (holeid  varchar(30) NOT NULL, g_from numeric(6,2)
NOT NULL,g_to numeric(6,2) NOT NULL,g_avc numeric(6,2), g_rec
numeric(4,2),g_box varchar(30), g_sum10 numeric(4,2),g_frac integer);"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE assay (holeid varchar(30) NOT NULL, a_from numeric(6,2)
NOT NULL,a_to numeric(6,2) NOT NULL,a_sample varchar(30) NOT NULL, a_batch
varchar(30) NOT NULL,a_dispatch varchar(30) NOT NULL,a_elem varchar(10) NOT
NULL,a_valor numeric(9,3),a_unidade varchar(30));"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE qaqc (holeid varchar(30) NOT NULL, q_aliq varchar(30) NOT
NULL, q_sample varchar(30) NOT NULL,q_batch varchar(30) NOT NULL, q_dispatch
varchar(30) NOT NULL, q_elem varchar(10) NOT NULL, q_valor numeric(9,3),
q_unidade varchar(30));"
> dbGetQuery(con,sql)
> sql<-"CREATE TABLE dhd (holeid varchar(30) NOT NULL, d_curve varchar(30) NOT
NULL,d_depth numeric(6,2) NOT NULL,d_value numeric(12,2));"
> dbGetQuery(con,sql)
> sql<-"CREATE INDEX ON dhd (holeid);"
> dbGetQuery(con,sql)
```

Inserindo dados

Uma vez criadas as tabelas, vamos inserir os dados nelas.
Inserindo os dados dos furos nas respectivas tabelas.

```
>  hd<-read.csv('collarA.csv',stringsAsFactors=FALSE)
> for (i in 1:14){
+ sql<-paste0("INSERT INTO collar (holeid,c_projeto,c_area, c_target,geom,
c_srid,c_az,c_dip,c_td,c_rig, c_drillco,c_inicio,c_fim)
VALUES('",hd[i,"holeid"], "','",hd[i,"projeto"], "','",hd[i,"area"],
"','",hd[i,"target"], "',ST_GeomFromText('POINT(",hd[i,"x"]," ",hd[i,"y"],"
",hd[i,"z"],")',32721)",",",hd[i,"srid"],",",hd[i,"az"],
",",hd[i,"dip"],",",hd[i,"td"],",'",hd[i,"rig"],"','",hd[i,"drillco"],"','",hd[
i,"inicio"],"','",hd[i,"fim"],"');")
+ dbSendQuery(con,sql)
+ }

> tmp<-read.csv('surveyA.csv',stringsAsFactors=FALSE)
```

```
> dbWriteTable(con, "survey", tmp, row.names=FALSE, append=TRUE)
> tmp<-read.csv('logA.csv',stringsAsFactors=FALSE)
> dbWriteTable(con, "log", tmp, row.names=FALSE, append=TRUE)
> tmp<-read.csv('geotechA.csv',stringsAsFactors=FALSE)
> dbWriteTable(con, "geotec", tmp, row.names=FALSE, append=TRUE)
> tmp<-read.csv('assayAL.csv',stringsAsFactors=FALSE)
> dbWriteTable(con, "assay", tmp, row.names=FALSE, append=TRUE)
> tmp<-read.csv('qaqcAL.csv',stringsAsFactors=FALSE)
> dbWriteTable(con, "qaqc", tmp, row.names=FALSE, append=TRUE)
```

Inserindo os dados LAS na tabela ‘dhd’.

```
> inserirLas<-function(archivo){
+ line<-""
+ arqui<-paste0(archivo,'.csv')
+ las.data<-read.csv(arqui)
+ cat(line, file="templas.csv", append=FALSE, sep = '')
+ for(i in 2:ncol(las.data)){
+  for(j in 1:nrow(las.data)){
+   line2<-paste0(' ',archivo,',',names(las.data)
[i],",",las.data[j,1],",",las.data[j,i])
+   cat(line2, file="templas.csv", append=TRUE, sep = "\n")
+  }
+ }
+ system("psql -c \"\\COPY dhd FROM 'templas.csv'
+ delimiter ',' csv NULL 'NA';\" -U user geobanco")
+ }
> inserirLas('PBAT-10-02')
> inserirLas('PBAT-10-05')
> inserirLas('PBAT-11-03')
> inserirLas('PBAT-11-07')
> inserirLas('PBAT-11-08')
> inserirLas('PBAT-11-09')
> inserirLas('PBAT-11-10')
> inserirLas('PBAT-11-11')
> inserirLas('PBAT-11-12')
> inserirLas('PBAT-12-13')
```

Todas as tabelas do geobanco:

```
geobanco=# \d
              List of relations
 Schema |       Name        |   Type   | Owner
--------+-------------------+----------+-------
 public | assay             | table    | andre
 public | b4                | table    | andre
 public | b4_rid_seq        | sequence | andre
 public | b8                | table    | andre
 public | b8_rid_seq        | sequence | andre
 public | collar            | table    | andre
 public | dem               | table    | andre
 public | dem_rid_seq       | sequence | andre
 public | dhd               | table    | andre
 public | geography_columns | view     | andre
 public | geometry_columns  | view     | andre
 public | geotec            | table    | andre
 public | log               | table    | andre
 public | pocos             | table    | andre
 public | qaqc              | table    | andre
 public | raster_columns    | view     | andre
 public | raster_overviews  | view     | andre
 public | sis2d             | table    | andre
```

```
 public | spatial_ref_sys   | table    | andre
 public | survey            | table    | andre
 public | tci               | table    | andre
 public | tci_rid_seq       | sequence | andre
(22 rows)
```

PARTE 6

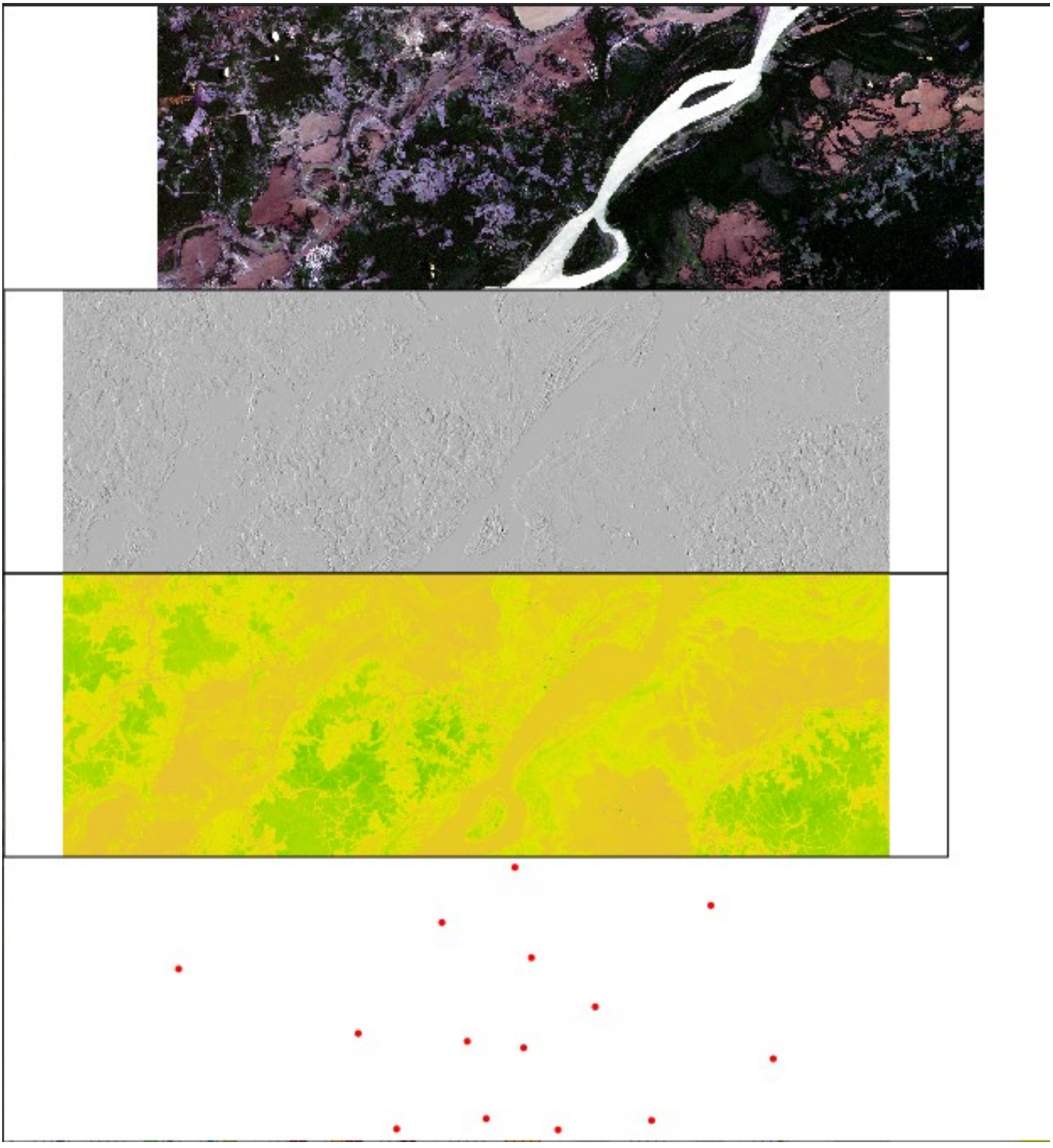
6.1 – Integração de dados geológicos e análise espacial

Vamos mostrar como é feita uma análise espacial de dados de exploração mineral com base nos dados adicionados na parte anterior e avaliação inicial de recurso.

6.1.1 - Dados na superfície

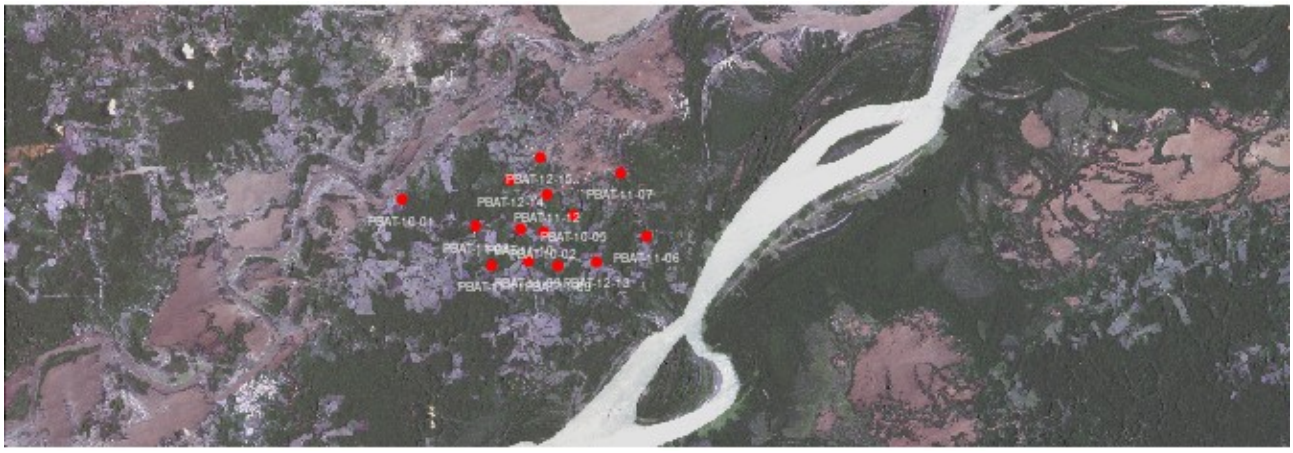
Visualizando dados de superfície tais como localização de furos, imagens de satélite, DEM, etc:

```
> library(raster)
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1",user="voce",password="segredo",
dbname="geobanco")
> dem<-pgGetRast(con,"dem")
> sent<-pgGetRast(con,"tci",bands=TRUE)
> b4<-pgGetRast(con,"b4")
> b8<-pgGetRast(con,"b8")
> sql<-"select st_x(geom) as x, st_y(geom) as y,st_z(geom) as z,
holeid, c_projeto,c_target,c_area,c_dip, c_az, c_td,c_rig,c_drillco,
c_inicio,c_fim from collar;"
> furos<-dbGetQuery(con,sql)
> coordinates(furos)<-~x+y+z
> gradient<-terrain(dem,opt='slope',unit='radians')
> aspect<-terrain(dem,opt='aspect',unit='radians')
> relevo<-hillShade(gradient,aspect,angle=45,direction=270,normalize= TRUE)
> par(mfrow=c(4,1))
> plotRGB(sent,r=3,g=2,b=1,stretch='lin')
> plot(relevo,axes=FALSE,col=grey(0:100/100),legend=FALSE)
> plot(dem,axes=FALSE,legend=FALSE)
> plot(furos,cex=1,pch=20,col='red')
```



Plotando informações sobrepostas gerando um mapa superficial:

```
> dev.off()
plotRGB(sent,r=3,g=2,b=1,stretch='lin')
plot(relevo,add=TRUE,alpha=0.4,col=grey(0:100/100),legend=FALSE)
plot(furos,add=TRUE,cex=1,pch=20,col='red')
text(furos,label=furos$holeid,pos=1,cex=0.4,col='white')
```



Visualizando NDVI e sombra de relevo. (Avaliação de terrenos, ambiental e acessos):

```
> ndvi<-(b8-b4)/(b8+b4)
> plot(ndvi,legend=FALSE,axes=FALSE,zlim=c(0,1))
> plot(relevo,add=TRUE,alpha=0.4,col=grey(0:100/100),legend=FALSE)
```

6.1.2 - Estruturando informação em espaço tridimensional

Vamos agora mostrar como colocar informações no contexto tridimensional. Para isso usaremos a informação do colar dos furos e a informação do perfilamento de desvio destes furos. Tratam-se de furos verticais ou bem próximo do vertical mas o método é demostrado sendo mais perceptível para furos inclinados.

Primeiro criamos dois data.frames que serão usadas para gerar a informação de coordenadas tridimensionais dentro de qualquer furo na profundidade informada.

Criando os dois data.frames (colar e estacoes):

```
> library(raster)
```

```
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1",user="voce",password="segredo",
dbname="geobanco")
> sql<-"select st_x(geom) as x, st_y(geom) as y,st_z(geom) as z,
holeid, c_projeto,c_target,c_area,c_dip, c_az, c_td,c_rig,c_drillco,
c_inicio,c_fim from collar;"
> furos2<-dbGetQuery(con,sql)
> colar<-data.frame(furo=furos2$holeid,prof=0,x=furos2$x,y=furos2$y,z=furos2$z,
mergulho=furos2$c_dip, az=furos2$c_az)
> sql<-"SELECT * FROM survey;"
> su<-dbGetQuery(con,sql)
> estacoes<-data.frame(furo=su$holeid,prof=su$s_depth,x=NA,y=NA,z=NA,
mergulho=su$s_dip,az=su$s_az)
> head(colar, n=1L)
       furo prof      x       y  z mergulho az
1 PBAT-10-01    0 271624 9614442 22      0  -90
> head(estacoes, n=1L)
       furo prof  x  y  z mergulho  az
1 PBAT-10-01    0 NA NA NA        0 -90
```

Os dois data.frames possuem as colunas idênticas. A data.frame colar contem informações tridimensionais da boca de cada furo e a data.frame contem informação de cada ponto de desvio perfilado em cada furo de sonda. As coordenadas 3D destes pontos ainda não foram calculadas.

Vamos calcular estas coordenadas usando a função abaixo:

```
> tresD<-function(da,es){
 linha<-data.frame()
 for(i in 1:nrow(da)){
  x<-da[i,3]
  y<-da[i,4]
  z<-da[i,5]
  di<-0
  fur<-es[es[[1]]==da$furo[[i]],]
  fur<-fur[order(fur$prof),]
  for(j in 1:nrow(fur)){
   angulo<-fur[j,7]
   d<-fur[j,2]-di
   di<-d+di
   deltaz<-abs(d*sin(fur[j,6]*pi/180))
   raio<-d*cos(fur[j,6]*pi/180)
   x<-round(x+raio*sin(angulo*pi/180))
   y<-round(y+raio*cos(angulo*pi/180))
   z<-round(z-deltaz)
   linha<-
rbind(linha,data.frame(furo=fur[j,1] ,prof=fur[j,2],x=x ,y=y ,z=z,mergulho=fur[
j,6],az=fur[j,7]))
   z<-z
  }
 }
 lin<-rbind(linha,da)
 lin<-lin[order(lin$furo,lin$prof),]
 return(lin)
}
```

E usando os dois data.frames como argumento da função tresD teremos:

```
> pts.XYZ<-tresD(colar,estacoes)
```

Com o data.frame criado pela função tresD podemos agora obter coordenadas 3D de qualquer ponto dentro de qualquer furo de sonda usando a função obterXYZ abaixo usando como argumento nosso data.frame acima criado, o ID do furo e a profundidade no furo:

```
> obterXYZ<-function(pontos,furo,profu){
  sete<-pontos[as.character(pontos$furo)==furo,]
  refa<-sete[sete$prof==max(sete[sete$prof<profu,]$prof),]
  ref<-refa[1,]
  d<-profu-ref$prof
  deltaz<-abs(d*sin(ref$mergulho*pi/180))
  raio<-d*cos(ref$mergulho*pi/180)
  z<-ref$z-deltaz
  angulo<-ref$az
  x<-ref$x+raio*sin(angulo*pi/180)
  y<-ref$y+raio*cos(angulo*pi/180)
  return(c(x,y,z))
 }
> obterXYZ(pts.XYZ,'PBAT-11-12',710)
[1]  279514 9614695    -685
```

Agora estamos prontos para botar em prática o posicionamento tridimensional de informações de furos de sondagem.

Vamos usar a informação de amostras para análises químicas do arquivo log e georreferenciar as mesmas em 3D.

Criando o data.frame *amostras* com a informação necessária:

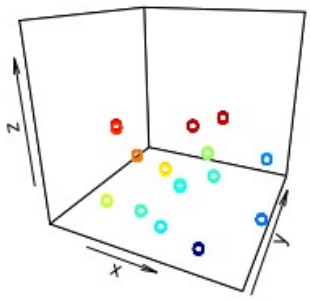
```
> sql<-"SELECT * FROM log"
> log<-dbGetQuery(con,sql)
> amostras<-data.frame(log[which( !is.na(log$l_sample)),cbind(1,2,3,7)],x=NA,
y=NA,z=NA)
> head(amostras,n=1L)
       holeid l_from   l_to l_sample  x  y  z
70 PBAT-10-01  763.1 763.68   152032 NA NA NA
```

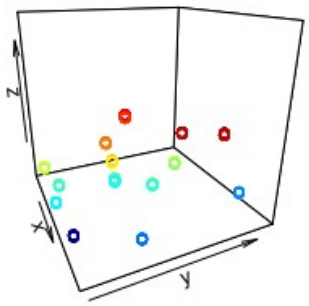
E finalmente calculando a localização central de cada amosta em espaço tridimensional:

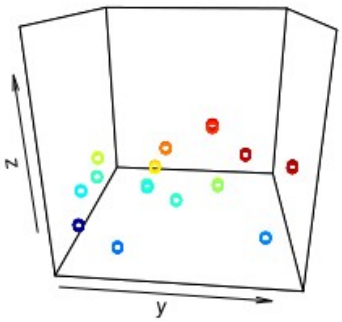
```
> for(i in 1:nrow(amostras)){
 amostras$x[i]<-obterXYZ(pts.XYZ,amostras$holeid[i],(amostras$l_to[i]-
           amostras$l_from[i])/2+amostras$l_from[i])[1]
 amostras$y[i]<-obterXYZ(pts.XYZ,amostras$holeid[i],(amostras$l_to[i]-
           amostras$l_from[i])/2+amostras$l_from[i])[2]
 amostras$z[i]<-obterXYZ(pts.XYZ,amostras$holeid[i],(amostras$l_to[i]-
           amostras$l_from[i])/2+amostras$l_from[i])[3]
}
```

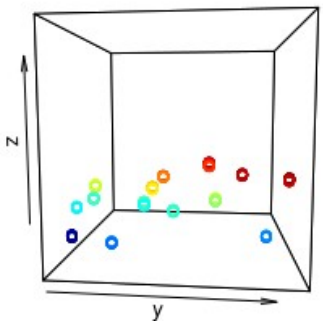
Vamos plotar as amostras agora usando

```
> library(plot3D)
> par(mfrow=c(2,2))
> scatter3D(amostras$x,amostras$y,amostras$z,theta=25,phi=20,colkey=FALSE,
zlim=c(-950,-300))
> scatter3D(amostras$x,amostras$y,amostras$z,theta=65,phi=20,colkey=FALSE,
zlim=c(-950,-300))
> scatter3D(amostras$x,amostras$y,amostras$z,theta=95,phi=20,colkey=FALSE,
zlim=c(-950,-300))
> scatter3D(amostras$x,amostras$y,amostras$z,theta=95,phi=5,colkey=FALSE, zlim=
c(-950,-300))
```









Agora que temos a localização tridimensional das amostras vamos carregar o resultado das análises químicas nesse data.frame.

Primeiro carregamos os dados:

```
> sql<-"SELECT * FROM assay;"
> smp<-dbGetQuery(con,sql)
> smp.largo<-reshape(smp,timevar='a_elem', idvar=c('holeid', 'a_from','a_to',
'a_sample', 'a_batch' , 'a_dispatch','a_unidade'),direction='wide')
> head(smp.largo,n=1L)
      holeid a_from   a_to a_sample        a_batch a_dispatch a_unidade
1 PBAT-10-01  763.1 763.68   152032 PBAT-10-01-002 G-2010-806   percent
  a_valor.k a_valor.mg a_valor.na a_valor.ca a_valor.so4 a_valor.cl a_valor.ri
1      0.18       0.02      34.28       0.89         2.3       51.5        8.1
  a_valor.tot a_valor.kcl a_valor.nacl a_valor.mgcl2 a_valor.caso4 a_valor.k2o
1       97.27        0.35        87.13          0.07          3.04        0.22
  a_valor.na2o a_valor.mgo a_valor.cao a_valor.br a_valor.k2o1 a_valor.na2o1
1         46.2        0.03        1.25          8           NA            NA
  a_valor.mgo1 a_valor.cao1 a_valor.s a_valor.moist
1           NA           NA        NA           1.7
```

E depois unimos esta data.frame com as amostras:

```
> ar<- merge(amostras, smp.largo, by.x='amostra', by.y= 'idamostra')
> head(ar,n=1L)
   l_sample   holeid.x l_from   l_to      x       y        z   holeid.y a_from
1   152001 PBAT-10-01 766.19 766.42 271624 9614442 -744.305 PBAT-10-01 766.19
    a_to        a_batch a_dispatch a_unidade a_valor.k a_valor.mg a_valor.na
1 766.42 PBAT-10-01-001 G-2010-624   percent      0.43       0.01      37.99
  a_valor.ca a_valor.so4 a_valor.cl a_valor.ri a_valor.tot a_valor.kcl
```

```
1       0.53         1.4       57.5        1.7       99.56        0.82
  a_valor.nacl a_valor.mgcl2 a_valor.caso4 a_valor.k2o a_valor.na2o a_valor.mgo
1        96.56          0.05           1.8        0.52         51.2        0.02
  a_valor.cao a_valor.br a_valor.k2o1 a_valor.na2o1 a_valor.mgo1 a_valor.cao1
1        0.74          8           NA            NA           NA           NA
  a_valor.s a_valor.moist
1        NA          0.1
```

6.1.3 Recursos Minerais usando poligonais

Neste formato os dados estão prontos serem analisados espacialmente. Vamos ver um simples exemplo a seguir usando poligonais para calculo de recursos. Vamos criar uma tabela com o intervalo onde o teor médio total de cada furo respeita um cut-off de 5% Kcl. Primeiro criando um data.frame com todos os furos e com teores zerados e outro com os furos com teor de KCl > 5% que substituirá o valor destes do data.frame zerado original:

```
> library(data.table)
> res.furo<-setDT(ar[which( ar$a_valor.kcl>=0),])[,.(x=mean(x),y=mean(y),
z=mean(z), esp.total=max(a_to)-min(a_from),esp.minerio=sum(a_to-a_from),
k=sum(a_valor.k*(a_to-a_from)/sum(a_to-a_from)),kcl=sum(a_valor.kcl*(a_to-
a_from)/sum(a_to-a_from)), k2o=sum(a_valor.k2o*(a_to-a_from)/sum(a_to-
a_from)),ri=sum(a_valor.ri*(a_to-a_from)/sum(a_to-a_from))), by=holeid.x]
> res.furo
      holeid.x      x       y         z esp.total esp.minerio        k
 1: PBAT-10-01 271624 9614442 -750.7018     18.00       18.00 0.4961056
 2: PBAT-10-02 279340 9612680 -819.7221     13.89       13.89 2.8186969
 3: PBAT-10-05 280941 9613595 -819.2750      8.15        8.15 0.5000245
 4: PBAT-11-07 283527 9615862 -844.1619      5.71        5.71 0.2111384
 5: PBAT-11-06 284922 9612434 -847.3917      6.15        6.15 0.1143089
 6: PBAT-11-08 275640 9612999 -766.9870      8.05        8.05 0.1452174
 7: PBAT-11-09 278503 9611094 -817.0330      7.47        7.47 5.3082597
 8: PBAT-11-03 280109 9610844 -826.5229      5.91        5.91 3.5108291
 9: PBAT-11-10 278079 9612824 -783.4813      9.60        9.60 0.8774688
10: PBAT-11-11 276498 9610863 -795.2580      7.87        7.87 0.1048666
11: PBAT-11-12 279514 9614695 -800.9001     11.41       11.41 4.0285802
12: PBAT-12-13 282202 9611053 -882.1034      8.48        8.48 0.4192571
13: PBAT-12-14 277513 9615481 -740.0872      8.00        8.00 0.2240500
14: PBAT-12-15 279144 9616715 -738.9640     10.51       10.51 3.8773454
          kcl       k2o        ri
 1:  0.9463778 0.6017833 2.5332778
 2:  5.3742981 3.3956803 1.4159827
 3:  0.9526258 0.6025153 2.4042945
 4:  0.4011909 0.2544133 0.5714536
 5:  0.2174309 0.1372846 1.0626016
 6:  0.2761491 0.1751677 0.4942857
 7: 10.1229853 6.3949665 1.5859438
 8:  6.6958037 4.2298985 2.6553299
 9:  1.6749063 1.0578646 1.1578125
10:  0.1967980 0.1258069 0.8509530
11:  7.6855127 4.8547853 1.6215600
12:  0.7995401 0.5058373 1.3137972
13:  0.4245375 0.2684625 0.8173750
14:  7.3966508 4.6724643 3.0641294
> res.furo[,5:10]<-0
> res.furo
     holeid.x      x       y         z esp.total esp.minerio k kcl k2o ri
 1: PBAT-10-01 271624 9614442 -750.7018         0           0 0   0   0  0
 2: PBAT-10-02 279340 9612680 -819.7221         0           0 0   0   0  0
 3: PBAT-10-05 280941 9613595 -819.2750         0           0 0   0   0  0
 4: PBAT-11-07 283527 9615862 -844.1619         0           0 0   0   0  0
 5: PBAT-11-06 284922 9612434 -847.3917         0           0 0   0   0  0
 6: PBAT-11-08 275640 9612999 -766.9870         0           0 0   0   0  0
```

```
 7: PBAT-11-09 278503 9611094 -817.0330         0           0 0   0   0  0
 8: PBAT-11-03 280109 9610844 -826.5229         0           0 0   0   0  0
 9: PBAT-11-10 278079 9612824 -783.4813         0           0 0   0   0  0
10: PBAT-11-11 276498 9610863 -795.2580         0           0 0   0   0  0
11: PBAT-11-12 279514 9614695 -800.9001         0           0 0   0   0  0
12: PBAT-12-13 282202 9611053 -882.1034         0           0 0   0   0  0
13: PBAT-12-14 277513 9615481 -740.0872         0           0 0   0   0  0
14: PBAT-12-15 279144 9616715 -738.9640         0           0 0   0   0  0
> res.furo2<-setDT(ar[which( ar$a_valor.kcl>=5),])[,.(x=mean(x), y=mean(y),
z=mean(z),esp.total=max(a_to)-min(a_from),esp.minerio=sum(a_to-a_from),
k=sum(a_valor.k*(a_to-a_from)/sum(a_to-a_from)),kcl=sum(a_valor.kcl*(a_to-
a_from)/sum(a_to-a_from)), k2o=sum(a_valor.k2o*(a_to-a_from)/sum(a_to-
a_from)),ri=sum(a_valor.ri*(a_to-a_from)/sum(a_to-a_from))), by=holeid.x]
> res.furo2
     holeid.x      x       y         z esp.total esp.minerio        k      kcl
1: PBAT-10-02 279340 9612680 -815.6178      1.66        1.66 18.84602 35.94229
2: PBAT-10-05 280941 9613595 -818.4500      0.44        0.44  3.42500  6.53500
3: PBAT-11-09 278503 9611094 -815.9675      1.82        1.82 20.94462 39.94500
4: PBAT-11-03 280109 9610844 -826.0021      1.37        1.37 14.04161 26.78394
5: PBAT-11-10 278079 9612824 -782.1225      0.43        0.43 10.42233 19.87279
6: PBAT-11-12 279514 9614695 -799.5696      2.07        2.07 20.46043 39.02338
7: PBAT-12-15 279144 9616715 -738.0264      2.06        2.06 17.03107 32.48296
        k2o        ri
1: 22.70663 3.5307229
2:  4.13000 1.2500000
3: 25.23473 2.3098901
4: 16.91942 2.0270073
5: 12.55349 0.4139535
6: 24.65169 0.5536232
7: 20.51990 4.1805825
> setDT(res.furo)[res.furo2, esp.total := i.esp.total , on =.(holeid.x)]
> setDT(res.furo)[res.furo2, esp.minerio := i.esp.minerio , on =.(holeid.x)]
> setDT(res.furo)[res.furo2, k := i.k , on =.(holeid.x)]
> setDT(res.furo)[res.furo2, kcl := i.kcl, on =.(holeid.x)]
> setDT(res.furo)[res.furo2, k2o:=i.k2o, on =.(holeid.x)]
> setDT(res.furo)[res.furo2, ri:=i.ri, on =.(holeid.x)]
> res.furo
     holeid.x      x       y         z esp.total esp.minerio        k      kcl
 1: PBAT-10-01 271624 9614442 -750.7018      0.00        0.00  0.00000  0.00000
 2: PBAT-10-02 279340 9612680 -819.7221      1.66        1.66 18.84602 35.94229
 3: PBAT-10-05 280941 9613595 -819.2750      0.44        0.44  3.42500  6.53500
 4: PBAT-11-07 283527 9615862 -844.1619      0.00        0.00  0.00000  0.00000
 5: PBAT-11-06 284922 9612434 -847.3917      0.00        0.00  0.00000  0.00000
 6: PBAT-11-08 275640 9612999 -766.9870      0.00        0.00  0.00000  0.00000
 7: PBAT-11-09 278503 9611094 -817.0330      1.82        1.82 20.94462 39.94500
 8: PBAT-11-03 280109 9610844 -826.5229      1.37        1.37 14.04161 26.78394
 9: PBAT-11-10 278079 9612824 -783.4813      0.43        0.43 10.42233 19.87279
10: PBAT-11-11 276498 9610863 -795.2580      0.00        0.00  0.00000  0.00000
11: PBAT-11-12 279514 9614695 -800.9001      2.07        2.07 20.46043 39.02338
12: PBAT-12-13 282202 9611053 -882.1034      0.00        0.00  0.00000  0.00000
13: PBAT-12-14 277513 9615481 -740.0872      0.00        0.00  0.00000  0.00000
14: PBAT-12-15 279144 9616715 -738.9640      2.06        2.06 17.03107 32.48296
        k2o        ri
 1:  0.00000 0.0000000
 2: 22.70663 3.5307229
 3:  4.13000 1.2500000
 4:  0.00000 0.0000000
 5:  0.00000 0.0000000
 6:  0.00000 0.0000000
 7: 25.23473 2.3098901
 8: 16.91942 2.0270073
 9: 12.55349 0.4139535
```

```
10:  0.00000 0.0000000
11: 24.65169 0.5536232
12:  0.00000 0.0000000
13:  0.00000 0.0000000
14: 20.51990 4.1805825
```

O resultado é um data.frame com os valores que respeitam o cut-off usado. Os valores abaixo do cut-off se torna zero.

Transformamos o nosso data.frame em um objeto SpatialPointDataFrame com CRS WGS-84 UTM Zone21 sul:

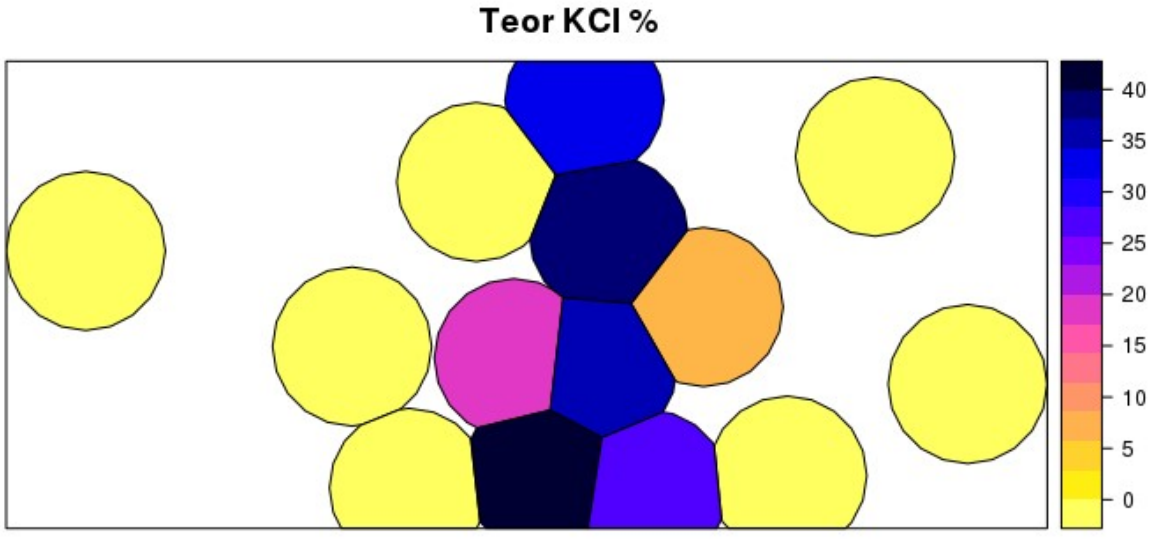
```
> coordinates(res.furo)<-~x+y+z
> crs(res.furo)<-CRS('+init=epsg:32721')
```

Gerando as poligonais usando Voronoi e um buffer de 1200 metros de cada furo:

```
> library(rgeos)
> buf<-gBuffer(res.furo,width=1200)
> library(dismo)
> v<-voronoi(res.furo)
> recurso<-intersect(v,buf)
> recurso@data
     idfuro.x esp.total esp.minerio        k      kcl      k20        RI
1  PBAT-10-01      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
2  PBAT-10-02      1.66         1.66 18.84602 35.94229 22.70663 3.5307229
3  PBAT-10-05      0.44         0.44  3.42500  6.53500  4.13000 1.2500000
4  PBAT-11-07      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
5  PBAT-11-06      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
6  PBAT-11-08      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
7  PBAT-11-09      1.82         1.82 20.94462 39.94500 25.23473 2.3098901
8  PBAT-11-03      1.37         1.37 14.04161 26.78394 16.91942 2.0270073
9  PBAT-11-10      0.43         0.43 10.42233 19.87279 12.55349 0.4139535
10 PBAT-11-11      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
11 PBAT-11-12      2.07         2.07 20.46043 39.02338 24.65169 0.5536232
12 PBAT-12-13      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
13 PBAT-12-14      0.00         0.00  0.00000  0.00000  0.00000 0.0000000
14 PBAT-12-15      2.06         2.06 17.03107 32.48296 20.51990 4.1805825
```

Plotando resultado dos polígonos voronoi:

```
> spplot(recurso,'kcl',col.regions=rev(get_col_regions()), main="Teor KCl %")
```



Calculando o recurso com base nos polígonos resultantes e seus dados.

Área de cada polígono:

```
> cbind(recurso$holeid.x,round(area(recurso)))
      [,1]         [,2]
 [1,] "PBAT-10-01" "4449845"
 [2,] "PBAT-10-02" "2871823"
 [3,] "PBAT-10-05" "3879207"
 [4,] "PBAT-11-07" "4449845"
 [5,] "PBAT-11-06" "4449845"
 [6,] "PBAT-11-08" "4432565"
 [7,] "PBAT-11-09" "2935622"
 [8,] "PBAT-11-03" "2875289"
 [9,] "PBAT-11-10" "3410341"
[10,] "PBAT-11-11" "3439290"
[11,] "PBAT-11-12" "3787818"
[12,] "PBAT-12-13" "3863509"
[13,] "PBAT-12-14" "4231838"
[14,] "PBAT-12-15" "3295504"
```

Volume de minério dos polígonos (m^3):

```
> cbind(recurso$holeid.x,round(area(recurso)*recurso$esp.minerio))
      [,1]         [,2]
 [1,] "PBAT-10-01" "0"
 [2,] "PBAT-10-02" "4767226"
 [3,] "PBAT-10-05" "1706851"
 [4,] "PBAT-11-07" "0"
 [5,] "PBAT-11-06" "0"
 [6,] "PBAT-11-08" "0"
 [7,] "PBAT-11-09" "5342832"
 [8,] "PBAT-11-03" "3939146"
 [9,] "PBAT-11-10" "1466447"
[10,] "PBAT-11-11" "0"
[11,] "PBAT-11-12" "7840783"
[12,] "PBAT-12-13" "0"
[13,] "PBAT-12-14" "0"
[14,] "PBAT-12-15" "6788737"
```

Volume total (m^3):

```
> sum(area(recurso)*recurso$esp.minerio)
[1] 31852023
```

Tonelagem total (assumindo densidade de 2.08):

```
> sum(area(recurso)*recurso$esp.minerio)*2.08
[1] 66252209
```

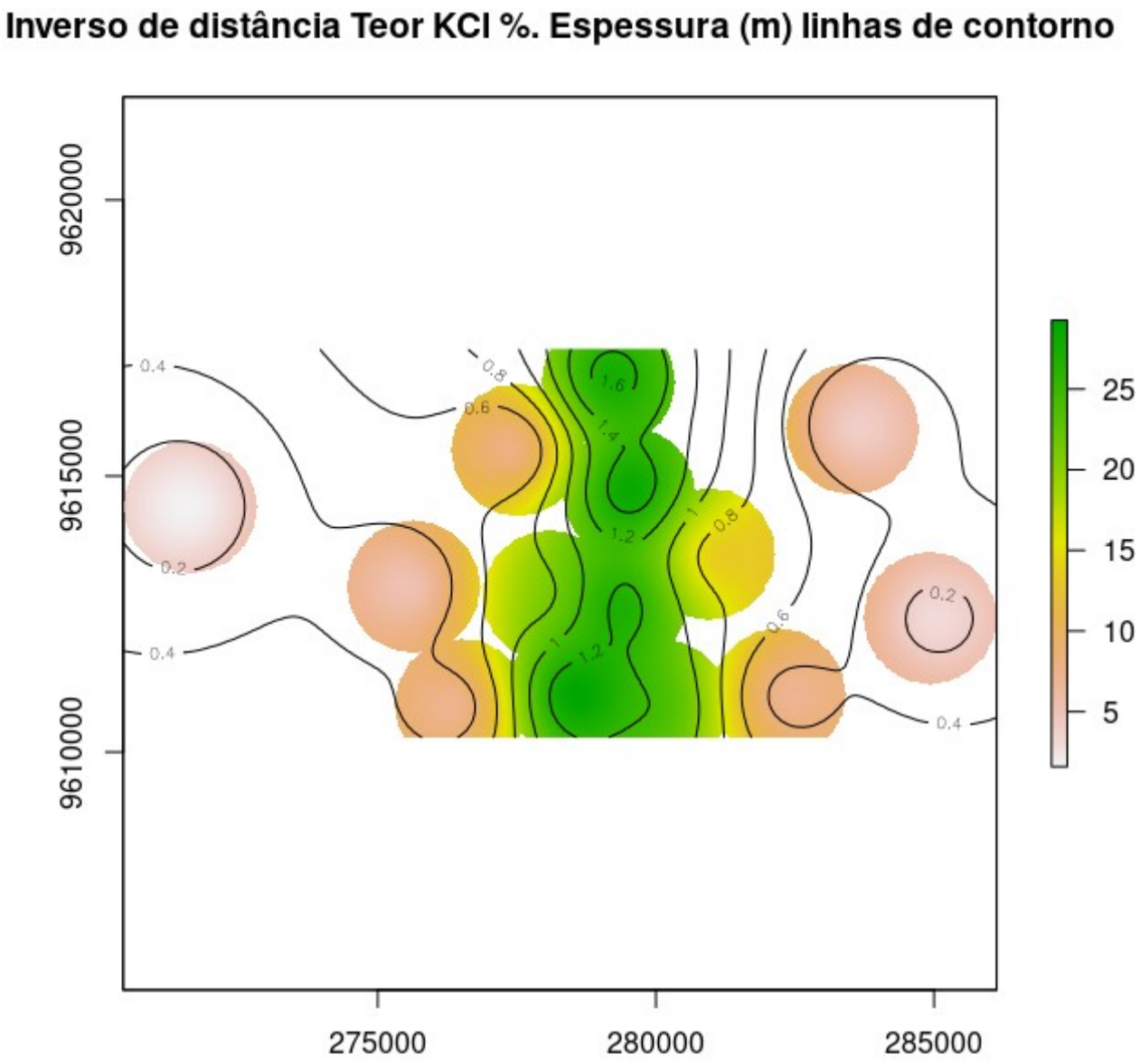
Espessura e teor médios de KCl (ponderados pelas áreas dos polígonos com teor acima do cut-off):

```
> sum(area(recurso)*recurso$esp.minerio)/sum(area(recurso[which
(recurso$esp.minerio>0),]))
[1] 1.381531
> sum(area(recurso)*recurso$kcl)/sum(area(recurso[which (recurso$esp.minerio>
0),]))
[1] 27.99665
```

Chegamos a conclusão que, com base nos furos apresentados, temos um recurso de:
66.252 milhões de toneladas @ 27.99% KCl com espessura média de mineralização de 1.38 metros

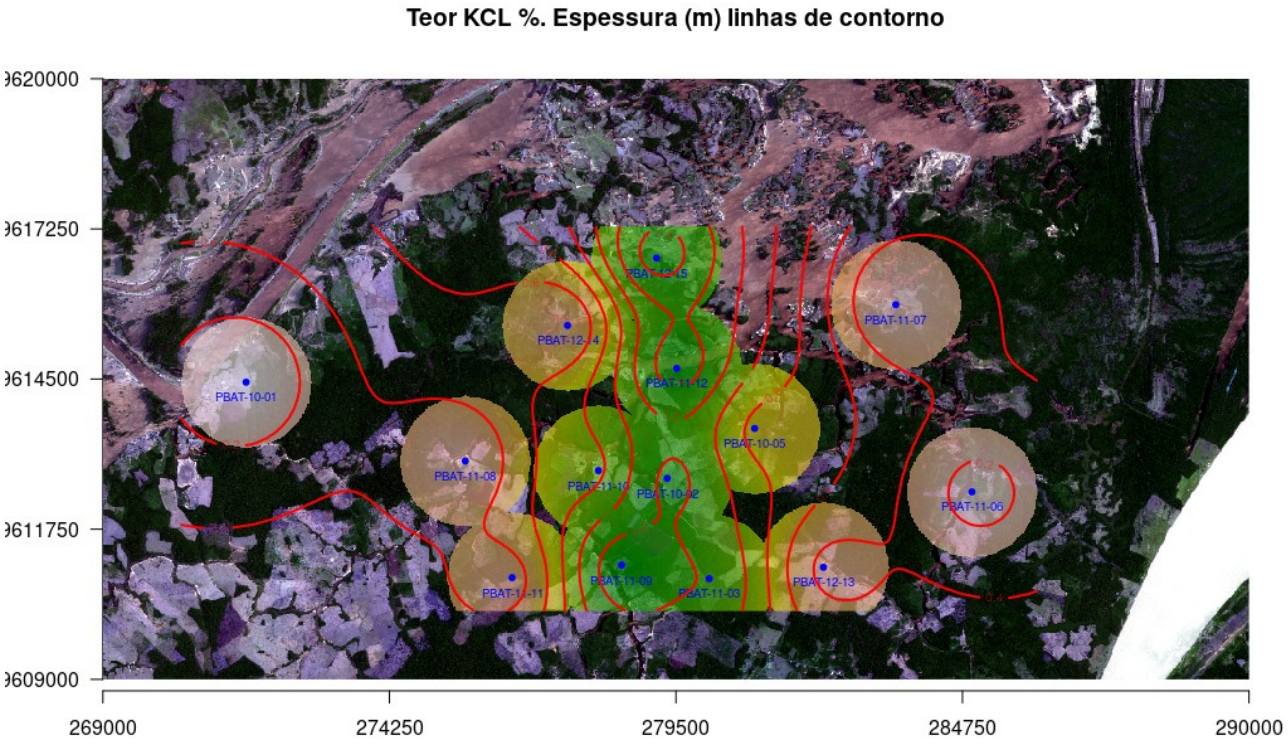
Interpolando espessuras e teores usando Inverso de distância:

```
> rast<-raster(recurso,res=10)
> library(gstat)
> metodo<-gstat(formula=kcl~1,locations=res.furo)
> metodo2<-gstat(formula=esp.minerio~1,locations=res.furo)
> idw<-interpolate(rast,metodo)
[inverse distance weighted interpolation]
> idw.esp<-interpolate(rast,metodo2)
[inverse distance weighted interpolation]
> idwmsc<-mask(idw,recurso)
> plot(idwmsc,main='Inverso de distância Teor KCl %. Espessura (m) linhas de
contorno')
> contour(idw.esp,add=TRUE)
```



Visualizando toda informação em conjunto:

```
> plotRGB(sent,r=3,g=2,b=1,stretch='lin',main='Teor KCL %. Espessura (m) linhas
de contorno',axes=TRUE,ext=extent(269000,290000,9609000,9620000))
> plot(idwmsc,add=TRUE,alpha=0.6,legend=F)
> plot(furos,add=TRUE,cex=1,pch=20,col='blue')
> text(furos,label=furos$holeid,pos=1,cex=0.6,col='blue')
> contour(idw.esp,add=TRUE,col='red',drawlabels=T,lwd=2)
```



6.2 – Modelagem de recursos usando R e GSLIB

Vamos falar superficialmente agora sobre geo estatística e modelo de blocos usando GSLIB que é um conjunto de programas em FORTRAN que cobrem todos os aspectos de modelagem geoespacial de recursos.

GSLIB são programas em Fortran77 ou Fortran90. Seu significado é **G**eostatistical **S**oftware **LIB**rary. Esse nome foi primeiramente utilizado para uma coleção de programas geo estatísticos desenvolvidos pela Universidade de Standford. Para mais detalhes baixe o livro de GSLIB (**http://claytonvdeutsch.com/wp-content/uploads/2019/03/GSLIB-Book-Second-Edition.pdf**).

Vamos usar PostgreSQL e R para prepare os dados e rapidamente gerar visualizações dos resultados.

Baixe as fontes do GSLIB para linux **http://www.statios.com/software/gslib90_ls.tar.gz**
ou os executáveis em DOS **http://www.statios.com/software/gslib90.zip**

6.2.1 – Instalando o GSLIB

Se você escolheu o executáveis em DOS para o ambiente windows descompacte os mesmo em uma pasta.

Caso use o linux compile a fonte usando make (substitua o compilador fortran no arquivo **`Makefile`** se necessário, o utilizado é os g95 mas pode ser substituído por f95 ou o compilador fortran 90 que estiver na sua máquina. Existe também um arquivo **`Makefile`** dentro da pasta gslib, faça a mesma coisa substituindo o compilador fortran se necessário.

gslib90/Makefile
```
FC=g95  #substitua se necessário
```
…

gslib90/gslib/Makefile
```
FC=g95  #substitua se necessário
...
```
compile usandp o comando abaixo na pasta gslib90
```
$sudo make
```

***** *Se um error de compilação aparecer durante a compilação do arquivo fonte vargplt.for substitua as linha 579 e 580 com:*
```
      parameter (MAXLEN=40)
      character str(MAXLEN)*1,strl*40
```
e execute novamente com:
```
$ sudo make
```

Adicione a pasta do programa no PATH do seu sistema. Se for Linux, adicione a seguinte linha no arquivo .bashsrc
```
export PATH="/onde/voce/instalou/gslib90:$PATH"
```
grave e digite no monitor
```
$ source ~/.bashrc
```

6.2.2 – Preparando os dados

Baixe os arquivos necessários localizados nos links abaixo:.

Collar - **http://amazeone.com.br/barebra/pandora/collar_JERICHO.csv**
Survey - **http://amazeone.com.br/barebra/pandora/survey_JERICHO.csv**
Assay - **http://amazeone.com.br/barebra/pandora/assay_JERICHO.csv**

O primeiro passo é carregar a informação desses arquivos em um banco de dados espacial conforme as instruções abaixo.

Criaremos um banco de dados chamado jericho para colocarmos as informação de collar, assay e survey dentro dele. Fazemos isso usando:

```
$ createdb jericho  --encoding=utf-8
$ psql jericho -c "CREATE EXTENSION postgis;"
$ psql jericho -c "CREATE TABLE collar(bhid varchar(30) NOT NULL, xcollar
numeric NOT NULL,ycollar numeric NOT NULL,zcollar numeric NOT NULL,
UNIQUE(bhid));"
$ psql jericho -c "CREATE TABLE assay(bhid varchar(30) NOT NULL, _from numeric
NOT NULL, _to numeric NOT NULL, cu numeric, au numeric, dmn varchar(10),
UNIQUE(bhid, _from, _to));"
$ psql jericho -c "CREATE TABLE survey(bhid varchar(30) NOT NULL,_at numeric NOT
NULL,az numeric NOT NULL,dip numeric NOT NULL, UNIQUE(bhid, _at));"
```

Agora, usando R vamos carregar as tabelas criadas acima. Substitua '**seuNome**' and '**segredo**' com a senha e usuário do seu banco de dados.

```
> collar<-read.csv('collar_JERICHO.csv')
> survey<-read.csv('survey_JERICHO.csv')
> assay<-read.csv('assay_JERICHO.csv')
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host='127.0.0.1',user='seuNome',
password='segredo', dbname='jericho')
> names(collar)<-c('bhid','xcollar','ycollar','zcollar')
> dbWriteTable(con, "collar", collar, row.names=FALSE, append=TRUE)
> names(assay)<-c('bhid','_from','_to','cu','au','dmn')
> dbWriteTable(con, "assay", assay, row.names=FALSE, append=TRUE)
> names(survey)<-c('bhid','_at','az','dip')
> dbWriteTable(con, "survey", survey, row.names=FALSE, append=TRUE)
```

Uma vez carregados o dados vamos gerar um arquivo de dados com as amostras individuais em espaço 3D que será a base do processamento usado pelo GSLIB. O código R abaixo calculará as coordenadas 3D de cada amostra nos furos de sondagem e criará o arquivo.

```
> dados<-dbGetQuery(con, "select TO_NUMBER(c.bhid,'99999999') as furo, c.xcollar
as Xm, c.ycollar as Ym, c.zcollar as Zm, c.xcollar as Xb, c.ycollar as Yb,
c.zcollar as Zb, c.xcollar as Xe, c.ycollar as Ye, c.zcollar as Ze,
(a._to+a._from)/2 as m, a._from as b, a._to as e, (a._to-a._from) as len, a.cu
as CU,a.au as AU from collar as c, assay as a where c.bhid = a.bhid and a.cu>0
order by furo,m")
> colar<-dbGetQuery(con, "select TO_NUMBER(c.bhid,'99999999') as furo,s._at as
prof, c.xcollar as x, c.ycollar as y, c.zcollar as z, s.dip as mergulho, s.az as
az from collar as c, survey as s where c.bhid = s.bhid and s._at=0")
> colar$mergulho<-colar$mergulho*-1
> estacoes<-dbGetQuery(con, "SELECT TO_NUMBER(bhid,'99999999') as furo, _at as
prof,null AS x, null AS y, null as z, dip as mergulho, az FROM survey")
> estacoes$mergulho<-estacoes$mergulho*-1
```

```
> tresD<-function(da,es){
 linha<-data.frame()
 for(i in 1:nrow(da)){
  x<-da[i,3]
  y<-da[i,4]
  z<-da[i,5]
  di<-0
  fur<-es[es[[1]]==da$furo[[i]],]
  fur<-fur[order(fur$prof),]
  for(j in 1:nrow(fur)){
   angulo<-fur[j,7]
   d<-fur[j,2]-di
   di<-d+di
   deltaz<-abs(d*sin(fur[j,6]*pi/180))
   raio<-d*cos(fur[j,6]*pi/180)
   x<-round(x+raio*sin(angulo*pi/180))
   y<-round(y+raio*cos(angulo*pi/180))
   z<-round(z-deltaz)
   linha<-
rbind(linha,data.frame(furo=fur[j,1] ,prof=fur[j,2],x=x ,y=y ,z=z,mergulho=fur[j
,6],az=fur[j,7]))
   z<-z
  }
 }
 lin<-rbind(linha,da)
 lin<-lin[order(lin$furo,lin$prof),]
 return(lin)
}
> pts.XYZ<-tresD(colar,estacoes)
> obterXYZ<-function(pontos,furo,profu){
 sete<-pontos[as.character(pontos$furo)==furo,]
 refa<-sete[sete$prof==max(sete[sete$prof<=profu,]$prof),]
 ref<-refa[1,]
 d<-profu-ref$prof
 deltaz<-abs(d*sin(ref$mergulho*pi/180))
 raio<-d*cos(ref$mergulho*pi/180)
 z<-ref$z-deltaz
 angulo<-ref$az
 x<-ref$x+raio*sin(angulo*pi/180)
 y<-ref$y+raio*cos(angulo*pi/180)
 return(c(x,y,z))
}
#Calculando as coordenadas 3D das amostras
> for(i in 1 : dim(dados)[1]){
 dado<-obterXYZ(pts.XYZ,dados$furo[i],dados$m[i])
 dado1<-obterXYZ(pts.XYZ,dados$furo[i],dados$b[i])
 dado2<-obterXYZ(pts.XYZ,dados$furo[i],dados$e[i])
 dados$xm[i]<-dado[1]
 dados$ym[i]<-dado[2]
 dados$zm[i]<-dado[3]
 dados$xb[i]<-dado1[1]
 dados$yb[i]<-dado1[2]
 dados$zb[i]<-dado1[3]
 dados$xe[i]<-dado2[1]
 dados$ye[i]<-dado2[2]
 dados$ze[i]<-dado2[3]
}
# Criando o arquivo de dados que seá usado no gslib
> out_str<-paste0('dataL1.dat','\n','16','\n','bhidnum','\n','xm','\n', 'ym','\
n','zm','\n','xb','\n','yb','\n','zb','\n','xe','\n','ye','\n','ze','\n', 'm','\
n','b','\n','e','\n','len','\n','cu','\n','au','\n')
> cat(out_str, file = 'dataL1.dat')
```

```
> library(data.table)
> fwrite(x = dados,file = "dataL1.dat", sep = " ", col.names=F, append=T)
```

O arquivo **dataL1.dat** foi criado e será a base do processamento que faremos daqui por diante. Este formato é por GSLIB e basicamente consiste um cabeçalho com o nome do arquivo na primeira linha, a quantidade de coluna de dados na segunda linha, o nome de cada coluna de dados e finalmente os dados.

```
dataL1.dat
16
bhidnum
xm
ym
zm
xb
yb
zb
xe
ye
ze
m
b
e
len
cu
au
1801 498674.939820935 7678687.17059919 9.95347380770994
498674.770475121 7678687.19439919 10.4233201181029
498675.109166748 7678687.14679919 9.48362749731697 206.5 206 207 1
0.15 0.03
…..
```

bhidnum é o número associado com o furo de sondagem (fortran não gosta de strings em estrutura de dados e por isso cada furo foi convertido para um número). As colunas **xm**, **ym** e **zm** são as coordenadas do meio da amostra. As colunas **xb**, **yb** e **zb** são as coordenadas do início da amostra (begin). As colunas **xe**, **ye** e **ze** são das coordenadas do fim da amostra (end), As colunas **m**, **b** e **e** são respectivamente as profundidades do início, meio e fim da amostra em profundidade dentro do furo de sonda. A coluna **len** é o cumprimento da amostra e as colunas **cu** e **au** são os valores analíticos de teor de cobre e ouro das amostra

6.2.3 – Gerando Histogramas e CDF (Frequência de distribuição acumulada)

O primeiro programa GSLIB que vamos executar gerará um histograma da frequência de distribuição do teor de Cobre e também gera um gráfico da frequência de distribuição acumulada (CDF).

A maioria dos programas GSLIB trabalham cou um arquivo de parametros .par e se você abre o aquivo fonte do programa ele geralmente se inicia com uma descrição dos campos de parâmetros na forma de comentário de programa (linhas iniciadas com ‘c’).

fragmento do código fonte **histplt.for**

```
c
c                          Histogram Plot
c                          **************
c
```

```
c This program generates a PostScript file with a histogram and summary
c statistics.
c
c INPUT/OUTPUT Parameters:
c
c   datafl      arquivo de entrada dos dados
c   ivr,iwt     coluna da variável e peso (0 se peso não usado)
c   tmin,tmax   limite de corte aceitável: >= tmin e < tmax
c   outfl       arquivo de saida com o histograma
c   hmin,hmax   limites do gráfico (automático se hmax<hmin)
c   fmax        máxima frequência plotada
c   ncl         número de classes usadas
c   ilog        1=scala logarítmica, 0=escala aritimética
c   icum        1=frequência acumulada, 0=histograma
c   ncum        número de pontos para hist (0 => automático)
c   ndec        número de casas decimais (0 => automático)
c   title       título do gráfico
c   spos        posição da estatísticao (ao longo do eixo X)
c   xref        valor de referência da posição do gráfico
c
```

e criaremos um arquivo **`hist.par`** como:

```
                                Parameters for HISTPLT
                                **********************
START OF PARAMETERS:
dataL1.dat
15 0
-1.0e21 1.0e21
histplt.ps
0.01 15.0
-1.0
25
1
0
200
2
Clustered Data
1.5
-1.1e21
```

Execute o programa usando:

```
$ histplt
```

(or `histplt.exe` if using dos)

As seguintes linhas irão aparecer:

```
 HISTPLT Version: 3.000

  Which parameter file do you want to use?
```

Entre com o nome do arquivo parâmetro

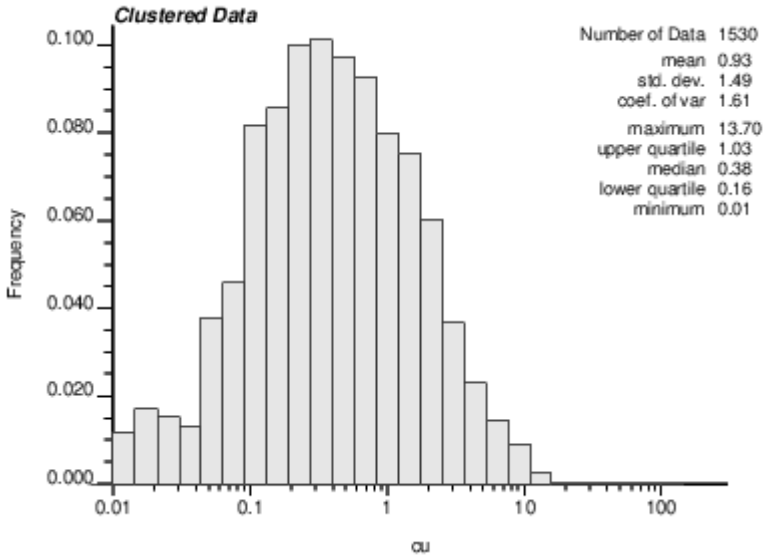
```
hist.par
```

e o seguinte resultado aparecerá:

```
  data file = dataL1.dat
  columns =            15           0
  trimming  limits =   -1.00000002E+21   1.00000002E+21
  output file = histplt.ps
  attribute limits =    9.99999978E-03   15.0000000
  frequency  limit =   -1.00000000
  number of classes =           25
  log scale option =            1
```

```
 cumulative frequency option =            0
 number of decimal places =           2
 title = Clustered Data
 position of stats =    1.50000000
 reference value =   -1.09999999E+21

There are     1530 data with:
  mean value          =      0.92686
  median value        =      0.37500
  standard deviation  =      1.49117
  minimum and maximum =      0.01000    13.70000


HISTPLT Version: 3.000 Finished
```

O arquivo **`histplt.ps`** é criado com o gráfico.



Agora executaremos novamente criando o seguinte arquivo de parâmetro **`histcdf.par`** :

```
                    Parameters for HISTPLT
                    **********************
START OF PARAMETERS:
dataL1.dat
15 0
-1.0e21 1.0e21
cdfplt.ps
0.01 15.0
-1.0
25
1
1
200
2
Clustered Data
1.5
-1.1e21
```

Execute o programa com
`$ histplt`
(or `histplt.exe` if using dos)

Irá aparecer:

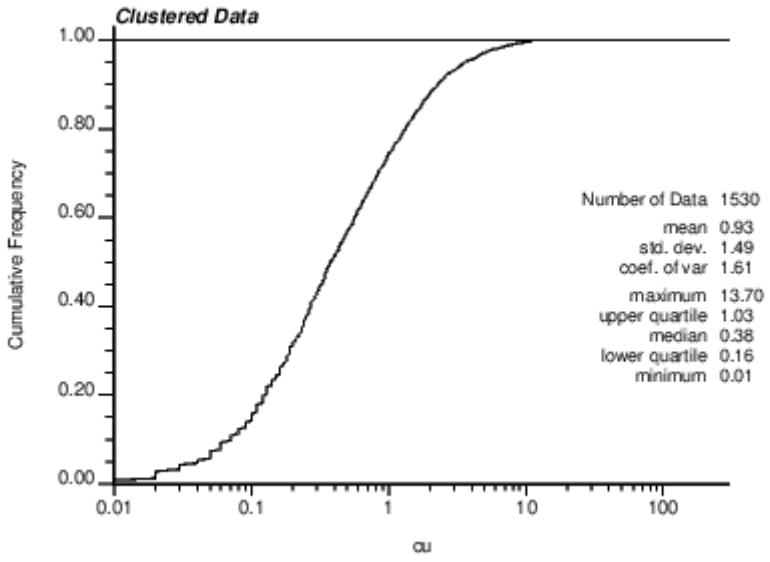
```
 HISTPLT Version: 3.000

 Which parameter file do you want to use?
```

Entre o nome do novo arquivo de parâmetro criado
`histcdf.par`

e o seguinte resultado aparecerá:

```
  data file = dataL1.dat
  columns =            15           0
  trimming  limits =   -1.00000002E+21   1.00000002E+21
  output file = cdfplt.ps
  attribute limits =    9.99999978E-03   15.0000000
  frequency  limit =   -1.00000000
  number of classes =           25
  log scale option =            1
  cumulative frequency option =           1
  number of cumulative points =          200
  number of decimal places =            2
  title = Clustered Data
  position of stats =    1.50000000
  reference value =   -1.09999999E+21

 There are     1530 data with:
   mean value          =      0.92686
   median value        =      0.37500
   standard deviation  =      1.49117
   minimum and maximum =      0.01000    13.70000


 HISTPLT Version: 3.000 Finished
```

e o arquivo **`cdfplt.ps`** é criado com o gráfico.



6.2.4 – Desagrupamento do dados (declustering)

Vamos calcular aqui os parâmetros ideais para desagrupar (decluster) os dados apropriadamente. Eles são (dos comentários do código fonte declus.for).

```
c         DECLUS: a three dimensional cell declustering program
c         ****************************************************
c
c See paper in Computers and Geosciences Vol 15 No 3 (1989) pp 325-332
c
c INPUT/OUTPUT Parameters:
c
c   datafl          arquivo com os dados
c   icolx,y,z,vr    colunas  X, Y, Z, e variável
c   tmin,tmax       limites de corte
c   sumfl           arquivo com saída do sumário resultante
c   outfl           arquivo de saída para dados e pesos do desagrupamento
c   yanis,zanis     anisotropia das células Y e Z (Ysize=size*Yanis)
c   iminmax         0=procura por média mínima desagrupada (1=máximo)
c   ncell,cmin,cmax número de tamanho de células , tam. mínimo, tam. máximo
c   noff            número de offsets da origem
```

A distribuição dos dados usados nessa parte expande 370 unidades na direção X, 2250 na direção Y e 370 na direção Z. Vamos usar esses números os valores de anisotropia em y (`yanis`) e em z (`zanis`) dividindo a extensão em Y por X e também a extensão em Z por X o que nos dá 6.1 e 1 respectivamente. Agora nos precisamos de definir o valor para `ncell`, `cmin` e `cmax` escolheremos 190 células de 10 a 200. O tamanho máximo das células é cerca da metade da extensão em X, o mínimo da célula é menor que a menor distância entre amostras em X. o número de células será **(max-min)/min**. O número de offsets da origem (`noff`) é geralmente entre 25 e 100. Usaremos o valor 100.

Desta forma, nosso arquivo de parâmetros para desagrupamento **`declus.par`** será:

```
                              Parameters for DECLUS
                              *********************
START OF PARAMETERS:
dataL1.dat
2 3 4 15
-1.0e21 1.0e21
dsum.txt
ddataL1.dat
6.1 1
0
190 10 200
100
```

Ao executarmos o programa declus usando os paraâmetros do arquivo **`declus.par`** teremos:

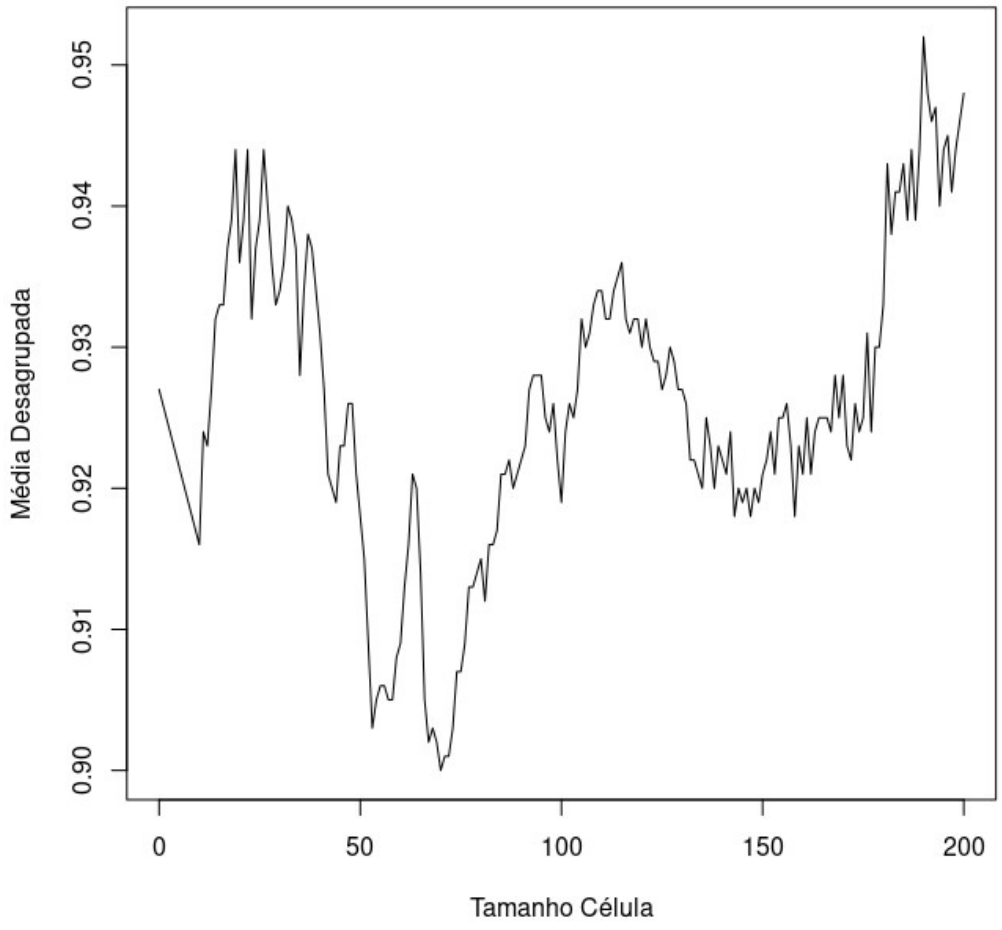
```
$ declus

 DECLUS Version: 3.000

 Which parameter file do you want to use?
declus.par
  data file = dataL1.dat
  columns =            2           3           4          15
  tmin,tmax =   -1.00000002E+21   1.00000002E+21
  summary file = dsum.txt
  output file = ddataL1.dat
  anisotropy =    6.09999990       1.00000000
```

```
 minmax flag =            0
 ncell min max =          190   10.0000000       200.000000
 offsets =          100


There are     1530 data with:
  mean value            =      0.92686
  minimum and maximum   =      0.01000    13.70000
  size of data vol in X =    354.06250
  size of data vol in Y =   2100.00000
  size of data vol in Z =    354.39316

  declustered mean      =      0.90013
  min and max weight    =      0.32065     5.73139
  equal weighting       =      1.00000


DECLUS Version: 3.000 Finished
```

Os arquivos **`dsum.txt`** e **`ddataL1.dat`** foram criados. Vamos plotar o resultado no arquivo **`dsum.txt`** usando R.

```
> df<-read.table('dsum.txt',skip=4)
> plot(df,type='l',xlab='Cell Size',ylab='Declustered Mean')
```

O resultado será:



Interpretando o gráfico temos que o valor ideal do tamanho da célula para desagruparmos os dados é de 70. Vamos executar novamente o programa declus usando o seguinte arquivo parâmetro **declusadj.par** f:

```
                                Parameters for DECLUS
                                *********************
START OF PARAMETERS:
dataL1.dat
2 3 4 15
-1.0e21 1.0e21
dsumdcl.txt
dataL1dcl.dat
6.1 1
0
1 70 70
100
```

Ao executar temos:

```
$ declus

 DECLUS Version: 3.000

 Which parameter file do you want to use?
declusadj.par
  data file = dataL1.dat
  columns =            2           3           4          15
  tmin,tmax =   -1.00000002E+21   1.00000002E+21
  summary file = dsumdcl.txt
  output file = dataL1dcl.dat
  anisotropy =    6.09999990       1.00000000
  minmax flag =            0
  ncell min max =            1   70.0000000       70.0000000
  offsets =          100


 There are     1530 data with:
   mean value            =      0.92686
   minimum and maximum   =      0.01000    13.70000
   size of data vol in X =    354.06250
   size of data vol in Y =   2100.00000
   size of data vol in Z =    354.39316

   declustered mean      =      0.90161
   min and max weight    =      0.32121     5.73048
   equal weighting       =      1.00000


 DECLUS Version: 3.000 Finished
```

e o nosso novo arquivo de dados **dataL1dcl.dat**, agora desagrupado, foi criado.

6.2.5 – Histograma e CDF dos dados desagrupados

Vamos novamente plotar o histograma e o CDF para os dados desagrupados. Primeiro criando os arquivos de parâmetros:

Arquivo **histdcl.par**

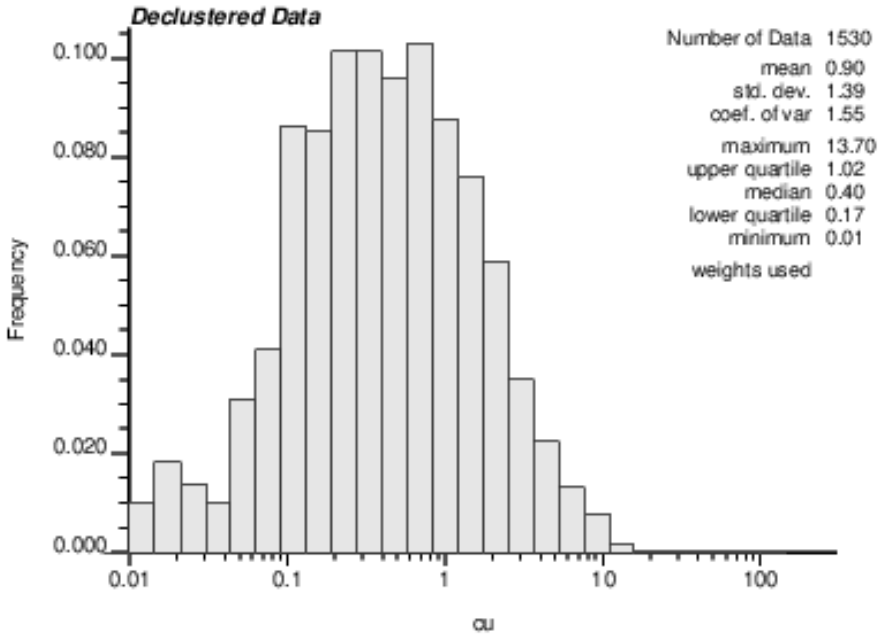
```
                                Parameters for HISTPLT
                                **********************
START OF PARAMETERS:
```

```
dataL1dcl.dat
15 17
-1.0e21 1.0e21
histpltdcl.ps
0.01 15.0
-1.0
25
1
0
200
2
Declustered Data
1.5
-1.1e21
```

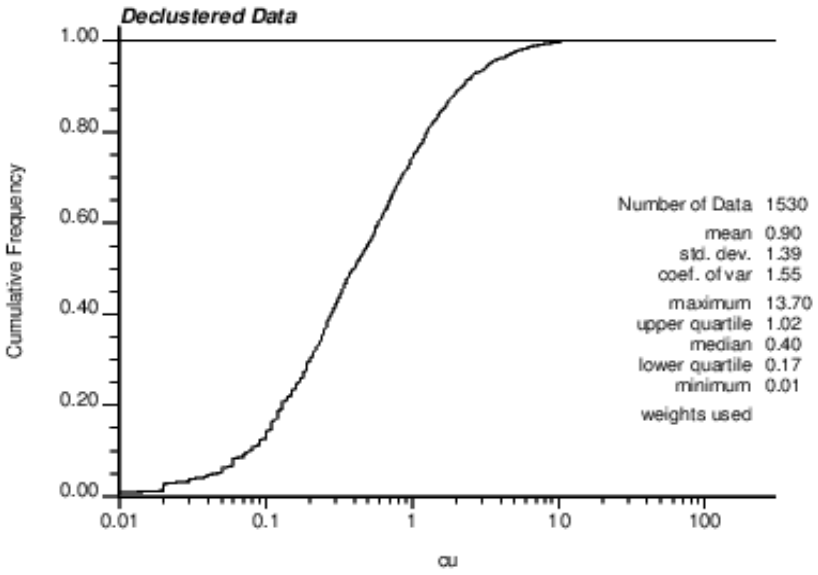
e arquivo **histcdfdcl.par**

```
                                Parameters for HISTPLT
                                **********************
START OF PARAMETERS:
dataL1dcl.dat
15 17
-1.0e21 1.0e21
cdfpltdcl.ps
0.01 15.0
-1.0
25
1
1
200
2
Declustered Data
1.5
-1.1e21
```

Ao executarmos teremos como resultado, usando **histdcl.par,** o arquivo **histpltdcl.ps** abaixo:

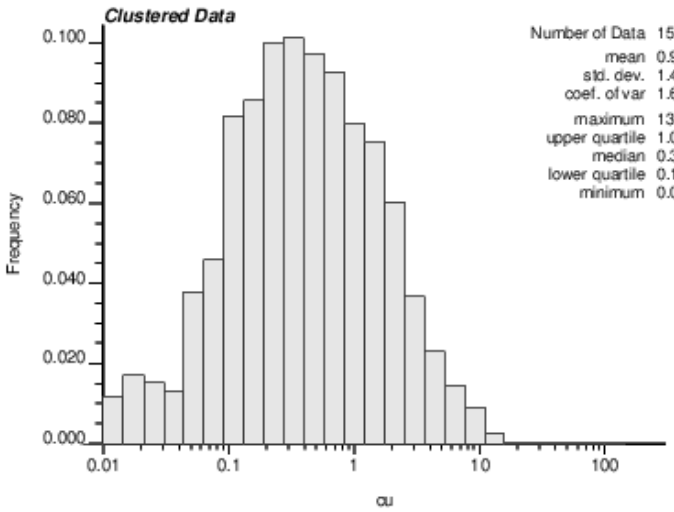


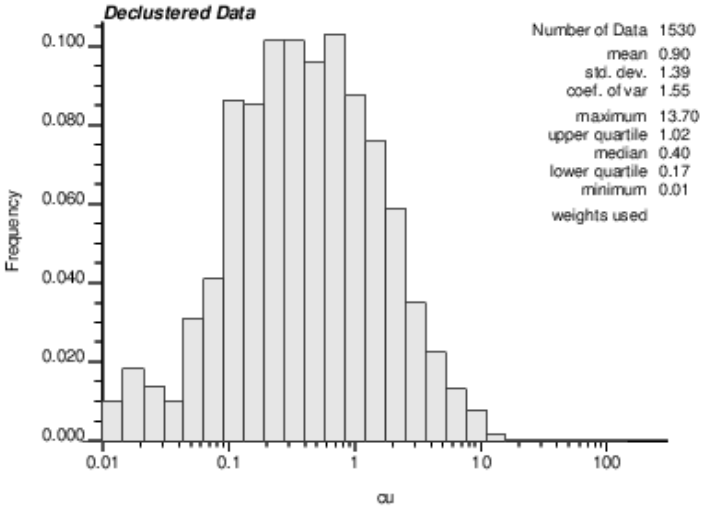
e usando **`histcdfdcl.par`** o arquivo **`cdfpltdcl.ps`** mostrará:

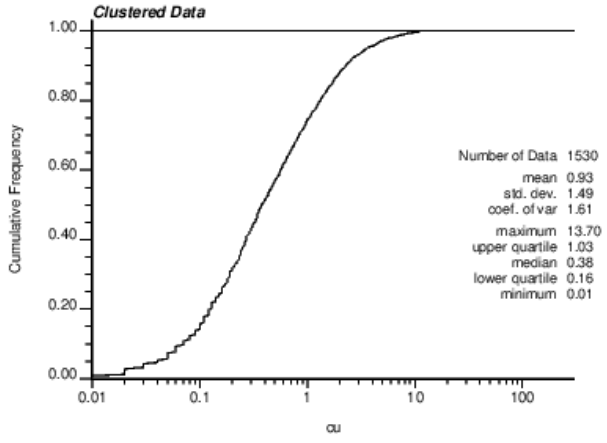


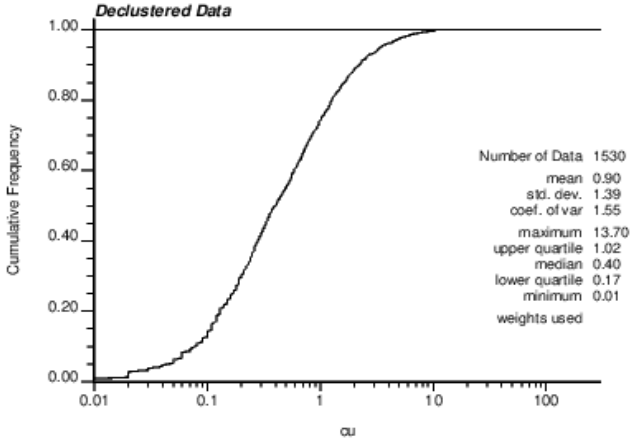
Comparando com o resultado anterior não desagrupado teremos.

Agrupado ***Desagrupado***





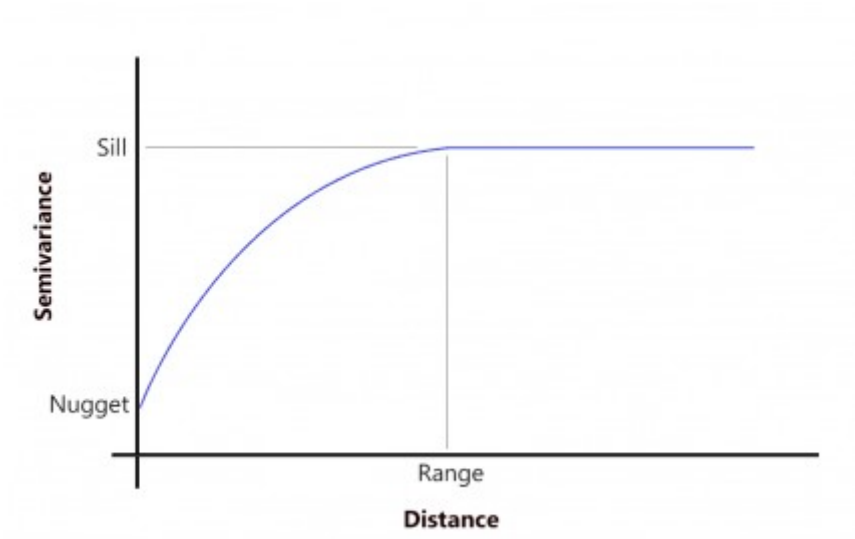




6.2.6 – Semivariograma

A criação e interpretação de um semi variograma é o processo mais importante na modelagem de recursos e seus parâmetros precisam ser bem calculados. Vamos tentar aqui, sem avançar muito em teoria cobrir os parâmetros usados por GSLIB em seu programa gamv.

As informações mais importantes que precisamos extrair do semi variograma são: nugget (efeito pepita), range (extensão) e sill (soleira) onde:
Sill – O valor onde o modelo fica plano.
Range – A distância na qual o modelo fica plano.
Nugget – O valor onde o modelo intercepta o eixo Y.



Para data distribuídos de forma irregular nós precisamos usar um procedimento para melhor entender como uma determinada informação varia com a distância. Valores próximos tendem a se relacionar melhor do que valores mais distantes. A maneira de medirmos estatisticamente como a informação se relaciona é através do calculo do semi variograma.

Um exemplo de um arquivo de parâmetros usado pelo programa gamv é mostrado abaixo.

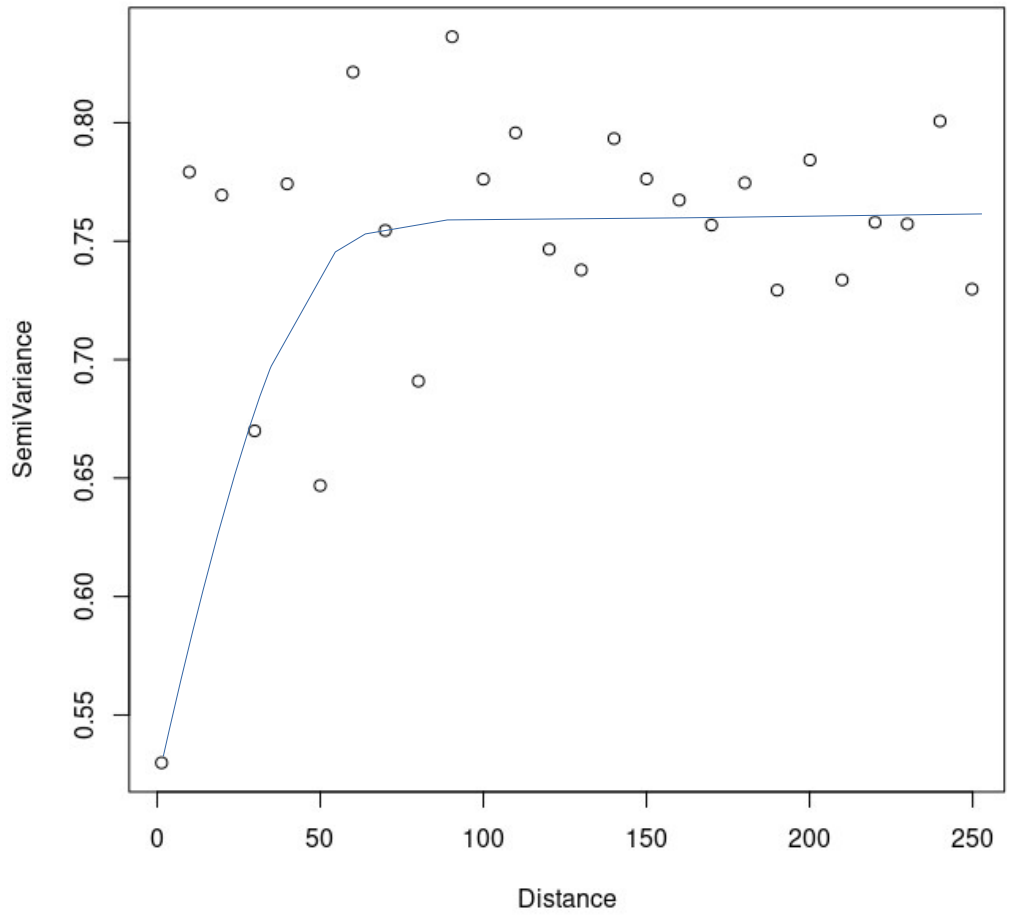
```
                      Parameters for GAMV
                      *******************
START OF PARAMETERS:
../data/cluster.dat           \arquivo com dados
2 3 4                         \colunas  X, Y, Z
1 15                          \numero de variáveis,número das colunas
-1.0e21 1.0e21                \limites de corte
gamv.out                      \arquivo saída do variograma
10                            \número de lags
5.0                           \distância separação entre lags
3.0                           \tolerancia do lag
1                             \número de direções
0.0 90.0 50.0 0.0 90.0 50.0   \azm,atol,bandh,dip,dtol,bandv
1                             \soleira padronizada? (0=não, 1=sim)
1                             \número de variogramas
1 1 1                         \tail var., head var., tipo variograma
```

Usando o aquivo de parâmetros **gamv.par** abaixo geraremos um arquivo **gamv.out** contendo o variograma resultante.

```
                        Parameters for GAMV
                        *******************
START OF PARAMETERS:
dataL1dcl.dat
2 3 4
1 15
-1.0e21 1.0e21
gamv.out
25
10
2
1
5.0 90.0 250.0 0.0 90.0 250.0
1
1
1 1 6
```

Podemos visualizar o resultado gráfico usando R.

```
> df<-read.table('gamv.out',skip=2)
> plot(df$V2,df$V3,xlab='Distance',ylab='SemiVariance')
```

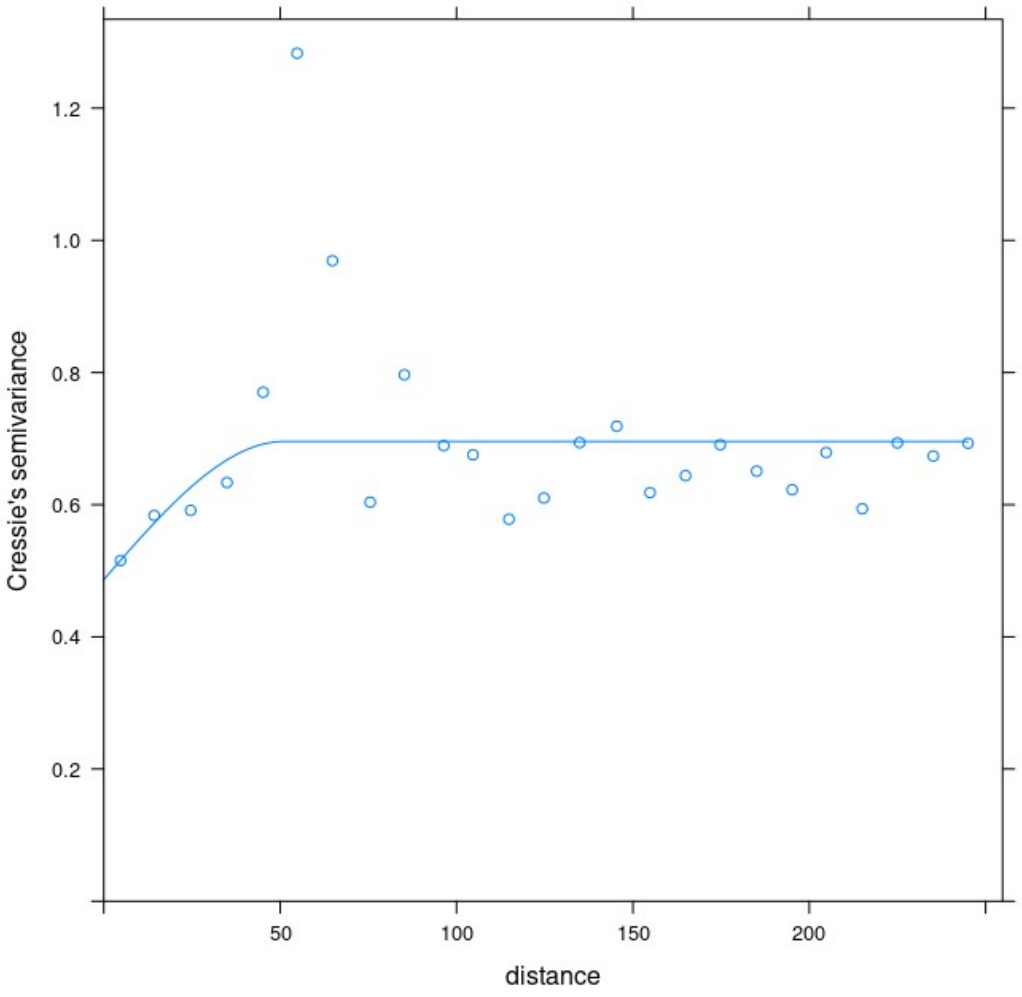


Da linha azul que melhor se ajusta ao resultado obtido extraímos os parâmetros necessários.
Nugget: 0.53
Sill (a partir do nugget): 0.22
Range: 50

Alternativamente podemos usar a biblioteca **gstat** do R para extrair os mesmos parâmetros a partir do arquivo **`dataL1dcl.dat`** .

```
> library(gstat)
> library(sp)
> dados<-read.table('dataL1dcl.dat',skip=19)
> coordinates(dados)=~V2+V3+V4
> v<-variogram(V15~1, dados,cressie=T, cutoff=250, width=10, alpha=5,
beta=0,tol.hor=90,tol.ver=90 )
> vario<-fit.variogram(v, vgm(c("Sph","Gau","Exp")))
> vario
  model     psill    range
1   Nug 0.4866216  0.00000
2   Sph 0.2089895 50.98447
> plot(v,vario)
```

O gráfico abaixo foi gerado para ilustrar o resultado mostrado acima com gstat.



Nós usamos ferramentas distintas e obtivemos resultados bastante semelhantes.
Nugget: 0.47
Sill (a partir do nugget): 0.21
Range: 51

Para finalizar a seção sobre semi variograma considere o ponto seguinte:

Somente uma direção (ou estrutura) foi considerada para demonstrar esta metodologia, mas sabemos que modelos geológicos são anisotrópicos e assim geralmente se faz necessário usar mais estruturas para melhor representar a anisotropia do mesmo.

Direções extra podem ser introduzidas nos parâmetros do programa gamv entrando o número de direções e linhas com os valores para **azm, atol, bandh, dip, dtol e bandv**:

```
...
3                               \número de direções
0.0 90.0 50.0 0.0 90.0 50.0   \azm,atol,bandh,dip,dtol,bandv
90.0 90.0 50.0 45.0 90.0 50.0 \azm,atol,bandh,dip,dtol,bandv
90.0 90.0 50.0 0.0 90.0 50.0  \azm,atol,bandh,dip,dtol,bandv
...
```

Cada direção irá gerar um novo semi variograma que poderá ser usado com seus respectivos pesos no processo de krigagem.

6.2.7 – Krigagem 3D

Com os parâmetros do semi variograma definidos, o próximo passo é efetuar a krigagem. A krigagem gerará um modelo de blocos 3D regularmente espaçado que é o nosso objetivo final.

Um arquivo de parâmetros típico para o programa kt3d.

```
                  Parameters for KT3D
                  *******************
START OF PARAMETERS:
../data/cluster.dat           \arquivo com os dados
1 2 3 0 5 6                   \ colunas para DH, X, Y, Z, var, sec var
-1.0e21 1.0e21                \limites de corte
1                             \opções: 0=grid, 1=cross, 2=jackknife
xvk.dat                       \arquivo com dados jackknife
1 2 0 3 0                     \colunas para X,Y,Z,var and sec var
3                             \nível debugging : 0,1,2,3
kt3d.dbg                      \arquivo de saída do debugging
kt3d.out                      \arquivo de saída da krigagem
50 0.5 1.0                    \nx,xmn,xsiz
50 0.5 1.0                    \ny,ymn,ysiz
1 0.5 1.0                     \nz,zmn,zsiz
1 1 1                         \x,y e z de discretização do bloco
4 8                           \dados min, max para krigagem
0                             \max por octant (0-> não usado)
20.0 20.0 20.0                \raio máximo de busca
0.0 0.0 0.0                   \ângulos para o elipsoide de busca
0 2.302                       \0=SK,1=OK,2=non-st SK,3=exdrift
0 0 0 0 0 0 0 0 0             \drift: x,y,z,xx,yy,zz,xy,xz,zy
0                             \0, variável 1, estimativa do trend
extdrift.dat                  \arquivo gridado com drift/média
4                             \ número da coluna no arquivo gridado
1 0.47                        \nst, efeito pepita(nugget)
1 0.8 0.0 0.0 0.0             \it,cc,ang1,ang2,ang3
10.0 10.0 10.0                \a_hmax, a_hmin, a_vert
```

No nosso exemplo o arquivo de parâmetros **kt3d.par** para executar o programa kt3d será:

```
                  Parameters for KT3D
                  *******************
START OF PARAMETERS:
dataL1dcl.dat
1 2 3 4 15 0
-1.0e21   1.0e21
0
xvk.dat
1   2   0    3    0
0
kt3d.dbg
kt3d.out
100 498449 5.0
449 7677948 5.0
80 -190 5.0
1    1      1
4    8
0
300.0 50.0 50.0
 5.0   0.0   0.0
1     2.302
0 0 0 0 0 0 0 0 0
0
extdrift.dat
4
1    0.47
1    1.0  5.0   0.0   0.0
         50.0  50.0  50.0
```

Esse processamento vai gerar um arquivo kt3d.out com o resultado da krigagem. Este arquivo contém a estimativa e variância da krigagem para cada ponto/bloco no grid, que se estrutura por x depois y, e por último z. Pontos fora do alcance da estimativa tem o valor -999.

6.2.8 – Estatísticas do recurso

Nas seções anterior nós vimos como GSLIB computa parâmetros estatísticos e também efetua a krigagem 3D dos dados.

Agora vamos usar o resultado da krigagem para extrair algumas informações estatísticas clássicas usando R. Basicamente usaremos os arquivos de resultados **kt3d.out** criado acima.

Carregando os valores de teor de Cobre e variância extraídos do arquivo resultante da krigagem **kt3d.out** dentro de um data.frame.

A partir da origem definida pelas coordenadas (498449, 7677948,190) nós aplicamos um offset de 2.5 metros para representar um ponto no centro de cada bloco do modelo. Em seguida substituímos os valores -999 por NA para compatibilidade com R.

```
> kri<-read.table('kt3d.out',skip=4)
> options(digits = 7)
> bmdf<-data.frame(X=rep(seq(498451.5,498946.5,5),35920),
                   Y=rep(seq(7677950.5,7680190.5,5),each=100),
                   Z=rep(seq(-187.5,207.5,5),each=44900),
                   Cu=kri$V1, var=kri$V2)
> bmdf[bmdf == -999] <- NA
```

Podemos visualizar o data.frame usando:

```
> head(bmdf,n=10)
         X         Y      Z Cu var
1  498451.5 7677950.5 -187.5 NA  NA
2  498456.5 7677950.5 -187.5 NA  NA
3  498461.5 7677950.5 -187.5 NA  NA
4  498466.5 7677950.5 -187.5 NA  NA
```

```
5  498471.5 7677950.5 -187.5 NA  NA
6  498476.5 7677950.5 -187.5 NA  NA
7  498481.5 7677950.5 -187.5 NA  NA
8  498486.5 7677950.5 -187.5 NA  NA
9  498491.5 7677950.5 -187.5 NA  NA
10 498496.5 7677950.5 -187.5 NA  NA
```

Removendo valores NA conforme abaixo:

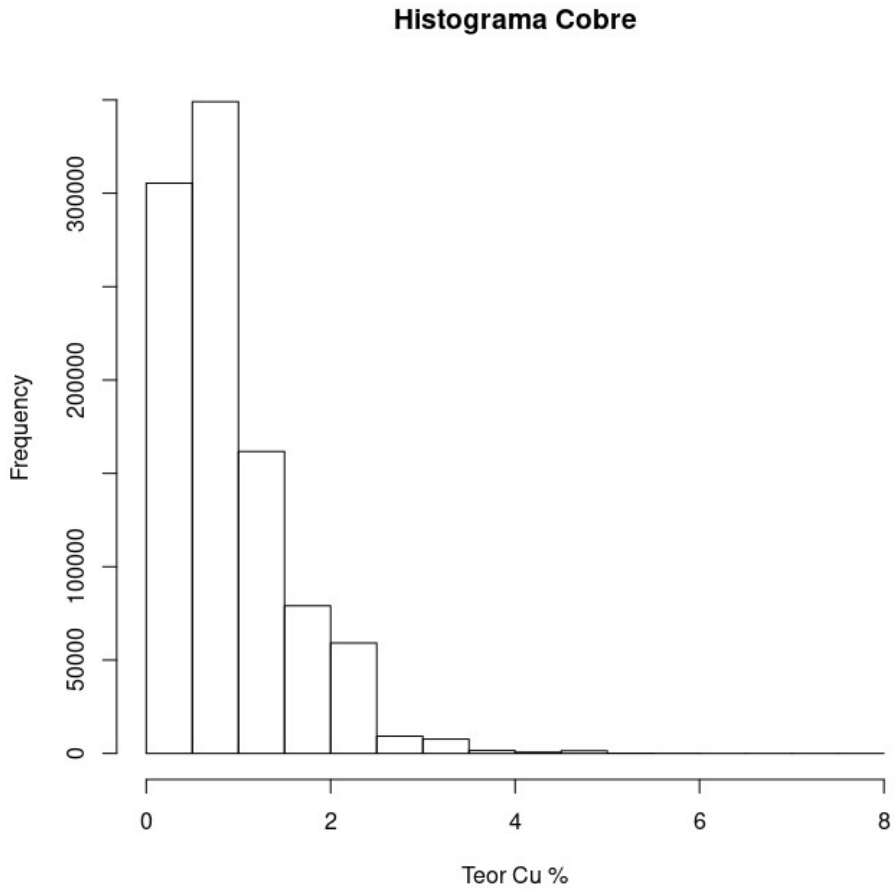
```
> bm<-bmdf[which(bmdf$Cu>0),]
> head(bm,n=10)
             X         Y      Z         Cu       var
24269 498791.5 7679160.5 -187.5 0.81485093 2.4465380
24270 498796.5 7679160.5 -187.5 0.81485093 2.4465380
24271 498801.5 7679160.5 -187.5 0.74757487 2.4454827
24368 498786.5 7679165.5 -187.5 0.45155278 2.4881620
24369 498791.5 7679165.5 -187.5 0.81485093 2.4465380
24370 498796.5 7679165.5 -187.5 0.81485093 2.4465380
24371 498801.5 7679165.5 -187.5 0.74757487 2.4454827
24372 498806.5 7679165.5 -187.5 0.62841624 2.4455087
24468 498786.5 7679170.5 -187.5 0.81485093 2.4465380
24469 498791.5 7679170.5 -187.5 0.81485093 2.4465380
```

Vamos ver alguns parametros estatísticos deste modelo.

```
> summary(bm$Cu)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.031769 0.418497 0.679251 0.891082 1.186710 7.567730
```

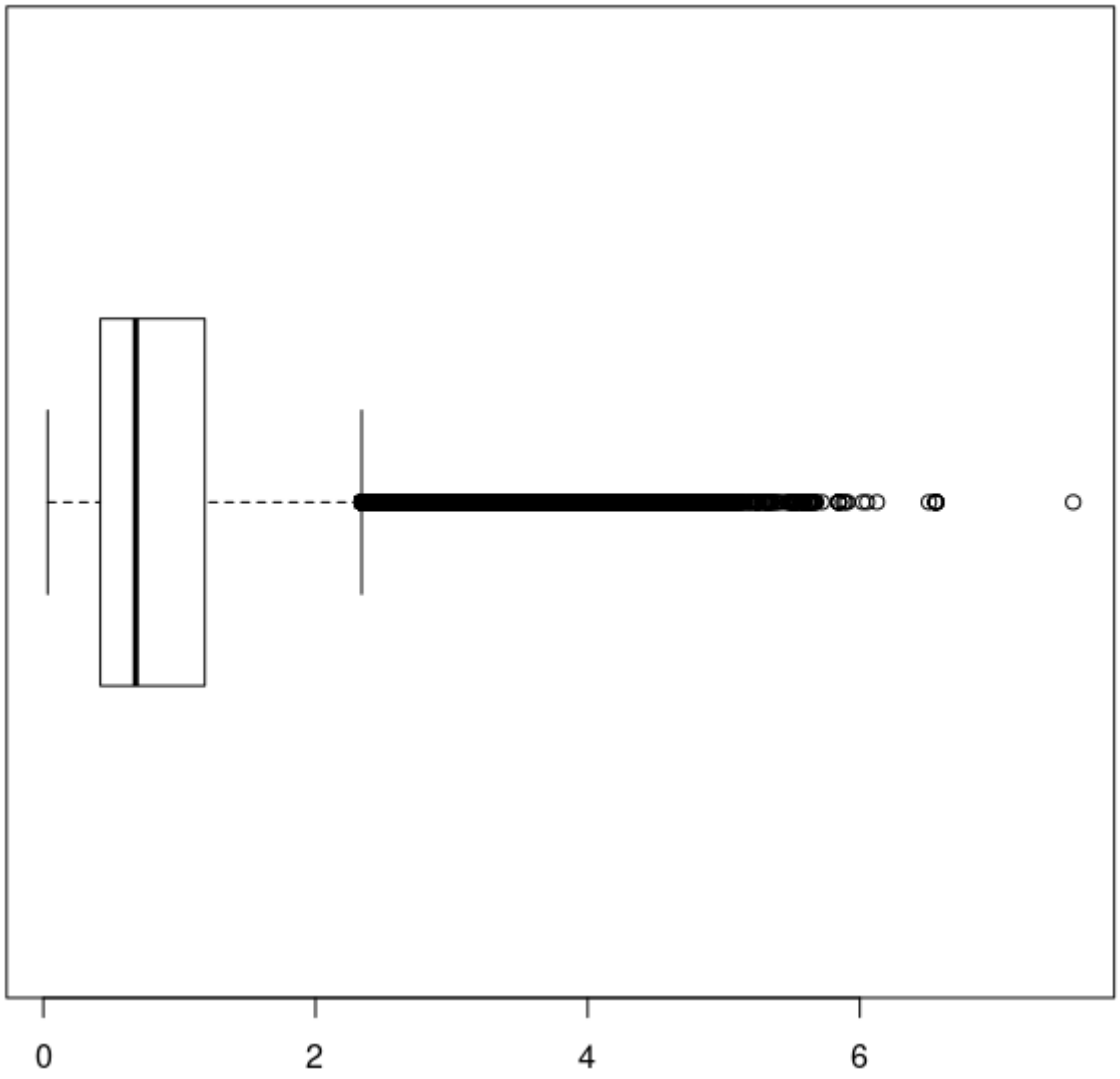
Distribuição do resultado modelado.

```
> hist(bm$Cu,main='Histograma Cobre', xlab='Teor Cu %')
```



Mais parâmetros estatísticos do modelo:

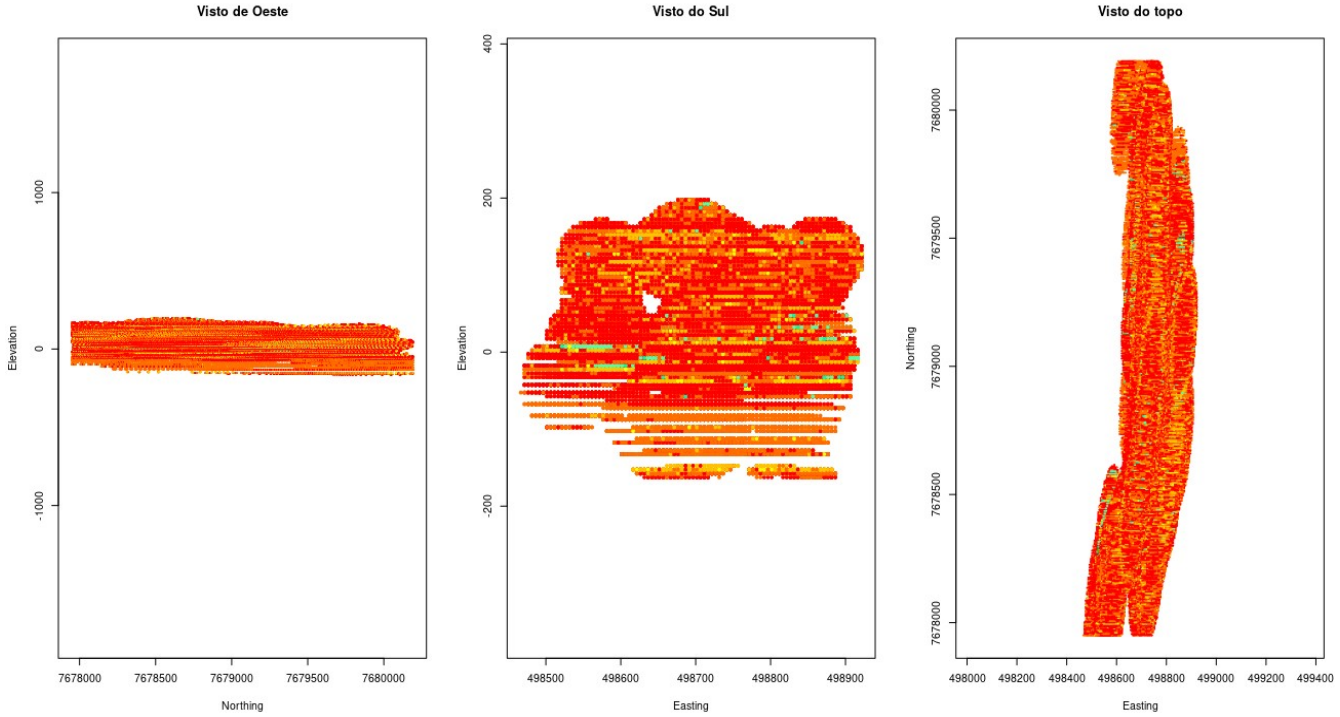
```
> #Quantis
> quantile(bm$Cu)
         0%         25%         50%         75%        100% 
0.031769037 0.418496670 0.679251370 1.186709900 7.567729500 
#definindo qunatis para exibir
> quantile(bm$Cu,c(0.05,0.95))
         5%         95% 
0.214179124 2.178487020 
#Inter Quartile Range
> IQR(bm$Cu)
[1] 0.76821323
#Box plot dos dados mostrando quantis, mediana e outliers
> boxplot(bm$Cu,horizontal=TRUE)
```



6.2.9 – Visualizando os Resultados

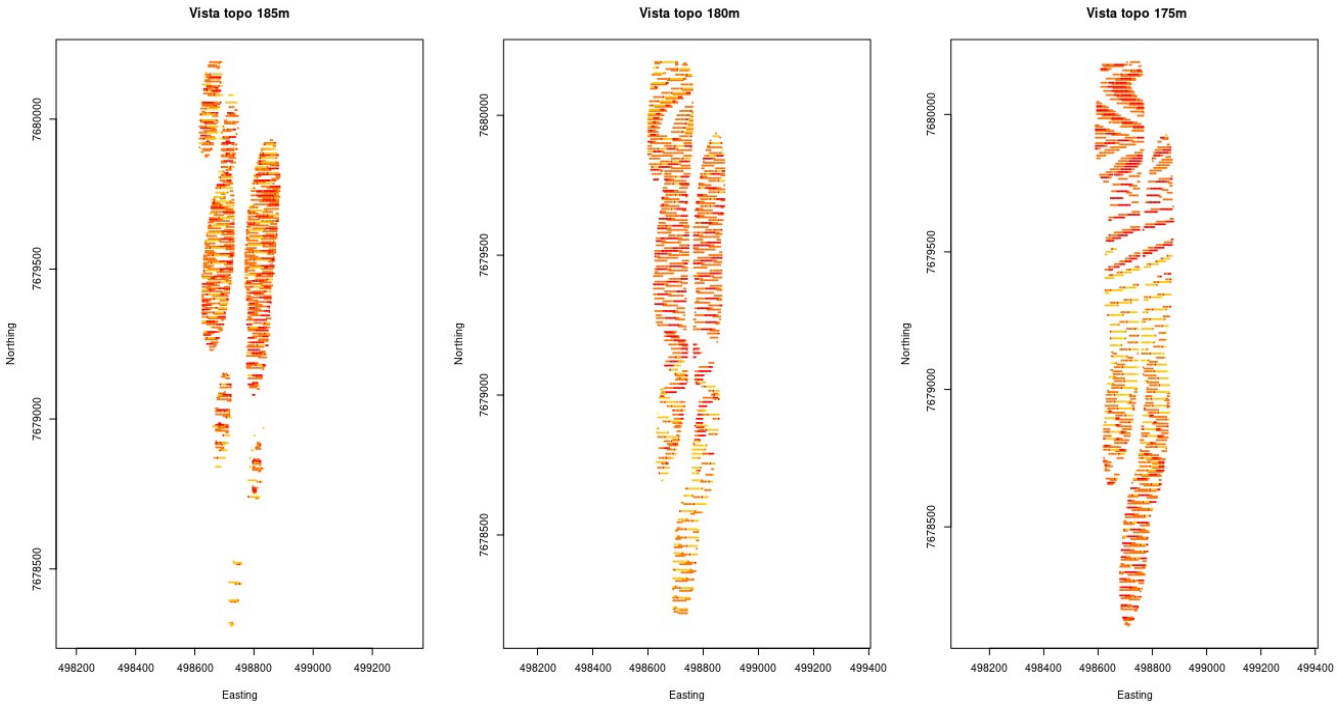
Verificando como nosso modelo esta distribuído espacialmente usando R.

```
> rbPal<-
colorRampPalette(c('red','orange','yellow','cyan','green','blue','violet'))
> dc <- rbPal(10)[as.numeric(cut(bmdf$Cu,breaks = 10))]
> par(mfrow=c(1,3))
> plot(bm$Y, bm$Z,cex=0.02,pch=20,asp=1,col = dc, main='Visto de Oeste',
xlab='Northing', ylab='Elevation')
> plot(bm$X, bm$Z,cex=0.9,pch=20,asp=1,col = dc, main='Visto do Sul',
xlab='Easting', ylab='Elevation')
> plot(bm$X, bm$Y,cex=0.02,pch=20,asp=1,col = dc, main='Visto do topo',
xlab='Easting', ylab='Northing')
```



Podemos fatiar o modelo usando.

```
> plot(bm$X[bmdf$Z > -190 & bmdf$Z < -185] , bm$Y[bmdf$Z > -190 & bmdf$Z < -
185],cex=0.02,pch=20,asp=1,col = dc, main='Vista topo 185m',xlab='Easting',
ylab='Northing')
> plot(bm$X[bmdf$Z > -185 & bmdf$Z < -180] , bm$Y[bmdf$Z > -185 & bmdf$Z < -
180],cex=0.02,pch=20,asp=1,col = dc, main=' Vista topo 180m',xlab='Easting',
ylab='Northing')
> plot(bm$X[bmdf$Z > -180 & bmdf$Z < -175] , bm$Y[bmdf$Z > -180 & bmdf$Z < -
175],cex=0.02,pch=20,asp=1,col = dc, main=' Vista topo 175m',xlab='Easting',
ylab='Northing')
```



Essa visualização é rápida e prática mas existem forma mais eficientes de visualizarmos os resultados do modelo de bloco usando Paraview, mas primeiro precisaremos de criar arquivos do tipo **VTK**. A seguir veremos como fazer isso usando R.

Visualizando os furos de sonda (criando arquivo vtk)

No código seguinte calcularemos a coordenada do fundo do furo usando as coordenadas do colar, direção e mergulho do furo. Este é um método calcularia as coordenadas intermediárias com base na informação da tabela survey, mas o presente conjunto de dados somente possui informação do colar e e profundidade total do poço, assim, na ausência da informação de survey, usaremos o início e o final do poço para definir a linha do furo de sonda para efeitos de simplificação. O objetivo aqui é mostrar como criar um arquivo .vtk que ilustra em 3D o furo de sonda com o Paraview.

Extraindo as coordenadas intermediárias do furo (nesse caso, somente início e fim).

```
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host='127.0.0.1',user='yourUserName',
             password='secret',dbname='jericho')
> collar<-dbGetQuery(con, "select TO_NUMBER(c.bhid,'99999999')
                  as holeid,s._at as depth, c.xcollar as x,
                  c.ycollar as y, c.zcollar as z, s.dip as dip,
                  s.az as az from collar as c, survey as s where
                  c.bhid = s.bhid and s._at=0")
> collar$dip<-collar$dip*-1
> station<-dbGetQuery(con, "SELECT TO_NUMBER(bhid,'99999999') as holeid,
                    _at as depth,null AS x, null AS y, null as z,
                     dip as dip, az FROM survey where _at>0")
> station$dip<-station$dip*-1
> tresD<-function(da,es){
 linha<-data.frame()
 for(i in 1:nrow(da)){
  x<-da[i,3]
  y<-da[i,4]
  z<-da[i,5]
  di<-0
  fur<-es[es[[1]]==da$holeid[[i]],]
```

```
  fur<-fur[order(fur$depth),]
  for(j in 1:nrow(fur)){
   angulo<-fur[j,7]
   d<-fur[j,2]-di
   di<-d+di
   deltaz<-abs(d*sin(fur[j,6]*pi/180))
   raio<-d*cos(fur[j,6]*pi/180)
   x<-x+raio*sin(angulo*pi/180)
   y<-y+raio*cos(angulo*pi/180)
   z<-z-deltaz
   linha<-rbind(linha,data.frame(holeid=fur[j,1] ,depth=fur[j,2],x=x,
                y=y ,z=z,dip=fur[j,6],az=fur[j,7]))
   z<-z
  }
 }
 lin<-rbind(linha,da)
 lin<-lin[order(lin$holeid,lin$depth),]
 return(lin)
}
> pts<-tresD(collar,station)
> head(pts)
   holeid depth          x          y           z dip az
74   1801     0 498605.000 7678697.00  204.0000000 -70 98
1    1801   290 498703.221 7678683.20  -68.5108600 -70 98
75   1802     0 498679.000 7679401.00  202.0000000 -70 88
2    1802   288 498777.442 7679404.44  -68.6314748 -70 88
76   1804     0 498450.000 7679950.00  200.0000000 -55 64
3    1804   405 498658.788 7680051.83 -131.7565779 -55 64
```

Note que cada furo apresenta somente dois pontos (início e final do furo). Caso existissem informações intermediárias de survey, estes pontos estariam presentes no data.frame resultante.

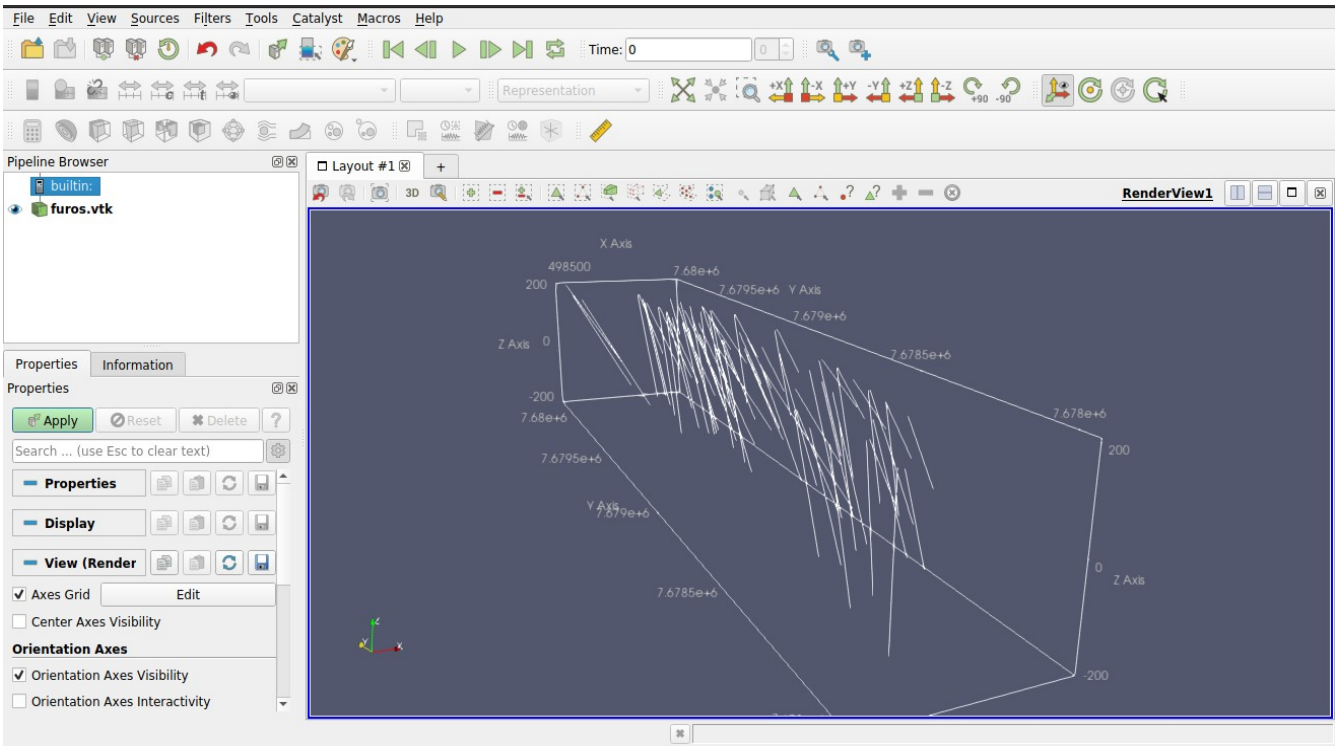
Agora criamos o arquivo **`furos.vtk`** para ser visualizado no Paraview.

```
> write('# vtk DataFile Version 4.2',file="furos.vtk")
> write('Furos de sonda Jericho',file="furos.vtk",append=TRUE)
> write('ASCII',file="furos.vtk",append=TRUE)
> write('DATASET POLYDATA',file="furos.vtk",append=TRUE)
> write(paste('POINTS',dim(pts)[1],'double'),file="furos.vtk",append=TRUE)
> for(v in 1:dim(pts)[1]){
   write(paste(pts$x[v],pts$y[v],pts$z[v]), file="furos.vtk",append=TRUE)
}
> write('',file="furos.vtk",append=TRUE)
> write('METADATA',file="furos.vtk",append=TRUE)
> write('INFORMATION 2',file="holes.vtk",append=TRUE)
> write('NAME L2_NORM_RANGE LOCATION vtkDataArray',file="furos.vtk",
      append=TRUE)
> write(paste('DATA 2 ',min(pts$y),max(pts$y)),file="furos.vtk",append=TRUE)
> write('NAME L2_NORM_FINITE_RANGE LOCATION vtkDataArray',file="furos.vtk",
      append=TRUE)
> write(paste('DATA 2 ',min(pts$y),max(pts$y)),file="furos.vtk",append=TRUE)
> write('',file="furos.vtk",append=TRUE)
> write(paste('LINES',dim(pts)[1]/2,(dim(pts)[1]/2)*3),file="furos.vtk",
      append=TRUE)
> for(i in seq(0, dim(pts)[1]-1, by = 2)){
  write(paste('2',i,i+1),file="furos.vtk", append=TRUE)
}
> write('',file="furos.vtk",append=TRUE)
> write('',file="furos.vtk",append=TRUE)
> write(paste('CELL_DATA',dim(pts)[1]/2),file="furos.vtk",append=TRUE)
> write('FIELD FieldData 1',file="furos.vtk",append=TRUE)
> write(paste('BHID 1',dim(pts)[1]/2,'string'),file="furos.vtk",append=TRUE)
```

```
> for(i in seq(1, dim(pts)[1], by = 2)){
  write(pts$holeid[i],file="furos.vtk",append=TRUE)
}
> write('METADATA',file="furos.vtk",append=TRUE)
> write('INFORMATION 0',file="furos.vtk",append=TRUE)
> write('',file="furos.vtk",append=TRUE)
```

Caso não tenha o programa Paraview, o mesmo pode ser baixado gratuitamente em : https://www.paraview.org/download/

Abrindo o arquivo furos.vtk usando o Paraview.



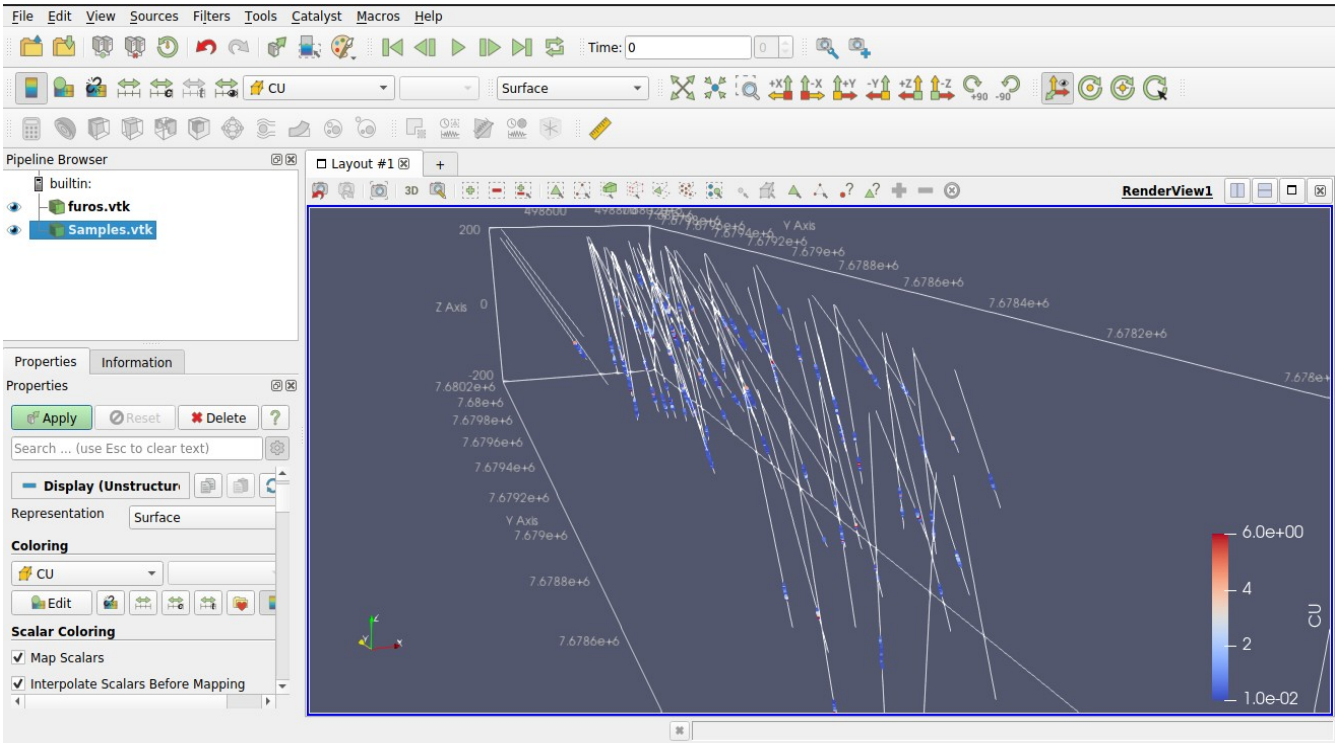
Visualizando as amostras (criando arquivo vtk)

Agora usando o arquivo **`dataL1dcl.dat`** criado previamente, vamos criar o arquivo **`samples.vtk`** com a posição das amostras e o teor de Cobre em cada uma delas.

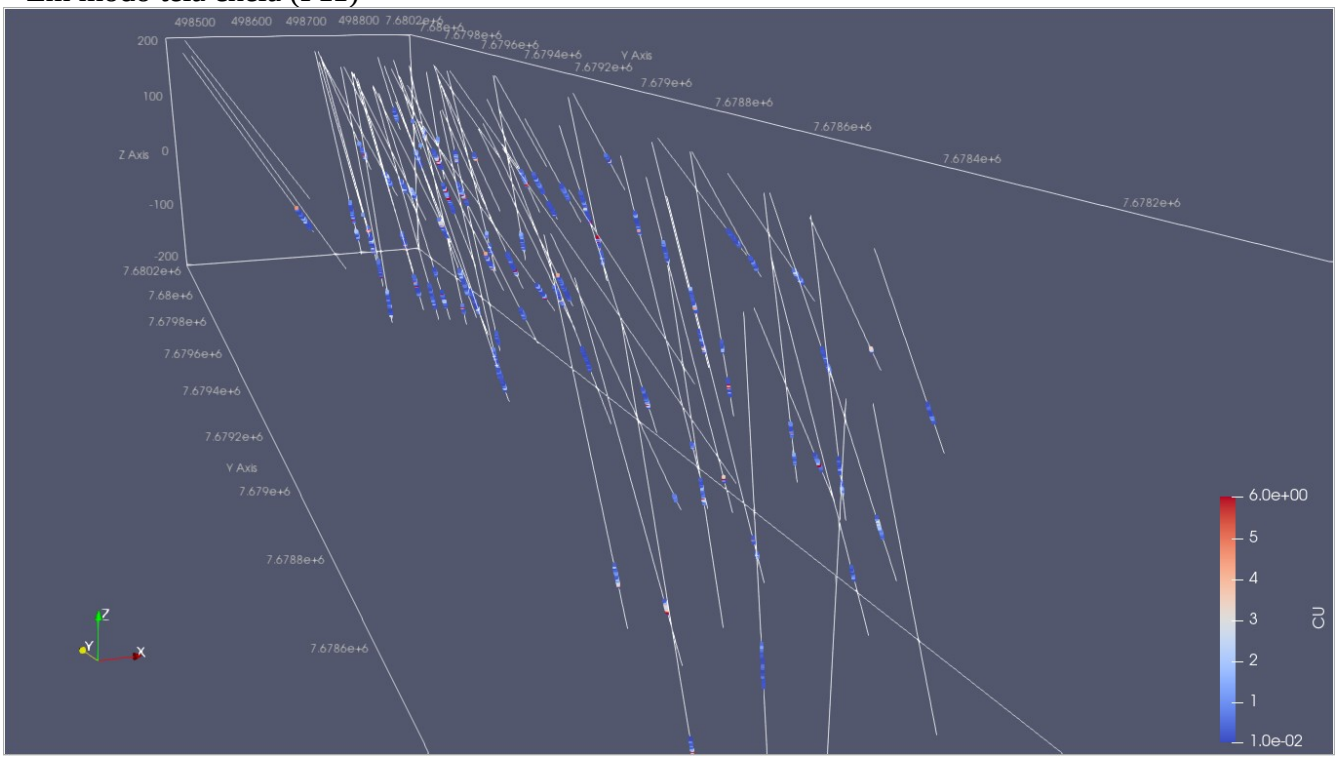
```
> dcl<-read.table('dataL1dcl.dat',skip=19)
> head(dcl,n=1)
> names(dcl)<-c('bhidnum','xm','ym','zm','xb','yb','zb','xe','ye','ze','dhdm',
              'dhdb','dhde','len','cu','au','wght')
> write('# vtk DataFile Version 4.2',file="Samples.vtk")
> write('Samples Jericho',file="Samples.vtk",append=TRUE)
> write('ASCII',file="Samples.vtk",append=TRUE)
> write('DATASET UNSTRUCTURED_GRID',file="Samples.vtk",append=TRUE)
> write(paste('POINTS',dim(dcl)[1],'double'),file="Samples.vtk",append=TRUE)
> for(v in 1:dim(dcl)[1]){
  write(paste(dcl$xm[v],dcl$ym[v],dcl$zm[v]), file="Samples.vtk",append=TRUE)
}
> write('',file="Samples.vtk",append=TRUE)
> write('METADATA',file="Samples.vtk",append=TRUE)
> write('INFORMATION 2',file="Samples.vtk",append=TRUE)
> write('NAME L2_NORM_RANGE LOCATION vtkDataArray',file="Samples.vtk",
      append=TRUE)
> write(paste('DATA 2 ',min(dcl$ym),max(dcl$ym)),file="Samples.vtk",append=TRUE)
```

```
> write('NAME L2_NORM_FINITE_RANGE LOCATION vtkDataArray',file="Samples.vtk",
      append=TRUE)
> write(paste('DATA 2 ',min(dcl$ym),max(dcl$ym)),file="Samples.vtk",append=TRUE)
> write('',file="Samples.vtk",append=TRUE)
> write(paste('CELLS',dim(dcl)[1],dim(dcl)[1]*2),file="Samples.vtk",append=TRUE)
> for(i in seq(0, dim(dcl)[1]-1, by = 1)){
  write(paste('1',i),file="Samples.vtk", append=TRUE)
}
> write('',file="Samples.vtk",append=TRUE)
> write(paste('CELL_TYPES',dim(dcl)[1]),file="Samples.vtk",append=TRUE)
> for(i in 1:dim(dcl)[1]){
  write('1',file="Samples.vtk",append=TRUE)
}
> write('',file="Samples.vtk",append=TRUE)
> write(paste('CELL_DATA',dim(dcl)[1]),file="Samples.vtk",append=TRUE)
> write('FIELD FieldData 1',file="Samples.vtk",append=TRUE)
> write(paste('CU 1',dim(dcl)[1],'double'),file="Samples.vtk",append=TRUE)
> for(i in seq(1, dim(dcl)[1], by = 1)){
 write(paste(dcl$cu[i]), file="Samples.vtk",append=TRUE)
}
> write('METADATA',file="Samples.vtk",append=TRUE)
> write('INFORMATION 0',file="Samples.vtk",append=TRUE)
> write('',file="Samples.vtk",append=TRUE)
```

Abra o arquivo samples.vtk no paraview e selecionet Cu como o campo de cor, ajuste o tamanho do ponto para 4 e reescale o campo de vlores de Cu para de 0 a 6&. Voce deverá ver algo semelhante à imagem abaixo (o arquivo **`furos.vtk`** é tambem mostrado como linhas brancas).



Em modo tela cheia (F11)



Visualizando modelo de bloco (criando arquivo vtk)

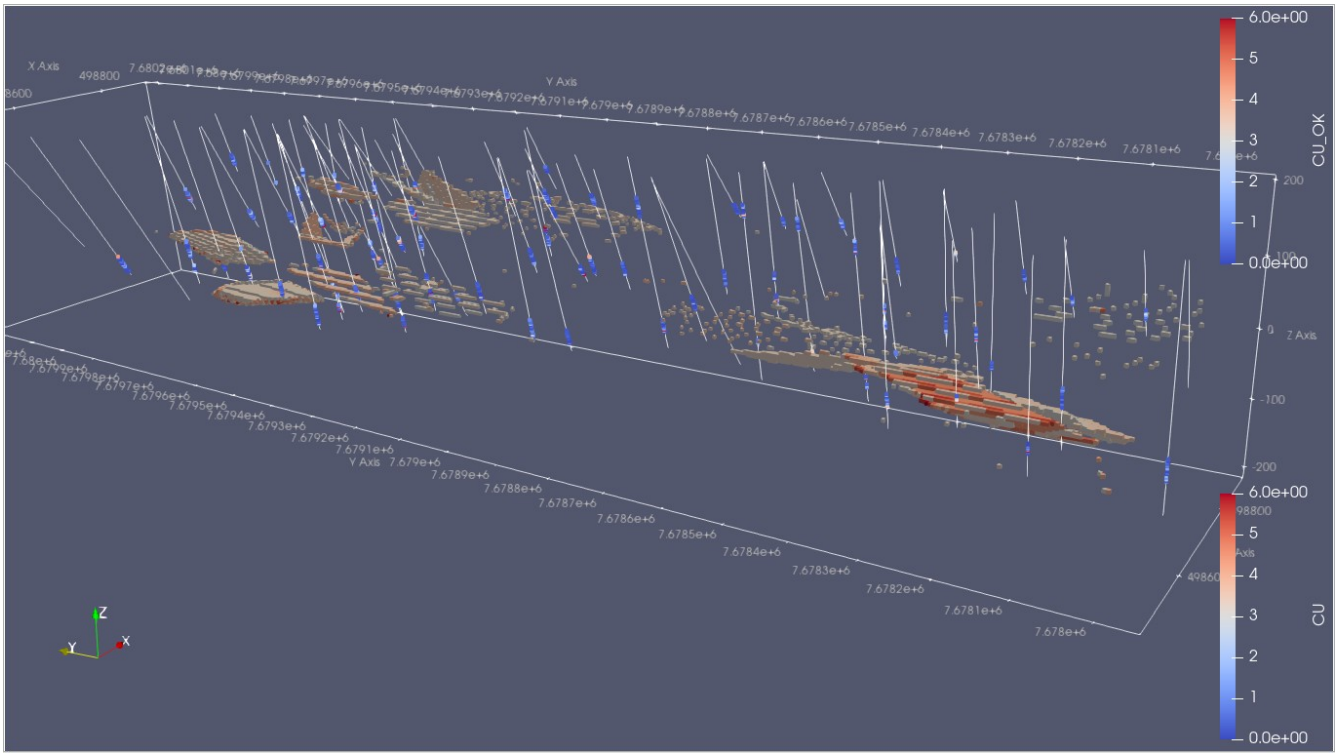
A series of vtk files, representing different cutoff grades, will be generated now. We will create 3.0, 3.5, 4.0, 4.5 and 5.0% cutoff files.

```
> kri<-read.table('kt3d.out',skip=4)
> options(digits = 7)
> bmdf<-data.frame(X=rep(seq(498451.5,498946.5,5),35920),
                   Y=rep(seq(7677950.5,7680190.5,5),each=100),
                   Z=rep(seq(-187.5,207.5,5),each=44900),
                   Cu=kri$V1, var=kri$V2)
> bmdf[bmdf == -999] <- NA
> blockMaker<-function(grade,fileName){
 bm<-bmdf[which(bmdf$Cu>grade),]
 write('# vtk DataFile Version 4.2',file=fileName)
 write('Block Model Jericho',file=fileName,append=TRUE)
 write('ASCII',file=fileName,append=TRUE)
 write('DATASET UNSTRUCTURED_GRID',file=fileName,append=TRUE)
 write(paste('POINTS',dim(bm)[1]*8,'float'),file=fileName,append=TRUE)
 for(v in 1:dim(bm)[1]){
  write(paste(bm$X[v]-2.5,bm$Y[v]-2.5,bm$Z[v]-2.5, bm$X[v]+2.5,bm$Y[v]-2.5,
      bm$Z[v]-2.5), file=fileName,append=TRUE)
  write(paste(bm$X[v]-2.5,bm$Y[v]+2.5,bm$Z[v]-2.5, bm$X[v]+2.5,bm$Y[v]+2.5,
      bm$Z[v]-2.5), file=fileName,append=TRUE)
  write(paste(bm$X[v]-2.5,bm$Y[v]-2.5,bm$Z[v]+2.5, bm$X[v]+2.5,bm$Y[v]-2.5,
      bm$Z[v]+2.5), file=fileName,append=TRUE)
  write(paste(bm$X[v]-2.5,bm$Y[v]+2.5,bm$Z[v]+2.5, bm$X[v]+2.5,bm$Y[v]+2.5,
      bm$Z[v]+2.5), file=fileName,append=TRUE)
 }
 write('',file=fileName,append=TRUE)
 write(paste('CELLS',dim(bm)[1],dim(bm)[1]*9),file=fileName,
      append=TRUE)
 for(i in seq(0, dim(bm)[1]*8-1, by = 8)){
  write(paste('8',i,i+1,i+2,i+3,i+4,i+5,i+6,i+7,collapse=" "),
      file=fileName, append=TRUE)
```

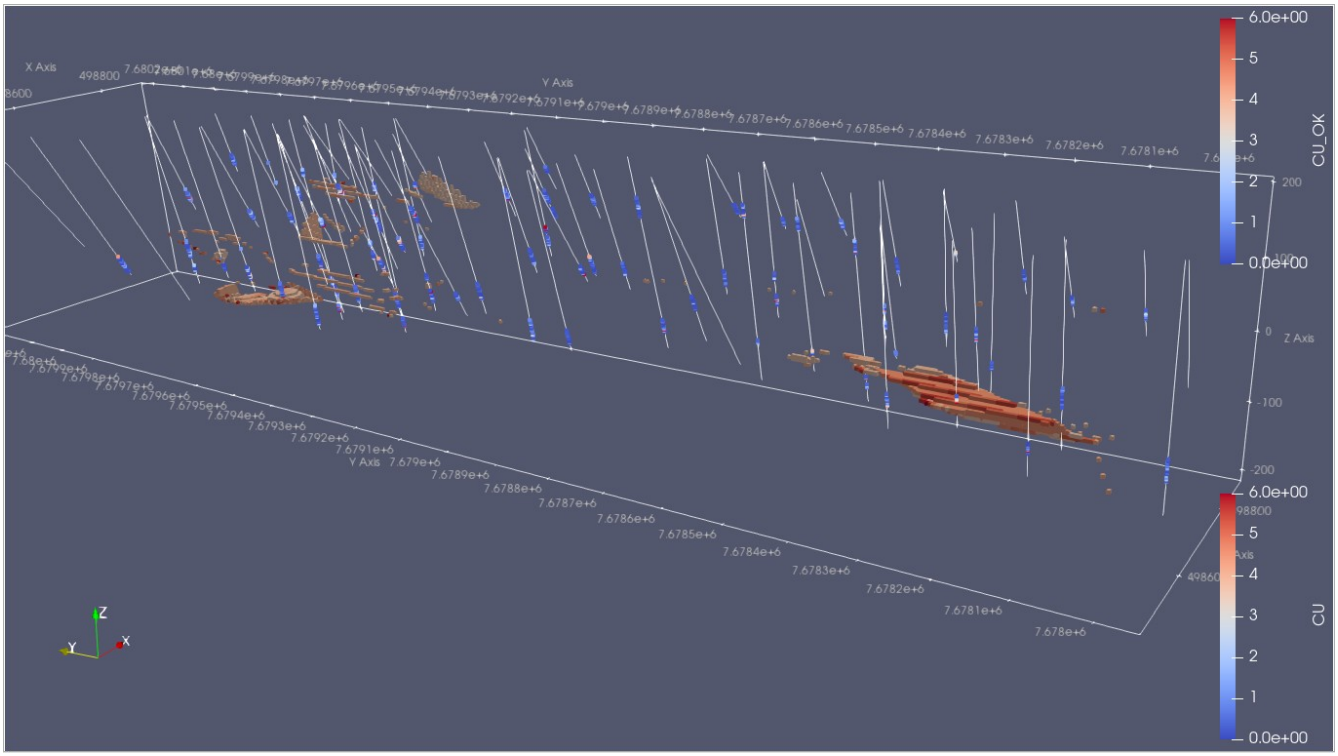
```
 }
 write('',file=fileName,append=TRUE)
 write(paste('CELL_TYPES',dim(bm)[1]),file=fileName,append=TRUE)
 for(i in 1:dim(bm)[1]){
  write('11',file=fileName,append=TRUE)
 }
 write('',file=fileName,append=TRUE)
 write(paste('CELL_DATA',dim(bm)[1]),file=fileName,append=TRUE)
 write('FIELD FieldData 1',file=fileName,append=TRUE)
 write(paste('CU_OK 1',dim(bm)[1],'float'),file=fileName,append=TRUE)
 for(i in seq(1, dim(bm)[1], by = 8)){
  write(paste(bm$Cu[i],bm$Cu[i+1],bm$Cu[i+2],bm$Cu[i+3],bm$Cu[i+4],
      bm$Cu[i+5],bm$Cu[i+6],bm$Cu[i+7],collapse=" "),
      file=fileName,append=TRUE)
 }
 write('METADATA',file=fileName,append=TRUE)
 write('INFORMATION 0',file=fileName,append=TRUE)
 write('',file=fileName,append=TRUE)
 print(paste(fileName,' criado...'))
 print(paste(' Cutoff: ', grade,'%'))
 print(paste(' Teor Médio: ',mean(bm$Cu),'%'))
 print(paste(' Volume: ',dim(bm)[1]*25,' cubic meters'))
 print(paste(' Tonelagem: ',dim(bm)[1]*25*3,' ton, assumindo densidade de 3 ton
por metro cúbico'))
 tx  <- readLines(fileName)
 tx2  <- gsub(pattern = "NA", replace = "", x = tx)
 writeLines(tx2, con=fileName)
}
> blockMaker(3.0,'block_3.vtk')
[1] "block_3.vtk  criado..."
[1] " Cutoff:  3 %"
[1] " Teor Médio:  3.58066972841469 %"
[1] " Volume:  289375  cubic meters"
[1] " Tonelagem:  868125  ton, assumindo densidade de 3 ton por metro cúbico"
> blockMaker(3.5,'block_35.vtk')
[1] "block_35.vtk  criado..."
[1] " Cutoff:  3.5 %"
[1] " Teor Médio:  4.22348828407216 %"
[1] " Volume:  97000  cubic meters"
[1] " Tonelagem:  291000  ton, assumindo densidade de 3 ton por metro cúbico"
> blockMaker(4.0,'block_4.vtk')
[1] "block_4.vtk  criado..."
[1] " Cutoff:  4 %"
[1] " Teor Médio:  4.55287305340017 %"
[1] " Volume:  57350  cubic meters"
[1] " Tonelagem:  172050  ton, assumindo densidade de 3 ton por metro cúbico"
> blockMaker(4.5,'block_45.vtk')
[1] "block_45.vtk  criado..."
[1] " Cutoff:  4.5 %"
[1] " Teor Médio:  4.70371268715881 %"
[1] " Volume:  40300  cubic meters"
[1] " Tonelagem:  120900  ton, assumindo densidade de 3 ton por metro cúbico"
> blockMaker(5.0,'block_5.vtk')
[1] "block_5.vtk  criado..."
[1] " Cutoff:  5 %"
[1] " Teor Médio:  5.39973631633663 %"
[1] " Volume:  5050  cubic meters"
[1] " Tonelagem:  15150  ton, assumindo densidade de 3 ton por metro cúbico"
```

O código acima mostrou o recurso para cada cutoff passado e criou os seguintes arquivos vtk para cada cutoff.

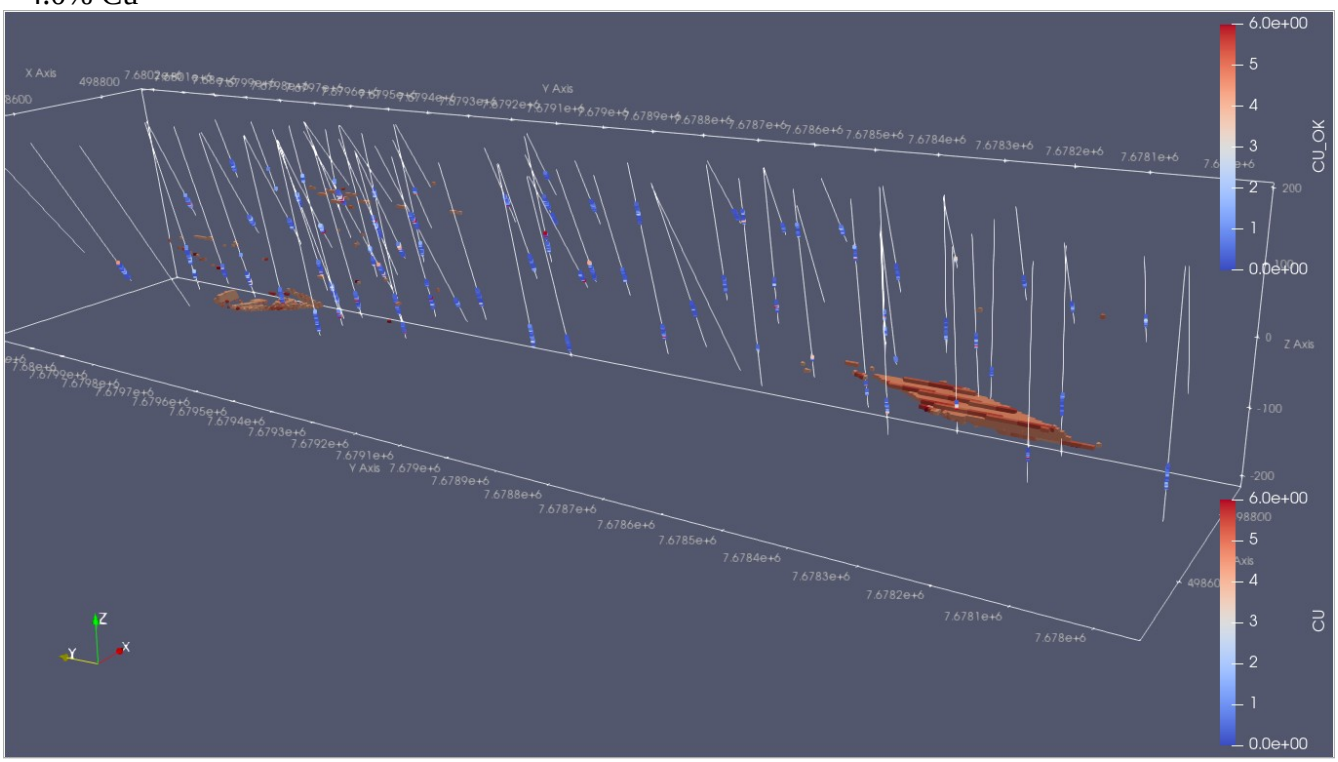
3.0% Cu


X Axis
Y Axis
Z Axis
CU_OK
CU
6.0e+00
5
4
3
2
1
0.0e+00
200
0
-100
-200

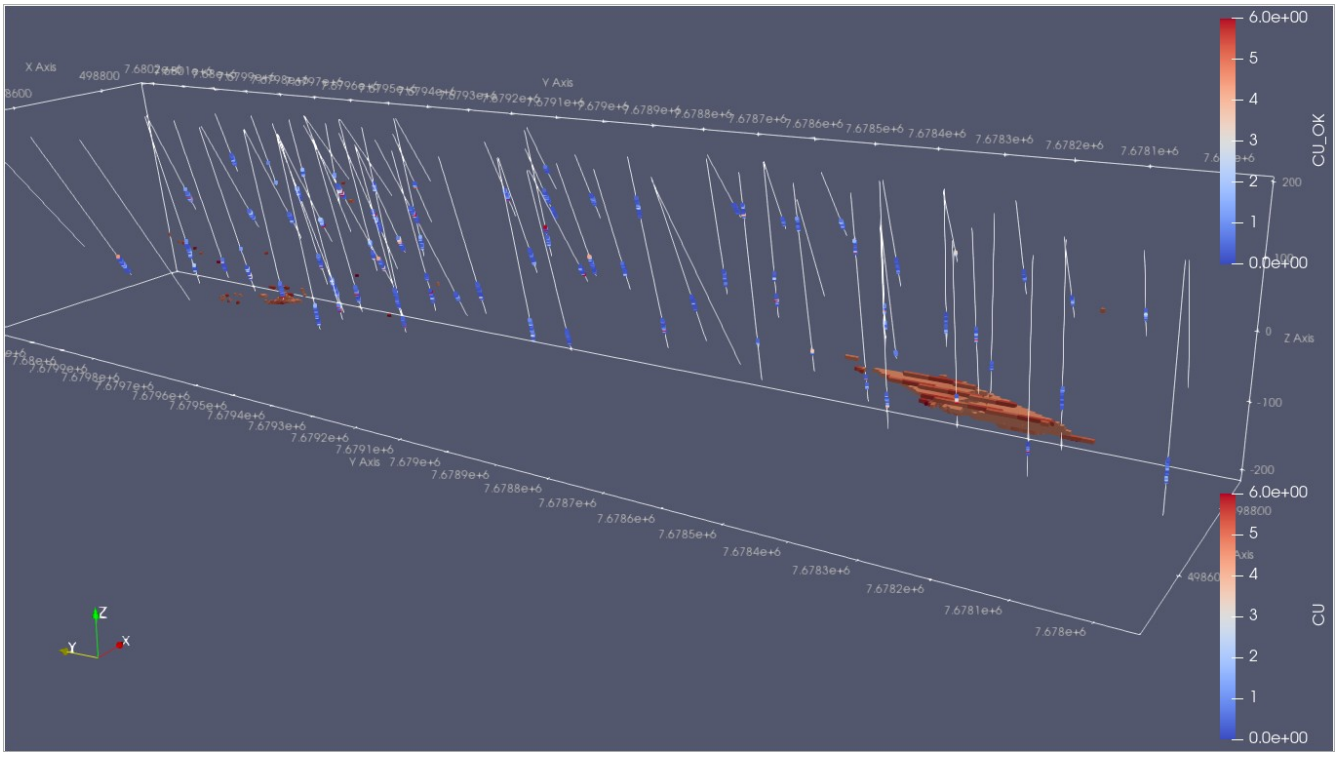
3.5% Cu


X Axis
Y Axis
Z Axis
CU_OK
CU
6.0e+00
5
4
3
2
1
0.0e+00
200
0
-100
-200

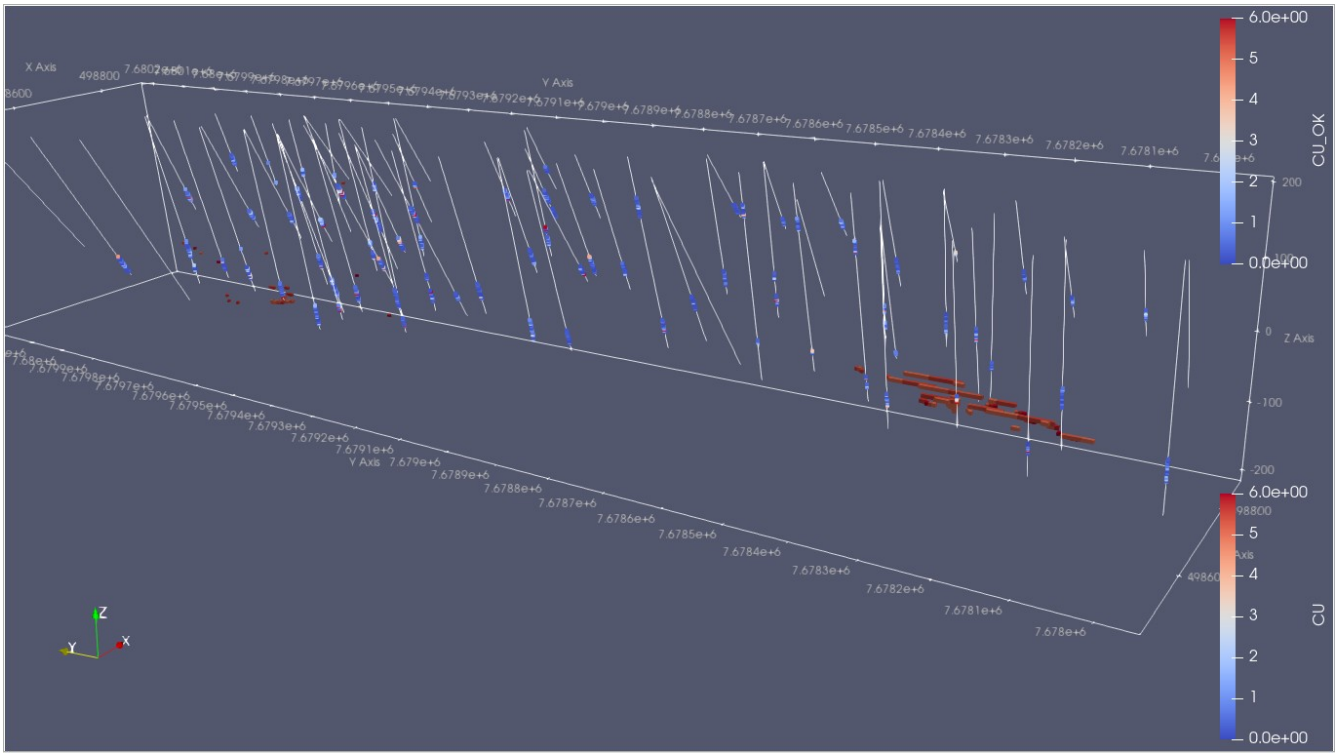
4.0% Cu


X Axis
Y Axis
Z Axis
CU_OK
CU
6.0e+00
5
4
3
2
1
0.0e+00
200
0
-100
-200

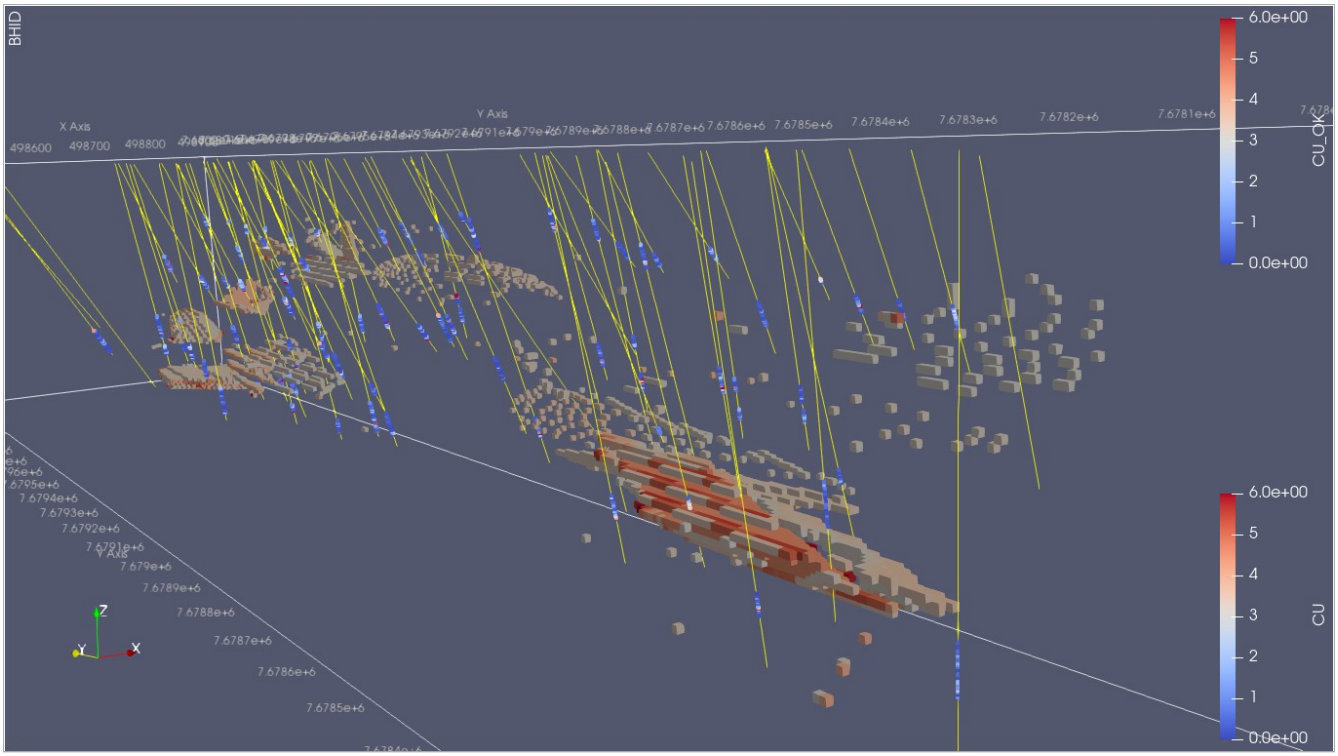
4.5% Cu


X Axis
Y Axis
Z Axis
CU_OK
CU
6.0e+00
5
4
3
2
1
0.0e+00
200
0
-100
-200

5.0% Cu



Visualização completa dos dados com Paraview.



6.3 – Curvas LAS e exemplo de análise petrofísica básica

Log ASCII Standard (ou LAS) é um formato comum de armazenamento de levantamentos de geofísicos comuns em poços de exploração de petróleo e gás e menos conhecido em exploração mineral.

6.3.1 – Lendo o arquivo LAS e carregando em formato data.frame

Vamos mostrar como ler um arquivo LAS versão 2 (unwraped) abaixo. As informações extraídas serão as curvas e seus respectivos valores. Valores nulos, comumente apresentados como -999.25, serão convertidos para NA no data.frame.

O arquivo LAS deste exemplo pode ser baixado em:
http://amazeone.com.br/barebra/pandora/1046531020.las

```
> las<-readLines('1046531020.las')
> data<-''
> hd<-''
> a<-1
> go<-0
> for (i in 1:length(las)){
 if(substr(las[i], 1, 2) == "~A"){
 data<- las[(i+1):length(las)]
 break
 }
}
> las<-readLines('1046531020.las')
> for (i in 1:length(las)){
 if(substr(las[i], 1, 2) == "~C"){go<-1}
 if(substr(las[i+1], 1, 2) == "~P" || substr(las[i+1], 1, 2) == "~A"
 || substr(las[i+1], 1, 2) == "~O"){break}
 if(go==1){value<-strsplit(trimws(las[i+1],"l"),'\\s{1,}')[[1]]
 hd[a]<-value[1]
 a<-a+1
 }
}
> las.data<-read.table(header=TRUE, text=data)
> names(las.data)<-hd
> las.data[las.data==-999.2500]<-NA
```

6.3.2 – Carregando os dados no banco de dados

Crie uma tabela básica para armazenar as curvas LAS. Colocaremos cada curva associada com sua profundidade e organizada por furo. Usando esse formato poderemos armazenar os dados de perfilagem de diversos furos em uma única tabela.

Primeiro criamos a tabela em um banco de dados já existente. Substitua o … com os valores apropriados para seu usuário e senha.

```
> library(RPostgreSQL)
>  con<-dbConnect(PostgreSQL(),host="127.0.1",  user=  "andre",password="devcor",
dbname="geobanco")
> sql<-"CREATE TABLE dhd_las (dhd_hole_id varchar(20), dhd_curve varchar(20),
> dhd_depth numeric(12,2),dhd_value numeric(12,2));"
> dbGetQuery(con,sql)
```

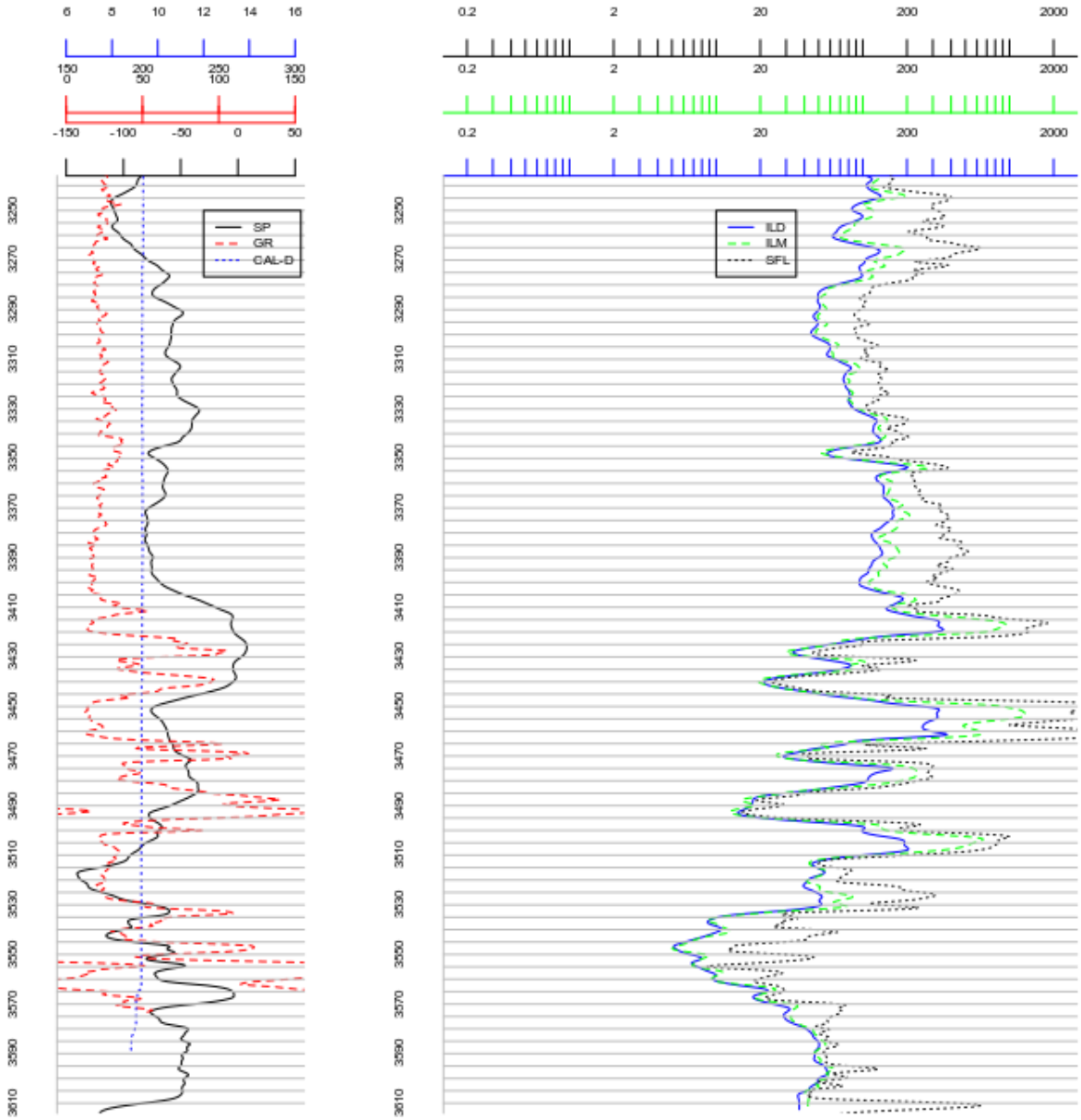
Agora vamos carregar o dataframe no banco de dados. Um arquivo texto chamado templas.csv será criado antes de carregarmos os dados (esse passo poderá demorar um ou dois minutos)

```
> line<-""
> cat(line, file="templas.csv", append=FALSE, sep = "")
> for(i in 2:ncol(las.data)){
   for(j in 1:nrow(las.data)){
   line2<-paste0(" 1046531020,",names(las.data) [i],",",las.data[j,1],
",",las.data[j,i])
   cat(line2, file="templas.csv", append=TRUE, sep = "\n")
  }
}
> system("psql -c \"\\COPY dhd_las FROM 'templas.csv' delimiter ',' csv NULL
'NA';\" -U usuário geobanco")
```

Vamos agora mostrar como plotar no formato tradicional.

```
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1", user='...', password='...',
dbname='geobanco')
> sql<-"select * from dhd_las where dhd_hole_id=' 1046531020'"
> las<-dbGetQuery(con,sql)
> las.df <- reshape(las,timevar = "dhd_curve",idvar = c("dhd_hole_id",
"dhd_depth"), direction = "wide")
> names(las.df)←c('holeID','depth','CILD','CALN','CALD', 'GR','ILD','ILM',
'MINV','LWTLB','CALM','MNOR','NPLS','DPLS','RXRT','SFL','SP','DRHO','RHOB','PE',
'INV_RF','NOR_RF')
> par(mar=c(1,2,5.5,2) + 0.1)
> layout(matrix(c(1,2),nrow=1), widths=c(1,2))
> plot(las.df$SP,las.df$depth,axes=FALSE,xlim=c(-150,50),type='l',xlab='',
ylab='',col='black',ylim=c(3600,3250))
> axis(3, xlim=c(-150,50),col='black',lwd=1,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.df$GR,las.df$depth,  axes=FALSE, xlim=c(0,150), type='l',xlab='',
ylab='',col='red',lty=2, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(0,150),col='red',lwd=1,line=1.6,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.df$GR,las.df$depth,  axes=FALSE, xlim=c(150,300), type='l',xlab='',
ylab='',col='red',lty=2, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(0,150),col='red',lwd=1,line=1.9,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.df$CALD,las.df$depth,  axes=FALSE, xlim=c(6,16), type='l',xlab='',
ylab='',col='blue',lty=3, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(6,16),col='blue',lwd=1,line=3.6,cex.lab=1,cex.axis=0.5)
> axis(2,pretty(range(las.df$depth),500),cex.lab=1,tck=1,cex.axis=0.5,
col='gray')
> mtext('depth',side=2,col="black",line=2)
> legend(x=12,y=3250,legend=c('SP','GR','CAL-D'),lty=c(1,2,3),col=c('black',
'red','blue'),cex=0.5)
> plot(las.df$ILD,las.df$depth,axes=FALSE,xlim=c(10^-1,10^3),type='l',xlab='',
ylab='',col='blue',ylim=c(3600,3250),log='x')
> at.y <- outer(1:9, (.5*10^(-1:3)))
> lab.y <- ifelse(log10(at.y) %% 1 == 0, at.y*2, NA)
> axis(3, xlim=c(10^-1,10^3),col='blue',lwd=1,cex.lab=1,cex.axis=0.5, at=at.y,
labels=lab.y, las=1)
> par(new=TRUE)
> plot(las.df$ILM,las.df$depth,  axes=FALSE, xlim=c(10^-1,10^3),
type='l',xlab='', ylab='',col='green',lty=2, ylim=c(3600,3250),lwd=1,log='x')
> axis(3, xlim=c(10^-1,10^3),col='green',lwd=1,line=1.9,cex.lab=1,cex.axis=0.5,
at=at.y, labels=lab.y, las=1)
> par(new=TRUE)
```

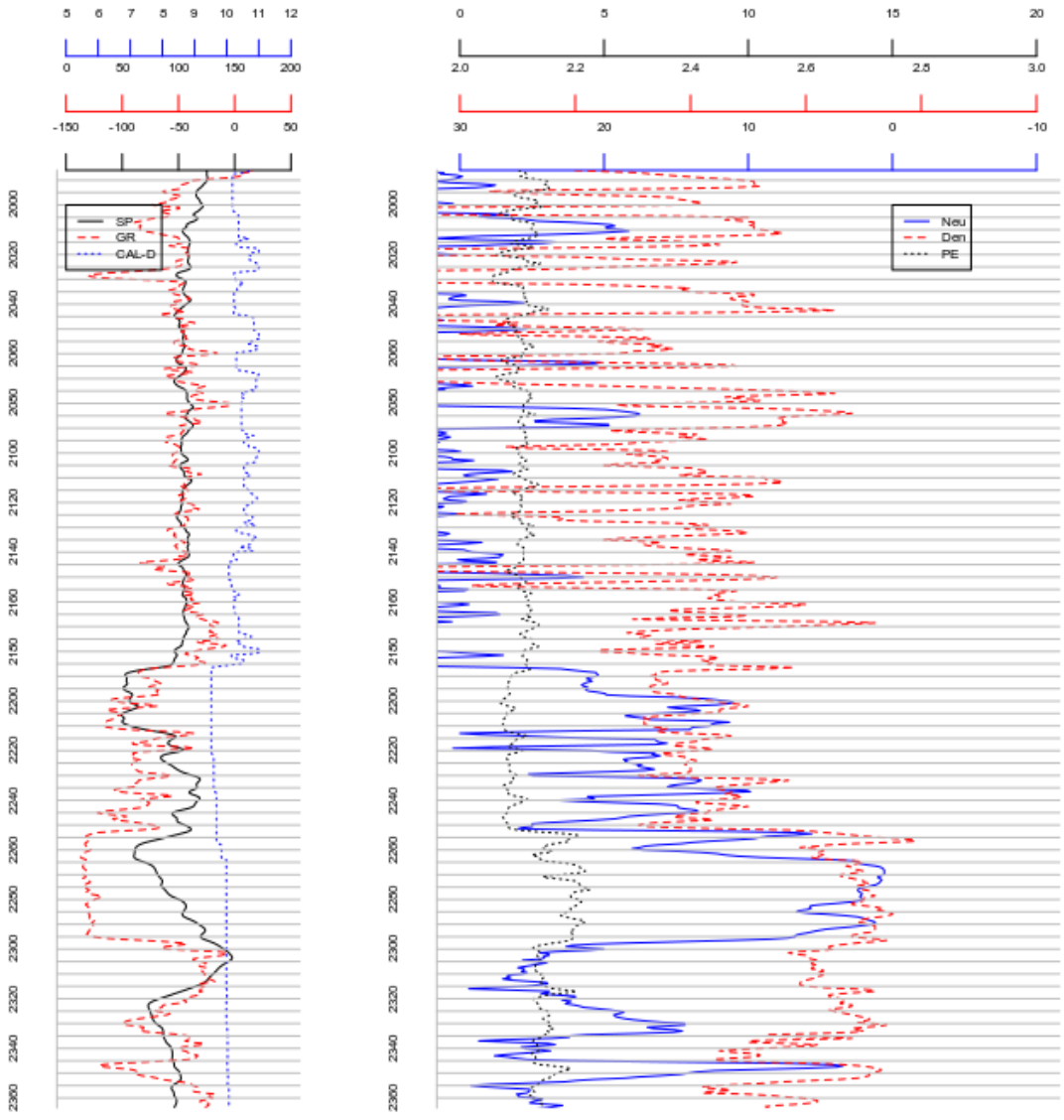
```
> plot(las.df$SFL,las.df$depth,  axes=FALSE, xlim=c(10^-1,10^3),
type='l',xlab='', ylab='',col='black',lty=3, ylim=c(3600,3250),lwd=1,log='x')
> axis(3, xlim=c(10^-1,10^3),col='black',lwd=1,line=3.6,cex.lab=1,cex.axis=0.5,
at=at.y, labels=lab.y, las=1)
> axis(2,pretty(range(las.df$depth),500),cex.lab=1,tck=1,cex.axis=0.5,
col='gray')
> legend(x=5,y=3250,legend=c('ILD','ILM','SFL'),lty=c(1,2,3),col=c('blue',
'green','black'),cex=0.5)
```



Ou:

```
> par(mar=c(1,2,5,2) + 0.1)
> layout(matrix(c(1,2),nrow=1), widths=c(1,2))
> plot(las.df$SP,las.df$depth,axes=FALSE,xlim=c(-150,50),type='l',xlab='',
ylab='',col='black',ylim=c(2350,2000))
> axis(3, xlim=c(-150,50),col='black',lwd=1,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.df$GR,las.df$depth,  axes=FALSE, xlim=c(0,200), type='l',xlab='',
ylab='',col='red',lty=2, ylim=c(2350,2000),lwd=1)
> axis(3, xlim=c(0,200),col='red',lwd=1,line=1.8,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
```

```
> plot(las.df$CALD,las.df$depth,  axes=FALSE, xlim=c(5,12), type='l',xlab='',
ylab='',col='blue',lty=3, ylim=c(2350,2000),lwd=1)
> axis(3, xlim=c(5,12),col='blue',lwd=1,line=3.5,cex.lab=1,cex.axis=0.5)
> axis(2,pretty(range(las.df$depth),500),cex.lab=1,tck=1,cex.axis=0.5,
col='gray')
> mtext('depth',side=2,col="black",line=2)
> legend(x=5,y=2000,legend=c('SP','GR','CAL-D'),lty=c(1,2,3),col=c('black',
'red','blue'),cex=0.5)
> plot(las.df$NPLS,las.df$depth,axes=FALSE,xlim=c(30,-10),type='l',xlab='',
ylab='',col='blue',ylim=c(2350,2000))
> axis(3, xlim=c(30,-10),col='blue',lwd=1,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.df$RHOB,las.df$depth,  axes=FALSE, xlim=c(2,3), type='l',xlab='',
ylab='',col='red',lty=2, ylim=c(2350,2000),lwd=1)
> axis(3, xlim=c(2.3),col='red',lwd=1,line=1.8,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> par(new=TRUE)
> plot(las.df$PE,las.df$depth,  axes=FALSE, xlim=c(0,20), type='l',xlab='',
ylab='',col='black',lty=3, ylim=c(2350,2000),lwd=1)
> axis(3, xlim=c(0,20),col='black',lwd=1,line=3.5,cex.lab=1,cex.axis=0.5)
> axis(2,pretty(range(las.df$depth),500),cex.lab=1,tck=1,cex.axis=0.5,
col='gray')
> legend(x=15,y=2000,legend=c('Neu','Den','PE'),lty=c(1,2,3),col=c('blue',
'red','black'),cex=0.5)
```



6.3.3 – Extraindo parâmetros petrofísicos

Existe muita coisa envolvida na análise petrofísica, aqui vamos manter simples por razões didáticas e ilustrativas. Aplicaremos essa análise no furo como um todo onde geralmente intervalos presselecionados são usados para ter maior representatividade.

Carregando as curvas

Selecionando curvas básicas que são usadas na análise petrofísica.

```
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host="...", user="...", password="...",
dbname="...")
> sql<-"select * from dhd_las where dhd_curve='GR' or dhd_curve='ILD'
or dhd_curve='ILM' or dhd_curve='NPLS' or dhd_curve='RHOB' or dhd_curve='PE';"
> las.petro<-dbGetQuery(con,sql)
```

Organizando o formato dos dados no formato 'wide' usando reshape e renomeando as colunas:

```
> petro.df <- reshape(las.petro,timevar = "dhd_curve",idvar = c("dhd_hole_id",
"dhd_depth"), direction = "wide")
> names(petro.df)<-c('holeID','depth','GR','ILD','ILM','NPLS','RHOB','PE')
```

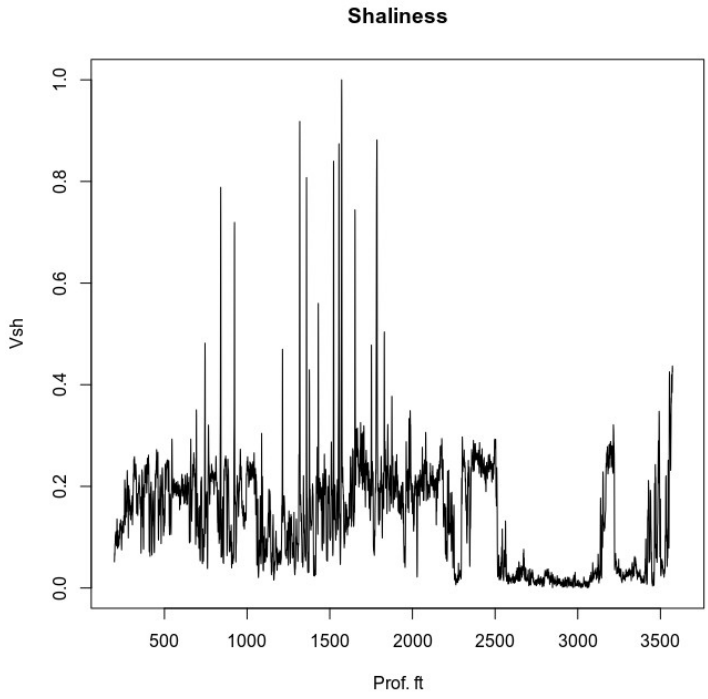
Calculando o VSh

Vamos agora calcular o primeiro parâmetro, a curva Vsh ou shaliness :

```
> GRmax<-max(petro.df$GR,na.rm=TRUE)
> GRmin<-min(petro.df$GR,na.rm=TRUE)
> petro.df$VSH<-(petro.df$GR-GRmin)/(GRmax-GRmin)
```

Podemos visualizar a curva Vsh usando:

```
> plot(petro.df$depth,petro.df$VSH,type='l',main='Shaliness',xlab='Prof. m',
ylab='Vsh')
```



Calculando a Porosidade

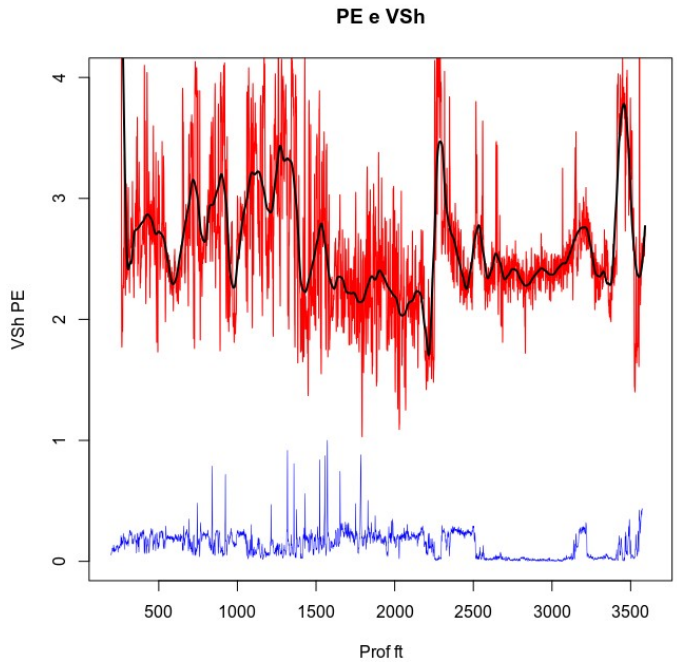
Antes de calcularmos a porosidade - densidade nós precisaremos de calcular dois parâmetros: Densidade da matriz e densidade do fluido. Nós assumiremos que a densidade do fluido seja 1 e a densidade da matriz será interpretada grosseiramente com a curva PE. Se PE é abaixo de 1 é carvão, se próximo a 2 é arenito, se próximo a 3 pode ser folhelho ou dolomito, se próximo a 5 é calcário ou anidrita. O valor de Vsh, que já foi calculado, pode ser usado para diferenciar folhelho de dolomito e a densidade pode ser usada para diferenciar anidrita de calcário.

Mas antes vamos filtrar os dados de PE com um filtro passa baixa.

```
> lowpass.PE <- loess(PE~depth,data=petro.df, span = 0.05)
> PE.lp<-predict(lowpass.PE, petro.df$depth)
```

Visualizando:

```
> plot(petro.df$depth,petro.df$PE,type='l',col='red',main='PE e VSh',xlab='Prof
ft', ylab='VSh PE',ylim=c(0,4))
> lines(petro.df$depth,PE.lp,type='l',lwd=2)
> lines(petro.df$depth,petro.df$VSH,type='l',lwd=.5,col='blue')
> petro.df<-cbind(petro.df,PE.lp)
```



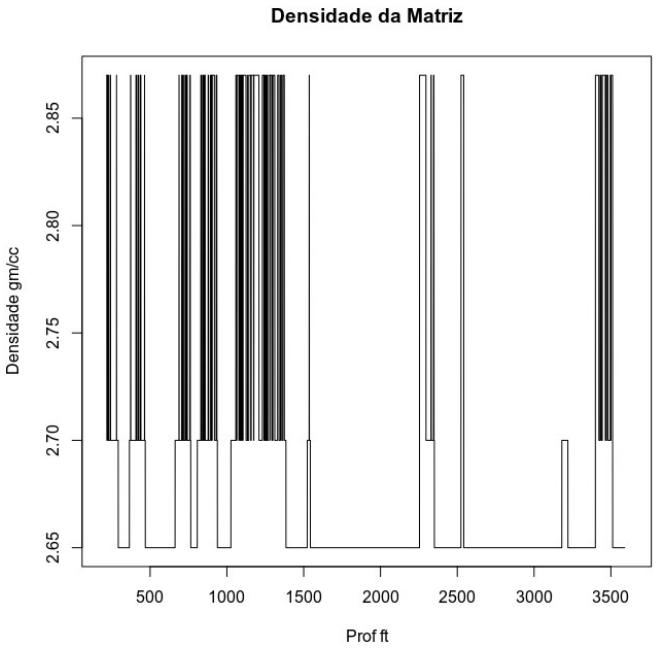
Com base nos valores de Pe e VSh(gamma) o poço pode ser dividido em 3 unidades principais: uma composta basicamente por folhelho e dolomita de 200 a 1600 pés, uma unidade siliciclástica composta por folhelho e arenito de 1600 a 3300 pés e uma unidade basal com a presença de folhelho e carbonatos a partir de 3300 pés.

Usaremos a porosidade da matriz de 2.87 gm/cc para o dolomito, 2.7 gm/cc para o folhelho e 2.65 gm/cc para o arenito. Pe < 2.5 será interpretado como arenito, Pe > 2.5 será folhelho quando o Vsh > 0.1 caso contrário será interpretado como dolomito.

Abaixo ajustaremos a densidade da matriz como o descrito acima. De uma forma geral executamos um processo de identificação litológica do poço como bônus nesse processo.

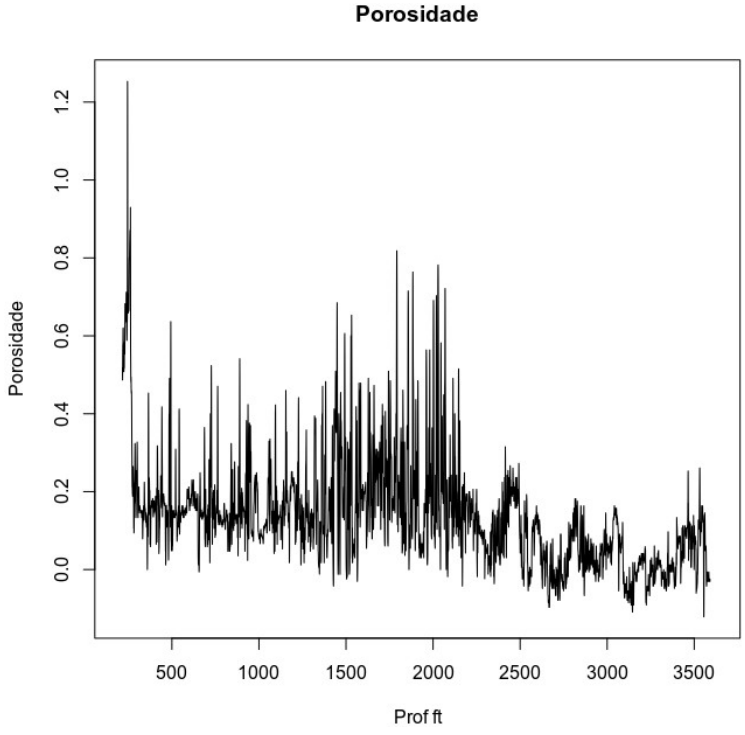
```
> petro.df$ro.matrix <- ifelse(petro.df$PE.lp >=2.75, ifelse(petro.df$VSH<
0.1,2.87,2.7), 2.65)
> plot(petro.df$depth,petro.df$ro.matrix,type='l',main='Densidade da Matrizy',
xlab='Prof ft',ylab='Densidade gm/cc')
```

NOTA - Lembre que usamos várias suposições neste processamento: Aplicamos no furo todo e não em intervalos alvo, usamos a densidade do fluido como sendo 1 e dividimos somente em três unidades litológicas alvo. Tudo isso diminuirá a precisão do tratamento mas o objetivo aqui é mostrar como o método é executado em R a partir de uma curva LAS que foi carregada no banco de dados.



Vamos agora calcular a porosidade usando um valor de densidade do fluido de 1.

```
> petro.df$phi<-(petro.df$ro.matrix-petro.df$RHOB)/(petro.df$ro.matrix-1)
> plot(petro.df$depth,petro.df$phi,type='l',main='Porosidade',xlab='Prof ft',
ylab='Porosidade')
```



Calculando o Sw

O primeiro passo é calcular o Rw a partir de uma “zona de água” (water zone) selecionada. Novamente, este é um exemplo didático e estamos aplicando para o poço inteiro, geralmente este processo é aplicado para intervalos de furo selecionados. Então, vamos usar um valor único de Rw que pode não ser apropriado para o poço inteiro pois a resistividade da zona da água tende a mudar com a profundidade.

Selecione um intervalo onde a resistividade é baixa e que seja bem poroso (arenito) para obter o valor de Rw. A 3042 pés de profundidade temos essa condição. Então calculamos o Rw usando:

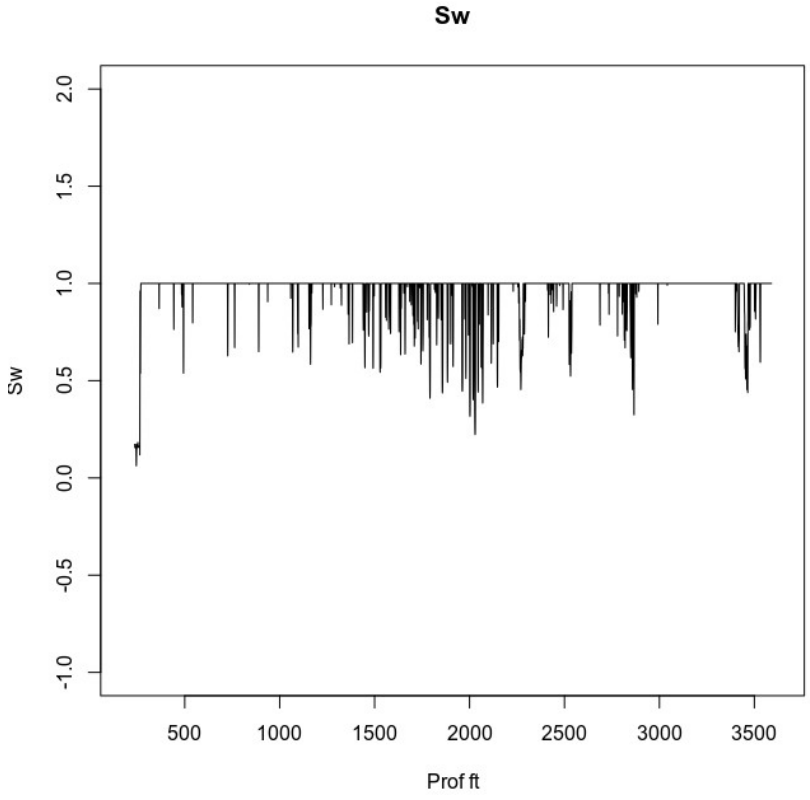
```
> petro.df[petro.df$depth == 3042,]
#          holeID depth    GR   ILD   ILM  NPLS RHOB   PE        VSH    PE.lp
#5696  1046531020  3042 14.91 17.93 19.78 25.97 2.38 2.35 0.01037538 2.427359
#     ro.matrix       phi
#5696      2.65 0.1636364
> Rt<-petro.df[petro.df$depth == 3042,]$ILD
> phiT<-petro.df[petro.df$depth == 3042,]$phi
> Rw<-phiT*phiT*Rt
> Rw
#[1] 0.4801091
```

Agora vamos calcular o Sw para o poço inteiro usando:

```
> petro.df$Sw<-1/petro.df$phi*sqrt(Rw/petro.df$ILD)
```

Valores maiores que 1 e menores que 0 serão convertidos para 1.

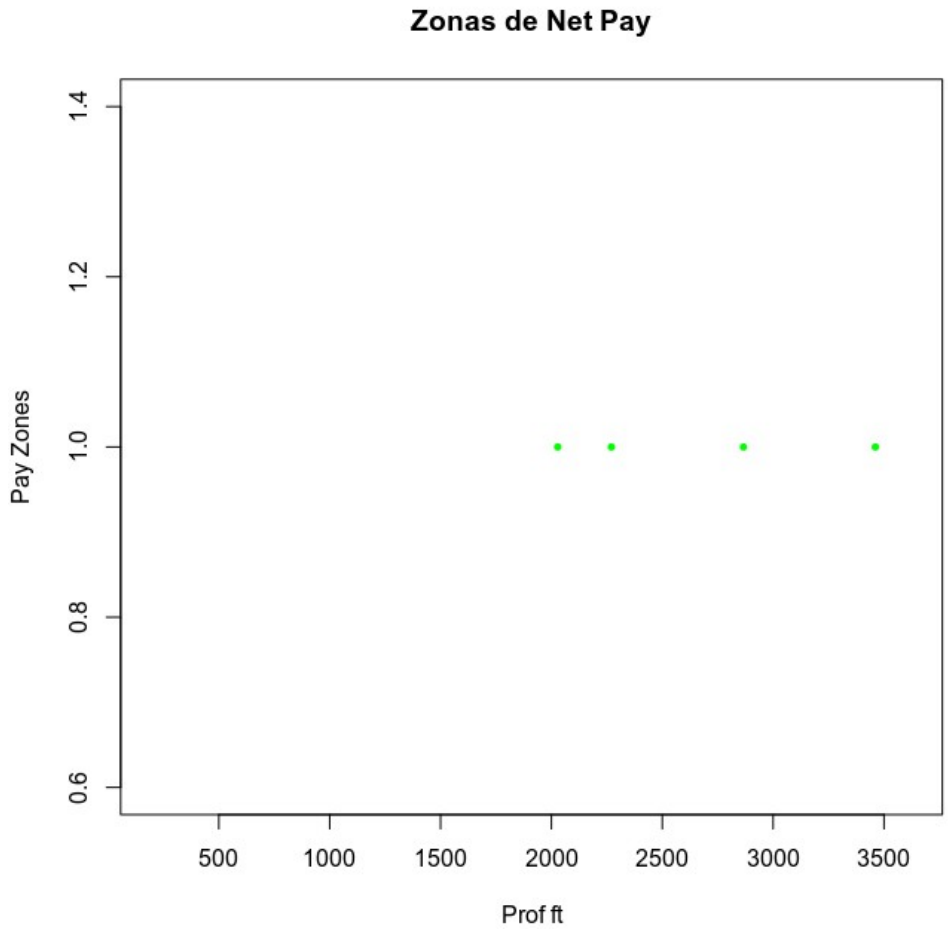
```
> index<- petro.df$Sw > 1
> petro.df$Sw[index]<- 1
> index<- petro.df$Sw < 0
> petro.df$Sw[index]<- 1
> plot(petro.df$depth,petro.df$Sw,type='l',main='Sw',xlab='Prof ft', ylab='Sw',
ylim=c(-1,2))
```



Calculando Net Reservoir e Net Pay

De forma simplificada, o Net Reservoir são as zonas onde o VSh é baixo e a porosidade é alta. O Net Pay são as zonas dentro do Net Reservoir onde a saturação de água e pequena. Usaremos os valores de corte de 0.05 para o Vsh, de 0.1 para a Porosidade e 0.5 para a Sw para definirmos nosso Net Pay.

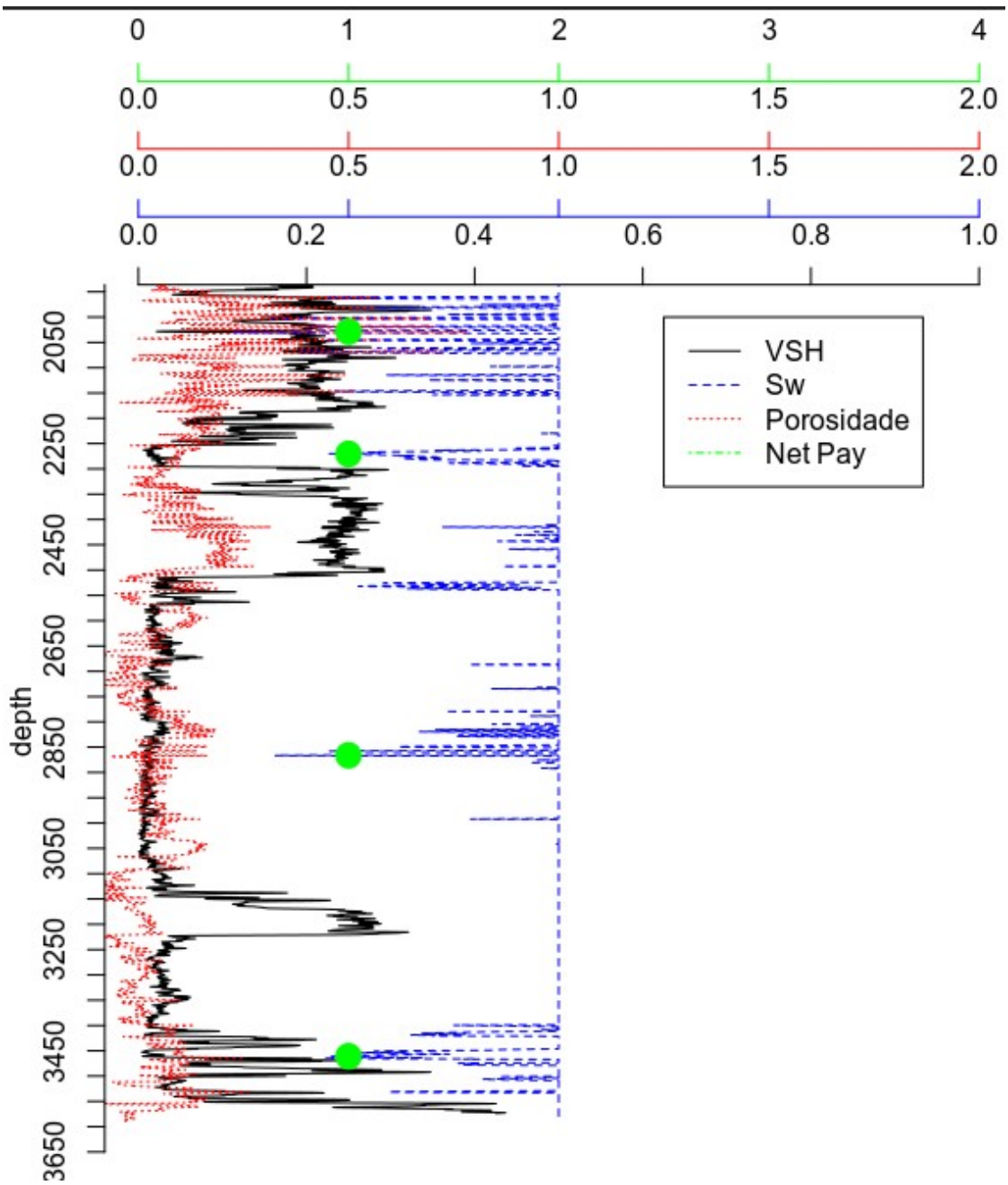
```
> petro.df$net.pay<-ifelse(petro.df$phi>0.1,ifelse(petro.df$VSH<0.05,
ifelse(petro.df$Sw<0.5,1,NA), NA),NA)
> plot(petro.df$depth,petro.df$net.pay,type='l',main='Zonas de Net
Pay',xlab='Prof ft',ylab='Pay Zones',lwd=5,col='green')
```



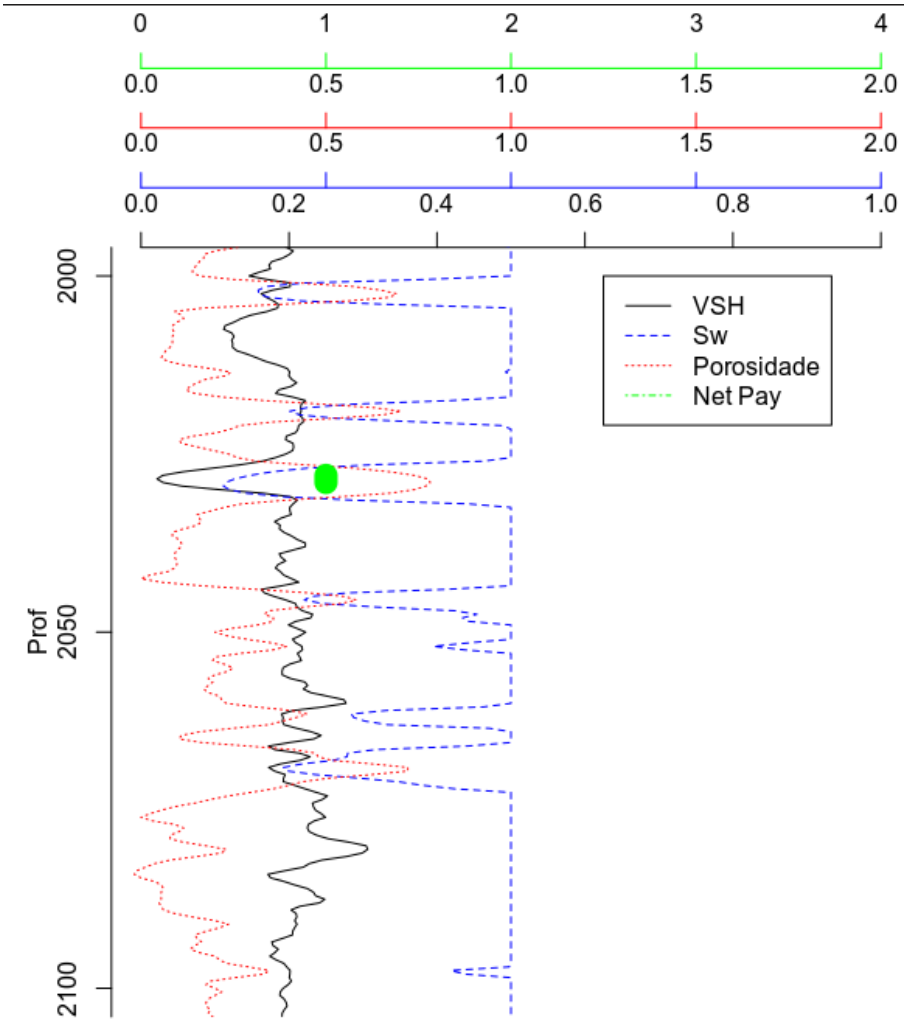
Vamos agora plotar os 4 parâmetros juntos num formato de melhor visualização e cada intervalo de Net Pay individualmente para ilustrar melhor o resultado.

```
> par(mar=c(1, 4, 8, 4) + 0.1)
> plot(petro.df$VSH,petro.df$depth,axes=FALSE,xlim=c(0,1),type='l',xlab='',
ylab='',col='black',ylim=c(3600,2000))
> axis(3, xlim=c(0,1),col='black',lwd=1)
> par(new=TRUE)
> plot(petro.df$Sw,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='blue',lty=2, ylim=c(3600,2000),lwd=1)
> axis(3, xlim=c(0,2),col='blue',lwd=1,line=2)
> par(new=TRUE)
> plot(petro.df$phi,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='red',lty=3, ylim=c(3600,2000),lwd=1)
> axis(3, xlim=c(0,2),col='red',lwd=1,line=4)
> par(new=TRUE)
> plot(petro.df$net.pay,petro.df$depth,  axes=FALSE, xlim=c(0,4),
type='l',xlab='', ylab='',col='green',lty=4,ylim=c(3600,2000),lwd=15)
```

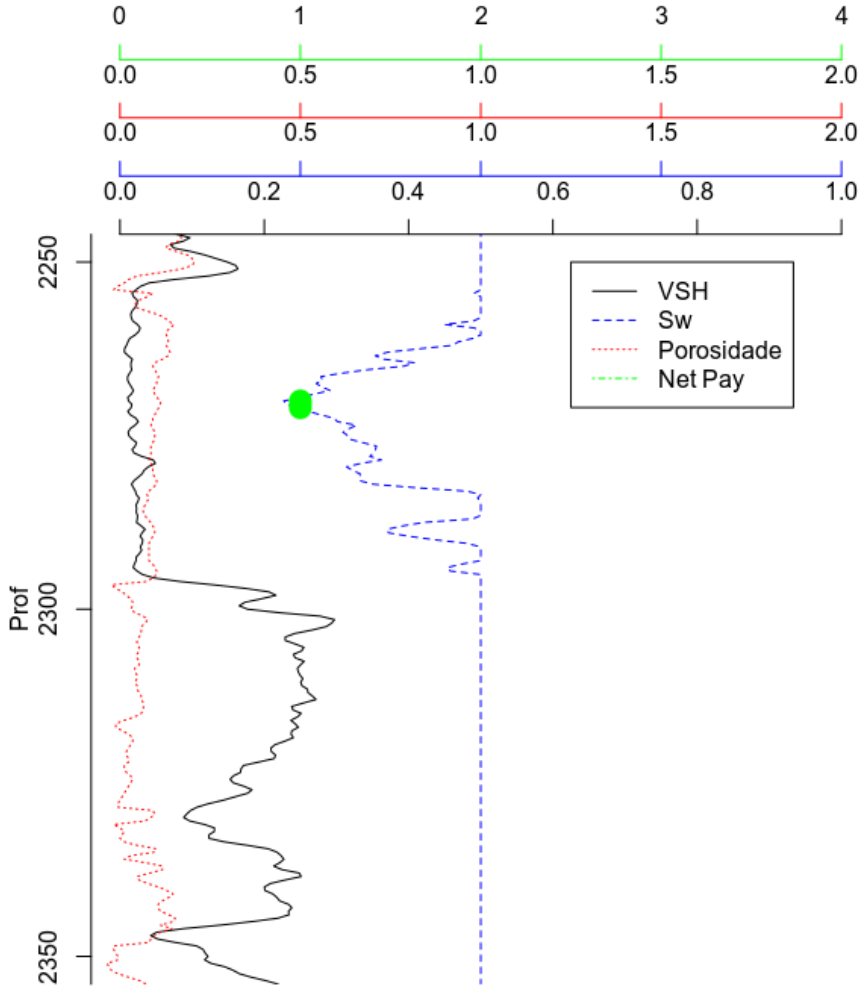
```
> axis(3, xlim=c(0,4),col='green',lwd=1,line=6)
> axis(2,pretty(range(petro.df$depth),100))
> mtext('depth',side=2,col="black",line=2)
> legend(x=2.5,y=2000,legend=c('VSH','Sw','Porosidade','Net
Pay'),lty=c(1,2,3,4),col=c('black','blue','red','green'))
```



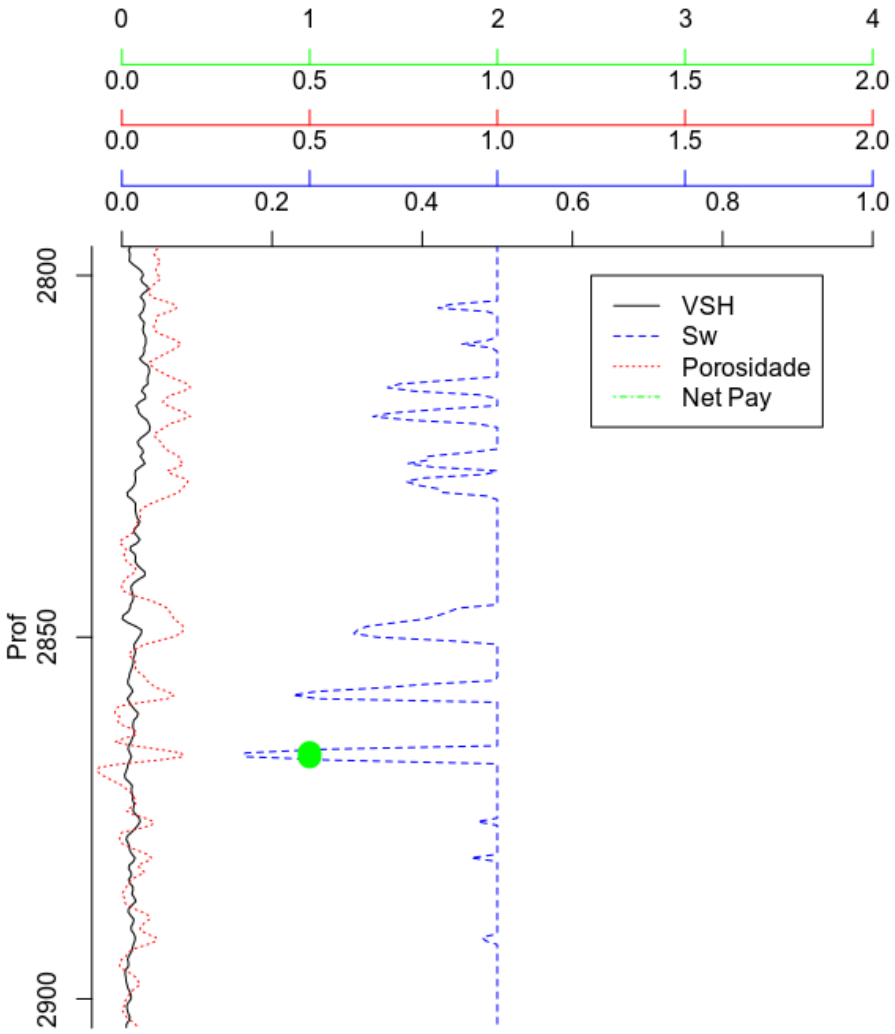
```
> par(mar=c(1, 4, 8, 4) + 0.1)
> plot(petro.df$VSH,petro.df$depth,axes=FALSE,xlim=c(0,1),type='l',xlab='',
ylab='',col='black',ylim=c(2100,2000))
> axis(3, xlim=c(0,1),col='black',lwd=1)
> par(new=TRUE)
> plot(petro.df$Sw,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='blue',lty=2, ylim=c(2100,2000),lwd=1)
> axis(3, xlim=c(0,2),col='blue',lwd=1,line=2)
> par(new=TRUE)
> plot(petro.df$phi,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='red',lty=3, ylim=c(2100,2000),lwd=1)
> axis(3, xlim=c(0,2),col='red',lwd=1,line=4)
> par(new=TRUE)
> plot(petro.df$net.pay,petro.df$depth,  axes=FALSE, xlim=c(0,4),
type='l',xlab='', ylab='',col='green',lty=4,ylim=c(2100,2000),lwd=15)
> axis(3, xlim=c(0,4),col='green',lwd=1,line=6)
> axis(2,pretty(range(petro.df$depth),100))
> mtext('Prof',side=2,col="black",line=2)
> legend(x=2.5,y=2000,legend=c('VSH','Sw','Porosidade','Net
Pay'),lty=c(1,2,3,4),col=c('black','blue','red','green'))
```



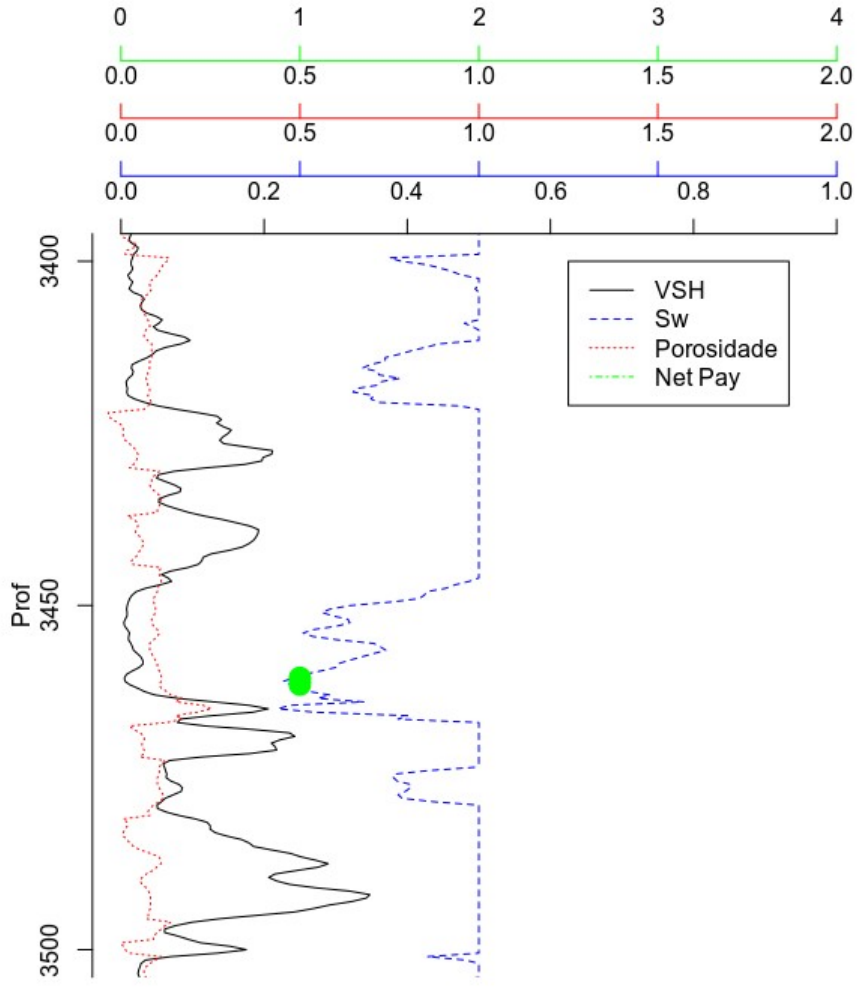
```
> par(mar=c(1, 4, 8, 4) + 0.1)
> plot(petro.df$VSH,petro.df$depth,axes=FALSE,xlim=c(0,1),type='l',xlab='',
ylab='',col='black',ylim=c(2350,2250))
> axis(3, xlim=c(0,1),col='black',lwd=1)
> par(new=TRUE)
> plot(petro.df$Sw,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='blue',lty=2, ylim=c(2350,2250),lwd=1)
> axis(3, xlim=c(0,2),col='blue',lwd=1,line=2)
> par(new=TRUE)
> plot(petro.df$phi,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='red',lty=3, ylim=c(2350,2250),lwd=1)
> axis(3, xlim=c(0,2),col='red',lwd=1,line=4)
> par(new=TRUE)
> plot(petro.df$net.pay,petro.df$depth,  axes=FALSE, xlim=c(0,4),
type='l',xlab='', ylab='',col='green',lty=4,ylim=c(2350,2250),lwd=15)
> axis(3, xlim=c(0,4),col='green',lwd=1,line=6)
> axis(2,pretty(range(petro.df$depth),100))
> mtext('Prof',side=2,col="black",line=2)
> legend(x=2.5,y=2250,legend=c('VSH','Sw','Porosidade','Net
Pay'),lty=c(1,2,3,4),col=c('black','blue','red','green'))
```



```
> par(mar=c(1, 4, 8, 4) + 0.1)
> plot(petro.df$VSH,petro.df$depth,axes=FALSE,xlim=c(0,1),type='l',xlab='',
ylab='',col='black',ylim=c(2900,2800))
> axis(3, xlim=c(0,1),col='black',lwd=1)
> par(new=TRUE)
> plot(petro.df$Sw,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='blue',lty=2, ylim=c(2900,2800),lwd=1)
> axis(3, xlim=c(0,2),col='blue',lwd=1,line=2)
> par(new=TRUE)
> plot(petro.df$phi,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='red',lty=3, ylim=c(2900,2800),lwd=1)
> axis(3, xlim=c(0,2),col='red',lwd=1,line=4)
> par(new=TRUE)
> plot(petro.df$net.pay,petro.df$depth,  axes=FALSE, xlim=c(0,4),
type='l',xlab='', ylab='',col='green',lty=4,ylim=c(2900,2800),lwd=15)
> axis(3, xlim=c(0,4),col='green',lwd=1,line=6)
> axis(2,pretty(range(petro.df$depth),100))
> mtext('Prof',side=2,col="black",line=2)
> legend(x=2.5,y=2800,legend=c('VSH','Sw','Porosidade','Net
Pay'),lty=c(1,2,3,4),col=c('black','blue','red','green'))
```



```
> par(mar=c(1, 4, 8, 4) + 0.1)
> plot(petro.df$VSH,petro.df$depth,axes=FALSE,xlim=c(0,1),type='l',xlab='',
ylab='',col='black',ylim=c(3500,3400))
> axis(3, xlim=c(0,1),col='black',lwd=1)
> par(new=TRUE)
> plot(petro.df$Sw,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='blue',lty=2, ylim=c(3500,3400),lwd=1)
> axis(3, xlim=c(0,2),col='blue',lwd=1,line=2)
> par(new=TRUE)
> plot(petro.df$phi,petro.df$depth,  axes=FALSE, xlim=c(0,2), type='l',xlab='',
ylab='',col='red',lty=3, ylim=c(3500,3400),lwd=1)
> axis(3, xlim=c(0,2),col='red',lwd=1,line=4)
> par(new=TRUE)
> plot(petro.df$net.pay,petro.df$depth,  axes=FALSE, xlim=c(0,4),
type='l',xlab='', ylab='',col='green',lty=4,ylim=c(3500,3400),lwd=15)
> axis(3, xlim=c(0,4),col='green',lwd=1,line=6)
> axis(2,pretty(range(petro.df$depth),100))
> mtext('Prof',side=2,col="black",line=2)
> legend(x=2.5,y=3400,legend=c('VSH','Sw','Porosidade','Net
Pay'),lty=c(1,2,3,4),col=c('black','blue','red','green'))
```



6.4 - Dados SEG-Y

O formato de arquivo SEG-Y é um dos padrões da Society of Exploration Geophysicists (SEG) para armazenar dados geofísicos de sísmica e GPR. É um padrão aberto e é controlado pelo comitê de padrões técnicos da SEG.

Arquivos SEG-Y são divididos em três partes principais (genericamente falando); Cabeçalho do arquivo (3200 bytes texto mais 400 bytes binário), Cabeçalho de Traço (240 bytes), e dado binário do Traço.

Mais informações sobre o formato podem ser encontradas em https://seg.org/Portals/0/SEG/News%20and%20Resources/Technical%20Standards/seg_y_rev1.pdf .

Vamos mostrar como ler este arquivo usando R.
Primeiro o Cabeçalho texto e binário:

```
> hdbn<-c('JOB_ID_NO','LINE_NUMBER','REEL_NUMBER','TRACES_PER_RECORD',
'AUXS_PER_RECORD','SMPL_RATE','SAMPLE_RATE_FIELD','NSAMPLES','NSAMPLES_FIELD',
'FORMAT_CODE','ENSEMBLE_FOLD','TRACE_SORT','VERTICAL_SUM','SWEEP_FREQ_START',
'SWEEP_FREQ_END','SWEEP_FREQ_LENGTH','SWEEP_TYPE','SWEEP_TRACE_NO',
'SWEEP_TAPER_LENGTH_START','SWEEP_TAPER_LENGTH_END','SWEEP_TAPER_TYPE',
'CORRELATED_TRACES','BIN_GAIN','AMP_RECOVERY_METHOD','UNITS','SIGNAL_POLARITY',
'VIBRATOR_POL_CODE','UNUSED','SEGY_FORMAT_REVISION_NO','SEGY_FIXEDLEN_FLAG',
'SEGY_NO_TEXTFHEADERS','UNUSED')
> con <- file("1104-30A.segy","rb")
> header.text<-c('bogus','data')
> for(i in 1:40){
  dec<-readBin(con,"raw",size=1,n=80)
  header.text[i]<-paste0(iconv(rawToChar(dec,multiple=T), 'cp500',
'utf8'),collapse='')
 }
> seek(con,3200) #go to byte 3201
> hd.bin<-c(1.0,1.7)
> step<-1
> for (i in step:(step+2)){
  hd.bin[step]<-readBin(con,'int',size=4,endian='big')
  step<-step+1
 }
> for (i in step:(step+23)){
  hd.bin[step]<-readBin(con,'int',size=2,endian='big')
  step<-step+1
 }
> hd.bin[step]<-0
> seek(con,3500)
#[1] 3260
> step<-step+1
> for (i in step:(step+2)){
   hd.bin[step]<-readBin(con,'int',size=2,endian='big')
   step<-step+1
 }
> hd.bin[step]<-0
> seek(con,3600)
#[1] 3506
> header.bin<-data.frame(field=hdbn,value=hd.bin)
```

Calculando o números de traços a amostras por traço

```
> fl.sz<-file.info('1104-30A.segy')$size
> fl.sz
#[1] 50995140
> header.repetitions<-header.bin[31,2]
> header.repetitions
#[1] 0
> num.samples<-header.bin[8,2]
> num.samples
#[1] 2751
> num.tr<-(fl.sz-3600-3600*header.repetitions)/(240+num.samples*4)
> num.tr
#[1] 4535
> seek(con,3600+3600*header.repetitions)
```

Lendo os dados dos traços:

```
> count<-c(7,4,8,2,4,46,5,2,3,4,6,1,2)
> byte.me<-c(4,2,4,2,4,2,4,2,2,2,2,2,4)
> trace.bin<-c(1,2)
> trace.header<-list(data.frame(field='oi',value=1))
> l.tr<-list(c(1,2,3))
> trace<-c(1,2,3)
> for (i in 1:num.tr){
  tbc<-1
  for (a in 1:13){
   for (c in 1:count[a]){
     trace.bin[tbc]<-readBin(con,'int',size=byte.me[a],endian='big')
     tbc<-tbc+1
   }
  }
  frame<-data.frame(value=trace.bin)
  trace.header[[i]]<-frame
  for(s in 1:num.samples){
   trace[s]<-readBin(con,'double',size=4,endian='big')
  }
  l.tr[[i]]<-as.single(trace)
 }
> close(con)
```

Colocando toda informação junta:

```
> segY<-list(header.text,header.bin,trace.header,l.tr)
```

E carregando no geobanco:

```
> segy.name<-'1104-30A'
> header.joined<-paste(segY[[1]],collapse='\n')
> header.text<-data.frame(segy=segy.name,text=header.joined)
> header.binary<-data.frame(segy=segy.name,segY[[2]])
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1", user= "...",
password="...", dbname="geobanco")
> dbWriteTable(con, "sgy_hd_txt",value=header.text,append= TRUE,
row.names=FALSE)
> dbWriteTable(con, "sgy_hd_bin",value=header.binary,append= TRUE,
row.names=FALSE)
> trhdbn<-c("TRACE_SEQ_NO","TRACE_SEQ_REEL","FIELD_RECORD_NO","CHANNEL_NO",
"SHOT_POINT_NO","CMP_NO","CMP_SEQ_NO","TRACE_ID_CODE","FOLD","TRACE_HSTACK",
"TEST_CODE","OFFSET_SH_REC","ELEV_REC","ELEV_SHOT","DEPTH_SHOT",
"DATUM_ELEV_REC","ELEV_FLOATDATUM_SRC","WATER_DEPTH_SHOT","WATER_DEPTH_REC",
"ELEV_DEPTH_SCALER","COORD_SCALER","XSHOT","YSHOT","XREC","YREC","UNITS",
```

```
"VELOCITY_WEATHER","VELOCITY_SUBWEATHER","UPHOLE_SHOT","UPHOLE_REC",
"STATIC_SRC","STATIC_REC","STATIC_TOTAL","LAG_TIME_A","LAG_TIME_B",
"DELAY_TIME","MUTE_TIME_START","MUTE_TIME_END","NSAMPLES","SAMPLERATE",
"GAIN_TYPE","GAIN_CONSTANT","GAIN_INIT","DATA_CORRELATED","SWEEP_FREQ_START",
"SWEEP_FREQ_END","SWEEP_LENGTH","SWEEP_TYPE","SWEEP_TAPERLEN_START",
"SWEEP_TAPERLEN_END","SWEEP_TAPERTYPE","ALIAS_FILTER_FREQ","ALIAS_FILTR_SLOPE",
"NOTCH_FILTER_FREQ","NOTCH_FILTER_SLOPE","LOWCUT_FREQ","HIGHCUT_FREQ",
"LOWCUT_SLOPE","HIGHCUT_SLOPE","DATARECORDED_YEAR","DATARECORDED_DAY",
"DATARECORDED_HOUR","DATARECORDED_MINUTE","DATARECORDED_SECOND",
"TIME_BASIS_CODE","TRACE_WEIGHTING_FACTOR","GEOPHONE_GROUP_NUMBER",
"GEOPHONE_GROUP_FIRSTTRACE","GEOPHONE_GROUP_LASTTRACE","GAP_SIZE",
"OVER_TRAVEL","SHOT_SEQUENCE_NUMBER","FIELD_STATION_NUMBER","IN_LINE",
"CROSS_LINE","SHOTLINE_NUMBER","SHOTLINE_NUMBER_SCALER","TRACE_VALUE_UNIT",
"TDCTE","TDCTE","TDCTE","TRANSDUC_UNIT","TRACE_DEV_ID","TRACE_HEADER_SCALER",
"SOURCE_TYPE","-","-","-","-","-","-","-","-","-")
> segy.name<-'1104-30A'
> for (c in 1:num.tr){
 df.temp<-data.frame(segy=segy.name,trace=c,code=trhdbn,value=segY[[3]][[c]])
 dbWriteTable(con, "sgy_trhd",value=df.temp,append= TRUE, row.names=FALSE)
 }
```

Este passo final pode demorar (meia hora) uma vez que esses arquivos podem ser muito grandes:

```
> sql<-"CREATE TABLE sgy_tr_data (segy text,trace int,value numeric(18,9)[]);"
> dbGetQuery(con,sql)
> for (t in 1:num.tr){
   trace<-"'{"
   for (c in 1:num.samples){
     if(c==1){trace<-paste0(trace,as.character(segY[[4]][[t]][c]*1))}
     else{trace<-paste0(trace,",",as.character(segY[[4]][[t]][c]*1))}
   }
   trace<-paste0(trace,"}'")
   sql<-paste0("INSERT INTO sgy_tr_data (segy,trace,value)
VALUES('",segy.name, "',",t,",",trace,");")
   dbSendQuery(con,sql)
 }
```

Extraindo os dados de um SEG-Y do geobanco:

```
> library(RPostgreSQL)
> con<-dbConnect(PostgreSQL(),host="127.0.0.1", user= "...",password="...",
dbname="geobanco")
> sql<-"select text from sgy_hd_txt where segy='1104-30A';"
> segy.header.txt<-dbGetQuery(con,sql)
> sql1<-"select field,value from sgy_hd_bin where segy='1104-30A';"
> segy.header.bin<-dbGetQuery(con,sql1)
> sql2<-"select trace,code,value from sgy_trhd where segy='1104-30A';"
> segy.trace.header<-dbGetQuery(con,sql2)
> sql3<-"select trace,value from sgy_tr_data where segy='1104-30A';"
> segy.trace.data<-dbGetQuery(con,sql3)
> lista.SGY<-list(segy.header.txt,segy.header.bin,segy.trace.header,
segy.trace.data)
```

Podemos visualizar a informação extraída usado:

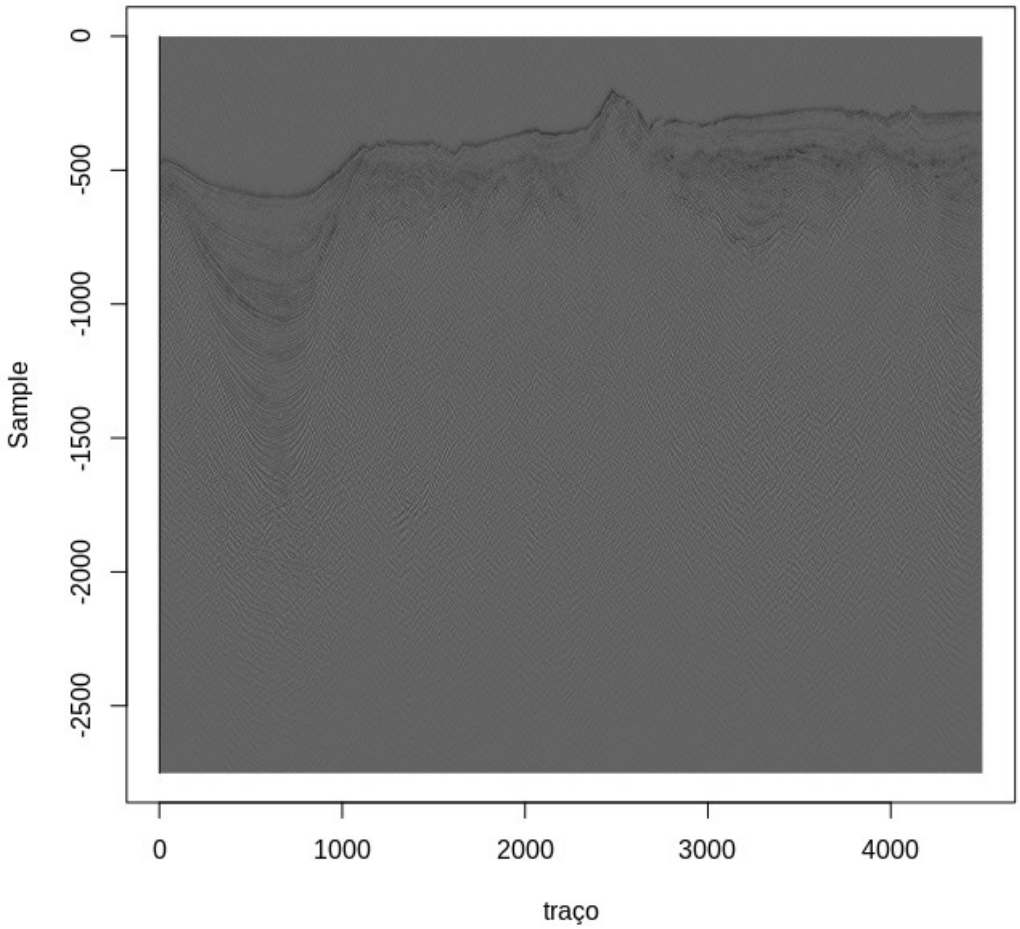
```
> lista.SGY[[1]]
1 C 1 CLIENT MMS                    COMPANY GECO                    CREW NO
C 2 LINE 1104-30A   AREA PHASE-30A              MAP ID
C 3 REEL NO 17784MIG0 DAY-START OF REEL     YEAR      OBSERVER
C 4 INSTRUMENT: MFG            MODEL            SERIAL NO
C 5 DATA TRACES/RECORD        AUXILIARY TRACES/RECORD        CDP FOLD
C 6 SAMPLE INTERVAL       4 SAMPLES/TRACE  2750 BITS/IN    0 BYTES/SAMPL
```

```
C 7 RECORDING FORMAT        FORMAT THIS REEL        MEASUREMENT SYSTEM
C 8 SAMPLE CODE: FLOATING PT 032 FIXED PT     FIXED PT-GAIN     CORRELAT
C 9 GAIN  TYPE: FIXED     BINARY     FLOATING POINT     OTHER
C10 FILTERS: ALIAS     HZ  NOTCH     HZ  BAND     -     HZ  SLOPE    -
C11 SOURCE: TYPE AIRGUN    NUMBER/POINT         POINT INTERVAL
C12     PATTERN:                            LENGTH        WIDTH
C13 SWEEP: START     HZ  END     HZ  LENGTH      MS  CHANNEL NO     TYPE
C14 TAPER: START LENGTH       MS  END LENGTH       MS  TYPE
C15 SPREAD: OFFSET        MAX DISTANCE        GROUP INTERVAL
C16 GEOPHONES: PER GROUP     SPACING     FREQUENCY     MFG          MODE
C17     PATTERN:                            LENGTH        WIDTH
C18 TRACES SORTED BY: RECORD     CDP     OTHER
C19 AMPLITUDE RECOVERY: NONE      SPHERICAL DIV       AGC    OTHER
C20 PROJECTION : SPHEROID :
C21 FAMILY 1 DEFAULTS: DATA TRACES/RECORD        AUXILIARY TRACES/RECORD
C22                    SAMPLE INTERVAL          SAMPLES/TRACE
C23 FAMILY 3 DEFAULTS: SAMPLE INTERVAL          SAMPLES/TRACE
C24 CENTRAL MERIDIAN :  0  0 0.0 E    ORIGIN PARALLEL :  0  0 0.0 N
C25 FALSE NORTHING :      0.0 FALSE EASTING :      0.0 UNIT TO METER :0.000000
C26
C27 MXYMRGOV 15.06.22 12- 8-88 A6602IM4 0526     1104-30
C28
C29
C30
C31
C32
C33
C34
C35
C36
C37
C38
C39
C40 END EBCDIC
> lista.SGY[[2]]
                    field value
1                  JOB_ID_NO   526
2               LINE_NUMBER  1104
3               REEL_NUMBER    30
4         TRACES_PER_RECORD     1
5           AUXS_PER_RECORD     0
6                SAMPLE_RATE  4000
7         SAMPLE_RATE_FIELD     0
8                   NSAMPLES  2751
9             NSAMPLES_FIELD  2751
10               FORMAT_CODE     1
11            ENSEMBLE_FOLD     1
12               TRACE_SORT     4
13             VERTICAL_SUM     2
14          SWEEP_FREQ_START     0
15            SWEEP_FREQ_END     0
16        SWEEP_FREQ_LENGTH     0
17               SWEEP_TYPE     0
18            SWEEP_TRACE_NO     0
19 SWEEP_TAPER_LENGTH_START     0
20   SWEEP_TAPER_LENGTH_END     0
21         SWEEP_TAPER_TYPE     0
22        CORRELATED_TRACES     0
23               BINARY_GAIN     0
24      AMP_RECOVERY_METHOD     0
25                    UNITS     1
26          SIGNAL_POLARITY     0
```

```
27        VIBRATOR_POL_CODE     0
28                   UNUSED     0
29  SEGY_FORMAT_REVISION_NO     0
30       SEGY_FIXEDLEN_FLAG     0
31     SEGY_NO_TEXTFHEADERS     0
32                   UNUSED     0

> head(lista.SGY[[3]],n=94L)
   trace                     code      value
1      1              TRACE_SEQ_NO          1
2      1             TRACE_SEQ_REEL          1
3      1           FIELD_RECORD_NO       9936
4      1                CHANNEL_NO          3
5      1             SHOT_POINT_NO      99430
6      1                    CMP_NO        231
7      1                CMP_SEQ_NO         27
8      1             TRACE_ID_CODE          1
9      1                      FOLD          2
10     1              TRACE_HSTACK          0
11     1                 TEST_CODE          1
12     1             OFFSET_SH_REC        310
13     1                  ELEV_REC          0
14     1                 ELEV_SHOT          0
15     1                DEPTH_SHOT          0
16     1            DATUM_ELEV_REC          0
17     1       ELEV_FLOATDATUM_SRC          0
18     1          WATER_DEPTH_SHOT          0
19     1           WATER_DEPTH_REC          0
20     1         ELEV_DEPTH_SCALER          1
21     1             COORD_SCALER       -100
22     1                    XSHOT   96025063
23     1                    YSHOT -337967999
24     1                     XREC  135312400
25     1                     YREC  967985300
26     1                    UNITS          1
...
...
91     1                        -     -14397
92     1                        -      -3851
93     1                        -          0
94     1                        -          0
> head(lista.SGY[[4]],n=1L)
[1] "{0.682663143,0.395595610,0.276713848,0.704893410,0.631148875,-
0.740177929,-0.969260693,-0.726061046,0.181251049,-0.699826598,-0.904339194,
...
-
0.301578641,1.829489350,2.355402708,2.352203608,1.950937629,0.994827628,0.81616
0142,1.077815175,2.294286251,2.520761013}"
```

Visualizando os dados carregados a partir do arquivo SEG-Y:

```
> plot((rep(1,2751)/10)+1,c(-1:-2751),type='l',xlim=c(1,4500),
xlab='traço', ylab='Sample')
> for(i in 1:4500){
 sepa<-gsub('[:{:]','',lista.SGY[[4]]$value[i])
 sepa2<-gsub('[:}:]','',sepa)
 sepa3<-as.numeric(unlist(strsplit(sepa2,",")),use.names=F))
 lines((sepa3/10)+i,c(-1:-2751),type='l',lwd=0.1)
}
```


Sample
0
-500
-1000
-1500
-2000
-2500
0
1000
2000
3000
4000
traço

6.5 – Dados de Geofísica aérea (Oasis Montaj)

Dados de geofísica aérea normalmente estão no seguinte formato:

```
/ ------------------------------------------------------------------------------
/ XYZ EXPORT [07/03/2015]
/ DATABASE   [.\mag_sub2.gdb]
/ ------------------------------------------------------------------------------
/
/         X          Y FIDUCIAL   GPSALT     BARO   ALTURA      MDT    MAGBASE     MAGBRU ...
/========== ========== ======== ======== ======== ======== ======== ========== ========== ...
/
//Flight 16
//Date 2014/05/24
Tie  19010
    217454    9225303  51679.8   459.43   461.28   160.64   298.78  25058.940  25019.602  ...
    217446    9225303  51679.9   459.38   461.30   159.88   299.50  25058.940  25019.560  ...
    217438    9225303  51680.0   459.32   461.32   159.11   300.20  25058.940  25019.522  ...
    217430    9225303  51680.1   459.25   461.34   158.39   300.85  25058.940  25019.486  ...
```

E precisamos colocar eles no seguinte formato para carregar no geobanco:

```
X,Y,FIDUCIAL,GPSALT,BARO,ALTURA,MDT,MAGBASE,MAGBRU,...
217011,9225307,60215.1,483.54,481.87,164.24,319.30,25046.860,24976.290,...
217011,9225315,60215.2,483.74,482.08,161.88,321.86,25046.860,24975.464,...
```

Fazemos isso em R criando um data.frame com essa característica usando:

```
> xmi<-180000
> xma<-190000
> ymi<-8940684
> yma<-8950684
> #lê arquivo .XYZ e o número de linhas do arquivo
> flcon<-file('1099_MAGLINE.XYZ',open='r')
> cmd <- paste("wc -l 1099_MAGLINE.XYZ | awk '{ print $1 }'")
> numberLines <- as.integer(system(command=cmd, intern=TRUE))
> # lendo as cinco primeiras linhas
> tmp <- readLines(flcon, n=5)
> # obtendo o nome das colunas
> coluna <-c(strsplit(substring(readLines(flcon, n=1),11),"\\s+")[[1]],
'line/tie')
> n.co<-length(coluna)+1
> # lendo mais 2 linhas
> tmp <- readLines(flcon, n=2)
> lin<-c('a')
> o<-1
> lineOrTie<-''
> for (i in 11:numberLines){
  l<-substring(readLines(flcon, n=1),2)
  if(substring(l,1,2)=='ie' || substring(l,1,2)=='in'){
   lineOrTie<-substring(l,5)
  }else{
   v<-as.numeric(strsplit(l, "\\s+")[[1]][2])
   w<-as.numeric(strsplit(l, "\\s+")[[1]][3])
   # se for usar limites remover comentário
   if(v>xmi & v<xma & w>ymi & w<yma){
    lin[o]<-paste(l,lineOrTie)
    o<-o+1
   }# remover para limites
  }
 }
> close(flcon)
> r <- strsplit(sub("^\\s+","",lin), "\\s+")
> s <-as.data.frame(do.call(rbind, r))
> names(s)<- coluna
> cols = c(1:19)
> s[,cols] = apply(s[,cols], 2, function(x) as.numeric(as.character(x)))
```

```
> library(raster)
> coordinates(s)<-~X+Y
> crs(s)<-CRS('+init=epsg:32721')
> s
class       : SpatialPointsDataFrame
features    : 23839
extent      : 180046.6, 189586.6, 8940685, 8950293  (xmin, xmax, ymin, ymax)
crs         : +init=epsg:32721 +proj=utm +zone=21 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 17
names       : FIDUCIAL, GPSALT,   BARO, ALTURA,    MDT,     MAGBRU,     MAGCOM,
MAGCOR,     MAGNIV,     MAGMIC, MAGIGRF,     IGRF,  LONGITUDE,  LATITUDE,
DATA, ...
min values  : 109652.0, 407.61, 402.67, 100.01, 307.94, 24409.5967, 24406.2140,
24429.4423, 24428.2533, 24428.6013, -0.0069, 24517.99, -53.826942, -9.484214,
2010/01/21, ...
max values  :  81107.0, 505.91, 502.66,  99.99, 409.51, 24634.4792, 24632.8914,
24667.1354, 24663.1780, 24664.9821, 99.8617, 24538.71, -53.914135, -9.571518,
2010/01/27, ...
```

Carregamos o SpatialPointDataFrame acima no geobanco usando:

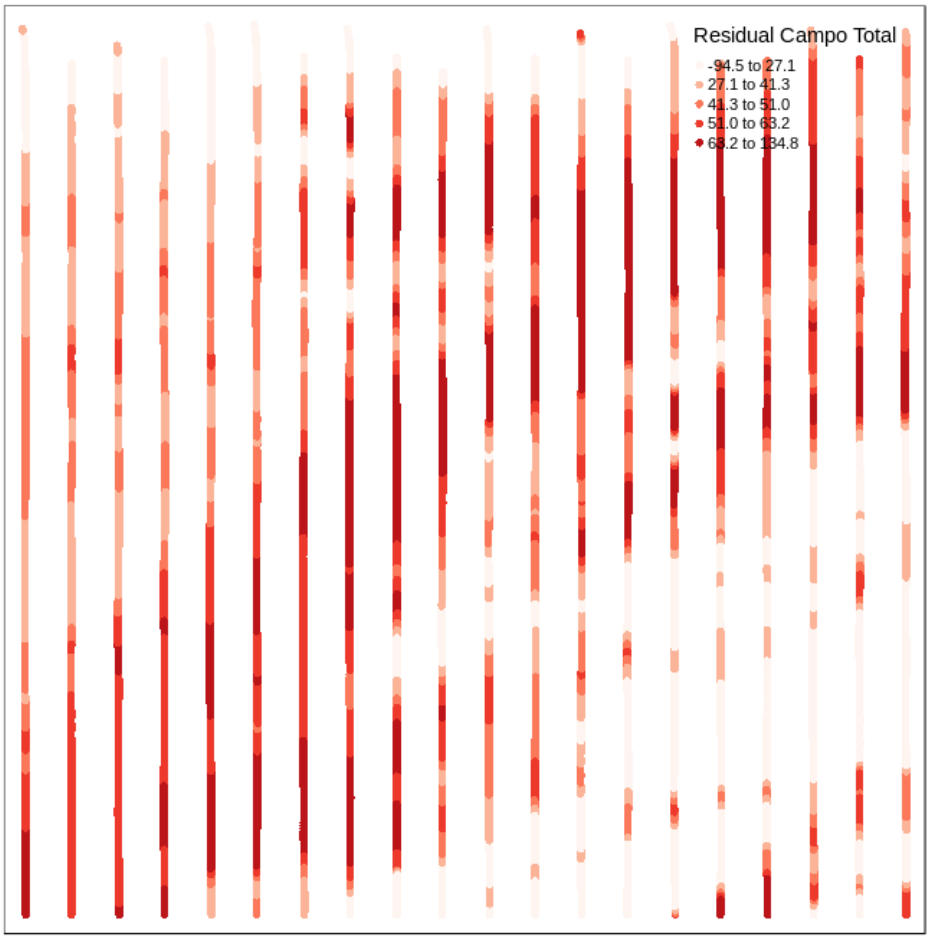
```
> library(rpostgis)
> con<-dbConnect(PostgreSQL(),dbname='geobanco',user='voce',password='segredo')
> pgInsert(con, name = c("public", "magnetometria"), data.obj = s)
```

E extraímos os dados do geobanco usando:

```
> library(rpostgis)
> library(raster)
> con<-dbConnect(PostgreSQL(),dbname='rteste',user='voce',password='segredo')
> mag<-pgGetGeom(con,c('public','magnetometria'))
```

Plotando:

```
> library(tmap)
> mag$MAGIGRF<-as.numeric(mag$MAGIGRF)
> tm_shape(mag) + tm_dots(size=0.15,col = "MAGIGRF", palette = "Reds", style =
"quantile", title='Residual Campo Total')
```


Residual Campo Total
-94.5 to 27.1
27.1 to 41.3
41.3 to 51.0
51.0 to 68.2
68.2 to 134.8