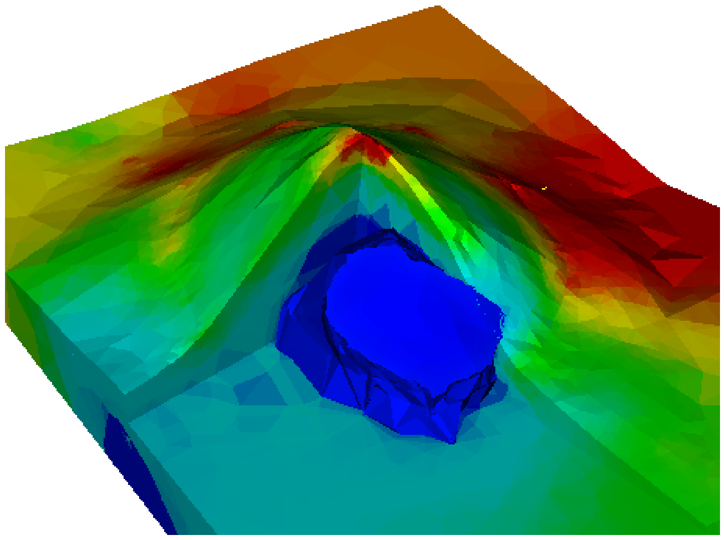
Boundless Electrical Resistivity Tomography BERT 2 – the user tutorial



Thomas Günther[*] & Carsten Rücker[†]

April 26, 2023
version 2.4.1

In this tutorial we demonstrate how to do ERT inversion using the software package BERT. Some small but instructive examples, all real field cases, are discussed to show how the different options in the configuration file can be used to yield case-specific inversion results.

The examples start from 2d inversion of surface measurements with and without topography. The same is later demonstrated in 3d, where mesh generations becomes more an issue. We show how to include structural information and how buried electrodes are handled.

Also, measurements on closed objects, such as trees, humans, soil columns and model tanks are shown. Finally we show how to handle time-lapse resistivity measurements. The user is invited to follow by processing the data in the examples directory.

[*]Leibniz Institute for Applied Geophysics, Hannover
[†]Berlin University of Technology, Department of Applied Geophysics

Contents

1. Introduction

1.1. BERT 1 and 2, DCFEMLib, GIMLi - history and names

Direct current electrical measurements are used in a wide range of applications such as medical imaging, geophysical surface or subsurface measurements or the investigation of trees and soil probes. This inverse problem is known under the terms ERT (electrical resistivity tomography), ERI (... imaging), EIT (... impedance tomography) or DC resistivity inversion. The aim of our software is to present an extremely flexible solution that works on all geometries.
Main advantage is the possibility to work on arbitrary geometries. Therefore we decided to consequently use unstructured[1] finite element meshes for forward calculation as well as for the parameter identification. By the use of triangles (2d) and tetrahedrons (3d) we can follow any prior geometry, probe or structural information we have from the subsurface. Due to this generality we decided to call it BERT - Boundless Electrical Resistivity Tomography. The name BERT can be associated to

- the technical solution described by Günther et al. (2006), Rücker (2010) and other works,
- the C++/Python software project (on `https://gitlab.com/resistivity-net/bert`),
- the distribution (Windows installer, conda package), and
- the main command-line tool.

BERT version 1 (released 2009-2013) has been a part of the C++ library DCFEMLib - Direct Current Finite Element Method Library that is still contained in the BERT source code. Parts of the DCFEMLib-based tools for mesh generation are still in use but will be replaced by Python tools soon. From version 2 on (starting with beta versions in 2010), BERT is based on GIMLi - Generalized Inversion and Modelling Library, a multi-method C++/Python library for various geophysical methods. Please have a look at `https://www.pygimli.org` for details how to retrieve, build and use the code. pyGIMLi is licensed under Apache License 2.0 [2] and BERT is licensed under GNU public license (GPL)[3]. Our vision is giving back to the academic community what we learned without companies letting exploit it, which is why we never cared for a graphical user interface.
The basic theory and technology of BERT is described by Günther et al. (2006) and bases on the finite element modelling techniques discussed by Rücker et al. (2006). First developed for the usual point electrodes, the finite element modelling was later extended to arbitrary electrode shapes using the complete electrode model (Rücker and Günther, 2011) and long

[1]Unstructured or irregular means that there is no order or rule of shape for the elements, regular discretizations with quadrangles, hexahedra or prisms are just special variants.
[2]See `https://www.apache.org/licenses/LICENSE-2.0` for Apache licensing conditions.
[3]See `https://www.gnu.org/licenses/gpl.html` for GPL licensing conditions

electrodes using the shunt electrode model (Ronczka et al., 2015). Generally the inversion is based on a smoothness-constrained Gauss-Newton inversion described by Günther et al. (2006). It was later formulated as a flexible minimization and regularization scheme described in detail by Rücker (2010).
The default inversion scheme is represented by a triple-grid scheme. Most inversion algorithm use a dual-grid scheme, i.e. the forward calculation is calculated on a mesh that is finer than the inversion one. We added another one in order to use a secondary field approach and thus have a very fast forward calculation[4] Figure 1 shows the three grids: On a coarse and resolution-dependent grid the parameters are defined. On a globally refined and prolonged mesh the forward calculation is done. And a very fine primary mesh is used to calculate the primary potentials (for a homogeneous subsurface), but only once directly after the mesh creation.

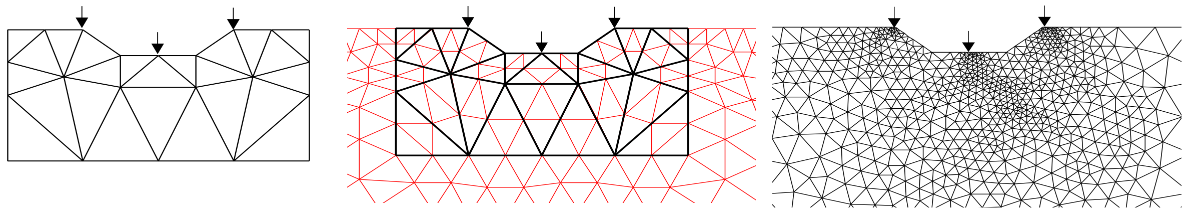


Figure 1: The three meshs of inversion for a 2d example, from Günther et al. (2006)

The overall scheme is visualised in Figure 2. It starts with the generation of the three meshes. Then the primary potentials are calculated and interpolated onto the secondary mesh. From this geometric factors are derived yielding the apparent resistivity and the sensitivity matrix is created for the homogeneous case. Finally the actual inversion is carried out: An inverse sub-problem is used to update the resistivity model, a forward calculation is carried out and checked against the data. The latter is done until the data are fitted well or the process stagnates.
BERT is available under Linux and Windows[5], either from pre-compiled binaries or self-compiled code[6]. The paths to the binaries and the library must be known, e.g. by setting
`$ export PATH=$PATH:/c/Software/BERT` for Windows or under Linux
`$ export PATH=$PATH:/path/to/BERT/bin`
`$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/BERT/lib` BERT comes along with the Python modules pyGIMLi and pyBERT that need to be found by setting the variable `PYTHONPATH` to its place (without the pygimli or pybert at the end!). All variables can be set permanently in a .bashrc file, either for all users in the directory where bert is, or for the current user in the home directory.

1.2. Options and commands

The inversion itself is controlled by the program bert (a bash script), which reads the so called configuration (cfg) file. In the cfg file all necessary information is stored in form of lines consisting of `KEY=value` type, as in bash everything behind the #-sign is ignored and can be

[4]However,you can use a dual-mesh or single-mesh approach as well, or completely different meshes for forward and inverse modelling.
[5]See appendix A.1 for using BERT in Windows.
[6]See `www.resistivity.net` for information about how to obtain the binaries/codes and to compile the code

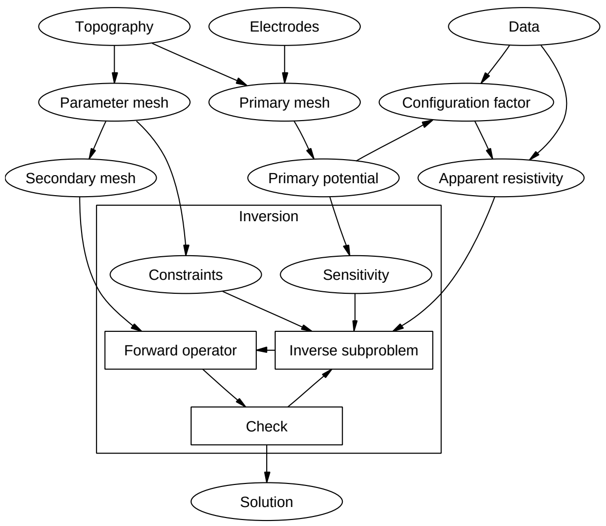


Figure 2: The BERT inversion scheme, from Günther et al. (2006): the geometrical information is used to prepare the actual inversion (rectangle).

used for comments. Note that the keys must be uppercase[7]. There is only one mandatory key: the **DATAFILE** key holding the name of the data file. Other important keys are **DIMENSION** (2 or 3) and **TOPOGRAPHY** (0 or 1, meaning false or true). We suggest to create a new directory for each problem or also for different strategies to solve it. The data file must be in the unified resistivity.net format (see `www.resistivity.net?unidata`) or any of the supported instrument files (see below, Python required!).
The current version can be show by calling

```
$ bert version
```

For list of possible options (with default values) see appendix D or call

```
$ bert opts
```

However only few of them are of frequent use[8]. At the beginning of each `bert` call, the global and user options are searched for and displayed if present.
In order to create a new project there are special commands for the individual tasks, namely bertNew2D, bertNew2DTopo, bertNew3D, bertNew3DTopo, for the cases 2d/3d with or without topography. For example,

```
$ bertNew2D datafile.dat > bert.cfg
```

creates a new configuration file bert.cfg with the lines **DATAFILE=datafile.dat**, **DIMENSION=2** and **TOPOGRAPHY=0**, but also adds a lot of possible options for this case with an explanation, most of them inactive/commented by a # character. All bertNew commands try to convert any data files with endings

- txt (ABEM, Resecs or Syscal Pro Ascii Output),
- tx0 (4-point light),

[7]There is only one exception: **cMin/cMax/cMap** for color scale. We might drop case-sensitivity.
[8]For changing default options permanently, we suggest creating a file `$HOME/.bertrc` for user-specific options or a file `.bertrc` in the directory where BERT is installed for computer-wide option (that are overridden by user options or project options). Typical entries are, e.g., SENSMATMAXMEM=8000 (available memory in MB), or favorite mesh options

- bin (Syscal Pro/Junior binary file),
- stg (AGI SuperSting data)
- flw (Geotom custom array data)
- amp (ABEM LS system deprecated format),
- res2dinv (to be renamed to the extension res2dinv to be recognized automatically).
- udf (unified dataformat defined by DC2dInvRes)

to BERT input and adds a .dat ending to the original file name.
The input data conversion can also be done by hand by calling (no renaming necessary)

```
$ bert convert datafile
```

Additionally to the instrument data, one can also use the unified data format (UDF, once defined by DC2dInvRes) with topography being specified at the end of the file. If the filename contains topography at the end of the file (res2dinv or udf), the tape-measured electrodes are unrolled along the topography the resulting data file holds the real positions in the electrode section. Note that in case of topography we prefer (and save) resistance over apparent resistivity as it is not disambiguous concerning how the geometric factor was calculated.
The user can now (or later) change the options and run single steps or the whole inversion by

```
bert cfgfile commands
```

where cfgfile is the configuration file and commands can consist of the following:

version just plot BERT version and default options

all makes all, that is probably the first step in most (at least small) cases

meshs just create meshes to check them first (suggested for bigger problems)

nomeshs do everything else but the meshes (after a successful mesh generation)

domain create only the mesh input PLC

pot calculates primary potentials and interpolates them to the secondary mesh (in BERT 1, this was divided into the two jobs primPot and interpolate)

filter filter input data using data limits, add geometric factor and estimate error

calc only filtering and inversion (no change of mesh), e.g. after changing inversion options

save saves all important results (model&response for each iteration, log file, cfg file, meshes) in a directory called `result<date>_<time>`

clean cleans the directory from temporary results

mrproper deletes all stuff except input and result directories (releases disk memory fully)

plotting functions (2D only, with Python):

showmesh show parameter mesh (2D)

showpoly show poly file as input (2D and 3D)

show show inversion result or other model vector (2D and 3D)

mkpdf generate a pdf of the resulting resistivity or other model vector (2D)

showdata plots apparent resistivity pseudosection (2D)

showerror plot error model pseudosection (2D)

showfit show data, model response and misfit as pseudosections (2D)

The show commands can be used in the project directory, also while inversion is still running, or in a result directory with saved results.

2. Basic concepts using a simple 2D example

2.1. First steps

The example in `examples/inversion/2dflat/gallery` was friendly provided by the University of Mining and Technology, Freiberg (F. Donner) and extensively discussed by Günther (2004). It is a very small profile over a known mining gallery that is used for draining water out of the mines. It represents a perfect two-dimensional anomaly since it strikes perpendicular to the profile and is a 2x2m cavity.
On a profile using 21 electrodes with 2m spacing, dipole-dipole measurements have been applied, the data quality was very good. The input in the data file `gallery.dat` is already the apparent resistivity, an error model is already in the data.
We create a configuration file redirecting the output of an initialization script

```
$ bertNew2D gallery.dat > bert.cfg
```

```
DATAFILE=gallery.dat
DIMENSION=2
PARADX=0.25          # size (in electrode spacings) of cells at the surface
PARA2DQUALITY=34.0   # defines how fast the mesh is growing (33-fast,35-slow)
```

The resulting file bert.cfg holds the data file, the dimension, some default options and some possible options that are deactivated by the hash. Before running an inversion we want to have a look at the data by calling[9]

```
$ bert bert.cfg showdata
```

Figure 3 shows the data, i.e. the apparent resistivity pseudosection. DD denotes dipole-dipole array and the number the spacing. You can also look at the error by replacing `showdata` with `showerror`.
The simplest inversion run is started by

```
$ bert bert.cfg all
```

and comprises mesh generation and actual inversion. The output, which is also stored in the file bert.log, ends with

```
4: Model: min = 69.271; max = 756.813
4: Response: min = 85.2701; max = 364.752
```

[9]Plotting any show command will only work if Python is installed (and found with the PATH environment variable or defined by PYTHON) and the modules pygimli and pybert are found via PYTHONPATH.

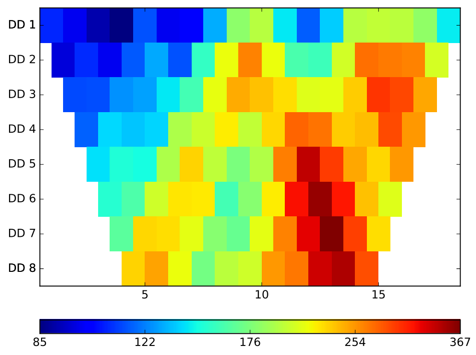


Figure 3: Apparent resistivity pseudosection with the showdata command.

```
4: rms/rrms(data, Response) = 3.8314/1.71759%
4: chi^2(data, Response, error, log) = 1.7069
4: Phi = 198.001+19.2525*20=583.052
```

In each iteration (number at the beginning) the minimum and maximum values are specified for the model and its forward response. Additionally, the inversion specifies several measures of data fit: an absolute root-mean square (rms) in Ohmm, a relative rms (rrms), the error-weighted chi-square fit $\chi^2 = \Phi_d/N$ and the objective function Φ consisting of data misfit $\Phi_d = \sum((d_i - f_i(m))/\epsilon_i)^2$ plus the regularization parameter λ (here 20) times the model roughness[10]. Obviously the data are fit with a good rrms of about 1.7%, which is not completely within the error model ($\chi^2 = 1$ means a perfect fit).

```
$ bert bert.cfg show
```

shows the inversion result displayed in Figure 4 using default options. It clearly images the cavity at about 20 m and another resistive anomaly whose origin is not completely clear from 2d measurements. See section 4.1 for inversion of a 3d data set.

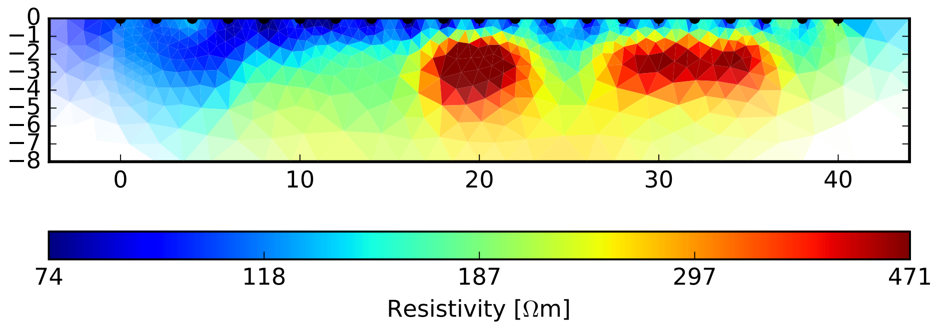


Figure 4: Result of the gallery example using default options.

It shows a rising well-conducting loamy overburden over a medium conductor, in which two distinct resistive bodies are embedded. The central one is the well-known mining gallery, the origin of the other is not completely clear. One can click into the model getting some information on the model cell.

[10]See Günther et al. (2006) for details and definitions.

The colour scale is chosen automatically by using 3% interpercentile values (this can be changed by setting the variable INTERPERC) unless not explicitly specified in the cfg file by the keywords **cMin** and **cMax**. The default color map depends on your *matplotlib* version, from v.2.0 on the perceptual[11] *viridis* color map replaced the old *jet* scale. As an alternative, you can use the rainbow-like *Spectral_r* that does not show the dominating yellow and cyan colors. You can use the key **cMap** to choose any of the *matplotlib* color scales[12]. The keyword **USECOVERAGE** (0 or 1-default) defines whether the coverage is used for alpha-shading the model). The key **SHOWELECTRODES** defines whether electrodes are plotted as black dots.
The model plot can be saved using

```
$ bert bert.cfg mkpdf
```

A visual inspection of the data fit can be obtained by calling

```
$ bert bert.cfg showfit
```

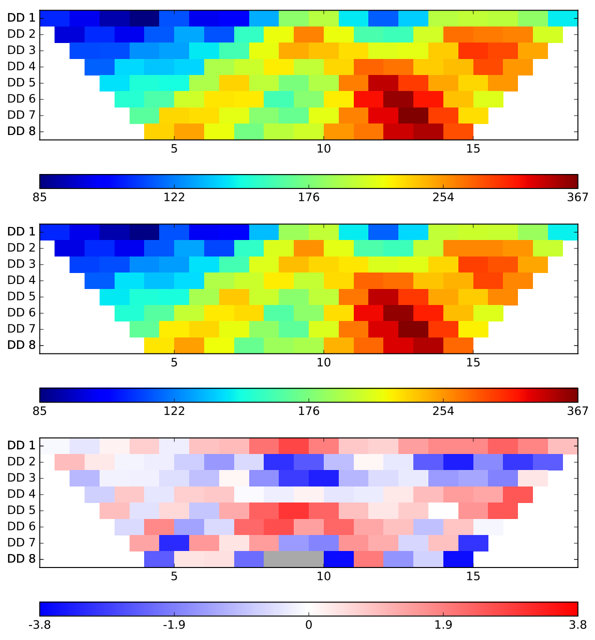


Figure 5: Measured apparent resistivity (top), model response (center) and percentage data misfit (bottom).

We see hardly any visual difference between the measured and modelled data. However, the misfit plot (that can also be obtained as a single plot by `bert bert.cfg showmisfit`) shows some systematic layering hinting that there might be still some information left. Ideally, the misfit plot is an uncorrelated random distribution. To save the result for further use, we use the command

```
$ bert bert.cfg save
```

It saves all input files, meshes, log files and results (vectors, vtk and pdf files) to a result folder. We can later go to this folder and do some visualization (or even start a new inversion). Save also cleans the working directory from temporary files. If you want to clean manually, call

[11] See discussions on `https://mycarta.wordpress.com/2012/05/29/the-rainbow-is-dead-long-live-the-rainbow-series-ou`

[12] For a list, see `https://matplotlib.org/users/colormaps.html`

`$ bert bert.cfg clean` (only the working directory) or
`$ bert bert.cfg veryclean` (deletes also mesh and potential directories).
`$ bert bert.cfg pack` even compresses all result folders.

2.2. Regularisation and data fit

We now might to change the characteristics of the model. The most important key for that is the regularization parameter **LAMBDA**. It controls the strengths of the smoothness constraints and thus defines how smooth the model will be. Therefore we test different values by commenting out the line with the parameter **LAMBDA** and setting it once to 200 and once to 2 instead of the default value of 20. After changing it we can combine calculation and plotting `bert bert.cfg calc show`

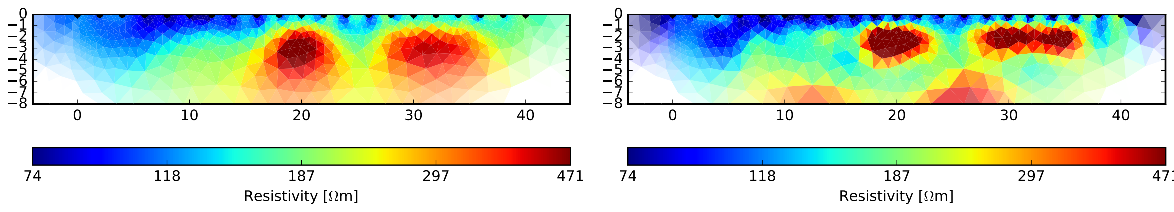


Figure 6: Result for a regularization parameter **LAMBDA** of 200 (left) and 2 (right).

Figure 6 shows the result for the two regularization strengst. The one for $\lambda = 200$ is clearly over-smoothed and cannot fit the data appropriately ($\chi^2 = 7.6$, rrms=3.1%). The value of $\lambda = 2$ can fit the data (it stops at $\chi^2 = 0.8$, rrms=1.2%) and could be a model. Even lower values will lead to oscillations in the model. By default, the regularization parameter remains constant, however in some cases it can be beneficial to start with a higher value and to decrease it in every iteration (say by 20%) using **LAMBDADECREASE=0.8**. The regularization strength can also be optimized by two ways: One is to adjust it such that the data are perfectly fit withing error ($\chi^2 = 1$) by using **CHI1OPT=1**. This is particularly interesting for synthetic data. Another option is to find the maximum curvature of the L-curve for a certain range of lambda values as described by Günther et al. (2006) using **LAMBDAOPT=1**.
Note that LAMBDA controls the smoothness compared to the data misfit term, in which the data errors serve as weighting. Higher errors correspond to lower regularization and vice versa. Note that stacking errors from instruments are not appropriate measures in the meaning of how well we can fit the data. A way to estimate errors is the analysis of reciprocal data as explained by Udphuay et al. (2011) and references therein. In the absence of errors in the data file an error estimation is made using a fixed percentage (**INPUTERRLEVEL**) and a voltage error (**INPUTERRVOLTAGE**). If the current is not in the file, a value of 100 mA is assumed. For different current strengths the voltage error has to be adapted. If there are errors in the file, they can be discarded by a new error estimation by using **OVERRIDEERROR=1**. Note that an L2 norm (squared) minimization assumes a Gaussian distribution of the (error-weighted) misfit. In case of significant outliers in the data one can use **ROBUSTDATA=1**, an L1 norm scheme based on iteratively reweighted least squares (IRLS) after Claerbout and Muir (1973). However, one can easily loose resolution, which is why it is not recommended by default, but should be used after observing systematic outliers in the data.
In many cases the default regularization strength will lead to a good start, providing the error estimation meets the data quality. We recommend to have a look at distribution of the misfit

(Fig. 5c). In almost all surface measurements there is a need for anisotropic regularization, i.e. for a lower penalty of vertical contrasts due to a predominant layering (and also supported by a lower resolution of the method). The variable **ZWEIGHT** defines the relative weight for a purely vertical boundary in the model. For arbitrary boundaries (as they occur for triangular and tetrahedral discretizations) they are computed as described by Coscia et al. (2011).

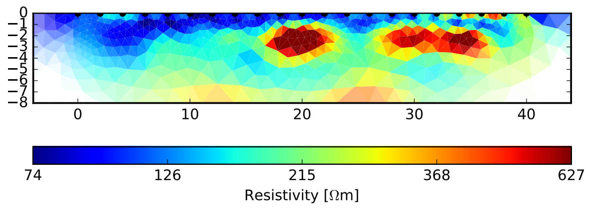


Figure 7: Result of the gallery example using **ZWEIGHT=0.4**.

Figure 7 shows the result for a vertical weight of 0.4, which probably improves the interpretation a bit. As a result, the layering in the misfit function disappeared. Note that reducing **ZWEIGHT** from the default value of 1 decreases the overall smoothness and can lead to overfitting by small-scale anomalies. In these cases the **LAMBDA** value should be increased. In order to overcome the smooth transitions in the model, one can use **BLOCKYMODEL=1**, i.e. the L1 norm minimization scheme. Just like the **ROBUSTDATA** option it used IRLS for weighting the individual gradients such that stronger contrasts are occurring.
So far we used first-order smoothness constraints, i.e. the constraint type **CONSTRAINT-TYPE** of 1 (default). One can also use second-order smoothness (2), i.e. the curvature of the model is minimized. Zeroth-order smoothness (0) means that only the norm of the model difference to the reference model is minimized. This can easily lead to strange oscillations in the model and is therefore only recommended if a very good starting model is known. Other possibilities comprise a mix of minimum-length constraints with first or second order smoothness using the values 10 and 20.
By default, a homogeneous starting model is generated with the median apparent resistivity. This can be overridden by the key **STARTMODEL**, which can be either a resistivity value or the name of a file containing the whole vector. As the length needs to match the unknown model cells, this varies with the mesh option, which is why non-homogeneous models are more easily controlled by using regions (see chapter 5.2). By default, the starting model is also the reference model, i.e. the model (including logarithmic transforms) difference is constrained. To prevent this, use the switch **NOREFERENCE=1**.

2.3. Mesh quality and refinement

Inversion results are of course to a certain degree mesh-depending. We go back to the mesh generation and have a look at the input of the mesh generation, the piecewise-linear complex (PLC), stored in mesh/mesh.poly by calling
`$ bert bert.cfg showpoly` and zooming in a little bit

We observe two bodies, an inner in red (the parameter domain with the marker 2), and a much bigger outer with the marker 1, which is needed for the forward calculation. The depth of the parameter domain is by default automatically determined based on 1d sensitivity studies (calling the command `paradepth gallery.cfg`, but can be adjusted using the key

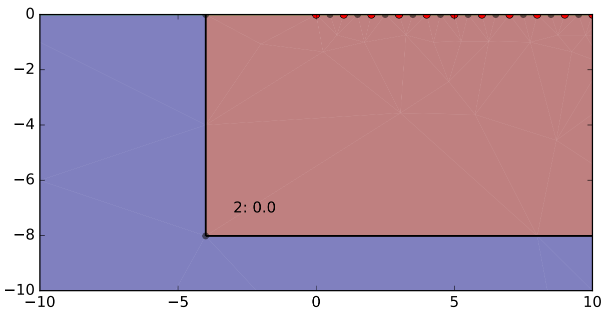


Figure 8: Piecewise linear complex (mesh input) for the gallery.

PARADEPTH. Sometimes it is advisable to increase the default value a bit. The value of **PARABOUNDARY** defines how far (in % of the electrode extension, default=5) the boundary is outside of the electrodes (red dots). Between the electrodes there are auxiliary points (in black) that define the mesh refinement at the surface, which is controlled by the key **PARADX**. A value of 0.5 means one additional point (as in Fig. 8), 0.3 means two points. Lower values means that two points are set to the left and right of each electrode (if **EQUIDISTBOUNDARY=0**). If **EQUIDISTBOUNDARY=1** is set (default), 1/PARADX is rounded to a natural number and equidistant points are set. The standard value for **PARADX** is 0.25 for flat topography and 0.33 with topography.
The coarse-ness towards the boundary is controlled by the mesh quality **PARA2DQUALITY**, which denotes a minimum angle. The higher the quality is, the more regular the triangles become, and the numerical accuracy increases but with an increasing number of cells and thus run-time. In triangle[13] (Shewchuk, 1996) version 1.6, our favoured 2d mesh generator, the range goes from 25-30 (bad quality) to 34-35 (good quality). We use the default tarting value of 34, but opt to increase it towards 35. Another way of avoiding a too coarse mesh is the maximum cell size by setting **PARAMAXCELLSIZE** (in m^2). Alternatively the parameter **PARAEDGELENGTH** is used to compute the area of a regular triangle to be used for **PARAMAXCELLSIZE**. We recommend editing the cfg file and calling `bert bert.cfg showmesh` until the user is satisfied.
In Figure 9 the resulting parameter meshes for different settings are displayed. Clearly, too low mesh qualities can lead to large and coarse triangles at the lower boundary, we recommend values between 34 and 34.6. Restricting the maximum cell size can help but also create artificially low triangle sizes somewhere in the model. For the outer model part there is the switch **BOUNDARY**, measured in multiples of the electrode extension, i.e. a value of 5 means that in our case there is 5x40 m space in -x, +x and -z direction.

2.4. Regular meshes

In contrast to irregular meshes, regular meshes need less control, but can lead to much higher number of cells and nodes compared to triangular meshes for the same refinement at the electrodes. Particularly the outer boundary becomes inefficient with rectangles. Therefore the regular parameter meshes are surrounded by triangles. Regular (quadrangle) meshes can be easily used by using **GRID=1** (Python+pygimli needed). Additionally to **PARADX** and **PARADEPTH**, the key **PARADZ** can be used to define the first layer thickness (otherwise

[13]see `https://www.cs.cmu.edu/~quake/triangle.research.html`

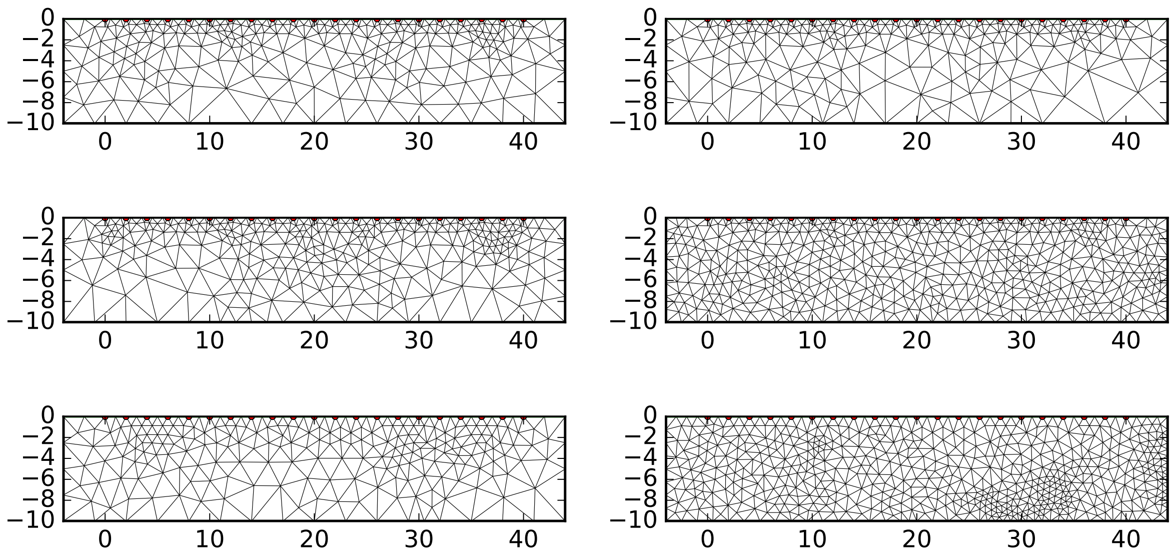


Figure 9: Meshes with different parameters: dx=0.25 & q=34 (top left, default), dx=0.25 & q=33 (top right), dx=0.25 & q=34.5 (center left), dx=0.25 & q=34, maxA=1 (center right), dx=0.5 & q=34 (bottom left), dx=0.5 & q=34.5, maxA=1 (bottom right)

equal to **PARADX**). The key **LAYERS** defines the number of layers (by default 11) created in such a way that the layer thickness increases linearly. Figure 10 shows the result of the default grid options, which is very comparable to Figs. 6b or 7.

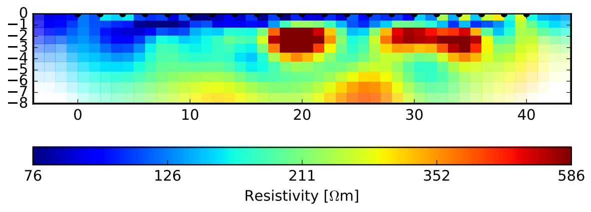


Figure 10: Inversion result for a regular parameter grid.

2.5. Induced polarization

There are several types of induced polarization (IP) data as addition to resistivity amplitude, either measured in the frequency domain (FD):

phase shifts using single or multiple frequencies

frequency effects calculated from resistivity measurements at two frequencies

or in the time domain (TD):

total chargeability representing the integral of the decay curve (in ms)

integral chargeability of a time window normalized by gate length & voltage (in mV/V)

The BERT IP inversion is oriented at single-frequency FD measurements, but can be applied to any other type of data as long as the effects are small due to linearity. IP data are represented as a column named ip in the data file, which is inverted after the (DC) resistivity inversion. The actual IP inversion uses, as described by Martin and Günther (2013), the Cauchy-Schwarz relations between real and imaginary parts to formulate the inversion of the imaginary resistivity as a linear problem. Finally, the imaginary resistivity is transformed back into a phase (in mrad). As for IP inversion a different regularization strength might be used, you can use the key **LAMBDAIP**, otherwise it is taking the **LAMBDA** value from the resistivity inversion. However, due to the linearity of small phases any other type of data (apparent chargeability in mV/V or msec or frequency effect in %) can be input and thus the output will have the same unit. As an example for this quite simple approach we provide the data of the healthy oak presented by Martin and Günther (2013) in the section 3.3.
In many cases, spectral IP data are gained by measuring multiple frequencies or analysing different parts of the decay curve. As this goes beyond the data format and analysis, spectral analysis is not part of the command line bert. Instead, we provide a Python class for SIP data handling, inversion and post-processing (Günter et al., 2016). It directly reads field SIP data and can do a spectrally constrained inversion of all frequencies followed by a Cole-Cole analysis as described by Günther and Martin (2016).

3. Other 2D geometries

3.1. 2D ERT with topography

In 2d inversion, topography is easily integrated by setting the heights of the electrodes. All the rest should be done automatically, if necessary, additional electrodes must be inserted. However, rarely all electrodes will be measured topographically. Often it is sufficient to have a few points. Note that in the current stage BERT requires the topographical information in the first section, not at the end of the file. For this case we recommend the use of DC2dInvRes (Günther, 2007) that will roll the positions along the surface. For this task, use `Data:Save Ohm file` and specify whether the x values are along measure tape or real x.

The slagdump profile

This field case is the very first case described by Günther et al. (2006), the base BERT article, as a 3D case, friendly provided by the M. Furche, Federal Institute of Geology and Natural Resources (BGR), Hannover. The data in `examples/inversion/2dtopo/slagdump` is one of the profiles crossing the slag heap (Fig. 10). A Wenner array with $a = 2\,\mathrm{m}$ spacing was applied yielding 222 data points. The topography was measured at 8 points by levelling and appended to the original file before converting it with DC2dInvRes. We initialize the inversion with standard options using the command:

```
$ bertNew2DTopo slagdump.ohm > bert.cfg
```

In case of non-trivial geometry we have another numeric task, i.e. to compute the primary potentials (that are otherwise computed analytically), which is associated by several key words starting with PRIM. Similar to the parameter mesh, the **PRIMDX** value specifies the (absolute, not relative!) refinement at the electrodes and **PRIM2DQUALITY** defines the mesh growth. Additionally, the **PRIMP2MESH** (default 1) determines the primary potentials being computed by quadratic shape functions (P2 refinement). As stated by Rücker et al.

(2006) the necessary refinement for a P2 mesh is about $a/10$ and $a/100$ for a P1 mesh. Usually the points at the surface are linearly interpolated, **SPLINEBOUNDARY=1** forces a spline interpolation, which is useful for "round" geometries or smooth topography.
After computing the primary potentials including interpolation onto the secondary forward mesh (done by `bert bert.cfg pot`), there are two files `primaryPot/pot.ohm` and `primaryPot/pot.collect` describing the potential for a unit conductivity for the chosen array and all electrode combinations, respectively. These are used to compute the geometric factor and the apparent resistivity (`bert bert.cfg filter`). By calling `bert bert.cfg topoeff` one can see the topographic effect (Rücker et al., 2006), i.e. the ratio of the geometric factors with and without topography $t = G_{flat} \cdot u_{topo}(\sigma = 1\,\mathrm{S/m})$. `bert bert.cfg showtopocorr` displays additionally the apparent resistivity based on the flat and topographic geometric factor (s. Fig. 11). Several anomalies can be explained solely by topographical undulations. The resistivity distribution shows a conductive interior and a resistive hard pan as discussed by Günther et al. (2006).

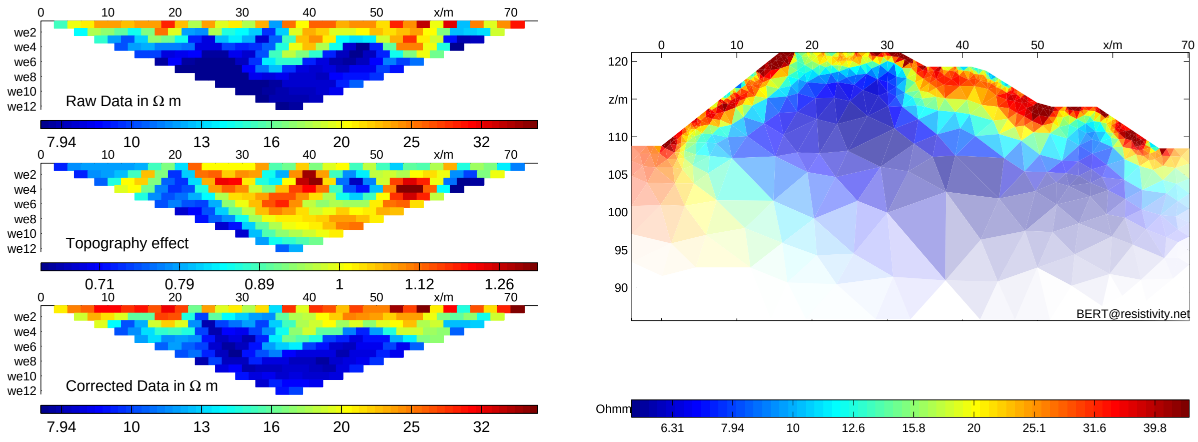


Figure 11: Topographic effect and inversion result of the slagdump site.

3.2. 2D cross-hole data

Of course cross-hole measurements can also be inverted using BERT. The height of each electrode must be set to the elevation minus depth. However, since we cannot distinguish whether it is topography or a buried electrode we must create the geometry by hand.
The example in `examples/inversion/2dxh` was produced by O. Kuras of the British Geological Survey (BGS) in the ALERT project (Kuras et al. (2009)) for time-lapse inversion (see section 6.3). It represents about 1300 data obtained by cross-hole measurement between 5 very shallow (0-1.6m) boreholes. In order to create an inversion mesh we would create a small box with marker 2 (inversion) inside of a big box that is used for forward calculation (marker 1) by 8 points and 8 edges.
This is more rigorously implemented by the PLC script `polyFlatWorld` which automatically calculates the size of the model and the boundary around the electrodes from the parameters **BOUNDARY**, **PARABOUNDARY**, **PARADX** and **PARADEPTH**. It is controlled by **PARAGEOMETRY="polyFlatWorld 2dxh.ohm"**[14]. As before, we can use **PARADX**

[14]Previous versions included the word source (run it in the current shell) which could lead to an error. As we detect it to be a command, we source automatically to be able to access the parameters given in the cfg file.

to refine the model at the electrodes by a node between each of the 0.1 m separated electrodes using **PARADX=0.05**. Note that, different from 2d surface measurements, it is treated absolutely by `polyFlatWorld`. Due to the layering, we use **ZWEIGHT=0.1**, a larger **LAMBDA=100** and **ROBUSTDATA=1**. Figure 12 shows the obtained resistivity distribution at the very beginning of a tracer experiment.

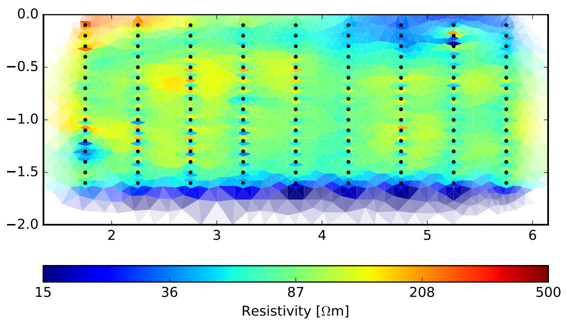


Figure 12: Inversion result of the crosshole data set.

In BERT 2, electrodes do not have to be points anymore. However, for surface measurements it makes almost always sense to use their positions as node to construct the mesh. This is different in cross-hole geometries. Similar to the script `polyFlatWorld` there is a script `polyFreeWorld` that replaced the former in the cfg file.
Since there a too few constraints to triangle, the resulting mesh is very coarse over the whole area. We could place some points to ensure locally small triangles and use a good mesh quality for slowly increasing edge lengths. Alternatively we can (globally) provide a maximum triangle area, e.g. by using the option **PARAEDGELENGTH**, where it the area is computed based on a regular triangle. By setting the actual value to 0.05 we obtain a fine enough mesh independent on electrode positions.
The third way is to create a regular mesh with either **GRID=1** or a user-defined Python script (`examples/inversion/2dhx/reg/mymesh.py`). Figure 13 shows the two alternative results. In general the resistivity distribution is very similar, i.e. the good conductor at depth and the resistor on the top left. However there are differences in the small-scaled aquifer anomalies.

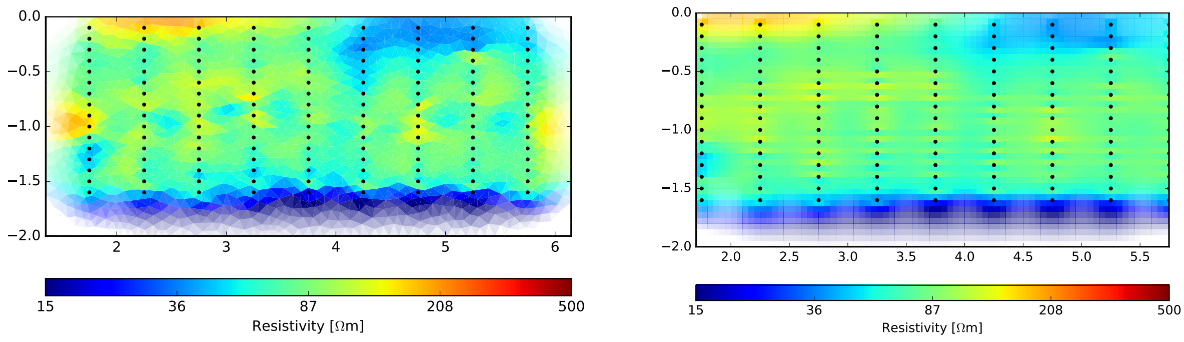


Figure 13: Inversion results on alternative mesh types: free electrodes (left) and rectangular mesh (right).

3.3. Closed 2d geometry - tree tomography

ERT on hollow lime tree

For 2d bodies the electrodes are usually on the whole boundary and the PLC can easily be formed by a close polygon. For tree geometry a dedicated GUI named TreeBERT (before DC2dTree) was created making it easy to process the data visually. EIT on trees has been successfully established to investigate decay of trees (Martin and Günther, 2013). The example in `examples/circle/tree` was measured and provided by Niels Hoffmann, formerly HAWK Göttingen. It represents a lime tree, measured by 24 steel electrodes that are plugged into the bark. Dipole-dipole measurements have been applied using a Geotom instrument.
The configuration file reads as follows

```
DATAFILE=hollow_limetree.ohm
DIMENSION=2
TOPOGRAPHY=1         # activates the primary mesh
CYLINDER=1           # defines a closed geometry
SURFACESMOOTH=1      # makes a nicer surface
EQUIDISTBOUNDARY=1   # equidistant refinement
PARADX=0.2           # 5 segments between the electrodes
PARA2DQUALITY=34.8   # very good quality, almost the upper limit
SPLINEBOUNDARY=1     # round geometry
PRIMDX_R=0.001       # refinement of primary mesh in radial direction
LAMBDA=10            # regularisation strength
BLOCKYMODEL=1        # enhance contrasts by robust (L1) methods
```

For this case an equidistant refinement, the use of splines and a high quality ensures a nice mesh with a round boundary. The primary refinement is done in radial direction. Additionally we used the robust modelling in order to obtain a clearer contrast of the high resistivity.

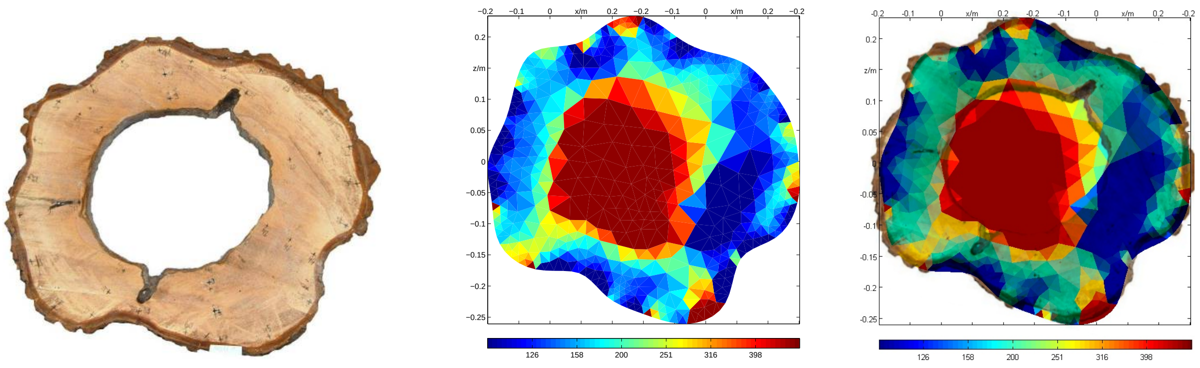


Figure 14: Tree cut (left), inversion result (center) and overlay.

After the measurements the tree was cut and revealed a cavity inside caused by decay. Figure 14 shows a photograph, the inversion result and an overlay of both. Clearly the cavity is marked by high resistivity that is in almost perfect accordance with the photo.

ERT and IP on healthy oak

The data provided in `examples/inversion/circle/oak/` are described by Martin and Günther (2013), who also give details on both the measurement and the inversion. Measurements were taken at a standing healthy oak (section HI in the paper) in both summer and winter. We take the measurements in summer (`hp5-s.dat`) and invert it with the given cfg file `hp5-s.cfg`. Looking into the output that is also stored in `bert.log` we see the following:

```
Linesearch tau = 0.12
3: Model: min = 211.068; max = 435.779
3: Response: min = 252.394; max = 359.383
3: rms/rrms(data, Response) = 82.4875/25.4984%
3: chi^2(data, Response, error, log) = 68.29
3: Phi = 19052.9+4.17273*20=19136.4
Reached data fit criteria (delta phi < 2%). Stop.
Processing ip data min=9.84261 max=37.4019
... # some more stuff here depending on verbosity
```

After inverting the DC data, dcinv looks for the IP data and would stop if there were none (min/max=0). Then, the IP data are inverted, by default the output is silent to that we do not see the individual inversions (as it would hinder looking at the DC data fit), but only the last iteration.

```
15: Model: min = 0.00531699; max = 33.0494
15: Response: min = 1.49463; max = 17.6232
15: rms/rrms(data, Response) = 0.893923/12.1318%
15: chi^2(data, Response, error, log) = 7.36795
15: Phi = 2055.66+290.928*20=7874.23
Resulting phase: min=0.0178055 max=111.453 mrad.
```

In this case the model (imaginary resistivity) lies between some milli-Ohms and several Ohms, corresponding to phase values of more than 100 mrad. The data fit is 12%, however this measure might not be meaningful when there are ip data close to zero present. The chi-square error is based on an assumed phase error of 1 mrad. The resulting phase image can be shown by
`$ bert hp5-s.cfg show phase.vector` or exported to pdf using
`$ bert hp5-s.cfg mkpdf phase.vector`.
The default regularization strength (**LAMBDA** value is taken for both) was in this case good for both DC and IP inversion. In general, different values might be appropriate. If the parameter **LAMBDAIP** is given then this value will be used for IP inversion only. Note that a rerun of the calculation (calc) needs to be done. More flexible possibilities for IP inversion are available through the Python classes Resistivity (for single DC/IP inversion) and SIPdata (for spectral IP inversion).

4. 3D geometries

3D surface measurements

3D surface measurements can be carried out in several variants:

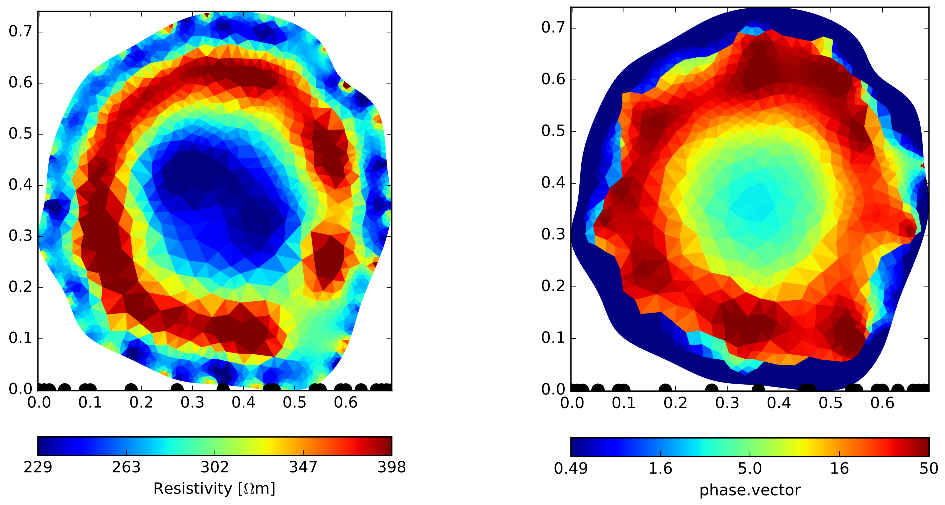


Figure 15: Resistivity (left) and phase (right) image for the healthy oak tree (in summer).

1. Layout of an electrode grid. However, due to the limited electrode number grids are restricted to small areas.
2. Parallel (and perpendicular) profiles along the coordinate axes.
3. Profiles in arbitrary directions due to accessibility limits.
4. Non-profile layout, e.g. large-scale dipole-dipole experiments.

In any case, the electrode positions and measurements must be defined according to the unified data format. The data for the first two types can be easily organized by hand. For number 3 (and 2) we suggest to prepare 2d files and to write a pro-file containing of lines with the 2d file name and x-y pairs of points where the line is going. This file can be read into DC3dInvRes Günther (2008) and used to write the 3d file. In case of topography it is best to do the tape correction on the 2d files before using DC2dInvRes and Export Ohm.
The most flexible element in 3d is the tetrahedron. The tetrahedralization is done by a mesh generator, our primary choice is Tetgen (Si, 2008), a free and versatile quality mesh generator (Si, 2015) that can be retrieved from `https://tetgen.org`, however you can also use the mesh generator GMesh (Geuzaine and Remacle, 2009) as described in the Appendix. The quality measure is different from 2d and describes a radius-to-edge ratio, note that small values point to higher quality. Appropriate values for (primary field) forward calculation are 1.12 to 1.2, for the inverse (and thus secondary) mesh values of 1.2-1.5 are appropriate, the keys are **PRIM3DQUALITY** and **PARA3DQUALITY**.

4.1. Flat-earth surface measurements

In `examples/inversion/3dflat/gallery` is a data set in the field where the 2dflat example (section 2.1) was measured. More information can be found in Günther (2004). It comprises a grid of 9x14 electrodes. Dipole-dipole measurements have been measured on all x and y profiles. In the data file is an error of constant 0.0 that will be overrided automatically. An inversion project file with default parameters is created by:

```
$ bertNew3D gallery.dat > bert.cfg
```

The inversion is then fully run (with command `all`) converging to a chi-squared misfit of about 2 (rrms=4-5%). In order to fit the data better, the regularization parameter is decreased using **LAMBDA=5**, which leads to an relative rms error of about 3% ($\chi^2 = 1$). Another way is to use an anisotropic regularization (**ZWEIGHT** below 1) also leading to $\chi^2 \approx 1$.
The inversion result is already saved a vtk file names `dcinv.result.vtk` and can be viewed by 3D visualizaton software such as ParaView[15].

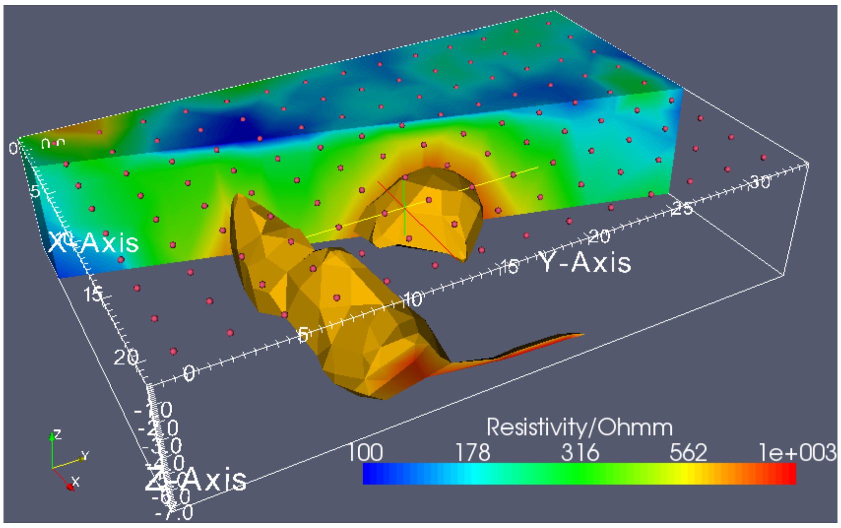


Figure 16: Inversion result of the 3d gallery data set using a smoothed iso-surface of 650Ωm and a Plane Clip, the red spheres are the used electrodes.

Figure 16 shows a Paraview visualisation that has been created by the following steps: i) Cell Data To Point Data, ii) Clip by Scalar 650 (Ωm), iii) Extract Surface, iv) Smooth Surface, v) Another Clip based on Cell2Point with Plane, vi) representation of the input as Outline and Cube Axes. The color bar is logarithmic with a manual range of 100-1000Ωm. The electrodes have been included as point vtk file and displayed by Glyph as Spheres of radius 0.05. After some exercise the reader will be able to create nice images, plots and calculate results such as extensions or volumes of geological bodies.

4.2. 3D Topography

The definition of a 3d topography is much more complicated than in 2d, where every shape can be described by a simple polygon. The input PLC consists of faces instead of edges, the resulting poly file has a similar but different format[16]. Generally the proceeding is the following: i) create a flat surface mesh, ii) interpolate heights from topographic information, iii) make a small (inversion mesh) and a large (forward mesh) box around it, iv) make refinement, if necessary, and v) create the mesh using tetgen. The whole procedure is introduced by Günther et al. (2006) and explained in more detail by Udphuay et al. (2011) for the example of a steep cliff.
For specifying topographic information, there are two different ways:

[15]see `https://www.paraview.org`
[16]See `https://tetgen.org`.

- the electrodes in the data file have an elevation and all other points are interpolated
- there is a digital elevation model (DEM) or at least a list of measured topo points (in a 3-column file containing x,y and z)

Whereas the first case is sufficient for smooth topography and/or dense electrode coverage, the latter is more general. The topographic points can be a (mostly regular) digital elevation model (DEM) or a list of irregularly measured points. In any case they are Delaunay triangulated to provide an elevation for every point in the surface of the mesh, i.e. the electrodes but also the whole forward mesh so that points outside the actual electrode spread are especially helpful. Both ways can be combined, however we recommend setting the height of the electrodes in the data file to zero so that these are interpolated from the DEM. Electrodes with measured elevations that are ensured to be consistent with the DEM can be appended to the topo file. The ASCII file containing the topographical points as x-y-z columns is specified by **TOPOPOINTS=filename**.

In `examples/inversion/examples/acucar` there is a project measured by the Federal Institute of Geology and Natural Resources (BGR) Hannover (M. Furche). The site is an old slag dump that comprises a topography reminding on the sugar hat in Rio. Two resistivity profiles have been measured crossing the top of the isolated hill. Another profile was realised around the hill in a more or less constant elevation. Although this is not a dense sampling as an electrode grid it should be sufficient to obtain a rough image.

Additionally to the electrodes, some topographical points have been measured and put into the file `points.xyz`. So we create a new project using

```
$ bertNew3DTopo acucar.ohm > bert.cfg
```

and add the line **TOPOPOINTS=points.xyz** to the model. If we now call

```
$ bert bert.cfg meshs
```

we see the mesh does not show the hill, since the topography overrides the electrode elevation. Therefore we have to add the electrode definition (lines 3-230) to the topography file and see then the hill (Figure 17 left). However due to the point density the electrode line appears as a sharp edge that is not really the truth but sufficient in this case.

In other cases we might have a digital elevation model. In order to show this on the same example, we created one by cubic interpolation of the available points on a regular grid of 2m spacing. In order to avoid interpolation errors between the electrodes (Udphuay et al., 2011, Fig. 10) we created a polygon file poly.xyz for the three profiles[17] and introduce it by **TOPOPOLY=poly.xyz**. Figure 17 shows the surface mesh of both variants. The sharp edges are now disappeared.

Finally the inversion result is visualised in Figure 18. It shows a conductive interior of the slag dump and different sediments at the surface, e.g. a resistive top. Of course the data coverage is low between the profiles and at the model boundaries. Therefore the model becomes more or less interpolated by the smoothness constraints.

For creating appropriate 3D meshes, one might need to play with the mesh options before going into inversion. Instead of the usual refinement parameter **PARADX** to refine the mesh at the surface, you can consider working with the parameters of the triangular surface: **SURFACEQUALITY** (the quality of triangle mesh, see 2D inversion), **SURFACEMAXTRISIZE** (the maximum element size) and **SURFACESMOOTH**. Frequently one observes ugly

[17]Several polygons are separated by a blank line.

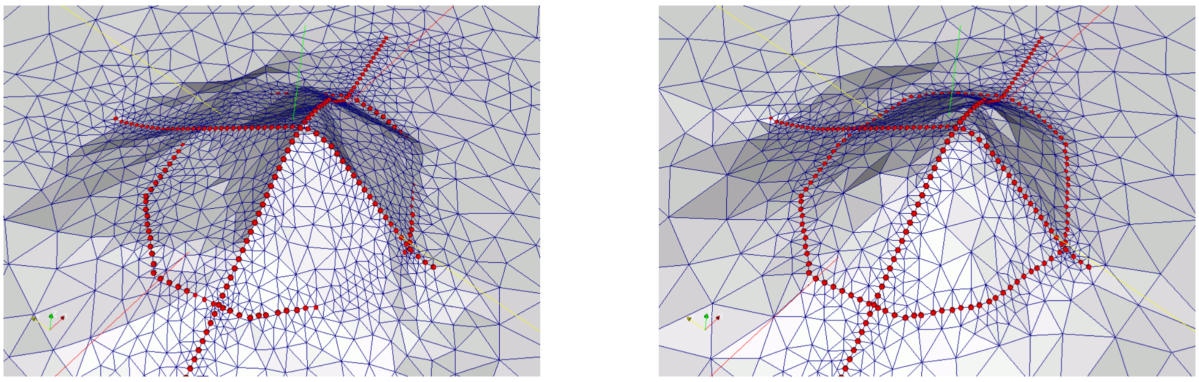


Figure 17: Surface mesh for the point-wise topographic information (left) and the digital elevation model (right), the electrodes are shown as red points.

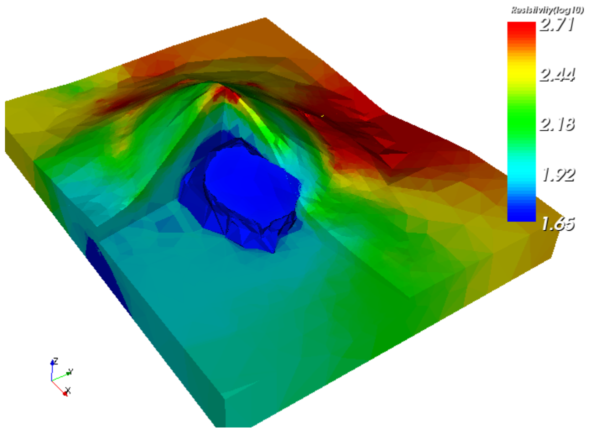


Figure 18: Inversion result of the 3dtopo case.

shaped tetrahedra at the boundary of the parametric domain and a correspondingly coarse discretization. This can be overcome by specifying **LOOPTETGEN=1**, i.e. the mesh generator is called in a loop to improve mesh quality step-by-step.

4.3. 3D-Crosshole measurements

Crosshole measurements can of course be applied three-dimensionally. The example in `examples/inversion/3` was presented by J. Doetsch from ETH Zurich. The data file `3dhx.ohm` comprises 753 data between 4 boreholes in the saturated zone (d=4-10m) and is part of a monitoring experiment. We create a cfg file using

```
DATAFILE=3dxh.ohm
DIMENSION=3
PARABOUNDARY=15   # to get a bit more space around the electrodes
PARAGEOMETRY="polyFlatWorld $DATAFILE"
```

By using **ZWEIGHT=0.2** we can enhance the predominantly layered structures. The inversion converges then with defaults down to about $\chi^2 = 1$. Figure 19 shows the final result.

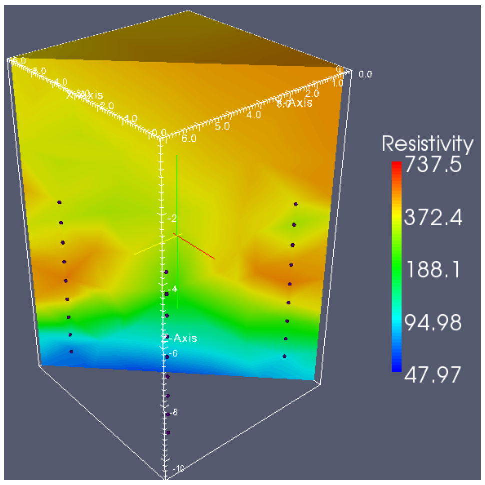


Figure 19: Inversion result for the 3D crosshole case.

4.3.1. Topography and buried electrodes

If both heights and depths below surface are given, we cannot use `createSurface`/`createParaMesh` and `polyFlatWorld`. In this case we suggest to create a data file with the surface electrodes (either with topo or with zeros and a topo file) and a list of subsurface electrodes including real. The normal PLC generation is done with the first and the latter are then added using `polyAddVIP`. Importantly the order of electrodes must be set such that first the surface electrodes and then the buries electrodes appear. The resulting script is introduced using **PARAGEOMETRY=myscript.sh** which can look as follows:

```
cp bert.cfg surface.cfg      # makes a copy with all options
echo DATAFILE=datafile-without-electrodes.dat >> surface.cfg
bert surface.cfg domain   # creates mesh/mesh.poly
polyAddVIP -m -99 -f borehole-electrodes.xyz mesh/mesh.poly
```

Finally all electrodes are in the PLC in the right order and all should be well.

4.4. Closed 3D geometries

Closed geometries are actually easier than open ones since we do not need a mesh prolongation and two different regions. However since the whole boundary is of Neumann type, we must ensure two additional conditions that are not necessary in the open case:

- The current cannot vanish in infinity, therefore we must use dipole sources, e.g. by a reference current node.
- Since only derivatives are present in the boundary value problem, we must make the forward solution unique, e.g. by adding a reference potential node, whose potential is forced to zero.

A 3d model tank

There are numerous examples of ERT measurements in model tanks that can be either cylindric (Garre et al., 2010) or cubic (Bechtold et al., 2012; Persson et al., 2015). In the Federal Institute of Geology and Natural Resources (BGR), Hannover, a cylindrical model tank was created (Falcon-Suarez et al., 2016) in order to make infiltration experiments with material from sawdust or from slag dumps. The column has diameter of 30cm and a height of 80cm. In each of 5 rings with 5 cm vertical distance 24 steel electrodes of 2cm length were installed. Dipole-dipole measurements have been applied to all rings yielding a number of 320 data. The example is located in `examples/inversion/3dtank`.
Since the parameterization cannot be detected automatically from the file, we have to create the mesh input, i.e. the PLC in `mesh/mesh.poly` by hand using a script. There is a poly tool `polyCreateCube` creating a unit cube. With the option *-Z* is creates a unit cylinder instead, which has to be scaled appropriately. Then we put in the electrodes as points[18] with the marker -99. We insert two additional nodes with markers -999 and -1000 that are used for current reference and potential reference. So the script reads:

```
MESH=mesh/mesh # PLC name
polyCreateCube -v -Z -s 48 -m 2 $MESH # create unit cylinder with 48 segments
polyTranslate -z -0.5 $MESH           # moves it such that top is zero
polyScale -x 0.3 -y 0.3 -z 0.8 $MESH  # scale to radius 0.15 & height 0.8
cat soil_column.dat | head -n 82 |tail -n 80 > elec.xyz # extract electrodes
polyAddVIP -m -99 -f elec.xyz $MESH   # add electrodes to mesh
polyAddVIP -m -999 -x 0 -y 0 -z 0 $MESH # current reference node
polyAddVIP -m -1000 -x 0 -y 0 -z -0.8 $MESH # potential reference node
polyConvert -V -o $MESH-poly $MESH    # convert to vtk to load it to paraview
```

We create an empty cfg file (or use bertNew3DTopo) with the lines

```
DATAFILE=soil_column.dat
DIMENSION=3
TOPOGRAPHY=1
CYLINDER=1   # ensures the closed geometry
```

We add our PLC script in form of a **PARAGEOMETRY** variable such that `mesh/mesh.poly` will be created by it.

```
PARAGEOMETRY=mymesh.sh
```

A further refinement can be achieved by quality improvement (**PARAQUALITY**) as it is quite coarse, local refinement (**PARADX**) or maximum cell size (**PARAMAXCELLSIZE**). In order to obtain an accurate we use a refinement for the primary mesh of 1cm and quadratic shape functions ending in about 32000 nodes.

```
PRIMDX=0.01
PRIMP2MESH=1
```

Figure 20 shows the course from the mesh input via the parameter mesh to the final result.

[18]Since the electrodes show no significant extension compared to the column size, we put the points not onto the surface but moved it 1cm inside, see Rücker and Günther (2011) for optimizing the optimum point.

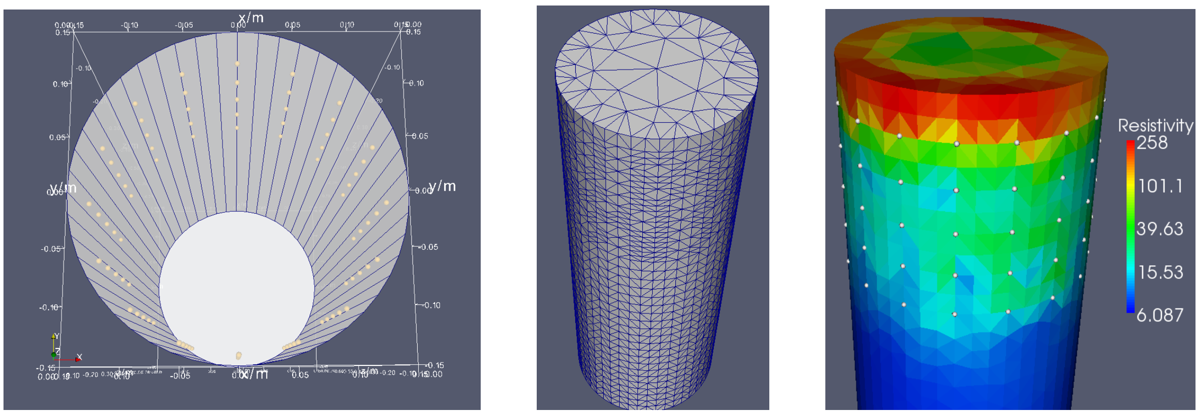


Figure 20: PLC (left), parameter mesh (center) and inversion result (right) of the soil column experiment.

For experiments with standing columns the z direction is different due to gravity. Therefore it might be helpful to run the inversion on a triangular prism mesh and to regularize the vertical direction through **ZWEIGHT**. Note that such a script that is applied through **PARAMESH** should be accompanied by a **PARAGEOMETRY** script generating the outer PLC for the primary potentials.

In important Note:

We computed the electrode positions such that they fall exactly on the boundary of our cylindrical prism (there is some snapping to avoid rounding errors). With a different number of prism sides it can easily happen that electrode positions fall outside of the PLC or very closely inside. Whereas the first case leads to errors, the latter leads to unnecessary fine discretization at these points. Generally, there are two alternatives that have been discussed by Rücker and Günther (2011):

- Electrodes are never infinitely small points (implying infinite current density) but have real sizes. If the electrode size is significant compared to their distance, one can model the real shape by the complete electrode model (CEM) as described by Rücker and Günther (2011). Although this is easy to model, it this implies effort when creating the different meshes in inversion.
- An alternative is to put the electrode inside. Rücker and Günther (2011) simulated a cubic model tank and found that a replacement point at about 60% of along the electrode axis decreases the errors to a minimum.

5. Incorporating prior information

Often there is additional information about the subsurface. An incorporation into the inversion process is always to be preferred over a comparison of the results.

5.1. Structural constraints

Günther and Rücker (2006) presented a more general minimisation approach that allows for arbitrary weights for each boundary between model cells. The numerical approach is explained in more details by Rücker (2010). In existence of a known discontinuity this can be set to zero allowing for (but not enforcing) an arbitrary jump in resistivity. Such constraints can be used in both 2D (Bazin and Pfaffhuber, 2013) and 3D Doetsch et al. (2012) cases.
The example also presented by Rücker (2010) is located in (`examples/inversion/inversion/-2dstruct`) was measured and friendly provided by the K-UTec GmbH Sondershausen (T. Schicht). Aim of the study was bedrock detection carried out with resistivity and refraction seismics. The velocity structure showed to be a very clear 2-layer case. So the result (layer boundary) of the refraction study can serve as structural information.
The file `bedrock.xz` contains the course of the boundary as x-z pairs. We now include this file into the configuration file using the **INTERFACE** option. In order to compare the result with and without structure we call

```
$ bertNew2D bedrock.dat > bert.cfg
$ bert bert.cfg all show
$ echo INTERFACE=bedrock.xz >> bert.cfg
$ bert bert.cfg all show
```

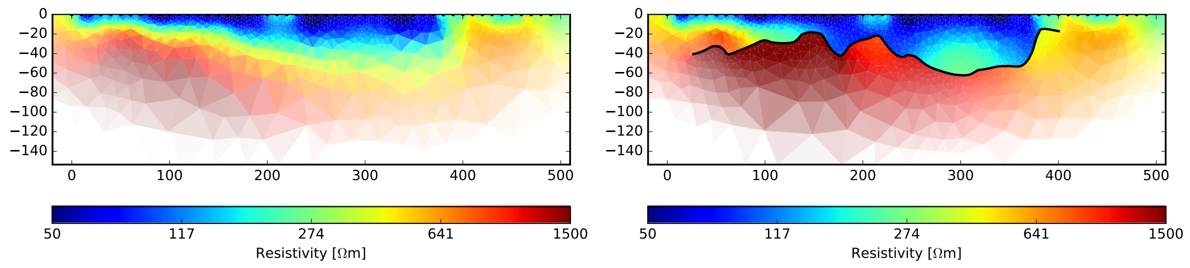


Figure 21: Resistivity distribution without (left) and with (right) structural information.

Figure 21 shows the subsurface images without and with the structural information. Obviously the additional information leads to a much clearer image of the subsurface. At most positions there is a sharp resistivity contrast at the boundary. However at some positions there is either a difference to velocity or the refraction result is ambiguous.
Several interfaces may be present in the **INTERFACE** file, separated by a blank line. Structural constraints may also come from borehole descriptions indicating a layer at a certain depth. This results in several small lines extending to the sides depending on the lateral representativity of the boreholes. Three-dimensional interfaces are point lists that are triangulated and put as facets in the model.

5.2. Region-specific control by an underwater example

In the preceding example we almost cut the model into two parts that are known from prior knowledge. Very often, different parts of the subsurface, i.e. regions, need different treatment. This can be geological units or artificial stuff like boreholes or cables in the ground. Flechsig et al. (2010) used only single regions to invert a very sparse data set. Doetsch et al. (2010) used boreholes as single regions to image an aquifer with cross-hole ERT. Coscia et al. (2011) embedded this into a large 3D model to monitor aquifer dynamics.

The technique behind regions is described in more detail by Rücker (2010). Regions are parts of the mesh with a constant marker, their behaviour can be easily described by a region file. The distribution of this marker can be looked at by calling `bert cfg showMarker`. By default, there is an inversion region (marker 2) embedded into a modelling region (marker 1) whose resistivities are not part of the inversion but are used in forward calculation by prolongation from the inversion region. We suggest using the lowest number for this background region as this is the default behaviour in case several regions are detected in a non-Neumann body.

The region file

The main control over the inversion is done by a so-called region file using the keyword **REGIONFILE** (and **TIMELAPSEREGIONFILE** for the time step inversion). Of course, many options such as **LAMBDA**, **ZWEIGHT**, **UPPERBOUND**, **LOWERBOUND** can be set directly in the cfg file for the whole inversion domain (whether it is one region or more), but can be adjusted for different regions individually. For a more rigorous description we refer to the GIMLi tutorial, appendix C. The region file is a column file with different parts separated by a token list (led by a # character), which can look like

```
#no start Ctype MC  zWeight Trans lBound uBound
0   100   1     1   0.2     log   50     1000
1   30    0     0.2 1       log   10     200
#no single start
2   1      100
#no background
-1  1
#interregion
0   2  0.2
```

The number No can also be a * for all regions and should lead the token list. The other tokens define values for the numbered token.

start starting resistivity

Ctype constraint type (1/-smoothness 1st/2nd order, 0-minimum length, 10/20 mixed 1st/2nd with zeroth order)

MC model control (individual lambda relative to the global lambda)

zWeight constraint weight for vertical boundaries (only for Ctype=1)

trans model transformation, log(LU) by default, others are lin or tan

lBound lower resistivity bound (for Trans=log)

uBound upper resistivity bound (for Trans=log)

single this region is treated such that the resistivity is constant

fix this region has a known conductivity that is used in the forward calculation but not in inversion

background this region is background and filled with resistivity of the neighbouring regions for the forward calculation, for a non-Neumann domain without region file the lowest number is the background by default

interregion two regions are by default decoupled (constraint weight is zero), but can be coupled[19]

In the above example there are two main regions with different starting values, valid ranges and constraints. Another region is treated constant and is coupled to region 0 weakly. Finally, there is one background region.

The lake case

The lake data set was measured by the Leibniz Institute for Applied Geophysics, Hannover (W. Südekum and T. Günther) and is located at `examples/inversion/region/lake`. Aim was to delineate sedimentation structures beneath the Feldungel lake near Osnabrueck. Electrodes have been spread out from one shore along the lake bottom onto the other shore. The spacing was 2m and both Wenner-alpha and Wenner-beta were measured and combined. Electrode positions (0 to -2.6m height) and resistances are the input file. Since the lake resistivity is a distinct known (both its geometry and the resistivity of 22.5 Ωm) body, it will be treated differently.

1. We start as for a topographic case and generate the meshing input `bert bert.cfg domain`
2. As a result we obtain the poly file `mesh/mesh.poly` which we copy to `mymesh.poly`[20]
3. We need to add the water surface by an edge between the left and the right shore (innermost points with zero altitude). A view into the poly file shows that they are represented by the points 3 and 138. So we add another edge at the end of the edges (line 303) by inserting the line *151 3 138 -1* (number n1 n2 edgemarker) and increase the number of edges in line 152 from 151 to 152
4. Finally we add a region marker somewhere in the lake with marker 1 (not inverted) by appending the line *2 50 -1 1 0.0* (number x y marker maxTriangleSize) and increasing the number of regions from 2 to 3.

We use this altered poly file in the inversion by introducing **PARAGEOMETRY=mymesh.poly** into the cfg file. This means that mesh.poly in the mesh directory is created by copying mymesh.poly. Alternatively we can put here a bash script or a Python script that generates the geometry (mesh/mesh.poly).
Figure 22 shows a section of the input PLC after calling `bert bert.cfg showpoly`. As usual, we have the outer region with marker 1 and the inner region with marker 2, additionally we have the marker 3 for the lake. By default the outer region is a background region in case several regions are available without a region file.
We use the following options

[19]You can use the former key **Inter-region** as well but this is erroneous due to the variety of dashes in different key tables we recommend using **Interregion**.
[20]see triangle page `https://www.cs.cmu.edu/~quake/triangle.research.html` for file description

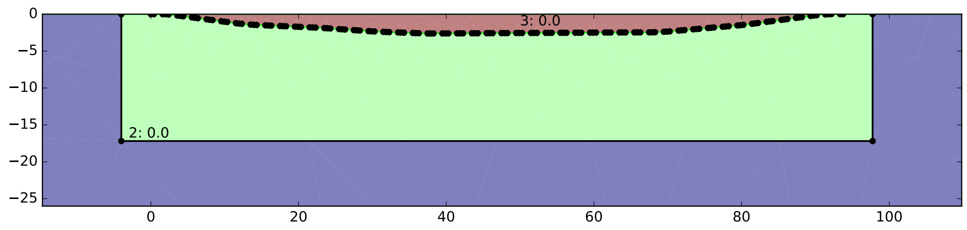


Figure 22: Representation of the input PLC for the lake case.

```
ZWEIGHT=0.2        # enhances layered (sediment) structures
OVERRIDEERROR=1  # do not use the measured errors in file (optimistic), but:
INPUTERRLEVEL=2        # 2% plus
INPUTERRVOLTAGE=20e-6 # 20 microvolts
cMin=15
cMax=500
```

and the inversion converges at $1 < \chi^2 < 2$. Figure 23 shows the resulting resistivity distribution. The lake sediments show generally by very low resistivities even below the measured value of 22.5 Ωm.

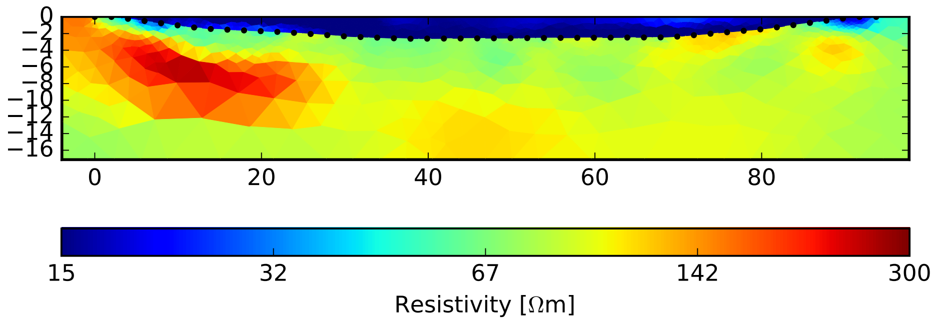


Figure 23: Inversion result of the water case using two regions that are automatically decoupled.

The regions are automatically decoupled (no smoothness constraints between them) as in the last example. Figure 24a shows the inversion result using normal treatment (by setting the region marker of the lake to 2 or by using `polyFlatWorld` as **PARAGEOMETRY**), i.e. smoothness constraints across the whole model. We see a lot of structures related to the lake bottom and irrealisticly high water resistivity. Now we start treating the

```
REGIONFILE=region.control
```

We first define the water region (marker 3) as a single-parameter region and the subsurface as a normal inversion region with smoothness constraints that enhance vertical structures, a range of 10-1000 Ωm and a starting resistivity of 100 Ωm:

```
#No start Trans zWeight Ctype lBound uBound
2   100   log   0.1     1     10     1000
#No single start
3   1     22.5
```

```
#No background
1 1
```

Figure 24b shows the result. The main image is similar but the oscillation at the sea-bottom are gone. The resistivity increase at the bottom disappeared and reveals more information about the deeper layers. However, the resistivity of the lake obtains values of about 16 Ωm, which is lower than the expected value of 22.5 Ωm measured at the side, even though we used it as starting value.
We could narrow the resistivity value of the lake by expanding the lines for region 3 by upper and lower resistivity bounds:

```
#No single start Trans lBound uBound
3 1    22.5  log   22     23
```

Whereas a single region is represented by one unknown, one can also associating it with a fixed value, thus making it a background region (i.e. removing it from the inversion completely).

```
#No fix
3 22.5
```

Whereas a normal background (region 1) is filled up (prolongated) from the inversion model with neighbouring resistivity, this region is filled with the fixed value.
The result, shown in Figure 24c, is equally well fitting the data. Both results are obviously equivalent, in this full-space problem the lake resistivity cannot be independently obtained and has to be added by additional information.
Alternatively we can assume that the lake bottom sediments have very similar resistivity to the water. Therefore we remove the fix again and introduce inter-region constraints between the two regions:

```
#No single start
3   1      22.5
#Interregion
2   3      0.5
```

So the otherwise decoupled regions are connected by smoothness constraints with a strength that is a factor 2 weaker than normal. The result, equivalently fitting the data, is shown in Figure 24d. It is very similar to Fig. 24c and thus supporting the measured conductivity.
On the other hand it is not clear from our simple probes that the water resistivity is really constant. So we can treat the water as normal region with proper upper and lower bounds. In order to have our target structure in the water layer we increase the model control (relative regularization parameter) for that region by a factor of 2. The inter-region constraints are kept.

```
#No start Trans zWeight Ctype lBound uBound MC
2   25    log   0.1     1     10     1000   1
3   22.5  log   0.01    1     10     30     3
#Interregion
2   3     0.5
```

The result (Fig. 24e) however, is not very different from the other but demonstrates the flexibility of the region approach.

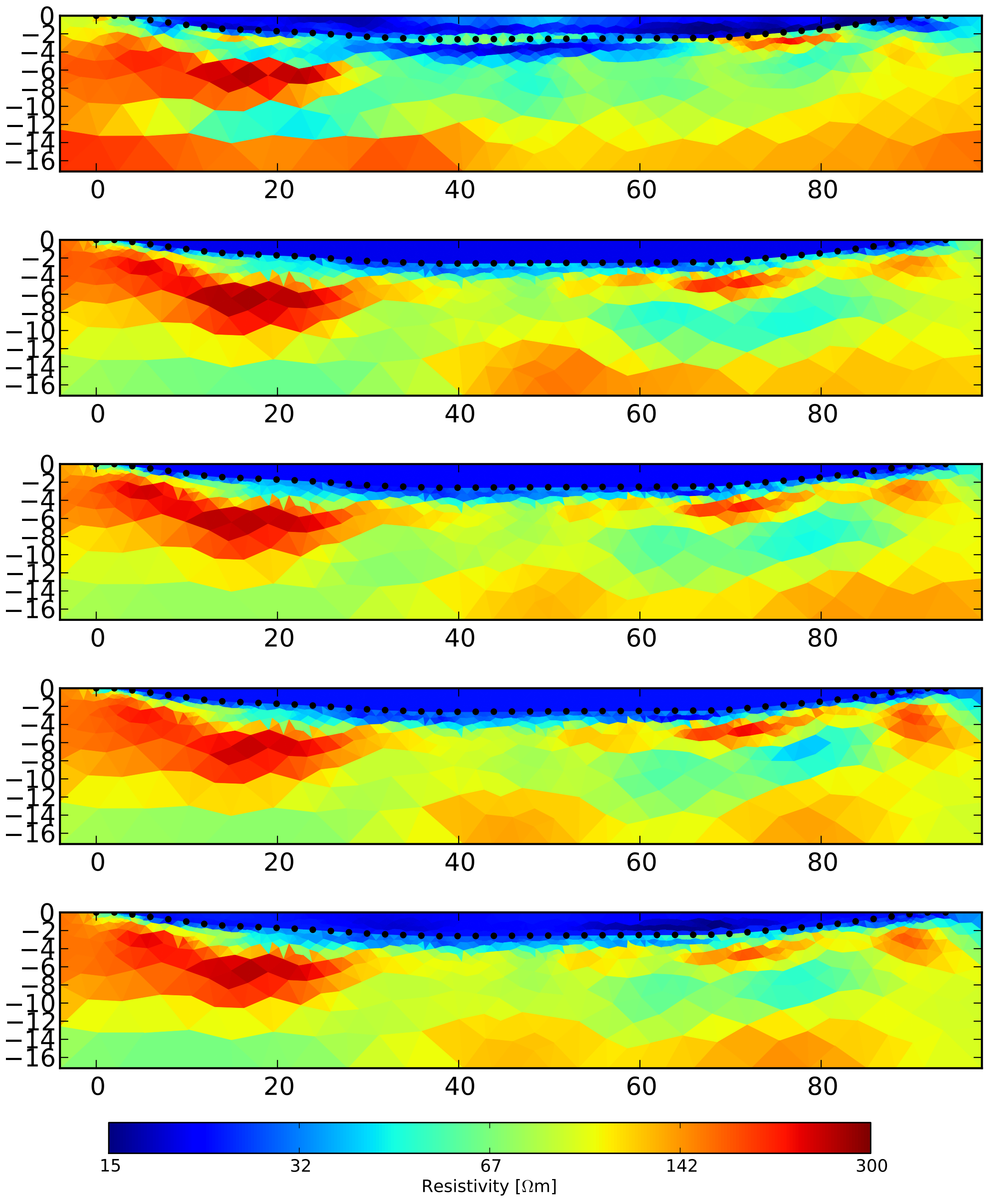


Figure 24: Inversion result with different options (top to bottom): a) smoothness-everywhere, b) lake as single-parameter region, c) lake as fixed region, d) inter-region constraints between lake and lake-bottom, e) like d but with variable water body.

6. Time-lapse ERT

We are often interested in ongoing physical processes and use ERT for monitoring experiments. An arbitrary number of subsequent data sets can be processed by writing their file names in a text file and passing it by **TIMESTEPS=filename**.

6.1. Strategies

There is a large number of different strategies for doing time-lapse inversion and accordingly a large number of associated keys.

1. The easiest is an independent inversion, which can be easily done using a simple shell script that just overwrites **DATAFILE**. However, very frequently artifacts arise, especially when looking at the changes (ratios).

2. Another variant is ratio or quotient inversion, i.e. calculating the data ratio and inverting them as apparent resistivity (Schütze et al., 2002). However, this approach is valid only for small contrasts since it does not take the sensitivity distribution as function of the real model into account. BERT1 used this approach by default, here it can still be used by **RATIOSTEP=1**.

3. There is the class of reference model based schemes, where for each frame a full minimization is done, but the models are constrained taking the model of either the first (default) or the preceding (**TIMELAPSESTEPMODEL=1**) frame as reference. It is the most general scheme since the used measuring arrays and even the electrode positions can vary over time. Therefore it is the default method.

4. A special scheme is the so-called difference inversion after LaBrecque and Yang (2001). It is based on the observation that there is a significant amount of systematic errors in all time steps. Therefore, if **TIMELAPSEREMOVEMISFIT=1**, the data d^k of k^{th} frame is corrected by the misfit of the first frame ($\mathbf{d^0}$) such that

$$\|\mathbf{d^k} - \mathbf{f}(\mathbf{m^k}) - \mathbf{d^0} + \mathbf{f}(\mathbf{m^0})\|$$

   is minimized next to the regularization of the model difference $\mathbf{m^k} - \mathbf{m^0}$ (alread in the last scheme). We recommend this in case of known systematic errors. If only systematic errors are present this will only increase error and thus smoothness. At any rate, identical arrays are used.

5. From an inversion point of view, the most rigorous method is a full discretization in time and simultaneous inversion. This is done efficiently using block matrices but up to now available only in the Python managers.

6.2. Processing and options

An issue of ongoing research are time-depending error models. Analysis of normal-reciprocal data can possibly overcome the problem of differing effective smoothness.
Generally, the arrays in the individual data files can be different. However, to use the difference inversion, they must be identical. To avoid stripping data by filter rules, we recommend using **DOWNWEIGHT=1**. Therefore the data are preprocessed into a data cube (number vs.

time). Non-existing data or measurements outside the filter rules stay in the file, but are down-weighted by a large error.
To define the time-lapse strategy, the above mentioned keys are used. By default, reference inversion against the first frame is done, except **TIMELAPSESTEPMODEL=1** determines regularization of time steps. La Brecque's method (different inversion) is used if **TIMELAPSEREMOVEMISFIT=1**.
Time-lapse inversion can take long computing times. If the changes are small, the main computational effort, the re-calculation of the Jacobian matrix can be omitted by using **FASTTIMELAPSE=1**, however only effective in case of identical arrays.
Since the model difference has generally different range and shape, its regularization can differ from the static inversion. The key **LAMBDATIMELAPSE** defines regularization strength, **TIMELAPSECONSTRAINT** the type of constraints (0, 1, 2, 10 or 20, see section 2.1). Whereas 0th order constraints (no smoothness) are often prohibitive for static inversion due to the large artifacts, it can be interesting for time-lapse problems, since smoothness does not systematically change the model space. More generally, a region file (see section 5.2) can be applied for the time step inversion specifically by **TIMELAPSEREGIONFILE**.
The results of the inversion are contained in the binary files `modelAbs.bmat` (absolute values) and `modelDiff.bmat` (relative differences in %). For 2D cases, they can be used as argument, along with a number specifying the time step, behind the `show/mkpdf` commands to display the model or its deviation, e.g.

```
$ bert cfg show modelAbs.bmat 5
```

shows the model for the 5th timestep,

```
$ bert cfg show modelDiff.bmat 5
```

shows the relative differences to the base model (default) or the preceding model (TIMELAPSESTEPMODEL=1). Note that in these cases **cMin/cMax** values MUST be given.
Omitting the time step number displays the baseline model (target `show`) or creates a multi-page pdf (target `mkpdf`). In both 2D and 3D, for each timestep i a file dcinv.result_i.vtk is created so that in Paraview they are automatically loaded as time steps if loading the group `dcinv.result..vtk`. The vtk files hold both absolute values and model ratios with respect to the baseline.

6.3. Crosshole timelapse measurements

Let's go back to the crosshole case 2dxh (see section 3.2) and unpack the time data files in 2dxh-timelapse.zip. In 2006 the BGS injected a highly saline tracer in borehole number 8 and measured 36 data sets every 40 minutes such that a whole day was covered. The subsequent files are named 01.dat, 02.dat, ... and are assembled in timesteps.txt.
By including **TIMESTEPS=timesteps.txt** and calling `bert bert.cfg calc` again. As a result we obtain several vtk files that are recognized by ParaView as time steps, allowing easy scrolling through the times.
Figure 25 shows some selected time steps that allow for seeing the tracer flow toward the left boundary. Note that these are only preliminary results that are used to present how BERT is working. With more sophisticated time lapse strategies the monitoring process can be traced more accurately.
Mostly, one is interested in the changes (ratios) and wants to test several algorithms, which is done for different settings in Fig. 26.

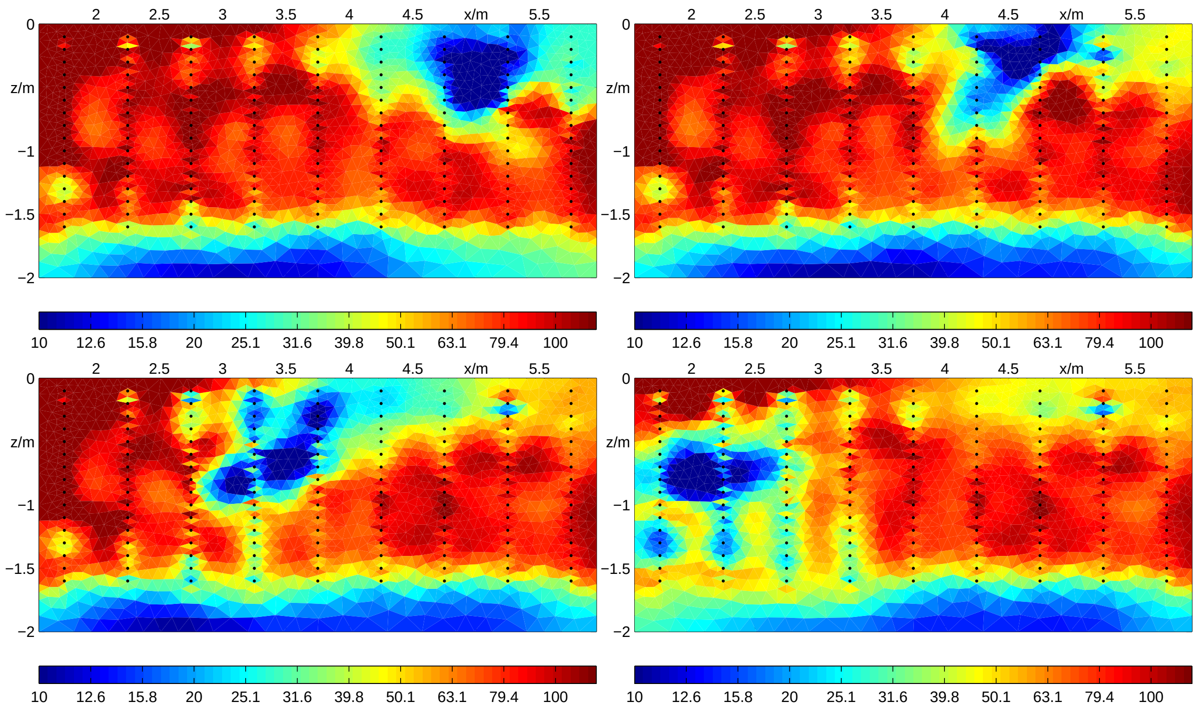


Figure 25: Inversion results 3 hours (upper left), 7 hours (upper right), 12 hours (lower left) and 16.5 hours (lower right) after tracer injection.

From the inversions with regularization orders 0, 1 and 2 (lines 3-5) the classical smoothness constraints (1st/2nd with slightly decreased vertical weights) exhibit the largest effects, but also the largest artifacts above and below the tracer. Minimum-length regularization of the model difference does not show such artifacts but increases at the electrodes interrupting the tracer shape. If (isotropic) smoothness and pure deviation are combined (last line), the least artifacts are observed, but the shape of the tracer remains interrupted. Further tuning may lead to even nicer images, however it is not clear beforehand which method is best and how reality looks like.

6.4. Soil column measurements

We go back to the soil column example from section 4.4. After irrigating a certain amount of water, every 2 hours a complete data set was measured and included in the **TIMESERIES** file. Since the changes are relatively low, we take a look at the relative differences in the files diff_i.vtk with respect to the initial resistivity. Figure 27 shows 5 selected time steps. We can see the water front intruding but at a certain stage the column is drying out again.

Acknowledgements

We like to thank all the guys that provided the very instructive data and user stories: Folker Donner (formerly TU Freiberg), Markus Furche and Ulla Noell (BGR Hannover), Thomas Schicht (K-UTec GmbH Sondershausen), Niels Hoffmann (formerly HAWK Göttingen), Oliver Kuras (British Geological Survey), Joseph Doetsch (ETH Zurich) and Ilaria Coscia (formerly

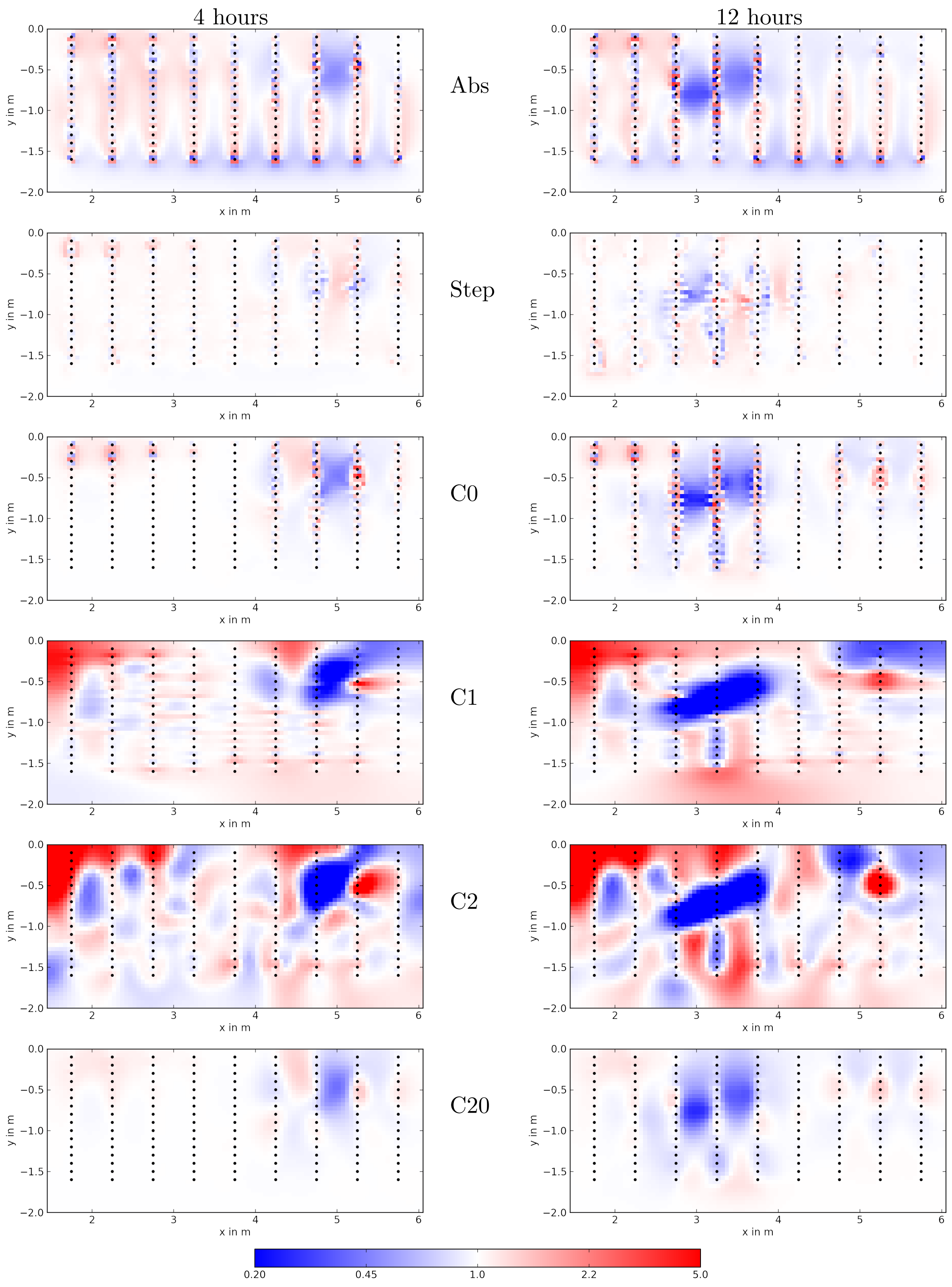


Figure 26: Resistivity ratio for timesteps 4 hours (left) and 12 hours (right) and inversion schemes: individual inversion, step-wise constrained and difference inversion using constraint orders 0, 1, 2 and 0+2

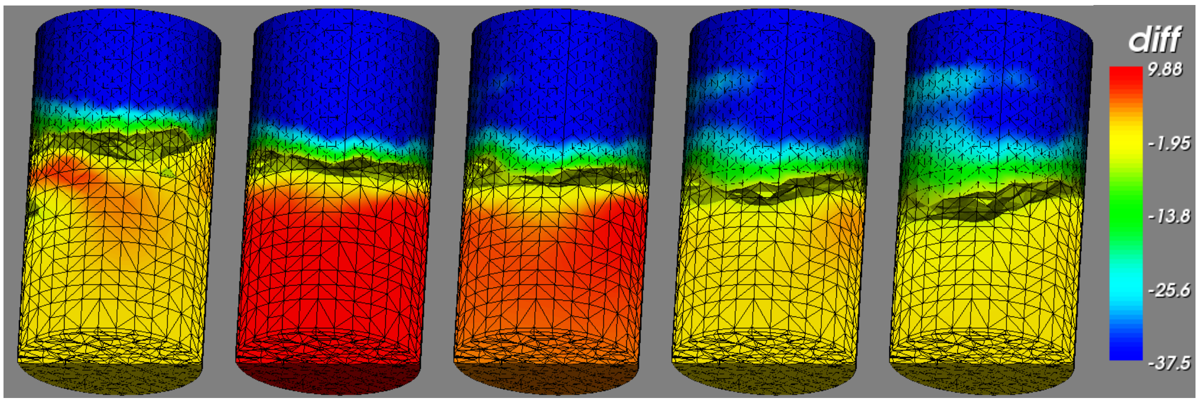


Figure 27: Relative resistivity difference (in %) for the repeated measurements at about 2, 4, 6, 10 and 16 hours after irrigation.

ETH Zurich), Sarah Garre (Uni Liege, Gembloux). Furthermore we acknowledge all the users and testers of BERT that made the software what it is now, a powerful expert tool.

References

Bazin, S. and Pfaffhuber, A. (2013). Mapping of quick clay by Electrical Resistivity Tomography under structural constraint. *Journal of Applied Geophysics*, 98:280–287.

Bechtold, M., Vanderborght, J., Weihermüller, L., Herbst, M., Günther, T., Ippisch, O., Kasteel, R., and Vereecken, H. (2012). Upward Transport in a Three-Dimensional Heterogeneous Laboratory Soil under Evaporation Conditions. *Vadose Zone Journal*, 11(2).

Claerbout, J. F. and Muir, F. (1973). Robust modeling with erratic data. *Geophysics*, 38(1):826–844.

Coscia, I., Greenhalgh, S., Linde, N., Doetsch, J., Marescot, L., Günther, T., and Green, A. (2011). 3D crosshole apparent resistivity static inversion and monitoring of a coupled river-aquifer system. *Geophysics*, 76(2):G49–59.

Doetsch, J., Coscia, I., Greenhalgh, S., Linde, N., Green, A., and Günther, T. (2010). The borehole-fluid effect in electrical resistivity imaging. *Geophysics*, 75(4):F107–F114. in print.

Doetsch, J., Linde, N., Pessognelli, M., Green, A. G., and Günther, T. (2012). Constraining 3-D electrical resistance tomography with GPR reflection data for improved aquifer characterization. *Journal of Applied Geophysics*, 78:68 – 76. ¡ce:title¿Developments in GPR and near-surface seismics - New applications and strategies for data integration, inversion, and modelling¡/ce:title¿.

Falcon-Suarez, I., Juncosa-Rivera, R., Vardon, P., Rammlmair, D., Günter, T., Noell, U., and Delgado-Martín, J. (2016). Hydrodynamic behaviour of compacted granite sawdust from the dimension stone industry of Pontevedra (Spain): experimental and modelling. *Environ Earth Sci*, 75(5).

Flechsig, C., Fabig, T., Rücker, C., and Schütze, C. (2010). Geoelectrical Investigations in the Cheb Basin/W-Bohemia: An Approach to Evaluate the Near-Surface conductivity structure. *Stud. Geophys. Geod*, 54:417–437.

Garre, S., Günther, T., Diels, J., and Vanderborght, J. (2012). Evaluating Experimental Design of ERT for Soil Moisture Monitoring in Contour Hedgerow Intercropping Systems. *Vadose Zone Journal*, 11(4).

Garre, S., Koestel, J., Günther, T., Javaux, M., Vanderborght, J., and Vereeken, H. (2010). Comparison of heterogeneous transport processes observed with ERT in two different soils. *Vadose Zone Journal*, 9(2):207–212.

Geuzaine, C. and Remacle, J.-F. (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Jounral for Numerical Methods in Engineering*, 79:1309–1331.

Günter, T., Martin, T., and Rücker, C. (2016). Spectral inversion of sip field data using pygimli/bert. In *4th International Workshop on Induced Polarization.*

Günther, T. (2002-2007). *DC2dInvRes - Direct Current 2d Inversion and Resolution.* resistivity.net productions, `http://dc2dinvres.resistivity.net`.

Günther, T. (2003-2008). *DC3dInvRes - Direct Current 3d Inversion and Resolution.* resistivity.net productions, `http://dc3dinvres.resistivity.net`.

Günther, T. (2004). *Inversion Methods and Resolution Analysis for the 2D/3D Reconstruction of Resistivity Structures from DC Measurements.* PhD thesis, University of Mining and Technology Freiberg. available at `http://fridolin.tu-freiberg.de`.

Günther, T. and Martin, T. (2016). Spectral two-dimensional inversion of frequency-domain induced polarisation data from a mining slag heap. *Journal of Applied Geophysics*, in press. accepted.

Günther, T. and Rücker, C. (2006). A general approach for introducing structural information - from constraints to joint inversion. In *Ext. Abstract, EAGE Near Surface Geophysics Workshop. 3.-6.9.06, Helsinki(Finland).*

Günther, T., Rücker, C., and Spitzer, K. (2006). 3-d modeling and inversion of DC resistivity data incorporating topography - Part II: Inversion. *Geophys. J. Int.*, 166(2):506–517.

Kuras, O., Pritchard, J., Meldrum, P. I., Chambers, J. E., Wilkinson, P. B., Ogilvy, R. D., and Wealthall, G. P. (2009). Monitoring hydraulic processes with Automated time-Lapse Electrical Resistivity Tomography (ALERT). *Compte Rendus Geosciences - Special issue on Hydrogeophysics*, 341(10-11):868–885.

LaBrecque, D. J. and Yang, X. (2001). Difference Inversion of ERT Data: a Fast Inversion Method for 3-D in Situ Monitoring. *J. Environ. Eng. Geophys.*, 6:83.

Martin, T. and Günther, T. (2013). Complex resistivity tomography (CRT) for fungus detection on standing oak trees. *European Journal of Forest Research*, 132(5):1–12.

Persson, M., Dahlin, T., and Günther, T. (2015). Observing Solute Transport in the Capillary Fringe Using Image Analysis and Electrical Resistivity Tomography in Laboratory Experiments. *Vadose Zone Journal*, 14(5):11p.

Ronczka, M., Rücker, C., and Günther, T. (2015). Numerical study of long-electrode electric resistivity tomography — Accuracy, sensitivity, and resolution. *Geophysics*, 80(6):E317–E328.

Rücker, C. (2010). *Advanced Electrical Resistivity Modelling and Inversion using Unstructured Discretization.* PhD thesis, University of Leipzig. in preparation.

Rücker, C. and Günther, T. (2011). The simulation of finite ERT electrodes using the complete electrode model. *Geophysics*, 76(4):F227–F238.

Rücker, C., Günther, T., and Spitzer, K. (2006). 3-d modeling and inversion of DC resistivity data incorporating topography - Part I: Modeling. *Geophys. J. Int.*, 166(2):495–505.

Schütze, C., Friedel, S., and Jacobs, F. (2002). Detection of three-dimensional transport processes in porous aquifers using geoelectrical quotient tomography. *Eur. J. of Env. and Engin. Geophysics*, 7:3–19.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Lin, M. C. and Manocha, D., editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag. From the First ACM Workshop on Applied Computational Geometry.

Si, H. (2002-2008). *TetGen - a quality-constrained tetrahedral mesh generator.* Weierstrass institute, Berlin, `http://www.tetgen.org`.

Si, H. (2015). TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Transactions on Mathematical Software*, 41(2):1–36.

Udphuay, S., Günther, T., Everett, M., Warden, R., and Briaud, J.-L. (2011). Three-dimensional resistivity tomography in extreme coastal terrain amidst dense cultural signals: application to cliff stability assessment at the historic D-Day site. *Geophys. J. Int.*, 185:201–220.

A. Installation

A.1. BERT for Windows users

BERT is successfully applied on Windows platforms, however, for bigger problems the performance on Linux is probably better. Although building has actually become as easy as under Linux, it is easily distributed as installer with binary files along with all examples and documentation.
Since BERT is controlled on the command line, Windows users need a command shell compatible with the Linux bash. Recently, the minimal system MSYS (minimal system) was brought to

new version 2 that includes 64bit support and a package manager (pacman). Please download the installer from `https://msys2.github.io`.
Install BERT, assuming under `d:\software\BERT`. Then this path must be known when working in the shell, either by changing the environment variable **PATH** under `System Control - System - Environment Variables` or in the shell by typing
`$ export PATH=$PATH:/d/software/BERT`[21]
The latter command can also be called automatically at startup by inserting it into the `$(HOME)/.bashrc` file in the home directory. Additionally, the variable **PYTHONPATH** is used to tell python where to look for modules (pygimli and pybert - do not include these names), therefore it should be set to `d:\software\BERT` in system control or using
`$ export PYTHONPATH=/d/software/BERT`.
Prepare your data and configuration file in a directory, go there in the shell by
`$ cd /c/data/profile1/trial` and run the inversion using `bert bert.cfg all` etc.
Note that if your Python distribution does not set the paths, you also need to do that, e.g. in the `.bashrc` file, by, e.g.
`$ export PATH=$PATH:/c/Software/Anaconda3`
On newer Anaconda versions, there might be some dll files missing so that you additionally need
`$ export PATH=$PATH:/c/Software/Anaconda3/Library/bin`
Of course you can also set up these paths, as the one for BERT, in the System Control.

A.2. BERT for Linux users

The easiest way to install BERT (and the necessary pyGIMLi) is using conda, a system that conveniently installs binary releases as well as their prerequisites and cares about the path variables to find the binaries and the Python libraries. It can be used from the Anaconda Python distribution or from its lightweight alternative miniconda. At any rate we recommend creating a separate environment into which it is installed. By doing so you can easily switch between several versions of BERT/pyGIMLi or Python, e.g. by:

```
$ conda config --add channels gimli --add channels conda-forge
$ conda create -n bert
$ source activate bert
$ conda install pybert
```

For updating, one can just call

```
$ conda update -f pygimli pybert
```

There might be reasons (incompatible system, version conflict with other modules, urgent update) when one needs to build pyGIMLi and the BERT binaries from the source. In this, case, first follow the installation procedure provided at `https://www.pygimli.org/installation.html` and the instructions given at `https://www.gitlab.com/resistivity-net/bert`. After building pyGIMLi and bert, one needs to add the paths for the binaries (bert/build/bin), the library (gimli/build/bin) and the Python modules, e.g. (best in the `.bashrc` file):

```
$ export PATH=$PATH:$HOME/src/bert/build/bin
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/src/gimli/build/bin
$ export $PYTHONPATH:$HOME/src/gimli/gimli/python:$HOME/src/bert/bert/python
```

[21]Note that file names are in different from Windows, i.e., `/c` instead of `c:` and slash instead of backslash.

B. Version history

BERT 2.4 (pyGIMLi 1.3)

2.4.1	19.01.2023	(pg 1.3.1) switch to Python 3.9, bug fixes
2.4.0	21.08.2022	(pg 1.3.0) improve TDIP/FDIP

BERT 2.3 (pyGIMLi 1.1/1.2, development limited to IP classes)

2.3.4	18.02.2022	(pg 1.2.6) avoid P2 mesh refinement in files, TDIP/HIRIP
2.3.3	12.12.2021	(pg 1.2.3) pyGIMLi update, TDIP/HIRIP, fix mesh bugs
2.3.2	01.08.2021	(pg 1.2.2) FDIP/TDIP updates
2.3.1	01.11.2020	(pg 1.1.1) update of FDIP/TDIP (Martin et al. 2020), Python 3.8
2.3.0	06.06.2020	(pg 1.1.0) upgrade to new pyGIMLi style, bug fixes

BERT 2.2 (pyGIMLi 1.0 with ERT core and ERTManager)

2.2.12	02.11.2019	(pg 1.0.12.1) mainly pyGIMLi fix, last pg 1.0
2.2.11	16.10.2019	(pg 1.0.12) improve TDIPdata class
2.2.10	01.04.2019	(pg 1.0.11) improve TDIPdata class
2.2.9	04.01.2019	(pg 1.0.10) first Python 3.7 version
2.2.8	30.10.2018	(pg 1.0.9) new TDIP class, minors and fixes
2.2.7	24.09.2018	(pg 1.0.7) convert and colors
2.2.6	13.02.2018	(pg 1.0.6 carneval edition): udf & res2dinv topo
2.2.5	12.01.2018	(pg 1.0.5) rather unknown version
2.2.4	17.11.2017	(pg 1.0.4 GELMON edition) minor stuff
2.2.3	15.09.2017	(pg 1.0.1) main tutorial update and bug fixes
2.2.2	21.08.2017	(pg 1.0.0) mainly pyGIMLi changes
2.2.1	21.07.2017	stable version available for Py 3.4, 3.5 and 3.6
2.2.0	16.06.2017	first version without C++ core library (all in GIMLi)

BERT 2.1 (end of libbert, increased Python use)

2.1.4	02.06.2017	last version with C++ BERT core library
2.1.3	03.03.2017	2nd Leipzig Workshop version
2.1.2	19.01.2017	first Python 3.5 build (from here default)
2.1.1	28.09.2016	stable bug-fixed post-workshop version (only Python 3.4)
2.1.0	21.09.2016	BERT Workshop Leipzig edition, new tutorial

BERT 2.0.x (originally 2.0RCx)

new core library libbert, establishing Python for visualization

2.0.19	17.08.2016	Günther&Martin (2016) paper defining FDIP
2.0.18	28.06.2016	do not use this version (a bug in 3D FEM)
2.0.17	24.05.2016	
2.0.16	14.04.2016	intermediate undertested version
2.0.15	24.03.2016	Easter Egg (rename from 2RCx to 2.0.x)
2.0.14	26.11.2015	GelMon edition

2.0.13	08.10.2015	(first working Resistivity Python class)
2.0.12	25.09.2015	from here only 64bit with WinPython3.4
2.0.11	14.09.2015	first 64bit version with Python3.4-64bit
2.0.11	14.08.2015	only 32bit version with Python3.4-32bit
2.0.10	09.07.2015	also Win64 with own Python64bit
2.0.9	18.05.2015	
2.0.8	21.03.2015	first FDIP (SIPData) class
2.0.7	16.12.2014	
2.0.6	17.10.2014	
2.0.5	03.06.2014	
2.0.4	02.12.2013	
2.0.3	08.08.2013	also Win64 with Python32bit
2.0.2	17.04.2013	

BERT 2 beta versions

2.0b23	06.08.2013	
2.0b22	05.04.2013	also Win64 23.01.2013
2.0b21	22.11.2012	also Win64
2.0b19/20	29./30.05.2012	
2.0b18	20.02.2012	
2.0b17	13.02.2012	also Win64
2.0b16	10.01.2012	only Win64
2.0b15	20.10.2011	also Win64 15.11.2011
2.0b14	17.08.2011	
2.0b13	06.07.2011	
2.0b12	08.06.2011	
2.0b11	13.04.2011	
2.0b10	07.02.2011	
2.0b8	02.11.2010	
2.0b4-6	5-13.08.2010	
2.0b3	28.07.2010	
2.0b1	21.04.2010	

BERT 1 (2004-2008-2013)

1.3.2	05.04.2013	final Win 32bit version
1.3.1	22.11.2013	also (final) Win64 version
1.3.0	30.05.2012	no more new features
1.2.5	30.08.2011	also first Win64 version
1.2.4	10.05.2011	Linux
1.2.3	21.04.2011	Linux
1.2.2	11.01.2011	
1.2.1	03.12.2010	
1.2.0	28.10.2010	Linux
1.1.0	25.06.2010	(also BERT2 beta versions starting)
1.1.1	27.10.2010	

`1.0.3    17.11.2009`

pre-versions 2006-2009

Main changes from version 1 to 2

Main improvements

- more flexible treatment of regions (different behaviour, constraints and transforms) , inter-region (de)coupling and therefore more possibilities to include prior knowledge
- more constraint types (zeroth, first and second order plus combinations of them) and a stabler way of anisotropic regularization
- more element types such as rectangular, hexahedral, or prism meshes or a combination of different elements
- use of different electrode types: nodes, surfaces (CEM electrodes) or node-free points (or a mix of different types)
- easy data filtering using different MIN/MAX keywords
- lower/upper resistivity bounds independent of the data (in BERT1 they were also applied to ρ_a)
- improved time-lapse strategies, e.g. full reference model or difference inversion
- improved direct visualization using pyGIMLi (Python bindings of GIMLi)
- improved speed (distributed computing thanks to BERTTHREADS) and convergence, easier build procedure
- improved phase inversion due to more rigorous IP scheme
- ERT manager for easy script-based handling

Changes in commands

primPot/interpolate $\Rightarrow$ **pot** - Formerly the primary potential was computed and interpolated in two steps. Now the potentials are interpolated onto the forward mesh on-the-fly unless otherwise stated by KEEPPRIMPOT=1.
correct $\Rightarrow$ **filter** - As the command correct (called before calc) was leading to confusions, the command filter now transforms the raw data into the desired inversion input including apparent resistivity, geometric factor and error. Additionally filtering is done using MIN/MAX keywords. For timelapse problems filter will homogenize the data array for the individual time steps. In future versions an automated normal reciprocal analysis can be called if the data allow for that.
calc/calcSensM $\Rightarrow$ **calc** - The calcSensM command is not needed any more and fully replaced by calc. Recalculation of the Jacobian is the default.

New commands

sensOnly - calculate only the Jacobian matrix (for sensitivity or resolution studies)
version - prints out the version
show, showmesh, showdata, showerror, showfit, mkpdf - plots of different stuff (in case of 2d geometry)
mkpdf, mkmeshpdf - automated pdf generation

C. Files and programs

Created files and their meaning

File types:

*.poly	triangle (2d) or tetgen (3d) PLC format
*.bms	binary mesh (house) format
*.vtk	visual toolkit mesh or poly format (paraview)
*.mesh	MEdit mesh format
*.vector	ascii vector of floats
*.collect	potential matrix of all electrodes

Directories and their content:

mesh	mesh input (mesh.poly) and meshes (meshPara,meshSec,meshPrim, meshParaDoma
primaryPot/pot(_s.bmat)	primary potentials (already on secondary mesh)
sens.bmat	sensitivity columns (smatrix.*) or rows smatrixCol.*
result*	saved result directory with most important files

Files in project or result directory:

command.history	history of commands executed by bert
bert.log	inversion log file
*.data	filtered data file with apparent resistivities and errors
dcinv.result.vtk	final result as vtk file
resistivity.vector	final (projected) resistivity distribution
response.vector	final model forward response
coverage.vector	coverage (sum of absolute sensitivies)
model_*.vector	model vectors of individual iterations
mesh/mesh.poly	mesh input PLC
mesh/meshParaDomain	pure parameter mesh (use for visualization)
in case of debug mode (SAVEALOT=1)	
response_*.vector	response vectors for each iteration
fop-model*.vtk	forward models
linesearchPhi(D).vector	linesearch vectors
constraint.matrix	constraint matrix (sparse matrix with I, J, value)

Program calls used for BERT

Inversion and parameterization:
bert - main executable running all based on cfg file

bertNew2D/2DTopo/2DCirc - CFG file generators for 2d cases (flat,topo,circle)
bertNew3D/3DTopo/3DCyl - CFG file generators for 3d cases (flat,topo,tank)
dcinv - actual inversion routine (see dcinv -h)
dcmod - forward modelling for synthetic (e.g. primary) potentials (see dcmod -h)
dcedit - filter raw data using numerical geometric factors
Mesh creation and alteration:

paradepth	estimate appropriate model depth by 1D sensitivities
createParaMesh	create parameter mesh from dat file
createSecondaryMesh	create secondary mesh out of parameters
createSurface	create 3d surface mesh from xyz point list
closeSurface	close 3d surface mesh by surrounding boxes
prepareMeshRefinement	insert refinemeht points
dctriangle	triangle call
meshconvert	convert mesh between various import formats

Poly tools - creating PLC objects:

polyCreateWorld	makes a world with 2 surface and interior boundary
polyFlatWorld	create world (2 regions) around electrodes=nodes
polyFreeWorld	create world (2 regions) around electrodes (no nodes)
polyCreateCube	create (unit) cube around origin
polyTranslate	translate PLC
polyScale	scale PLC
polyRotate	rotate PLC
polyMerge	merge 2 PLCs into a new one
polyAddVIP	add points (e.g. electrodes) to PLC
polyAddProfile	add profile of electrodes
polyRefineVIPS	refine points by local refinement
polyConvert	convert PLC to VTK or STL format
polyScripts.sh	various PLC functions

D. Complete list of options and their default values

```
#
# Global settings
#
VERSION=2.0.19
DATAFILE=notexisting.dat   # data filename (required)
DIMENSION=3       # dimension of the problem (2 for 2d and 3 for 3d)
TOPOGRAPHY=0      # defines if topography is present (0 or 1)
TOPOPOINTS=       # file with additional coordinates for 3d topography (x y z)
TOPOPOLY=         # file with additional polygons (sep. by blank line) for 3d topo
CYLINDER=         # geometry is closed
TIMESTEPS=        # file with the names of additional datafiles for timelapse inversion
PARAGEOMETRY=     # script (python or bash) creating geometry input (mesh/mesh.poly) or file
PARAMESH=         # mesh or command creating the inversion mesh (mesh/mesh.bms)
INTERFACE=        # file with x/z columns describing (2d) interface(s) (separated by blank line)
ELECTRODENODES=1 # electrodes are represented by nodes (not yet interpreted!)
USEBERT1=0        # uses previously generated BERT 1 meshes, potentials and sensitivity (BERT2)
KEEPPRIMPOT=0     # keep primary potentials on primary mesh, otherwise interpolate on-the-fly (BERT2)
DEBUG=0           # prints even more detailed information and save important temporary files
SAVEALOT=0        # save a lot of files for debugging purposes

#
# Data settings
#
OVERRIDEERROR=0 # overrides given errors with INPUTERRLEVEL and INPUTERRVOLTAGE
INPUTERRLEVEL=3 # input error level (in percent) if no error given (dcedit -p but still -e)
```

```
INPUTERRVOLTAGE=100e-6    # input voltage error (V) if no error given
KMIN=-9e99        # minimum geometric factor (BERT2)
KMAX=9e99         # maximum geometric factor (BERT2)
RMIN=0            # minimum apparent resistivity (BERT2)
RMAX=9e99         # maximum apparent resistivity (BERT2)
IPMIN=-9e99       # minimum IP value (BERT2)
IPMAX=9e99        # maximum IP value (BERT2)
ERRMAX=9e99       # maximum error estimate (BERT2)
DOWNWEIGHT=0      # downweight instead of deleting data to keep data structure (BERT2)
ABSRHOA=0         # Force all apparent resistivities to be positive

#
# Inversion settings
#
CONSTRAINT=1       # constraint type (1/2-first/second order, 0-min.length, 10/20 mix)
LAMBDA=20          # regularization strength
ZWEIGHT=1          # weight for purely vertical gradients (BERT2)
LAMBDAOPT=0        # optimze lambda by using l-curve
CHI1OPT=0          # optimize lambda such that CHI^2=1 (BERT2)
REGIONFILE=        # region file for inversion options for each region (BERT2)
BLOCKYMODEL=0      # iteratively (L1) reweighted model roughness
ROBUSTDATA=0       # iteratively (L1) reweighted data misfit
LOWERBOUND=0.0     # lower resistivity bound (logarithmic barrier)
UPPERBOUND=0.0     # upper resistivity bound (0.0 = deactivated)
RECALCJACOBIAN=1 # recalculate jacobian each iteration step (default in BERT2)
MAXITER=20         # maximum number of iteration steps
STARTMODEL=        # starting model, either a value (homogeneous model) or a filename holding a vector
RESOLUTIONFILE=  # file with indices/positions to compute resolution kernels for (BERT2)
LAMBDADECREASE=1 # decrease lambda in each iteration by factor (e.g. 0.8)
LOCALREGULARIZATION=0 # local regularization (constrain only model update instead of model)
#SINGVALUE=-1       # potential value at electrodes, for sensitivity (BERT1)
SENSMATDROPTOL=0    # drop tolerance for Jacobian values (sparse storage)
SENSMATMAXMEM=2000 # available memory in MB for pre-allocation of sparse Jacobian
#
# Time lapse inversion settings
#
LAMBDATIMELAPSE=         # separate lambda value for timesteps (BERT2)
TIMELAPSECONSTRAINT=1 # constraint type (see CONSTRAINT)
TIMELAPSEREGIONFILE=  # region file (see REGIONFILE)
TIMELAPSESTEPMODEL=0  # use preceding model as reference instead of first
FASTTIMELAPSE=0          # no jacobian recalculation for timesteps (only identical arrays) (BERT2)
RATIOSTEP=0              # very simple (low-contrast) and fast method
TIMELAPSEREMOVEMISFIT=0 # difference inversion after LaBreque et al. (1996) (BERT2)

#
# Mesh settings
#
PARAMAXCELLSIZE=0       # maximum cell size volume (m3) (DIMENSION=3); area (m2) (DIMENSION=2) for para mesh
PARAEDGELENGTH=0        # compute PARAMAXCELLSIZE by volume of regular triangle/tetrahedron (easier to control)
PRIMMAXCELLSIZE=0       # maximum cell size volume (m3) (DIMENSION=3); area (m2) (DIMENSION=2) for prim mesh
PARADEPTH=0             # maximum depth of parameter domain in meter (0 = automatic estimation by 1d coverage)
PARABOUNDARY=5          # boundary around electrodes in parameter domain (percent)
SPLINEBOUNDARY=0        # spline circle boundary instead of piecewise linear interpolation
EQUIDISTBOUNDARY=1      # equidistant refined space between electrodes (2d) (default in BERT2)
BOUNDARY=500            # size of boundary area around parameter domain
MESHGEN=tetgen          # command or location 3d mesh generator
TETGENTOLERANCE=1e-12     # tetgen tolerance limit
TETGENPRESERVEBOUNDARY=0 # tetgen should suppress splitting of boundary facets or segments
LOOPTETGEN=0            # use tetgen itself to slightly improve mesh quality due to repeating tetgen calls
PARADX=0.0              # refinement for para mesh (values >0.5 will be forced to 0.5)
GRID=0                  # generate regular (quadriliteral or hexahedral) mesh (only TOPOGRAPHY=0, BERT2)
LAYERS=10               # number of layers to use for grid (for GRID=1 only)
PARADZ=1                # layer thickness to use for grid (for GRID=1 only)
PRIMDX=0.1              # electrode refinement (prim mesh) (note: relative in 2d, absolute in 3d)
PRIMDX_R=0.0            # electrode refinement (prim mesh) dr in center direction (overides PRIMDX)
PARA2DQUALITY=33.0      # parameter grid (from 25 (very bad) to 35 (good))
PRIM2DQUALITY=33.4      # primary grid (from 25 (very bad) to 35 (good))
PARA3DQUALITY=1.5       # parameter grid (from 1.11 (good) to 2 (bad))
PRIM3DQUALITY=1.2       # primary grid (from 1.11 (good) to 2 (bad))
SURFACEQUALITY=30       # quality of topographical surface grid (from 20 (bad) to 35 (good))
SURFACEMAXTRISIZE=0.0 # maximal triangle area of paramatric surface grid
SURFACESMOOTH=0         # improve quality of topographical surface grid
ICDROPTOL=0.0           # if number of nodes  200k drop tolerance is set for ICCG solver (only BERT1)
TOTALPOTENTIAL=0        # don't use secondary potential approach (i.e. for CEM meshes)
OMITSECREFINE=0         # omit global refinement for forward calculation (for fine parameter meshes)
#SECMESHREFINE=1        # the opposite (BERT1)
SECP2MESH=0             # use quadratic shape functions in forward mesh (BERT2)
PRIMP2MESH=1            # use primary p2 mesh (default in BERT2)

#
# Directory settings
#
MESHBASENAME=mesh         # basename for mesh files
DIRMESHS=mesh             # directory name for mesh files
DIRPOT=primaryPot         # directory name for primary and interpolated potentials
DIRPRIMPOT=potentials   # subdirectory name for primary potentials (obsolete)
DIRINTERPOLPOT=interpolated # subdirectory name for interpolated potentials (obsolete)
DIRSENS=sensM             # directory name for sensitivity matrix
OLDPRIMMESHSTYLE=0        # alternative way to create primary mesh (for internal use only)
```

```
#
# Data filtering options
#
KMIN=-9e99              # minimum geometric factor
KMAX=9e99               # maximum geometric factor
RMIN=0                  # minimum apparent resistivity
RMAX=9e99               # maximum apparent resistivity
IPMIN=-9e99             # minimum apparent phase
IPMAX=9e99              # maximum apparent phase
ERRMAX=9e99             # maximum error estimate"

#
# Plotting settings (2d plots with pytripatch)
#
PYTHON=python           # command or location of python executable
PYTRIPATCH=pytripatch   # command and location of pytripatch (2d mesh plotting)
USECOVERAGE=1           # use coverage for alpha shading
SHOWELECTRODES=1        # show electrode positions as black dots
INTERPERC=3             # use 3% interpercentiles for model display with pytripatch
cMin=                  # minimum for colour scale
cMax=                  # minimum for colour scale
```

Permanently active options can be saved in a file `.bertrc` in the home directory (user-specific) or in the folder where `bert` is located (for all users). This is particulaly interesting for **SENSMATMAXMEM** (recommend 50-80% of the available RAM in MB), the number of CPUs to use in parallel (BERTTHREADS), or the Python executable (PYTHON).

E. User stories/HowTo's

Sometimes users have very specific tasks requirements, mostly geometries, that are to be incorporated. We present solutions, usually by some Python or bash scripts, that go beyond the usual documentation, the corresponding input files are located in sub-folders of `doc/HowTo`. Some of them are actually workarounds that are to be integrated in the following releases of BERT 2 via new keywords but can still show users how to create solutions generally. Others are user stories that demand quite complicated approaches that will never be integrated into BERT directly. We hope that the existing stories tell you how to create solutions by combination of the existing ones. If you have an interesting case already solved or that cannot be solved easily but could be of interest for many users, don't hesitate to contact us to integrate it.
Note that the solutions mainly deal with creating appropriate meshes and are written as shell scripts, python scripts or a combination of both. Hence a basic knowledge in programming is required to understand it. Here we present the main components used part by part, for the full scripts see the files in the indicated directories. Usually we skip some general parts of it such as the importing of libraries, usually

```python
import pygimli as pg
import numpy as np
import matplotlib.pyplot as plt
```

For a complete list of the available functions, see the documentation of pyGIMLi (`http://www.pygimli.org`), Numpy/SciPy (`http://www.scipy.org`) or Matplotlib. Main data type to be worked with is pg.Vector, which is compatible with np.array.

E.1. How to do 2d inversion with external topographic information

(To be included into BERT using INPUTFILE or with a python createParaMesh)
File: `doc/HowTo/2d_topofile`
Task: use unidata format file with topography at end for inversion
Problem: the BERT 1 mesh tools does not support this yet
Solution: data have to be split and transformed to standard format.
TODO: generate a pygimli-only solution

Before, the usual recommendation was to use DC2dInvRes and its option Save-Ohm-File, which does the tape correction. Now we solve the problem in a shell script and invoke Python directly. First we extract the number of electrodes and data and write the first parts (electrodes+data) of the file such that be read by pygimli. Moreover we read the topography list and export it to another file:

```
infile='profile-topo.dat'
datfile='profile.dat'
outfile='profile.ohm'
topofile='topo.xz'
read line < $infile
nel=${line%#*}
line=`head -n $[nel + 3] $infile|tail -n 1`
ndata=${line%#*}
head -n $[nel + $ndata + 4] $infile > $datfile
line=`head -n $[nel + $ndata + 5] $infile|tail -n 1`
ntopo=${line%#*}
echo $nel $ndata $ntopo
tail -n $ntopo $infile > $topofile
```

We now switch to python and read in the file and extract electrode positions. We compute the tape distance along the profile and use the topography to interpolate it onto the electrode positions.

```
data = pg.DataContainer('$datfile')
print(data)
x, z = np.loadtxt('$topofile', unpack=True)
tt = np.hstack((0.,np.cumsum(np.sqrt(np.diff(x)**2+np.diff(z)**2))))
xe = [el.pos()[0] for el in data.electrodes()]
z = np.interp(xe, x, z)
```

However, usually the electrode positions have tape distances and not real x coordinates. Therefore we have to account for their heights such that the tape distance remains constant. After it we set

```
x = xe[0]+np.hstack((0, np.cumsum(np.sqrt(np.diff(xe)**2-np.diff(z)**2))))
de = np.sqrt(np.diff(x)**2+np.diff(z)**2)
for i in range(data.sensorCount()):
    data.setSensorPosition(i,pg.RVector3(x[i], 0.0, z[i]))

data.save('$outfile', 'a b m n u i R', 'x z')
```

That's it! With the created file we can do a very classical inversion.

E.2. 2D inversion with given 3d coordinates

(To be included into standard bert automatically or via DIMENSION=2.5?)
Files: `doc/Howto/3d_coordinates`: `mkcoord.py`, `mk3dvtk.py`
Task: do 2d inversion of profile with 3d coordinates and create appropriate vtk file
Problem: fit an appropriate line through inexact positions and transform result back to 3d

1. Read data file and fit a profile line through electrodes

Input could be:
1a) 2d data file with 3d coordinates (x,y,z) for each electrode
1b) 2d flat earth data file and additional GPS positions file
I suppose the latter, more general, case with GPS positions for some (not necessarily all) electrodes in a separate file gps.txt (profile distance, height, northing and easting).

```
tt, hh, yy, xx = np.loadtxt('gps.txt', unpack=True)
# tape distance, height, northing and easting
```

After reading and plotting the coordinates we see typical errors of up to a few meters due to GPS inaccuracies, which we want to eliminate. Therefore we fit a smooth curve by using harmonic functions implemented in the pygimli function harmfit. From the resulting x and y we compute the distance t along the tape, which is used for inversion.

```
x = harmfit(xx, tt, nCoefficients=10)[0]
y = harmfit(yy, tt, nCoefficients=10)[0]
z = harmfit(hh, tt, nCoefficients=15)[0]
t = np.hstack((0.,np.cumsum(np.sqrt(np.diff(x)**2+np.diff(y)**2)))) # 2d distance
```

The number of coefficients nCoefficients defines the complexity of the fitted curve and should be large enough to describe the course, but not too large to avoid small-scaled oscillation. Experience with different numbers with easily give good values for any data.

```
plt.plot(tt, hh, 'bx-', tt, z, 'r-')
plt.plot(xx, yy, 'bx-', x, y, 'b-')
```

Figure 28 shows how the coordinates are approximated.

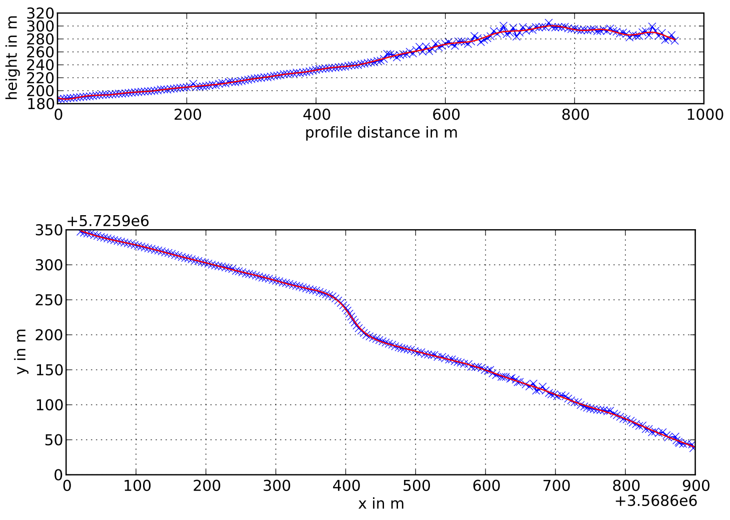


Figure 28: Measured positions (blue) and approximated profile (red) along the tape (top) and in the x-y plane

2a. project positions onto profile

We now load the data container and interpolate the (tape) coordinates xe onto the 2d positions t and z. We then run through all electrodes and change their 3d position to the 2d position before saving the data to a 2d file.

```
data = pg.DataContainer('profile.dat')
xe   = [pos[0] for pos in data.sensorPositions()]
x2d = np.interp(xe, tt, t)
z2d = np.interp(xe, tt, z)
for i in range(data.sensorCount()):
    data.setSensorPosition(i, pg.RVector3(x2d[i], 0.0, z2d[i]))
data.save('profile2d.ohm', 'a b m n u i', 'x z')
```

Furthermore we create a map from the 2d to the 3d coordinates for result projection.

```
POS = np.vstack((x2d,x,y))
np.savetxt('pos.map', POS.T)
```

2b. calculate 2d and 3d geometric factors and correct u or R

We could actually take into account that the profile is not strictly straight. ($G_{2d} > G_{3d} \Rightarrow u_{2d} < u_{3d}$ for const $\rho_a \Rightarrow$ correct $u_{2d} = u_{3d} * G_{3d}/G_{2d}$) Create standard meshes, refine and run dcmod with them.
NOT DONE YET

3. carry out 2d inversion and change positions in vtk file

A 2d inversion of the 2d data set is carried out using standard tools:

```
bertNew2DTopo profile2d.ohm > bert.cfg
# add some other useful options as value bounds or whatever
bert bert.cfg all
```

4. Projection of the result back to 3d domain

As a result we have a vtk file with 2d coordinates that need to be brought to 3d. We read the mesh and save the node positions in the variable tn and zn:

```
mesh = pg.Mesh('dcinv.result.vtk')
tn = [n.pos()[0] for n in mesh.nodes()]
zn = [n.pos()[1] for n in mesh.nodes()]
```

Then we load the previously saved map and interpolate the tn to real x and y:

```
tt, xx, yy = np.loadtxt('pos.map', unpack=True)
xn = np.interp(tn, tt,xx)
yn = np.interp(tn, tt,yy)
```

Finally we set the positions of the individual nodes and export the file.

```
for i, n in enumerate(mesh.nodes()):
    n.setPos(pg.RVector3(xn[i],yn[i],zn[i]))

mesh.exportVTK('result3d.vtk')
```

The result is then displayed in Figure 29. Doing so for several profiles they can easily displayed together in Paraview or also exported to other programs such as GoCad or any GIS software.

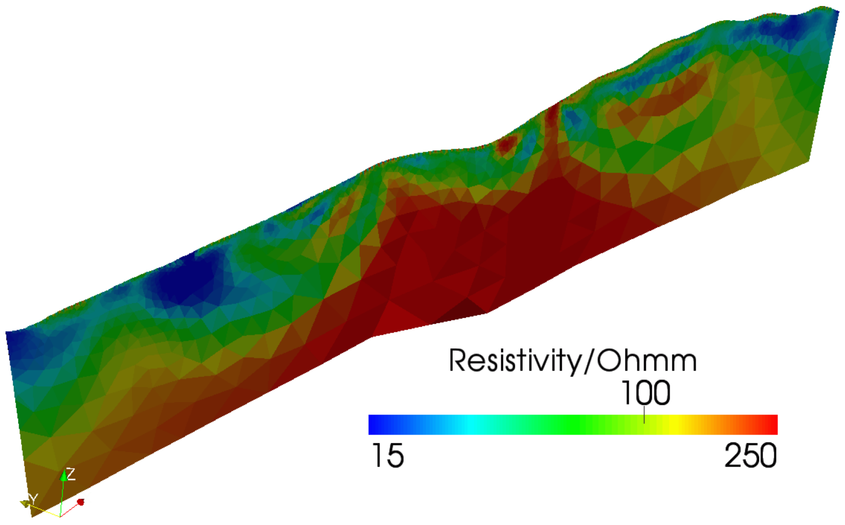


Figure 29: Inversion result displayed in 3D.

E.3. User-defined regular 2d mesh

(to be included into BERT using any DX keyword or a script like polyRectWorld)
`doc/howto/regular_mesh2d`
Task: inversion on regular rectangular meshes
Problem: necessary boundary should consist of triangles in order to keep node numbers small
Solution: create regular 2d mesh for the parameters and add a triangle boundary box for accurate solution due to boundary conditions.
As in the next example, a grid is created and its boundary is integrated into a bigger box, which is merged and combined with the regular mesh. In 2D this task is very easy and implemented in the pygimli function appendTriangleBoundary. The following python script should be self-readable:

```python
from pygimli.viewer import showMesh
from pygimli.meshtools import appendTriangleBoundary

data = pg.DataContainer('gallery.dat')
xmin = data.sensorPosition(0)[0]
xmax = data.sensorPosition( data.sensorCount() - 1 )[0]
dx   = ( data.sensorPosition(1)[0] - xmin ) / 2.
zmax = 10

nb = 2
x = pg.asvector(np.arange(xmin-dx*nb,xmax+dx*(nb+1), dx))
z = pg.asvector(np.arange(- np.ceil( zmax / dx ), 1. ) * dx)

mesh = pg.Mesh(2)  # new 2d mesh
mesh.create2DGrid( x, z )
for c in mesh.cells():
        c.setMarker(2)  # set all markers to 2

mesh2 = appendTriangleBoundary(mesh, 50., 50.) # the main thing!
showMesh(mesh2)
mesh2.save('mesh.bms')
```

We now include the mesh by specifying `PARAMESH=mesh.bms`. Figure 30 shows the created mixed element mesh and the inversion result.

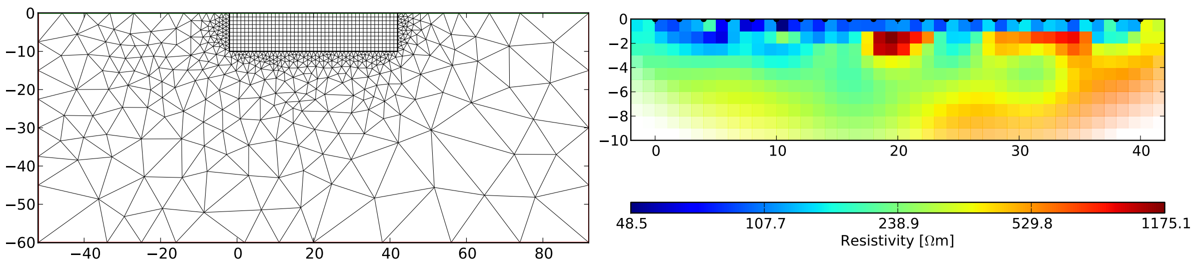


Figure 30: Created mixed mesh (left) and inversion result using regular meshes

Regular meshes may be advantageous over unstructured meshes if either (i) a predominant layering should be better imaged or (ii) if cells with constant size are required, e.g. for crosshole measurements.

E.4. How to invert 1d resistivity in a 2D domain

(could be option after integrating regular 2d meshes)
`doc/Howto/1d_vertical`
Task: inversion of resistivity data from a vertical electrode chain
Problem: Gimli 1D forward operator does not handle subsurface sources
Solution: use 2D (BERT) forward operator and special quasi-1D mesh
The essential part is to create an appropriate mesh that requires our demands. We chose a regular mesh, which can be easily created using the pygimli functions create2DMesh or create3DMesh and set the markers such that all rectangles obtain an identical marker and can later be treated as one single region. We could distribute the markers by hand but the above functions already contain typical solutions: The markerType (0, 1, 2 or 12) command specifies, which index (x and y) is increasing the marker with the axis. By default, all cells will have the same marker. We use the index 2 to use constant y markers:

```
data=pg.DataContainer('vest-ost.dat')
z1 = data.sensorPosition(0)()[2] # z of first electrode
z2 = data.sensorPosition(1)()[2] # z of second electrode
zN = data.sensorPosition( data.sensorCount() - 1 )[2]
dz = z2 - z1   # regular spacing with electrode distance
nb = 2         # number of boundary elements
nx2= 20        # half number of x cells
# position vectors, regular spacing
xx = np.arange( -nx2, nx2 ) * dz
zz = np.arange( z1 - nb * dz, zN + nb * dz, dz)
mesh = pg.createMesh2D(xx, zz, 2)
```

Note that the similar exists for create3DMesh. For appropriate outer space we insert this mesh into a big box and fill the intermediate space by triangles using appendTriangleBoundary:

```
mesh2 = appendTriangleBoundary(mesh, 70., 70., marker=0, isSubSurface=True)
print(mesh, mesh2)
marker2 = mesh2.cellMarker()
showMesh(mesh2, data=marker2, linear=True)
mesh2.save('mesh/mesh.bms')
```

The python script applied using PARAMESH=mkmesh.py. Additionally we need to specify a region file to treat each layer as a constant single region and the outside as background. Between the layers smoothness constraints are defined using the inter-region constraints:

```
#No single trans lBound uBound
*   1      log   1      300
#No background
0   1
#Inter-region
*   *   1
```

Figure shows the inversion result derived from a data set using vertical buried electrodes in the freshwater-saltwater interface below the north sea island Borkum. Since the data fit is at about 1%, we presume that ehe lithology is just a rough image of fine layerings. Below the last clay layer, the salt-water with resistivities below 1 Ωm starts.

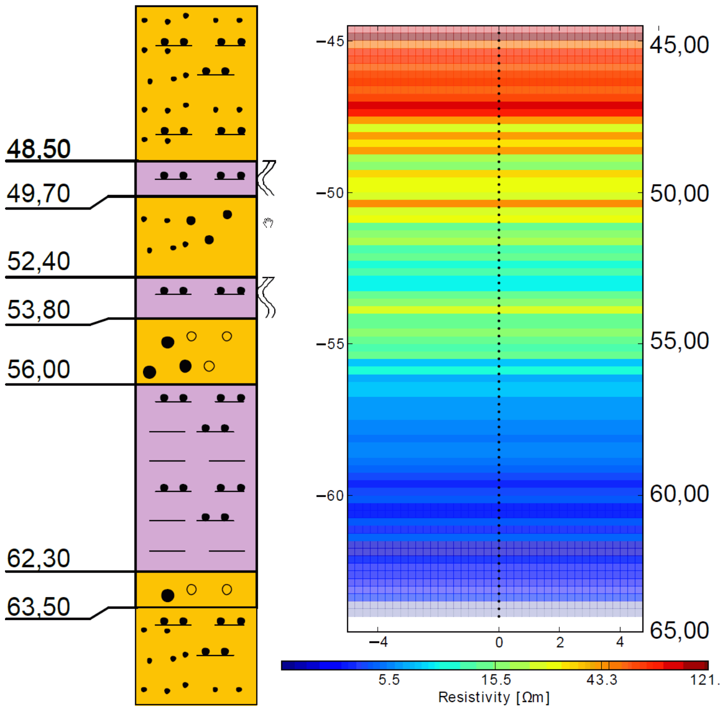


Figure 31: Inversion result (right) with lithology (left)

E.5. Inversion on user-defined regular 3d mesh

(not really stable due to tetgen face creation)
`doc/howto/regular_mesh3d`
Task: Use a regular 3D (FD-like) discretization for inversion with BERT
Problem: Create mixed mesh of tetrahedra (inversion part) and hexahedra (outer box)
BERT 2 supports hexahedral finite element computation both naturally or by dissecting each hexahedron into 5 or 6 tetrahedra. Whereas the first variant is restricted to orthogonal hexahedra, the latter could be used for a deformed regular mesh, e.g. with surface topography. For inversion in unbounded domains, an outer box around the inversion domain is needed to ensure the correctness of boundary conditions and thus accuracy. If the meshes are prolongated regularly, we obtain very ugly cells with bad numerical behaviour and have a huge number of nodes in the forward computation. Therefore we want to prolongate the mesh with tetrahedra.
A regular mesh can be created in pygimli using built-in functions. First, vectors are created (specify min/max/dx) and used to create a mesh:

```
x = np.arange(xmin-dx*nb,xmax+dx*(nb+1), dx)
y = np.arange(ymin-dx*nb,ymax+dx*(nb+1), dx)
z = np.arange(-np.ceil( zmax / dx), 1.) * dx
mesh = pg.Mesh(3)
mesh.create3DGrid(x, y, z)
```

A second mesh is created by H2 refinement of the hexahedra into tetrahedra: The outer mesh boundaries, which only have one neighboring cell, are set to marker 1 to be identified later.

```
mesh2 = pg.Mesh(3) # new 3d mesh
mesh2.createH2Mesh(mesh) # refined
print mesh, mesh2   # display node/cell/boundary numbers
for b in mesh2.boundaries():
    if not (b.leftCell() and b.rightCell()):
        b.setMarker(1)

mesh.save('para')
mesh.exportVTK('para')
```

Finally we extract the outer boundaries of the mesh and save it as poly file:

```
poly = pg.Mesh(3)
poly.createMeshByBoundaries(mesh2, mesh2.findBoundaryByMarker(1))
poly.exportAsTetgenPolyFile('paraBoundary')
```

We now create a world (big box with FE modelling boundary markers), merge it with the parameter outer boundary and add a hole marker in the middle (in bash):

```
polyCreateWorld -x100 -y100 -z50 world    # make big box
polyMerge world paraBoundary worldSurface # can take a while
polyAddVIP -x 0 -y 0 -z -0.1 worldSurface # hole marker in the middle
```

The PLC has now the faces of both and can be meshed with moderate quality

```
tetgen -pazVACq2 worldSurface  # result will be worldSurface.1.*
meshconvert -vBDM -it -o worldBoundary worldSurface.1 # convert
```

We now extract the outer surface of the resulting mesh by their -2 markers

```
worldBoundary = pg.Mesh('worldBoundary.bms')
worldPoly = pg.Mesh(3)
worldPoly.createMeshByBoundaries(worldBoundary,
    worldBoundary.findBoundaryByMarker(-2, 0))
worldPoly.exportAsTetgenPolyFile('worldBoundary.poly')
```

and obtain a triangulated surface mesh of the outer box without the inner. Therefore we have to merge both surface meshes

`$ polyMerge worldBoundary paraBoundary allBoundary`

Note that the latter procedure can take really long since intersections between all face pairs have to be checked. We can avoid this by adding the option -N to polyMerge, because we know there are no intersections. As a side effect the nodes at the inner box boundary are doubled and have to be removed, which we can establish by the script `readmypolyfile.py`:

```
Poly=pg.Mesh(3)
f=open('allBoundary.poly','r')
line1=f.readline()
nnodes=int(line1.split()[0])
nodes=[]
for i in range(nnodes):
    pos=f.readline().split()
    p=pg.RVector3(float(pos[1]),float(pos[2]),float(pos[3]))
    n=Poly.createNodeWithCheck(p)
    nodes.append(n)

line2=f.readline()
nfaces=int(line2.split()[0])
for i in range(nfaces):
    bla = f.readline()
    ind = f.readline().split()
    fa = Poly.createTriangleFace(nodes[int(ind[1])],nodes[int(ind[2])],
                                 nodes[int(ind[3])],0)
    fa.setMarker(-2) # for outer boundary (mixed boundary conditions)

f.close()
Poly.exportAsTetgenPolyFile('test.poly')
```

The resulting test.poly can now be meshed using tetgen -Y (keep faces):

`$ tetgen -pazVACY -q2 test`

If there are errors due to intersecting facets they can be identified using

```
$ tetgen -d test
$ meshconvert -V -it -o wrong test.1
```

We observed that tetgen 1.4.3 is doing a good job in contrast to 1.4.2. Check which version you use (tetgen -h) and make sure you use 1.4.3. If tetgen meshes correctly, we have two meshes, which have to be merged

```
Outer = pg.Mesh('right.bms') # outer box (world)
Inner = pg.Mesh('para.bms')  # inner box (parameters)
print(Outer, Inner)
# set all outer cells to 1 and inner cells to 2
for c in Outer.cells(): c.setMarker(1)
    for c in Inner.cells():
        c.setMarker(2)
        Outer.copyCell(c)

print(Outer)
Outer.save('mesh')
Outer.exportVTK('mesh')
```

And there we go. The resulting mesh.bms can be used for inversion easily just by specifying `PARAMESH=mesh.bms` in the cfg file. The whole script is either in doall.sh calling the individual python files or in makeall.sh, where the python commands are directly invoked.

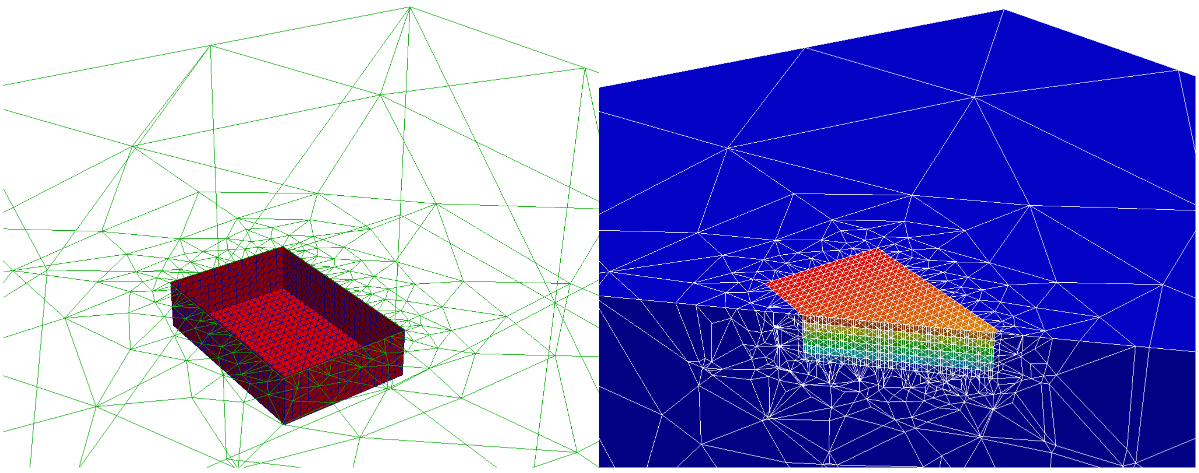


Figure 32: Boundary of the inner 3D grid (red faces and blue lines) and of the outer box (green lines)

Note that the cells in the interior have the marker 2, i.e. the describe one region as usual in inversion, however with tetrahedral cells since we divided each hexahedron as such. In order to avoid this, we do not set the marker and keep the subsequently increasing numbers from create3DGrid, so that every tetrahedron is one region with 5 tetrahedra. By specifying the single attribute in a region file

```
#No single trans
*   1      log
#No background
0   1
#Inter-region
*   *    1
```

we actually invert for hexahedra with smoothness constraints between each other. Besides the `REGIONFILE=region.control` keyword, `SECMESHREFINE=0` is useful to avoid refinement of the already refined cells.

E.6. 3D inversion and visualization of 2d profiles in a 3d topography

`doc/HowTo/2d3dtopo`
Origin: Thomas Günther & Dave Tanner (LIAG Hannover)
Task: Generate 2d and 3d data files from 3d DEM and visualize results
Problem: 2d profiles are local (tape-measure) and DEM global
Solution: reading pro-file, tape-unrolling and vtk file manipulation

Since true 3d measurements (grid of electrodes) are often prohibitive due to limited electrodes, usually pseudo-3d, i.e. 2d profiles - not necessarily parallel and perpendicular, are measured. For georeferencing, total stations yield a digital elevation model and the course of the profiles by a polygon of 2 or more points. The location of the profile is stored in a pro-file (used e.g. by DC3dInvRes) with a line for each 2d profile looking like:

```
filename_of_2dfile x1 y1 x2 y2 ...
```

First we read in the filenames and points:

```
XL, YL, FILES = [], [], []
fid = open('kilmore2011.pro')
for line in fid:
    FILES.append( line.split()[0] )
    XL.append(np.double(line.split()[1::2]))
    YL.append(np.double(line.split()[2::2]))

fid.close()
```

Next, we read in the DEM and make a Delaunay triangulation

```
import matplotlib.tri as mtri
x, y, z = np.loadtxt('topo.xyz', unpack=True)
tri = mtri.Triangulation(x, y)
interp_lin = mtri.LinearTriInterpolator(tri, z)
```

We create a new 3d mesh with the DEM points as nodes and the triangulatation as cells:

```
for x1, y1, z1 in zip(x,y,z):
    mesh.createNode(x1, y1, z1)

for v in tri.triangles:
    mesh.createCell(pg.Triangle(mesh.node(int(v[0])),
          mesh.node(int(v[1])), mesh.node(int(v[2]))))

mesh.exportVTK('topo.vtk')
```

For each of the profiles we need to roll-on the tape coordinates from the datafile on the topography. Therefore we create a densely sampled point list along the profile course (given by xl/yl arrays) and create a pure node mesh of it, before the topography is achieved using the GIMLi surface mesh interpolation routine.

```
le = np.hstack((0., np.cumsum(np.sqrt(np.diff(xl)**2+np.diff(yl)**2))))
xi = np.interp(np.linspace(le[0],le[-1],nump), le, xl)
yi = np.interp(np.linspace(le[0],le[-1],nump), le, yl)
zi = interp_lin(xi, yi)
```

For the obtained point list we compute the tape distance along the topography and interpolate all three coordinates from it.

```
dt = np.sqrt(np.diff(xi)**2 + np.diff(yi)**2 + np.diff(zi)**2 ) # differences
ti = np.hstack((0., np.cumsum(dt))) # tape distance along profile
data = b.DataContainerERT(datfile)
te=[pos.x() for pos in data.sensorPositions()]
ze = np.interp(te, ti, zi)
xe = np.interp(te, ti, xi)
ye = np.interp(te, ti, yi)
```

For a 2d inversion we need the direction along the profile. The tape-true coordinate mapping is saved to some temporary file that will later be used.

```
x2d = np.hstack((0.,np.cumsum(np.sqrt( np.diff(xe)**2+np.diff(ye)**2 ))))
ALL = np.vstack((te, xe, ye, ze))
np.savetxt('positions/'+datfile.replace('.dat','.txyz'), ALL.T)
```

First we want to generate a valid 2d file using the latter coordinate, saving exactly the field read in:

```
for i in range(data.sensorCount()):
    data.setSensorPosition(i, pg.RVector3( x2d[i], 0.0, zn[i]))

filename2d = '2dfiles/' + datfile.replace('.dat', '_2d.dat')
data.save(filename2d, data.inputFormatString())
```

Next, we set the coordinates to the real position and subsequently add the data to a previously created empty data container.

```
for i in range(data.sensorCount()):
    data.setSensorPosition(i, pg.RVector3(xe[i], ye[i], ze[i]))

data3d.add(data)
```

We are now doing inversion of the individual 2d files and the 3d file. Whereas the 3d file already has the correct 3d coordinates, we must back-transform the resulting vtk file. Assume we have the results named the same way as the 2d data files but with the extension vtk. We read in the vtk file and the map file previously created:

```
posfile = 'positions/' + datafile.replace('.dat','.txyz')
t, x, y, z, x2d = np.loadtxt(posfile, unpack=True)

vtkfile = datfile.replace('.dat','.vtk')
mesh = pg.Mesh(vtkfile)

xm = pg.x(mesh.positions()) # 2d x
zn = pg.x(mesh.positions()) # z
```

Then we interpolate from the 2d coordinates to the 3d coordinates. Instead of numpy.interp we need also to extrapolate since the mesh usually goes beyond the electrodes. For this reason we write a simle interpolation-extrapolation function:

```
def myextrap(x, xp, yp):
    """ numpy.interp function with linear extrapolation """
    y = np.interp(x, xp, yp)
    y = np.where(x<xp[0], yp[0]+(x-xp[0])*(yp[0]-yp[1])/(xp[0]-xp[1]), y)
    y = np.where(x>xp[-1], yp[-1]+(x-xp[-1])*(yp[-1]-yp[-2])/(xp[-1]-xp[-2]), y)
    return y
```

and use this function for determining x and y positions of the mesh nodes

```
xn = myextrap(xm, x2d, x)
yn = myextrap(xm, x2d, y)
```

Finally we set the positions of the mesh nodes and export the vtk.

```
for i in range(mesh.nodeCount()):
    mesh.node(i).setPos(pg.RVector3(xn[i], yn[i], zn[i]))
mesh.exportVTK('3dvtk/' + vtkfile)
```

Now all vtk files in the folder 3dvtk can be loaded into paraview to form a fence diagram. Additionally the result from the 3d inversion can be loaded and compared to the 2d inversions. If the topography does not show a large curvature, alternatively the 2d inversions could be done with the original (tape measure) files and the back-transformation will add the topography:

```
xn = myextrap(xm, t, x)
yn = myextrap(xm, t, y)
zn = zn + myextrap(tn, t, z)
for i in range(mesh.nodeCount()):
    mesh.node(i).setPos(pg.RVector3(xn[i], yn[i], zn[i]))
```

The resulting vtk files can be viewed together in ParaView and their visualization is very easy to control by grouping the files. For even more convenience, the meshes can be joined into a single vtk. Finally, the result looks like in Figure 33.

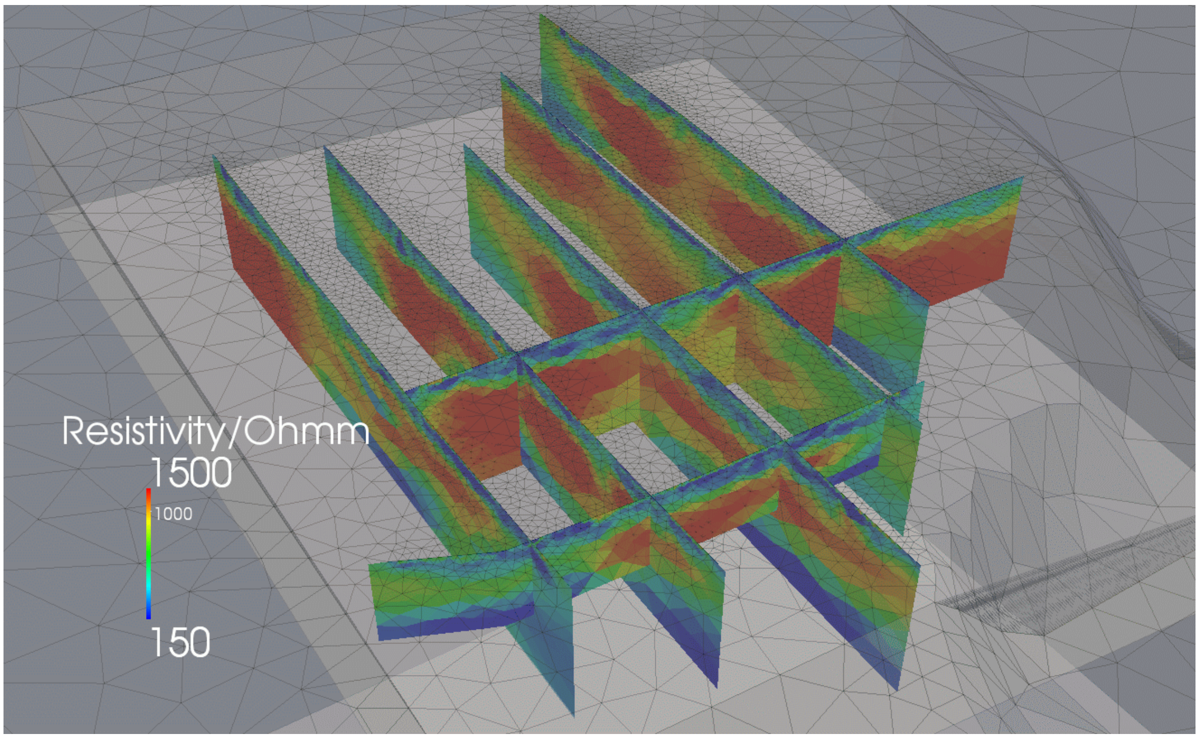


Figure 33: Fence diagram of 2d results with 3d surface mesh

E.7. Use a Hydrus3D mesh of a soil column for forward calculation

(note the limitation due to renumbering the boundary nodes)
`doc/howto/Simulate_Hydrus_results`
Origin: Ulla Noell (BGR Hannover)
Task: Use discretization and resistivity vector from Hydrus3D simulation
Problem: Append unstructured mesh boundary to (usually regular) tetrahedral mesh

With the monitoring of hydraulic processes using ERT the demand for an efficient forward calculation for a given water content and thus resistivity arises. Hydrus3D uses meshes of tetrahedra that are usually regularly arranged. For an appropriate forward calculation boundaries far away from the parameters are needed. A further regular prolongation will cause a huge increase of nodes and thus runtime. Therefore the parameter box is surrounded by unstructed tetrahedra whose resistivity is determined using parameter prolongation as done in inversion. The problem is very similar to the one of creating regular 3d meshes for inversion, except that (a) the input is already a tetrahedal mesh and (b) the output must be defined such that a forward calculation using dcmod can be directly called. According to (a), the first part of creating the mesh and refining it is omitted. Instead, the Hydrus mesh is imported and saved using the following pygimli script. (Unzip MESHTRIA.TXT.gz using gzip -d to obtain the Hydrus3D mesh)

```
mesh = pg.Mesh(3) # new mesh
f = open('MESHTRIA.TXT','r')
for i in range(6): # read first 6 lines
    line1 = f.readline()

nnodes = int(line1.split()[0]) # number of nodes
ncells = int(line1.split()[1]) # and cells
print nnodes, ncells
line1 = f.readline()
nodes = []
for ni in range(nnodes): # read all nodes and append in mesh
    pos = f.readline().split() # no.,x,y,z -> position
    p = pg.RVector3(float(pos[1]), float(pos[2]), float(pos[3]))
    n = mesh.createNode(p)
    nodes.append(n)

line1=f.readline()
line1=f.readline()
cells=[]
for ci in range(ncells): # read all cells and append in mesh
    pos = f.readline().split()
    i,j,k,l = int(pos[1]),int(pos[2]),int(pos[3]),int(pos[4])
    c = mesh.createTetrahedron(nodes[i-1],nodes[j-1],nodes[k-1],nodes[l-1])
    cells.append(c)

f.close()

print mesh # print mesh properties and export as bms and vtk
mesh.save('hydrus')
mesh.exportVTK('hydrus')
```

As a result, we have hydrus.bms and hydrus.vtk and can visualize it using ParaView. Just as in regular3d meshes, its boundary is exported as vtk file.

```
mesh.createNeighbourInfos()
for b in mesh.boundaries():
    if not (b.leftCell() and b.rightCell()):
        b.setMarker(1)

poly = pg.Mesh(3)
```

```
poly.createMeshByBoundaries(mesh, mesh.findBoundaryByMarker(1))
poly.exportAsTetgenPolyFile('paraBoundary')
```

The rest is just as in the other example, i.e. - Part 2: create world, mesh to obtain boundary - Part 3: merge surfaces of inner and outer box and mesh together except that the markers for the individual cells are not constantly 2 (inversion), but counted starting from 1.

```
# merge inner mesh into outer
for m, c in enumerate(Inner.cells()): # run through all cells
    nodes = pg.stdVectorNodes()
    for i, n in enumerate(c.nodes()): # add nodes if not existent
        nodes.append(Outer.createNodeWithCheck(n.pos()))
    Outer.createCell( nodes, m+1 );
```

Note further that the cells in the outer box keep the marker 0. In order to make a forward calculation we first need to map the markers into resistivities by creating a two-column rho.map file starting with 0 0 (marker zero is mapped into 0 resistivity, which means filling up with neighboring values):

```
res = ...      # load from file
ind = np.arange(len( res ) + 1)
A = np.vstack((ind, np.hstack((0, res)))).T
f = open('rho.map', 'w')
for row in A:
    f.write('%d' % row[0] + '\t' + '%f' % row[1] + '\n')
f.close()
```

The final call to dcmod then reads

```
$ dcmod -v -S -a rho.map -s datafile mesh.bms
```

E.8. How to use Hydrus2D simulations for synthetic data inversion

`doc/HowTo/Hydrus_mesh2d`
Contribution: Sarah Garre (KU Leuven)
Task: use a Hydrus2D mesh/resistivity for forward simulation and inversion the given triangular mesh (MESHTRIA.TXT) exhibits topography
The background and results are explained in more details by Garre et al. (2012).
Problem: the given Hydrus mesh does not have

1. a boundary far enough to ensure accurate solutions
2. appropriate boundary conditions (mixed BC on the 3 outer boundaries in the subsurface)

Solution:

1. read in the native Hydrus2d format
2. add a box around the mesh with appropriate BC
3. triangulate outer space and merge with inner mesh
4. refine and do the forward calculation within pygimli
5. set up an inversion with surface/subsurface electrodes

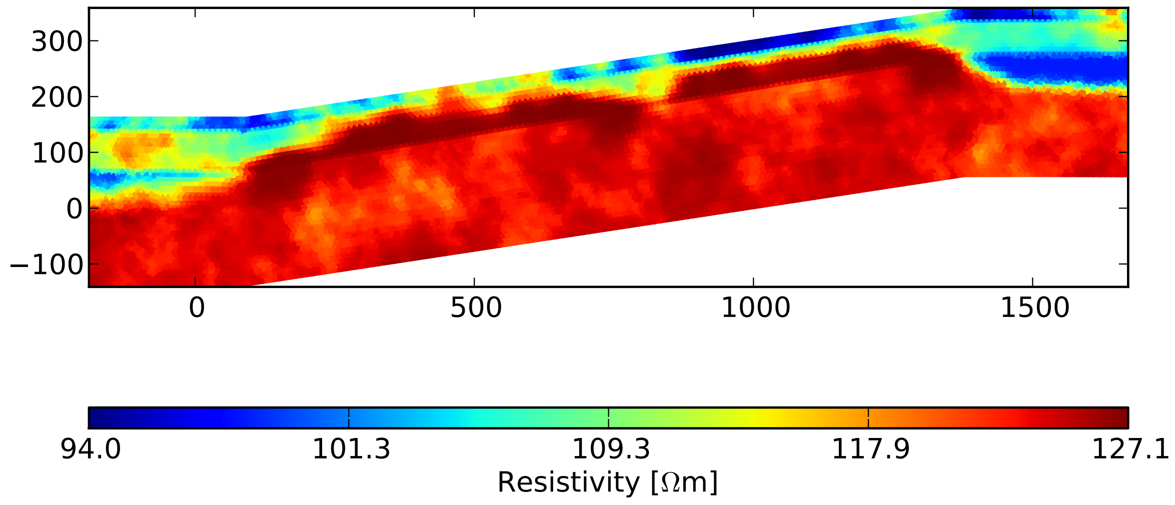


Figure 34: Resistivity mesh as obtained by Hydrus2d modelling

Figure 34 shows the mesh and given resistivity distribution as obtained from a hydraulic simulation and application of Archie's Law.
First we read in the MESHTRIA.TXT using the function readHydrus2dMesh, see `pygimli/utils/mesh.py` for the code as an example how to read in ASCII files. Then we create neighboring information, number the markers of all cells beginning by 1. With showMesh(mesh) we can display it (from pygimli.viewer import showMesh).

```
import pygimli as pg

mesh = readHydrus2dMesh( 'MESHTRIA.TXT' )
mesh.createNeighbourInfos()
for i,c in enumerate( mesh.cells() ):
    c.setMarker( i + 1 )
```

Next, we find all boundary nodes and sort them clock-wise:

```
x, y, bNodes = [], [], []
for b in mesh.boundaries():
    if not b.leftCell() or not b.rightCell():
        for n in b.nodes():
            if n not in bNodes:
                bNodes.append( n )
                x.append( np.round( n.pos().x() * 1e3 ) / 1e3)
                y.append( np.round( n.pos().y() * 1e3 ) / 1e3)

xm = np.mean(x)
ym = np.mean(y)
angles = np.array( np.angle( x - xm + ( y - ym ) * 1j ) )
ind = np.argsort( angles )
```

We create an empty mesh, create the boundary nodes and connect them by creating edges:

```
poly = pg.Mesh(2)
for i in range(len(ind)):
    poly.createNode( bNodes[ind[i]].pos() )
```

```
for id in range( 0, poly.nodeCount() ):
    poly.createEdge( poly.node( id ), poly.node( (id + 1)%
        poly.nodeCount() ), pg.MARKER_BOUND_HOMOGEN_NEUMANN )
```

We now extract the four corners to construct the outer box.

```
y1b = min( y[ x==min(x) ] )
y1t = max( y[ x==min(x) ] )
y2b = min( y[ x==max(x) ] )
y2t = max( y[ x==max(x) ] )
x2t = max( x[ y==max(y) ] )
xbound, ybound = 500., 500.
n1 = poly.createNode( pg.RVector3( min(x) - xbound, y1t, 0.0 ) )
n2 = poly.createNode( pg.RVector3( min(x) - xbound, y1b - ybound, 0.0 ) )
n3 = poly.createNode( pg.RVector3( max(x) + xbound, y2b - ybound, 0.0 ) )
n4 = poly.createNode( pg.RVector3( x2t + xbound, max(y), 0.0 ) )
```

The four corners are connected with each other and with the two upper corners of the inner box:

```
poly.createEdge( n1, n2, pg.MARKER_BOUND_MIXED )
poly.createEdge( n2, n3, pg.MARKER_BOUND_MIXED )
poly.createEdge( n3, n4, pg.MARKER_BOUND_MIXED )
i1=int( np.nonzero( ( x[ind]==min(x) )&( y[ind] == y1t ) )[0][0])
i2=int( np.nonzero( ( x[ind]==x2t )&( y[ind] == max(y) ) )[0][0])
poly.createEdge(n1, poly.node(i1), pg.MARKER_BOUND_HOMOGEN_NEUMANN )
poly.createEdge(poly.node(i2), n4, pg.MARKER_BOUND_HOMOGEN_NEUMANN )
```

We can output the poly file as vtk and show it:

```
poly.exportVTK('out.poly')
showMesh(poly)
```

We create a new triangle object, put a marker in and generate the mesh:

```
tri = pg.TriangleWrapper( poly ) # new triangle object
tri.addRegionMarkerTmp( 0, pg.RVector3( mesh.xmin() + 1.,
                                        mesh.ymin() + 1. ), -1 )
tri.setSwitches( '-pzeAfaq34' ) # plc,0 index,attribute, quality 34
mesh2 = pg.Mesh(2)
tri.generate( mesh2 )
```

Finally we copy all cells from the original mesh into the obtained mesh:

```
for cell in mesh.cells():
    mesh2.copyCell( cell )

showMesh( mesh2 )
mesh2.save('mesh.bms')
```

Since a forward calculation requires higher accuracy we decide to do it by using total potential calculation on a quadratic grid by making:

```
mesh3=pg.Mesh(2)
mesh3.createP2Mesh(mesh2)
mesh3.save('meshFor.bms')
print mesh3
```

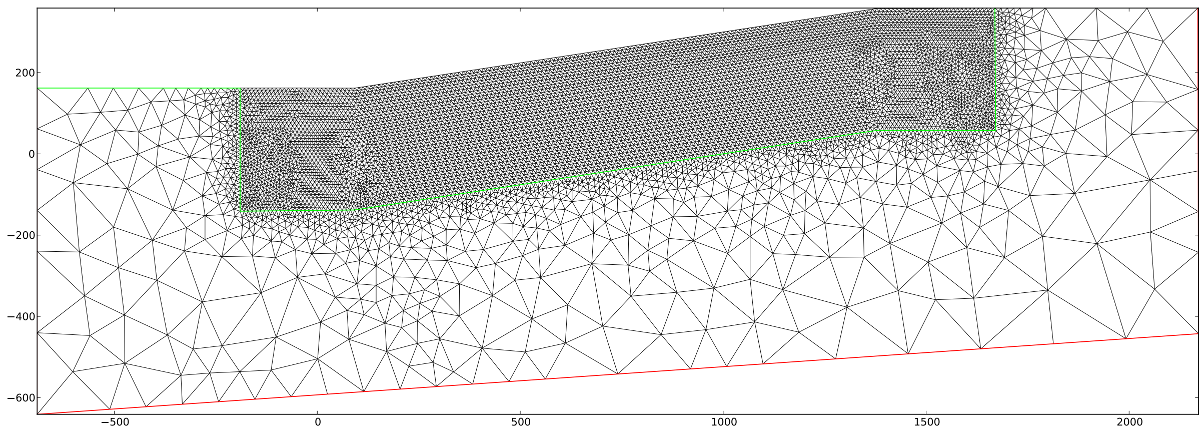


Figure 35: Input for meshing (red and green lines) and combined mesh

Now we come to the actual forward calculation: The resistivity information is retrieved from a VTK file by importing:

```
vtk=pg.Mesh(2)
vtk.importVTK('resistivity.vtk')
res=np.array(vtk.exportData('Resistivity'))
```

The resistivity vector is brought into a map file using a std C++ map with the number(marker) in the first column (key) and res in the second:

```
nr=np.arange(len(res))+1
resmap=np.vstack((nr.T,res.T)).T
cellMap = pg.stdMapF_F()
for key, val in resmap:
    cellMap[key]=val
```

The resulting resistivity map is applied and empty cells, i.e. cells in the created outer region (marker 0) obtain resistivity by prolongation:

```
mesh3.mapCellAttributes( cellMap )
mesh3.fillEmptyCells( mesh3.findCellByAttribute( 0.0 ), -1.0 )
```

We now load the scheme file (data without actual data), create a new modelling operator and calculate it. The resulting u are treated as R.

```
data=pg.DataContainer('hydrus2d.shm')
f=pg.DCMultiElectrodeModelling(mesh3,data,True)
f.calculate(data)
data.set('r',data.u())
data.save('out.dat','a b m n r')
```

Now we can do inversion of the data set by using the bert script. The problem here is that we have both surface and subsurface electrodes which is not yet automatically covered by bert (but will be soon). Therefore we create a dummy data file with only the surface electrodes, use it for mesh generation and invert the synthetic data after adding some Gaussian noise to it:

```
nsel=36 # number of surface electrodes
ERR=1
```

```
infile=out.dat
outfile=out0.dat
datfile=syn.dat
# create dummy data file with only surface electrodes
echo $nsel > $outfile
head -n 2  $infile|tail -n 1 >> $outfile
head -n $[nsel + 2] $infile|tail -n $nsel | tac >> $outfile
echo 0 >> $outfile
# create cfg file for generating meshes
bertNew2DTopo $outfile > bert0.cfg
echo PARADEPTH=300 >> bert0.cfg
echo PARADX=0.2 >> bert0.cfg
# do only the meshes with the dummy file
bert bert0.cfg meshs
# Noisify synthetic data with Gaussian Noise
dcedit -vSEN -e $ERR -u 0 -o $datfile $infile
# create normal cfg file and do inversion
bertNew2DTopo $datfile > bert.cfg
echo LAMBDA=300 >> bert.cfg
echo ZWEIGHT=0.1 >> bert.cfg
echo BLOCKYMODEL=1 >> bert.cfg
# cp mesh.bms mesh/  # use Hydrus mesh for inversion
bert bert.cfg pot calc
```

Figure 36 shows the obtained inversion result, which is close to the synthetic model. The very low contrasts in the model and limited number of electrodes prevent a better reconstruction. In case of realistic heterogeneities such a result could only be retrieved by a time-lapse scheme, where all systematic effects and small-scale structures will be cancelled out.

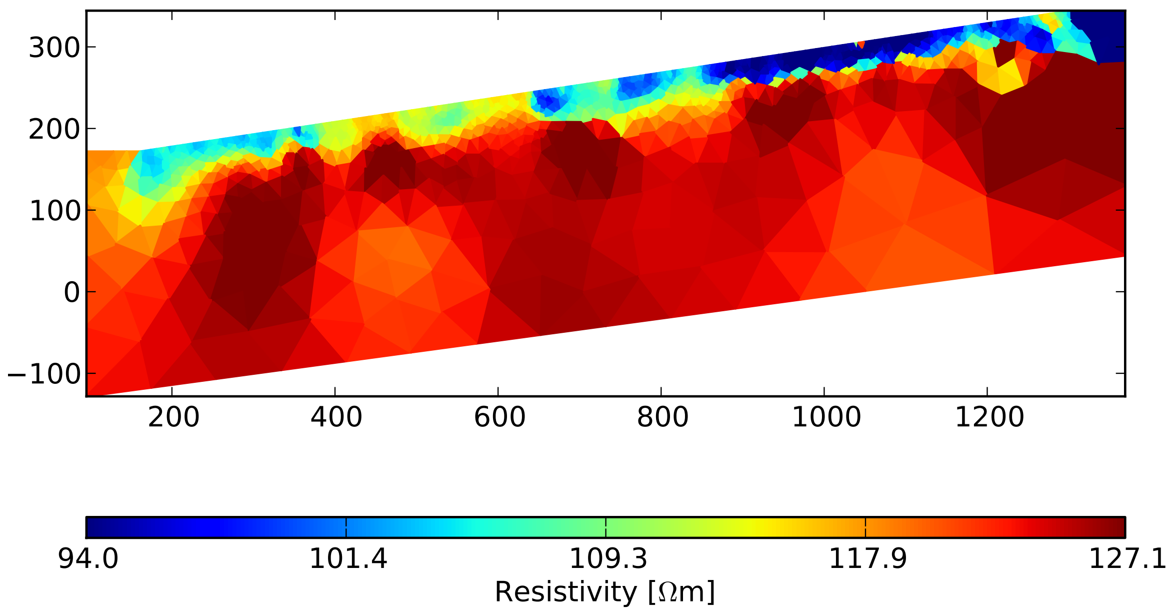


Figure 36: Obtained inversion result

E.9. Simulate CEM ring electrodes in tilted boreholes

`doc/howto/tilted_boreholes`
Contribution: Laure Beff (UC Louvain)
Task: Simulate the real geometric factors of ring electrodes using CEM-FEM
Problem: Define a mesh that contains tilted borehole tools with ring electrodes
The borehole probes are cylinders with metal ring electrodes at certain positions. Cylinders are approximated by regular prisms (e.g. 6 segments), which can be created by

```
    createCube -Z -s 6 cube
```

creating a 1x1x1m cylinder centered at the origin. The space of the cylinder is disregarded from meshing by adding a hole marker using -H. The probe tool itself consists of a sequence of non electrodes and electrodes, which are subsequently merged with each other and after translation to the borehole position they are merged with the big box (world in which the simulation is done) with the typical boundary conditions:

```
polyCreateWorld -x $SIZE -y $SIZE -z $SIZE -m1 $MESH # big box=world
```

Before some lengths are stored in scalar and vectorial variables, e.g.

```
RAD=0.045 # radius of borehole
DIST=(0.103 0.152 0.202 0.252 0.302 0.353 0.100) # electrode distances
```

At the beginning, we create prototypes of the two different cylinders and shift it such that the top is zero and they have the correct radius

```
polyCreateCube -H -Z -s $SEG cyl # create (closed) unit cylinder
polyTranslate -z -0.5 cyl        # shift to upper boundary=surface
polyScale -x $RAD -y $RAD cyl    # scale to correct radius
```

Since the distances differ, their height will be scaled later, the electrode now

```
cp cyl.poly elec0.poly           # unit electrode
polyScale -z $DRING elec0        # right height
```

Each borehole b is constructed at the origin by starting with a cylinder element

```
cp cylC.poly borehole.poly # uppermost piece
```

A variable z holds the current depth and is initialized with the distance to the first electrode, which is retrieved from the vector PD by using

```
z=${PD[$b]}
```

After each adding of an element z is increased by adding the individual length using awk

```
z=`echo $z $DRING|awk '{ print $1 + $2}'` # `` # output in z
```

The standard electrode is copied and translated and an electrode marker $ELM. is associated, which is later used to identify the CEM electrode (starting from -10000 downwards). Finally the electrode is merged with the borehole

```
cp elec0.poly elec.poly          #
polyAddVIP -B -m -$ELM elec
polyTranslate -z -$z elec
polyMerge borehole elec borehole
```

Similar is done with the pieces between the electrode, i.e. scale, translate and merge. The inner loop

```
for e in {0..6}
do
    ...
done
```

runs over the electrodes and the outer (b) over the borehole tools. Before merging with the world, the electrodes are translated to their positions. However, the boreholes are tilted by a specific angle (also stored the vector ROT). The rotation around the y axis (i.e. y remains constant) is done by

```
polyRotate -R -y ${ROT[$b]} borehole # -R for rad (default=deg)
```

The uppermost face (of each of the 6 electrodes) is intersecting the earth's surface and must therefore be moved to z=0. Therefore a copy of the borehole poly file is created and the lines with the points close to the surface are changed accordingly using head and tail commands

```
head -n1 borehole.poly > newborehole.poly # keep line 1, change 2+3
head -n3 borehole.poly |tail -n2|
awk '{print $1 "\t" $2 "\t" $3 "\t" 0 "\t" $5}' >> newborehole.poly
head -n6 borehole.poly |tail -n3 >> newborehole.poly # keep 4-6 etc.
```

The problem is that the rectangle side faces of the uppermost segment are not coplanar anymore due to the movement. One solution could be a more sophisticated positioning. Another way is to dissect all rectangles into triangles that are coplanar by definition. This is achieved by transforming the very first piece (the beginner) to the common STL format (that uses triangles) and then back to POLY using polyConvert:

```
polyConvert -S borehole.poly
polyConvert -P borehole.stl
```

Unfortunately, the hole marker is lost in the STL file, so it must be re-added:

```
polyAddVIP -H -x 0 -y 0 -z -0.01 borehole
```

Note that all PLCs ca be converted in to VTK (and viewed in ParaView) using

```
polyConvert -V file.poly
```

When looking at the PLC, we see that the lower face of the upper piece is dissected, whereas the upper face of the adjacent electrode at the same position has still 6 nodes. Therefore we create all the electrode cylinders with the option -O so that the top and bottom faces are omitted. As a consequence, we have a valid mesh input, which can be meshes using tetgen and transformed into a BMS/MESH/VTK format files using:

```
tetgen -pazVACq1.15 $MESH # mesh $PLC with quality=1.2 - mesh.1.*
meshconvert -vBDMV -it -o $MESH $MESH.1 # convert to BMS/MESH/VTK
```

To achieve high accuracy we further transform the mesh to quadratic elements using -p

```
meshconvert -vBD -p -o ${MESH}P $MESH.bms
```

Note that if there are meshing errors the problematic zones can be investigated using

```
tetgen -d meshfile
```

and the resulting mesh only contains the mesh dissections or non-coplanar elements. If there are additional surface electrodes they can be added using polyAddVIP

```
polyAddVIP -f surface_elec.xyz -m -99 $MESH
```

They should be locally refined to a/10, where a is the electrode distance:

```
polyRefineVIPS -z -0.015 $MESH
```

The whole script reads as below.
Now we want to model with the script and investigate the difference to point electrodes. First we run dcmod with -1 and the scheme file applied

```
dcmod -v -1 -s datafile mesh/meshP.bms # outputs num.ohm
```

Note that although dcmod tries to find electrodes by their position, there is a priority, first CEM electrodes and then are searched and then nodes. In cases of close distances between different types a wrong association may occur resulting in "free electrodes". We therefore recommend to always use CEM before Node and free electrodes by sorting data. Next, we create a file containing the analytical geometric factor using

```
dcedit -f "a b m n k" -o geom.dat datafile
```

Finally we use this file to "filter" the numerical results with these geometric factors.

```
dcedit -vSB -c geom.dat -o eleff.dat num.ohm
```

As the modeled $R = 1/k_{num}$, the resulting $\rho_a = k_{ana} * R = k_{ana}/k_{num}$ is the electrode effect.