# E4D-HR User Guide

Version 1.1.0

ExploreGeo

September 30, 2025

# Contents

# 1 Introduction

E4D-HR is a modified version of E4D, with changes made for mineral exploration use cases and some additional user-friendly features. E4D is a scalable, 3D geophysical modelling and inversion code designed to efficiently handle large datasets from direct current resistivity (DCR) and induced polarisation (IP) surveys.

This document is intended to guide geophysicists, who are familiar with DCR and IP methods, in using E4D to invert their data.

# 2 Installation

## 2.1 System Requirements

E4D-HR is compatible with the following Linux distributions. Windows users with versions that support WSL2 (Windows Subsystem for Linux) are also supported.

- **Ubuntu:** 20.04 (Focal Fossa), 22.04 (Jammy Jellyfish), 24.04 (Noble Numbat)
- **WSL2:** Windows 10 (Version 1903 or later), Windows 11

## 2.2 Required Software

- **Intel oneAPI Base Toolkit** 2024.2.0 or later
- **Intel HPC Toolkit** 2024.2.0 or later
- **build-essential** (Ubuntu and Debian)
  - Install by running: `sudo apt-get update && sudo apt-get install build-essential`

## 2.3 Downloading E4D

To download E4D-HR, follow these steps:

1. Visit the E4D-HR GitHub repository.
2. On the repository page, click on the **Code** button.
3. You can either:
   - **Clone the repository** using Git by running the following command in your terminal:

     ```
     git clone https://github.com/ofgn/E4D-HR.git
     ```

   - **Download the ZIP file** by clicking on **Download ZIP**, and then extract the contents to your desired directory.

### 2.4 Automatic Installation

For an automatic installation of E4D, use the provided `install.sh` script. This script handles the downloading, extracting, compiling, and setting up of all dependencies and E4D itself. To use the script, follow these steps:

1. Ensure that the necessary environment variables for Intel MKL are set.
2. Open a terminal and navigate to the directory where `install.sh` is located.
3. Make the script executable:

   ```
   chmod +x install.sh
   ```

4. Run the script:

   ```
   ./install.sh
   ```

5. The script will automatically handle the installation of dependencies and E4D compilation. Follow any prompts provided by the script.

Using the automatic installation script simplifies the setup process and ensures that all components are installed correctly with minimal manual intervention.

### 2.5 Manual Installation

To manually install E4D, follow these steps:

1. **Prepare Your Environment**:
   - Ensure that the Intel MKL environment variables are set. Source the Intel environment variables script (replace `<intel-compiler-dir>` with the path to your Intel installation):

     ```
     source <intel-compiler-dir>/setvars.sh
     ```

   - Verify that the `MKLROOT` environment variable is set by running:

     ```
     echo $MKLROOT
     ```

   - If `MKLROOT` is not set, refer to Intel's documentation or support for configuration instructions.

2. **Install Dependencies**:
   - **Install PETSc**:
     – Download and extract the PETSc source tarball (version 3.21.4):

       ```
       tar -xzf petsc-3.21.4.tar.gz
       ```

     – Navigate to the extracted directory:

       ```
       cd petsc-3.21.4
       ```

  - Configure PETSc with the appropriate options:
    ```
    ./configure --with-debugging=0 --with-fc=mpiifort -fc=ifx \\
    --with-cc=mpiicc -cc=icx --with-cxx=mpiicpc -cxx=icpx \\
    --with-blaslapack-dir=$MKLROOT
    ```
  - Compile PETSc:
    ```
    make all
    ```
- **Install Triangle**:
  - Download and extract the Triangle source tarball (version 1.6):
    ```
    tar -xzf triangle-1.6.tar.gz
    ```
  - Navigate to the extracted directory:
    ```
    cd triangle-1.6
    ```
  - Compile Triangle:
    ```
    make
    ```
- **Install TetGen**:
  - Download and extract the TetGen source tarball (version 1.6.0):
    ```
    tar -xzf tetgen-1.6.0.tar.gz
    ```
  - Navigate to the extracted directory:
    ```
    cd tetgen-1.6.0
    ```
  - Compile TetGen:
    ```
    make
    ```

3. **Compile E4D**:
   - Navigate to the E4D source directory:
     ```
     cd path/to/E4D-HR/src
     ```
   - Clean any previous builds:
     ```
     make clean
     ```
   - Compile E4D with the configured PETSc:
     ```
     make FC=mpiifort -fc=ifx \\
     FFLAGS="-O2 -xHost -r8 -heap-arrays" \\
     PETSC_DIR=path/to/petsc-3.21.4
     ```
   - Move the compiled E4D executable to the `bin` directory:
     ```
     mv e4d path/to/E4D-HR/bin
     ```

# 3 Background

This section provides some technical details about E4D, which can be skipped. It offers context on the software's capabilities and underlying principles, primarily for readers interested in the specific functionalities and computational methods employed in the modelling process.

## 3.1 Complex Conductivity

IP can be effectively described using the concept of complex conductivity, which captures both the conduction and polarisation properties of subsurface materials. E4D opts to use complex conductivity by decoupling the real and imaginary parts in the real number domain.

Complex conductivity $\sigma^*$ is used to describe the electrical properties of subsurface regions. It is expressed as:

$$\sigma^* = \sigma' + i\sigma'', \tag{1}$$

where $\sigma'$ represents the conduction strength due to electromigration, and $\sigma''$ represents the polarisation strength associated with IP effects.

The phase $\phi$ associated with $\sigma^*$ is defined as:

$$\sigma^* = |\sigma^*|e^{i\phi}, \tag{2}$$

Here, $\phi$ represents the phase lag of the induced alternating current (AC) electric field behind the injected AC current. The real part $\sigma'$ is linked to the movement of charges through the material, while the imaginary part $\sigma''$ is associated with charge storage mechanisms, such as polarisation effects.

### 3.1.1 Calculating Phase from Real and Imaginary Conductivity

The phase $\phi$ can be computed from the real part $\sigma'$ and the imaginary part $\sigma''$ of the complex conductivity using the following equation:

$$\phi = \arctan\left(\frac{\sigma''}{\sigma'}\right), \tag{3}$$

where $\arctan$ denotes the inverse tangent function. This equation gives the phase in radians, representing the lag of the AC field relative to the current.

### 3.1.2 Calculating Imaginary Conductivity from Real Conductivity and Phase

The imaginary part of the complex conductivity $\sigma''$ can be calculated from the real part $\sigma'$ and the phase $\phi$ using:

$$\sigma'' = \sigma' \cdot \tan(\phi), \tag{4}$$

where $\tan$ denotes the tangent function. This formula derives the imaginary conductivity based on the known real conductivity and the phase angle.

### 3.1.3 Magnitude of Complex Conductivity

The magnitude of the complex conductivity $|\sigma^*|$ is calculated as:

$$|\sigma^*| = \sqrt{(\sigma')^2 + (\sigma'')^2}. \tag{5}$$

### 3.2 Decoupling Real and Imaginary Components

The use of complex numbers in E4D allows for the decoupling of the real and imaginary components of conductivity during forward and inverse modelling. This decoupling simplifies the computational process by allowing independent treatment of the real and imaginary components.

In typical IP surveys, the phase angle $\phi$ is relatively small (e.g., less than 0.2 radians), which means the influence of the imaginary component on the real component is often negligible. This allows the real and imaginary parts to be modelled separately, reducing memory requirements by a factor of two and improving computational efficiency.

Moreover, this decoupling enables the specification of noise levels separately for the real and imaginary components of the potential (or equivalently, for the magnitude and phase), removing ambiguities in data misfit residuals that exist in the complex number domain. This approach ensures that the inversion process appropriately fits the data, leading to more accurate subsurface characterisation.

### 3.3 DCR Data

E4D expects the transfer resistance ($R_T$) as the DCR observation. $R_T$ is given in ohms and is the ratio of the primary voltage ($V_p$) and the applied current ($I$). Alternatively, it can be calculated as the ratio of the apparent resistivity ($\rho_a$) and the geometric factor ($K$) of the measurement's electrode configuration.

$$R_T = \frac{V_p}{I} \tag{6}$$

$$R_T = \frac{\rho_a}{K} \tag{7}$$

### 3.4 IP Data

E4D uses the phase difference in radians between the source and measured potentials for IP modelling. This approach differs from other inversion programs that may use integral chargeability, Cole-Cole parameters, or alternative methods. To accommodate data obtained through various techniques, a correction factor is applied for conversion purposes. While the specific value of the correction factor varies with different instruments and acquisition settings, Table 1 provides rough conversion factors for common IP measurements, which have generally yielded satisfactory results.

| Unit | Conversion |
|---|---|
| 1 mrad | 0.001 rad |
| 1 mV/V | 0.001 rad |
| 1 ms | 0.001 rad |
| 1 Percent Frequency Effect | 0.01 rad |

Table 1: IP unit conversion.

### 3.5 Triangular Mesh

A triangular mesh is a 2D mesh composed of interconnected triangles. In E4D, this type of mesh is primarily used to represent surfaces such as topography or other planar features. The triangles in the mesh are defined by vertices that correspond to control points, which could be surface electrodes or specific points on the topography. This triangular mesh serves as the foundation for more complex 3D meshes by accurately capturing the shape and variations of the surface it represents. The simplicity and flexibility of triangular meshes make them ideal for creating detailed surface representations that can be used as boundaries in 3D modelling processes.

## 3.6 Tetrahedral Mesh

E4D uses unstructured tetrahedral meshes for DCR and IP modelling. A tetrahedral mesh is a 3D mesh composed of tetrahedra, which are four-faced polyhedra where each face is a triangle. This structure allows the mesh to fill three-dimensional space effectively, making it suitable for modelling subsurface volumes.

Tetrahedral meshes offer significant geometric flexibility, allowing for the accurate representation of irregular surfaces and complex subsurface structures. This flexibility is crucial when incorporating complex topographies and heterogeneous subsurface conditions into the model. By using tetrahedra, the mesh can conform to the intricate details of the subsurface environment, ensuring that the simulations reflect real-world conditions as closely as possible.

Additionally, tetrahedral meshes enable targeted refinement around critical areas, such as around electrodes. This refinement improves numerical accuracy by increasing the mesh density where it is most needed, without significantly increasing the overall computational load. The result is a more efficient and accurate model, capable of handling the complexities of DCR and IP simulations.

## 3.7 Constrained Tetrahedralisation

In the process of generating a 3D mesh for subsurface modelling, accurately defining the upper surface is crucial. This surface is often irregular due to variations in topography or the placement of surface electrodes, which serve as control points in the mesh generation process.

To create this upper surface, E4D employs Triangle, a tool designed for generating high-quality triangulations. Triangle takes the control points, defined by the locations of surface electrodes or topographical data, and creates a triangulated surface. This triangulation ensures that the mesh closely conforms to the actual surface, capturing all significant features and variations. The resulting mesh is a collection of triangles, each defined by three vertices corresponding to the control points.

Once the upper surface has been triangulated, E4D uses this as a key component in constructing the 3D mesh. The vertical and lower surfaces, which are generally simpler and often defined as planes, are then joined with the upper surface. Together, these surfaces form the constraint for the tetrahedral mesh.

TetGen is then utilised to create the fully constrained tetrahedral mesh. It incorporates the triangulated upper surface from Triangle, along with the joined vertical and lower surfaces, as boundary conditions for the 3D mesh. This process, known as constrained tetrahedralisation, ensures that the triangles forming the upper surface are preserved as faces of the tetrahedra in the final mesh. This approach maintains the complexity and accuracy of the upper surface, leading to more reliable simulations of subsurface properties.

## 3.8 Inversion

Inversion can be understood as comprising two key components: forward modelling and the minimisation of an objective function. Forward modelling involves simulating how the earth's subsurface would respond to geophysical survey techniques, based on an initial set of model parameters. This simulated response is then compared to actual survey data, highlighting discrepancies between the model and reality. The process of minimisation of an objective function comes into play to adjust the model parameters iteratively, aiming to reduce the differences between the simulated and observed data. This minimisation effort seeks to find the set of model parameters that best fits the observed data, thereby enhancing the accuracy of the subsurface model. This iterative process continues until the model sufficiently conforms to the observed data, achieving a balance between fitting the data and maintaining a geologically plausible model structure.

## 3.9 Forward Modelling

E4D performs forward modelling by solving systems of equations for both real and imaginary components using the finite element method on an unstructured tetrahedral mesh, as detailed by [2] and [3] respectively.

The unstructured design of the finite element mesh allows for the extension of mesh boundaries far from the source region, effectively mitigating boundary effects without significantly increasing the number of mesh elements.

E4D uses the third-party mesh generator TetGen [4] for mesh creation and a Krylov subspace solver from PETSc [1] to solve the equations.

$$\mathbf{A}'\boldsymbol{\phi}' = I + \mathbf{A}''\boldsymbol{\phi}'' \tag{8}$$

$$\mathbf{A}'\boldsymbol{\phi}'' = -\mathbf{A}''\boldsymbol{\phi}' \tag{9}$$

$\mathbf{A}'$ and $\mathbf{A}''$ are sparse $n \times n$ finite element coupling matrices, constructed using the real and imaginary components of conductivity respectively, including appropriate boundary conditions, where $n$ is the number of mesh nodes.

$\boldsymbol{\phi}'$ and $\boldsymbol{\phi}''$ represent the $n$ element solution vectors holding the real and imaginary components of the potential field, respectively.

$I$ is an $n$ element vector holding the source current magnitude.

## 3.10 Objective Function

The goal of the inversion process is to refine the model by minimising an objective function that quantifies the discrepancy between the model's predictions and the observed data. The nonlinear conjugate gradient least squares (CGLS) algorithm is used to iteratively adjust the model to achieve this minimisation.

Relying solely on the data misfit function, which measures the difference between the model's predictions and the collected data, would lead to an excessive number of free parameters. This could result in a wide range of potential models with varying degrees of geological realism. To address this, it is crucial to impose additional constraints on the model. A smoothness constraint is typically applied to guide the model towards a more geologically plausible configuration by penalising excessive variations.

The objective function can be divided into two main components:

$$\Phi(\mathbf{m}^{\text{est}}) = \Phi_d(\mathbf{m}^{\text{est}}) + \beta\Phi_m(\mathbf{m}^{\text{est}}) \tag{10}$$

$\Phi_d$ represents the data misfit. This term quantifies the difference between the observed data and the predicted data generated by the model.

$\Phi_m$ represents the model norm, which imposes regularisation on the model parameters. This function is used to enforce desired structures in the recovered model, ensuring that the solution is both stable and geologically plausible. The value of $\Phi_m$ is determined by applying structural metrics and weighting functions, which define the specific structures to be imposed and the conditions under which they influence the inversion process.

$\beta$ is a tunable coefficient that controls the weighting of the model constraints, most commonly the smoothness of the model. E4D has the option to start with a high $\beta$ to prioritise a smooth model and then gradually decrease $\beta$ to allow for increased model complexity as data fitting improves.

$\mathbf{m}^{\text{est}}$ is a vector that consists of estimated conductivity values for each element in the computational mesh. The goal of the inversion is to find $\mathbf{m}^{\text{est}}$ which accurately reflects the earth of the survey area.

**Important Note:** In the E4D log file, $\beta\Phi_m$ and $\Phi_m$ are reported as $\Phi_m$ and $\Phi_m/\beta$ respectively.

## 3.11 Structural Metrics

A structural metric in E4D defines a relationship between the property (conductivity or phase) of a target element and one of the following:

- The property of the target element's neighbouring elements.
- The specified reference property value (see `ref_value` in the inverse options file syntax).
- The property of the corresponding element in the reference model file.

When a particular structural metric is assigned to a zone, E4D applies that metric to every element in that zone, and optionally to elements bounding that zone if specified in the corresponding constraint block. E4D then minimises the structural metric, thereby imposing the desired structure in the inverse solution.

The conditions under which E4D applies these constraints are determined by the weighting function associated with each structural metric. Therefore, the impact of each structural metric on constraining the inversion depends on the corresponding weighting function. The equations below describe each structural metric:

`struct_met = 1`: The difference between the log of the property of the target element ($m_t$) and the log of the property of its neighbouring element ($m_n$). Mathematically, $X = m_t - m_n$.

`struct_met = 2`: The absolute difference between the log of the property of the target element ($m_t$) and the log of the property of its neighbouring element ($m_n$). Mathematically, $X = |m_t - m_n|$.

`struct_met = 3`: The difference between the log of the property of the target element ($m_t$) and the log of the reference property value (`v_ref`). Mathematically, $X = m_t - \texttt{v\_ref}$, where `v_ref` is taken either from the constraint block or from the reference model file.

`struct_met = 4`: The absolute difference between the log of the property of the target element ($m_t$) and the log of the reference property value (`v_ref`). Mathematically, $X = |m_t - \texttt{v\_ref}|$.

`struct_met = 5`: Similar to `struct_met = 1`, but with anisotropic weighting applied. Mathematically, $X = m_t - m_n$, with anisotropic weighting in specific directions.

`struct_met = 6`: Similar to `struct_met = 2`, but with anisotropic weighting applied. Mathematically, $X = |m_t - m_n|$, with anisotropic weighting in specific directions.

`struct_met = 7`: Typically used for time-lapse inversion. It calculates the difference between the temporal change in the log of the property for spatially adjacent elements. Mathematically, $X = (m_t - \texttt{v\_reft}) - (m_n - \texttt{v\_refn})$, where $m_t$ and $m_n$ are the logs of the property of the target and neighbouring elements, respectively, and `v_reft` and `v_refn` are the reference values at a previous time.

`struct_met = 8`: Also used for time-lapse inversion, but it calculates the absolute difference between the temporal change in the log of the property for spatially adjacent elements. Mathematically, $X = |(m_t - \texttt{v\_reft}) - (m_n - \texttt{v\_refn})|$.

`struct_met = 9`: Similar to `struct_met = 1`, but applied only at specified boundaries. Mathematically, $X = m_t - m_n$.

`struct_met = 10`: Similar to `struct_met = 2`, but applied only at specified boundaries. Mathematically, $X = |m_t - m_n|$.

### 3.12 Weighting Functions

The primary purpose of the weighting function is to determine the conditions under which the corresponding structural metric should or should not be used to constrain the inversion. In essence, the weighting functions activate or deactivate the structural metrics and determine the degree of emphasis on minimising the structural metric during the transition between activation and deactivation.

The independent variable for each weighting function is the value of the structural metric, denoted as $X$. Each weighting function is based on the normal and cumulative normal distribution functions, and the output of these functions ranges between 0 and 1. If the weighting function evaluates to zero, the structural metric is inactive and does not influence the inversion. If it evaluates to one, the structural metric is fully active.

**Weighting Function 1:** Minimize $X$ if $X$ drops below mn $+ 2$sd, reaching full weight if $X$ drops below mn $- 2$sd.

$$f(X) = 0.5 \times \left(1 - \text{erf}\left(\frac{X - \text{mn}}{\sqrt{2} \times \text{sd}}\right)\right) \tag{11}$$

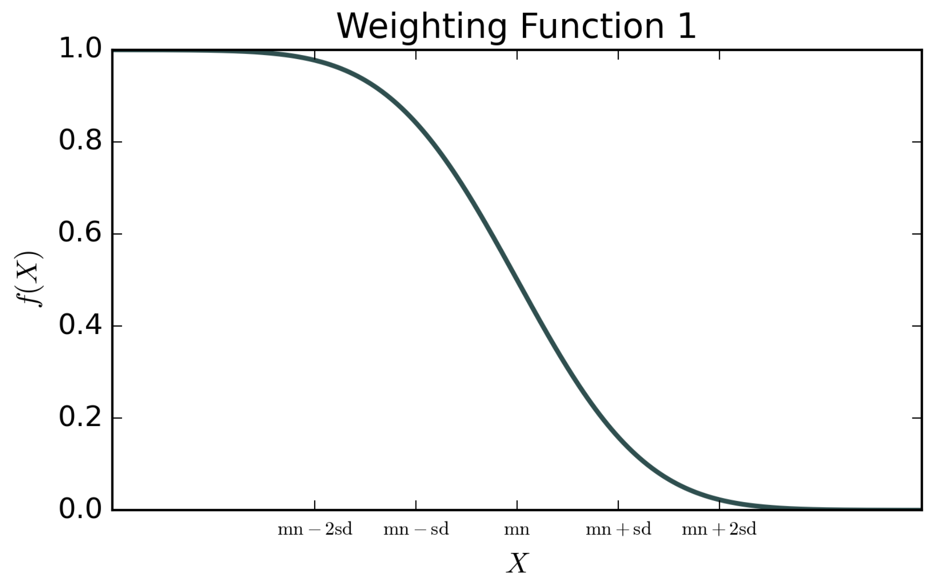

Figure 1: Weighting Function 1: $f(X)$ decreases as $X$ drops below mn.

**Weighting Function 2:** Minimize $X$ if $X$ rises above mn $-$ $2$sd, reaching full weight if $X$ rises above mn $+$ $2$sd.

$$f(X) = 0.5 \times \left(1 + \text{erf}\left(\frac{X - \text{mn}}{\sqrt{2} \times \text{sd}}\right)\right) \tag{12}$$

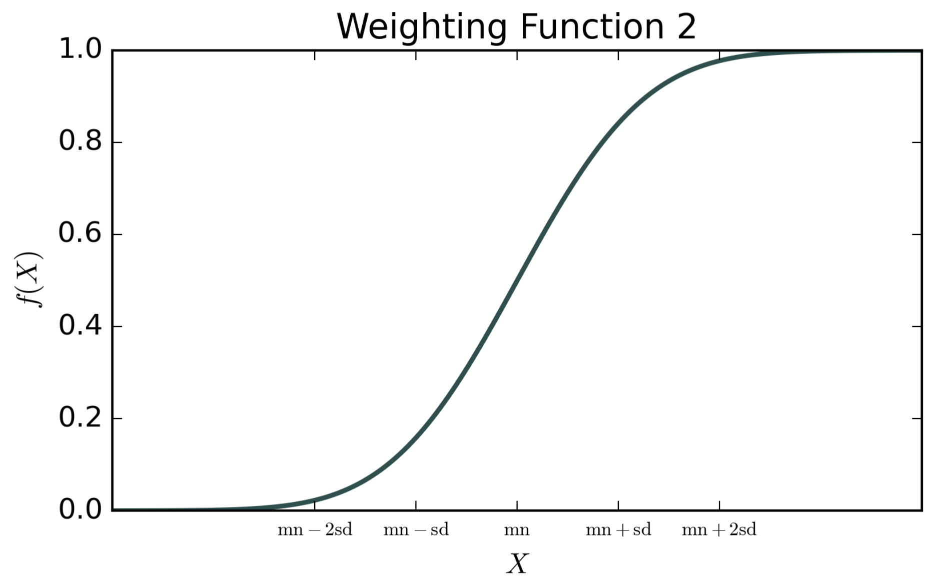

Figure 2: Weighting Function 2: $f(X)$ increases as $X$ rises above mn.

**Weighting Function 3:** Minimize $X$ if $X$ deviates from mn, reaching full weight if $X$ deviates from mn by more than approximately $2$sd.

$$f(X) = 1 - \exp\left(-\frac{(X - \text{mn})^2}{2 \times \text{sd}^2}\right) \tag{13}$$

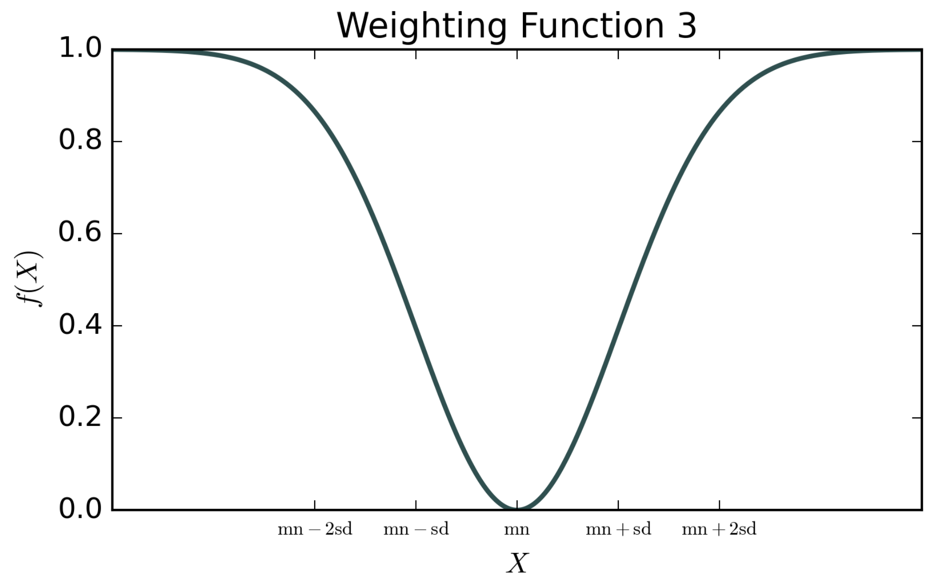

Figure 3: Weighting Function 3: $f(X)$ increases as $X$ deviates from mn.

**Weighting Function 4:** Minimize $X$ as $X$ approaches mn, reaching full weight when $X$ equals mn.

$$f(X) = \exp\left(-\frac{(X - \text{mn})^2}{2 \times \text{sd}^2}\right) \tag{14}$$

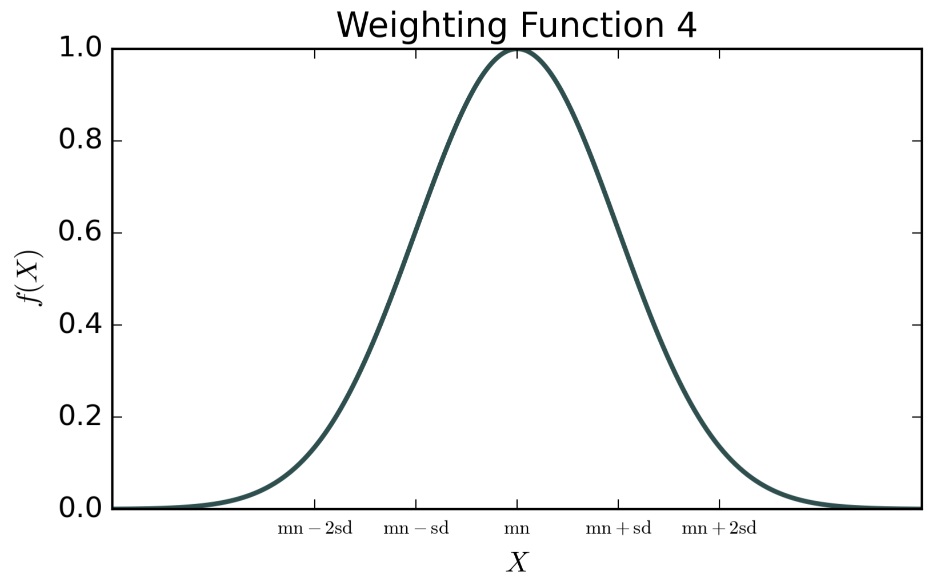

Figure 4: Weighting Function 4: $f(X)$ decreases as $X$ approaches mn.

The variables used in these equations are as follows:

- $f(X)$: The value of the weighting function, where $0 \leq f(X) \leq 1$.
- $X$: The value of the structural metric.
- erf: The error function.
- exp: The exponential function, where $e = 2.718281828$.
- mn: The mean of the weighting function, as specified in the constraint block in the inversion options file.
- sd: The standard deviation of the weighting function, as specified in the constraint block in the inversion options file.

## 4 Types of Constraints

### 4.1 Smoothness Constraint

- **Structural Metric**: Metric 2
  - Encourages smooth variations between adjacent elements.
- **Weighting Function**: Function 1
  - **Mean**: Typically set to a high value (e.g., 10).
  - **Standard Deviation**: Small (e.g., 0.01).
  - Ensures that the smoothness constraint is applied only when the difference between adjacent elements is not extreme.

### 4.2 Minimum Conductivity Constraint

- **Structural Metric**: Metric 3
  - Enforces a lower bound on conductivity values.
- **Weighting Function**: Function 1
  - **Mean**: Typically set to 0.
  - **Standard Deviation**: Small (e.g., 0.1).
  - **Reference Value**: Defines the minimum allowable conductivity (e.g., 0.002 S/m). When the log conductivity drops below this value, the constraint activates to ensure that conductivity does not fall below the specified minimum.

### 4.3 Maximum Conductivity Constraint

- **Structural Metric**: Metric 3
  - Enforces an upper bound on conductivity values.
- **Weighting Function**: Function 2
  - **Mean**: Typically set to 0.

    - **Standard Deviation**: Small (e.g., 0.1).
    - **Reference Value**: Defines the maximum allowable conductivity (e.g., 0.01 S/m). When the log conductivity exceeds this value, the constraint activates to ensure that conductivity does not surpass the specified maximum.

### 4.4 Blocky Constraint

- **Structural Metric**: Metric 2
    - Enforces blocky variations between adjacent elements.
- **Weighting Function**: Function 1
    - **Mean**: Lower than in smoothness constraints (e.g., 0.1).
    - **Standard Deviation**: Small relative to the mean (e.g., 0.05).
    - Enforces similarity constraints only if the difference between adjacent elements is relatively small, making the model more blocky rather than smooth.

### 4.5 Anisotropic Weighting and Inequality Constraints

- **Structural Metric**: Metric 6
    - Encourages continuity normal to a specific plane.
- **Weighting Function**: Function 1
    - **Mean**: Set to a high value (e.g., 5.0).
    - **Standard Deviation**: Small (e.g., 0.02).
    - Applies constraints in an anisotropic manner based on the orientation of the plane.
- **Inequality Constraints**:
    - **Lower Bound**: Metric 3 with Weighting Function 1
        - **Reference Value**: Defines the minimum bound (e.g., 0.001 S/m).
    - **Upper Bound**: Metric 3 with Weighting Function 2
        - **Reference Value**: Defines the maximum bound (e.g., 0.02 S/m).

## 5 User Defined Input File Formats

This section includes information about the files that the user will need to manually define. It does not cover files generated automatically by E4D, Triangle, or TetGen.

E4D uses a single input file named `e4d.inp` that can contain multiple runs executed sequentially. This approach allows users to chain together different operations such as mesh generation, forward modelling, and inversion in a single workflow.

### 5.1 General `inp` File Format

**Important Note:** Must have the `e4d.inp` file name.

```
Total number of runs: N

! Comments can be added using ! # $ = symbols

! Run 1 - Example: Mesh Generation
====================================================================
Run number: 1
Run mode: SIP1
[Run settings for mesh generation - see Mesh Generation section]
====================================================================

! Run 2 - Example: Forward Modelling
====================================================================
Run number: 2
Run mode: SIP2
[Run settings for forward modelling - see Forward Modelling section]
====================================================================

! Run 3 - Example: Inversion
====================================================================
Run number: 3
Run mode: SIP3
[Run settings for inversion - see Inversion section]
====================================================================
```

`N` (int)
    The total number of runs to execute sequentially.

`Run number` (int)
    Sequential identifier for each run block.

`Run mode` (char)
    Specifies the operation type:

- `SIP1/DCR1`: Mesh generation

  - `SIP2/DCR2`: Forward modelling
  - `SIP3/DCR3`: Inversion

`========`
: Separators used to clearly delineate different run blocks (optional but recommended for readability).

**Key Features**

- **Sequential Execution**: Runs are executed in the order specified, allowing output from one run to be used as input for the next.
- **Comments**: Lines beginning with `!`, `#`, `$`, or = are treated as comments.
- **Integrated Settings**: Inversion parameters are embedded directly in the input file, eliminating the need for separate `.inv` files.
- **Flexible Workflows**: Users can combine any sequence of operations in a single file.

## 5.2 Mesh Generation

### 5.2.1 `inp` File Format

**Important Note:** Must have the `e4d.inp` file name. For mesh generation, only the run block shown below is required.

```
    Total number of runs: 1

    Run number: 1
    Run mode: run_mode
    Config file name: cfg_filename
```

`run_mode` (char)
Specifies the mesh generation mode:

- `run_mode = DCR1`: Mesh generation for DCR surveys.
- `run_mode = SIP1`: Mesh generation for IP surveys.

`cfg_filename` (char)
The filename of the `cfg` file. Must have the `.cfg` file extension.

### 5.2.2 `cfg` File Format

**Important Note:** Must have the `.cfg` file extension.

```
mesh_qual mesh_vol_max
mesh_z_min
tetgen_flag
tetgen_path
triangle_path

np
1 x[1] y[1] z[1] flag[1]
2 x[2] y[2] z[2] flag[2]
...
np x[np] y[np] z[np] flag[np]

nf
nv[1] boundary_id[1]
v[1][1] v[1][2] ... v[1][nv[1]]
nv[2] boundary_id[2]
v[2][1] v[2][2] ... v[2][nv[2]]
...
nv[nf] boundary_id[nf]
v[nf][1] v[nf][2] ... v[nf][nv[nf]]

nh
1 xh[1] yh[1] zh[1]
2 xh[2] yh[2] zh[2]
...
nh xh[nh] yh[nh] zh[nh]

nz
1 xz[1] yz[1] zz[1] vol_max[1] sig[1] isig[1]
2 xz[2] yz[2] zz[2] vol_max[2] sig[2] isig[2]
...
nz xz[nz] yz[nz] zz[nz] vol_max[nz] sig[nz] isig[nz]

vis_flag
vis_loc
translate_flag
```

`mesh_qual` (real)
: The maximum radius-to-edge ratio of any element in the mesh. The radius refers to the radius of the sphere that passes through all vertices of a given tetrahedron. Recommended values are in the range of 1.2 to 1.5, with 1.2 specifying a higher quality mesh.

`mesh_vol_max` (real)
: The maximum volume constraint for all elements in the mesh. For elements with a

radius-to-edge ratio close to 1, the edge length as a function of element volume can be approximated as $L \approx 2V^{\frac{1}{3}}$.

`mesh_z_min` (real)
: The elevation of the bottom of the computational mesh, i.e., the lower boundary. The bottom of the mesh should be set so that all defined regions exist within the mesh and far enough below the surface to allow for the potential of any current source to vanish.

`tetgen_flag` (int)
: Determines whether TetGen is called to build the mesh:

  - `tetgen_flag = 0`: Builds the `.poly` input file for TetGen, but does not call TetGen to build the mesh.
  - `tetgen_flag = 1`: Calls TetGen to build the mesh.

`tetgen_path` (char)
: Specifies the path to the TetGen executable and should be enclosed in single quotes.

`triangle_path` (char)
: Specifies the path to the Triangle executable and should be enclosed in single quotes.

`np` (int)
: The total number of defined control points.

`x[i]` (real)
: The x coordinate of control point `i`.

`y[i]` (real)
: The y coordinate of control point `i`.

`z[i]` (real)
: The z coordinate of control point `i`.

`flag[i]` (int)
: The flag of control point `i`:

  - `flag[i] = 2`: External boundary control points are used to define the outer boundary of the mesh and are given a flag of 2. E4D uses these nodes to construct the outer vertical boundaries of the computational mesh by connecting adjacent points and constructing a series of vertical planes down to the elevation `mesh_z_min`.
  - `flag[i] = 1`: Surface control points are used to specify points of known elevation and/or surface electrode locations on the mesh and are given a flag of 1. E4D, Triangle, and TetGen use these points to define surface topography and add nodes at surface electrode locations. Surface points cannot be placed on or beyond the outer boundary of the mesh, which is specified by a set of surface boundary points.
  - `flag[i] = 0`: Internal control points are used to define internal boundaries, buried electrode locations, and mesh refinement points, and are given a flag of 0 unless they are used to define a metallic boundary. All internal control points must be placed within the mesh boundaries formed by the surface boundary, vertical external boundaries, and lower boundary.

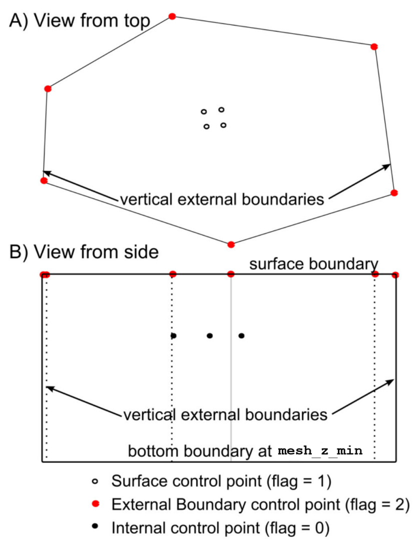
A) View from top
vertical external boundaries
B) View from side
surface boundary
vertical external boundaries
bottom boundary at mesh_z_min
Surface control point (flag = 1)
External Boundary control point (flag = 2)
Internal control point (flag = 0)

Figure 5: A diagram showing the three types of control points used to define a mesh.

`nf` (int)
: The number of internal facets. Facets are polygons used to define zone boundaries. For example, if the user intends to have a rectangular prism-shaped zone, they must define each rectangle that constitutes a face of the prism. A zone must have a closed boundary; thus, any point within the zone must be fully enclosed by facets defined by the user.

The exception to this rule is if a facet lies entirely on the surface, meaning all its vertices have `flag[i] = 1`. In this case, E4D automatically generates the facet. This is done to avoid excessive complexity when using E4D with complicated surface topographies.

For a rectangular prism-shaped zone exposed to the surface, the user must define the four vertical boundaries and the lower boundary. If the zone lies completely beneath the surface, the user must also define the upper boundary.

If two zones share a common facet, it only needs to be defined and counted once.

`nv[i]` (int)
: The total number of vertices defining facet `i`, e.g., `nv[i] = 4` for a rectangular facet.

`boundary_id[i]` (int)
: The boundary ID identifies which boundary facet `i` belongs to. It is required that `boundary_id[i] > 2` as negative numbers, 0, 1, and 2 are reserved in E4D.

`v[i][j]` (int)
: The control point number corresponding to vertex `j` in facet `i`.

- Control points must be specified to define a facet such that no segment connecting two consecutive control points crosses any other segment connecting two consecutive control points (i.e., the list of control points should be either clockwise or counter-clockwise). E4D does not check this condition before executing TetGen, and TetGen will fail if this condition is violated.
- Each of the points specified must lie in the same plane according to numerical precision, or TetGen will fail.

- Do not define facets on any external mesh boundary. E4D will construct these automatically.
- The segments of any facet that intersect the surface boundary must be explicitly specified in the plane definition. In other words, any plane segment that intersects the surface boundary must not be defined by the first and last points specified in the facet definition (i.e., the points defining segments on the surface must be listed sequentially in the facet definition).
- It is often useful to specify a series of facets that form a closed boundary (thereby forming a zone) whose upper boundary is the surface of the mesh, which is automatically generated by E4D. In this case, E4D must correctly connect the segments of facets intersecting the surface such that those segments define the upper boundary of the zone. To do so, E4D takes the list of control points defining the segments that intersect the surface and connects each point to the next nearest point. With this in mind, it is important to define the facets that intersect the surface in such a manner that E4D correctly connects the surface segments. Attention to this issue is only required for boundaries whose surface intersection forms convex angles.
- In the case where the upper boundary of a zone is the surface of the mesh, do not manually specify the facets that make up the upper boundary i.e. do not define any facets where all the control points have `flag[i] = 1`.

`nh` (int)
: The total number of user-defined holes in the mesh.

`xh[i]` (real)
: The x coordinate of any point within hole `i`.

`yh[i]` (real)
: The y coordinate of any point within hole `i`.

`zh[i]` (real)
: The z coordinate of any point within hole `i`.

`nz` (int)
: The total number of user-defined zones in the mesh.

`xz[i]` (real)
: The x coordinate of any point within zone `i`.

`yz[i]` (real)
: The y coordinate of any point within zone `i`.

`zz[i]` (real)
: The z coordinate of any point within zone `i`.

`vol_max[i]` (real)
: The maximum volume constraint for all elements within zone `i`.

`sig[i]` (real)
: The real component of the conductivity of zone `i` in S/m.

`isig[i]` (real)
: The imaginary component of the conductivity of zone `i` in S/m. Not required for `run_mode = DCR1`.

`vis_flag` (int)
: Rework needed, set to 0 for the time being.

`vis_loc` (int)
: Rework needed, set to ‘none’ for the time being.

`translate_flag` (int)
: Set to 1 to enable E4D automatic translation of the mesh or 0 to disable it. It is recommended to enable translation to minimize potential precision errors. A .trn file will be generated with the corresponding translation values for x, y and z.

## 5.3 Forward Modelling

### 5.3.1 `inp` File Format

**Important Note:** Must have the `e4d.inp` file name. For forward modelling, only the run block shown below is required.

```
Total number of runs: 1

Run number: 1
Run mode: run_mode
Mesh file name: mesh_prefix.1.node
Survey file name: srv_filename
Initial model: sig_filename
Output file name: out_filename
```

`run_mode` (char)
Specifies the forward modelling mode:

- `run_mode = DCR2`: Forward modelling for DCR surveys.
- `run_mode = SIP2`: Forward modelling for IP surveys.

`mesh_prefix.1.node` (char)
The mesh file name from the mesh generation process. These files must be generated beforehand by using the `DCR1` or `SIP1` run modes.

The following files must be present for forward modelling:

- `mesh_prefix.1.node`
- `mesh_prefix.1.ele`
- `mesh_prefix.1.edge`
- `mesh_prefix.1.face`
- `mesh_prefix.1.neigh`
- `mesh_prefix.sig`
- `mesh_prefix.trn`

`srv_filename` (char)
The filename of the `srv` file. Must have the `.srv` file extension.

`sig_filename` (char)
The filename of the `sig` file containing the conductivity model. Must have the `.sig` file extension.

`out_filename` (char)
The filename of the `out` file. Must have the `.out` file extension.

### 5.3.2 `srv` File Format

**Important Note:** Must have the `.srv` file extension.

```
ne
1 x[1] y[1] z[1] flag[1]
2 x[2] y[2] z[2] flag[2]
...
ne x[ne] y[ne] z[ne] flag[ne]

nm

1 c1[1] c2[1] p1[1] p2[1] dcr_obs[1] dcr_sd[1] ip_obs[1] ip_sd[1]
2 c1[2] c2[2] p1[2] p2[2] dcr_obs[2] dcr_sd[2] ip_obs[2] ip_sd[2]
...
nm c1[nm] c2[nm] p1[nm] p2[nm] dcr_obs[nm] dcr_sd[nm] ip_obs[nm] ip_sd[nm]
```

`ne` (int)
The total number of unique electrodes in the survey.

`x[i]` (real)
The x coordinate of electrode `i`.

`y[i]` (real)
The y coordinate of electrode `i`.

`z[i]` (real)
The z coordinate of electrode `i`.

`flag[i]` (int)
The flag of electrode `i`:

- `flag[i] = 1`: Surface electrode.
- `flag[i] = 0`: Sub-surface electrode.

`nm` (int)
The total number of survey measurements.

`c1[i]` (int)
The electrode index of the positive current electrode in measurement `i`.

`c2[i]` (int)
The electrode index of the negative current electrode in measurement `i`.

`p1[i]` (int)
The electrode index of the positive potential electrode in measurement `i`.

`p2[i]` (int)
The electrode index of the negative potential electrode in measurement `i`.

`dcr_obs[i]` (real)
: The observed transfer resistance of measurement `i` in ohms. For forward modelling, use an arbitrary placeholder value.

`dcr_sd[i]` (real)
: The transfer resistance standard deviation of measurement `i` in ohms. For forward modelling, use an arbitrary placeholder value.

`ip_obs[i]` (real)
: The observed phase of measurement `i` in radians. For forward modelling, use an arbitrary placeholder value. Not required for `run_mode = DCR2`.

`ip_sd[i]` (real)
: The phase standard deviation of measurement `i` in radians. For forward modelling, use an arbitrary placeholder value. Not required for `run_mode = DCR2`.

### 5.3.3 `sig` File Format

**Important Note:** A `sig` file is created during the mesh generation stage based on the defined zone properties. Often, it is adequate to use this, although the user may choose to define their own.

```
nel
1 sigma[1] isigma[1]
2 sigma[2] isigma[2]
...
nel sigma[nel] isigma[nel]
```

`nel` (int)
: The total number of elements in the mesh.

`sigma[i]` (real)
: The real conductivity of element `i` in S/m.

`isigma[i]` (real)
: The imaginary conductivity of element `i` in S/m. Not required for `run_mode = DCR2`.

### 5.3.4 `out` File Format

**Important Note:** Must have the `.out` file extension.

```
residual_flag
residual_prefix
npot
pot[1]
pot[2]
...
pot[npot]
```

`residual_flag` (int)
: Determines whether or not to output a residual data file for the given model. For forward modelling, the observed data will be the placeholder values specified in the `srv` file and the predicted values will be the forward response.

- `residual_flag = 0`: Does not output a residual data file.
- `residual_flag = 1`: Outputs a residual data file.

`residual_prefix` (char)
: The prefix of the residual data file. For forward modelling, the filename will be `residual_prefix.tx`

`npot` (int)
: The number of potential distribution files to output.

`pot[i]` (int)
: The index of measurement `i` from the `srv` file for which to output the potential distribution.

## 5.4 Inversion

**Important Changes:** Separate `.inv` files are no longer required. All inversion settings are now specified directly in the `e4d.inp` file as shown below.

### 5.4.1 `inp` File Format

**Important Note:** Must have the `e4d.inp` file name. For inversion, only the run block shown below is required.

```
Total number of runs:                     1

Run number:                               1
Run mode:                                 run_mode
Mesh file name:                           mesh_prefix.1.node
Survey file name:                         srv_filename
Initial model:                            initial_model
Output file name:                         out_filename
Reference model file name:                reference_model

! DC inversion Settings
Inversion settings:                       DC
Total number of zones:                    nc_dc

Zone number:                              zone[1]
Structural metric:                        struct_met[1]
Relative weighting of structural
   metric (x,y,z):                        wx[1] wy[1] wz[1]
Weighting function:                       f[1]
Mean and sd of weighting function:        mn[1] sd[1]
Links:                                    nl[1] link[1][1] ...
Reference value:                          ref_value[1]
Relative weight:                          ref_weight[1]

!  ...  additional zones

Initial beta:                             initial_beta
Minimum fractional decrease in beta:      min_decrease
Beta reduction proportion:                beta_reduction
Chi2 target value:                        chi2_target
Moving average window size:               chi2_hist_size
Minimum change in moving average:         chi2_conv_target
Inner iteration bounds:                   min_iter max_iter
Conductivity constraints:                 min_sig max_sig
Beta mode:                                beta_mode
Outlier removal mode:                     outlier_mode
Maximum outlier sd:                       outlier_sd

! IP inversion settings (if applicable)
Inversion settings:                       IP
Total number of zones:                    nc_ip
!  ...  (similar structure for IP
    settings with phase constraints)
```

`N` (int)
: The total number of runs to execute. E4D can now handle multiple sequential runs in a single input file.

`run_mode` (char)
: Specifies the mode that E4D should run in for each run.

- `run_mode = SIP3`: IP inversion.
- `run_mode = DCR3`: DCR inversion.

`config_filename` (char)
: The configuration file name for mesh generation (used with SIP1/DCR1 modes).

`mesh_prefix.1.node` (char)
: The mesh file name from the mesh generation process. These files must be generated beforehand by using the `SIP1` or `DCR1` run modes.

The following files must be present for inversion:

- `mesh_prefix.1.node`
- `mesh_prefix.1.ele`
- `mesh_prefix.1.edge`
- `mesh_prefix.1.face`
- `mesh_prefix.1.neigh`
- `mesh_prefix.sig`
- `mesh_prefix.trn`

`srv_filename` (char)
: The filename of the `srv` file. Must have the `.srv` file extension.

`initial_model` (char)

- `initial_model = average`: Sets the initial model to use the average apparent conductivity and apparent phase.
- `initial_model = median`: Sets the initial model to use the median apparent conductivity and apparent phase.
- `initial_model = sig_file_name`: Alternatively, a `sig_filename` can be specified.

`out_filename` (char)
: The filename of the `out` file. Must have the `.out` file extension.

`reference_model` (char)

- `reference_model = none`: Set to none unless the user requires inversion constraints based on a reference model.
- `reference_model = sig_file_name`: Alternatively, a `sig_filename` can be specified.

`nc_dc`, `nc_ip` (int)
: The total number of constraint zones for DC and IP inversions respectively. Each zone in the mesh should have at least one constraint assignment.

`zone[i]` (int)
: The zone number for which this constraint applies.

`struct_met[i]` (int)
: The structural metric to use for constraint `i`. For more details, see section 3.11.

`wx[i], wy[i], wz[i]` (real)
: The relative weighting of this structural metric in the x, y, and z directions respectively. These may not be used by every structural metric but must be present.

`f[i]` (int)
: The weighting function to use for constraint `i`. For more details, see section 3.12.

`mn[i]` (real)
: The mean of the weighting function in natural log space.

`sd[i]` (real)
: The standard deviation of the weighting function in natural log space.

`nl[i]` (int)
: The number of linked zones for this constraint.

`link[i][j]` (int)
: The zone numbers that are linked to zone `i`. Use 0 for no links.

`ref_value[i]` (real)
: The reference value used for this constraint. Typically set to 1.0.

`ref_weight[i]` (real)
: The relative weight applied to the regularisation constraints in this zone.

`initial_beta` (real)
: The global constraint weighting value at the beginning of the inversion.

`min_decrease` (real)
: The minimum fractional decrease in the objective function between outer iterations before beta is reduced.

`beta_reduction` (real)
: The factor by which beta is reduced when convergence criteria are met.

`chi2_target` (real)
: The normalised $\chi^2$ value at which the inversion is considered converged.

`chi2_hist_size` (int)
: The number of previous $\chi^2$ values to consider for moving average convergence.

`chi2_conv_target` (real)
: The target change in moving average of $\chi^2$ that indicates convergence.

`min_iter, max_iter` (int)
: The minimum and maximum number of inner CGLS iterations.

`min_sig, max_sig` (real)
: The minimum and maximum conductivity values allowed (safety constraints).

`min_phase`, `max_phase` (real)
The minimum and maximum phase values allowed for IP inversion (safety constraints).

`beta_mode` (int)

- `beta_mode = 2`: Beta reduces as the inversion progresses.
- `beta_mode = 3`: Beta remains constant throughout the inversion.

`outlier_mode` (int)

- `outlier_mode = 0`: No outlier removal.
- `outlier_mode = 1`: Remove outlier data from current inversion iteration.

`outlier_sd` (real)
The outlier removal standard deviation threshold.

### 5.4.2 `srv` File Format

**Important Note:** Must have the `.srv` file extension.

```
ne
1 x[1] y[1] z[1] flag[1]
2 x[2] y[2] z[2] flag[2]
...
ne x[ne] y[ne] z[ne] flag[ne]

nm

1 c1[1] c2[1] p1[1] p2[1] dcr_obs[1] dcr_sd[1] ip_obs[1] ip_sd[1]
2 c1[2] c2[2] p1[2] p2[2] dcr_obs[2] dcr_sd[2] ip_obs[2] ip_sd[2]
...
nm c1[nm] c2[nm] p1[nm] p2[nm] dcr_obs[nm] dcr_sd[nm] ip_obs[nm] ip_sd[nm]
```

`ne` (int)
: The total number of unique electrodes in the survey.

`x[i]` (real)
: The x coordinate of electrode `i`.

`y[i]` (real)
: The y coordinate of electrode `i`.

`z[i]` (real)
: The z coordinate of electrode `i`.

`flag[i]` (int)
: The flag of electrode `i`:

  - `flag[i] = 1`: Surface electrode.
  - `flag[i] = 0`: Sub-surface electrode.

`nm` (int)
: The total number of survey measurements.

`c1[i]` (int)
: The electrode index of the positive current electrode in measurement `i`.

`c2[i]` (int)
: The electrode index of the negative current electrode in measurement `i`.

`p1[i]` (int)
: The electrode index of the positive potential electrode in measurement `i`.

`p2[i]` (int)
: The electrode index of the negative potential electrode in measurement `i`.

`dcr_obs[i]` (real)
: The observed transfer resistance of measurement `i` in ohms.

`dcr_sd[i]` (real)
: The transfer resistance standard deviation of measurement `i` in ohms.

`ip_obs[i]` (real)
: The observed phase of measurement `i` in radians. Not required for `run_mode = DCR3`.

`ip_sd[i]` (real)
: The phase standard deviation of measurement `i` in radians. Not required for `run_mode = DCR3`.

### 5.4.3 `out` File Format

**Important Note:** Must have the `.out` file extension. Defaults used if omitted.

```
residual_flag
residual_prefix
npot
pot[1]
pot[2]
...
pot[npot]
```

`residual_flag` (int)
Determines whether or not to output a residual data file for the given model.

- `residual_flag = 0`: Does not output a residual data file.
- `residual_flag = 1`: (default) Outputs a residual data file.

`residual_prefix` (char)
The prefix of the residual data file. For forward modelling, the filename will be `residual_prefix.txt`.

`npot` (int)
(Default `0`) The number of potential distribution files to output.

`pot[i]` (int)
The index of measurement `i` from the `srv` file for which to output the potential distribution.

### 5.4.4 Obsolete `inv` File Format

**Important Note:** The separate `.inv` files are no longer required. All inversion settings are now specified directly in the `e4d.inp` file as shown in the previous section.

# 6 Miscellaneous File Formats

## 6.1 `node/ele/edge/face/neigh` File Format

These files are created by TetGen during the `DCR1` or `SIP1` run modes. They define the mesh and are required for both forward modelling and inversion. They're saved in the mesh subdirectory.

If `cfg_filename = mesh_prefix.cfg`, then the following files are generated:

- `mesh_prefix.1.node`
- `mesh_prefix.1.ele`
- `mesh_prefix.1.edge`
- `mesh_prefix.1.face`
- `mesh_prefix.1.neigh`

A detailed description of each mesh file format is available in the TetGen manual [4].

## 6.2 `trn` File Format

**Important Note:** A `trn` file is created during mesh generation and contains timestamped coordinate translations used internally by E4D. Each run appends a new entry to this file rather than overwriting it. E4D uses the last (most recent) entry in the file.

```
timestamp              delta_x delta_y delta_z
YYYY-MM-DD HH:MM:SS    delta_x delta_y delta_z
YYYY-MM-DD HH:MM:SS    delta_x delta_y delta_z
...                    ...
```

`timestamp` (char)
: The date and time when the mesh generation or processing run was executed, in the format `YYYY-MM-DD HH:MM:SS`.

`delta_x` (real)
: The translation of the mesh in the x-direction in meters.

`delta_y` (real)
: The translation of the mesh in the y-direction in meters.

`delta_z` (real)
: The translation of the mesh in the z-direction in meters.

## 7 IP Inversion Tutorial

**Important Note:** The required input files for this tutorial are located in: `/tutorial/two_blocks/`

This section provides a step-by-step tutorial for building a synthetic model, calculating the forward response, and inverting the data with smoothness constraints to attempt the recovery of the original model. Through this tutorial, users will gain practical experience with the process of forward modelling and inversion using E4D.

The synthetic model consists of a flat surface with two buried blocks, each measuring 200 m × 200 m × 200 m. The model parameters are as follows:

- **Background:**
  - Resistivity: 500 ohm-m
  - Phase: 1 mrad
- **Block 1:**
  - Resistivity: 5 ohm-m
  - Phase: 100 mrad
- **Block 2:**
  - Resistivity: 5000 ohm-m
  - Phase: 10 mrad

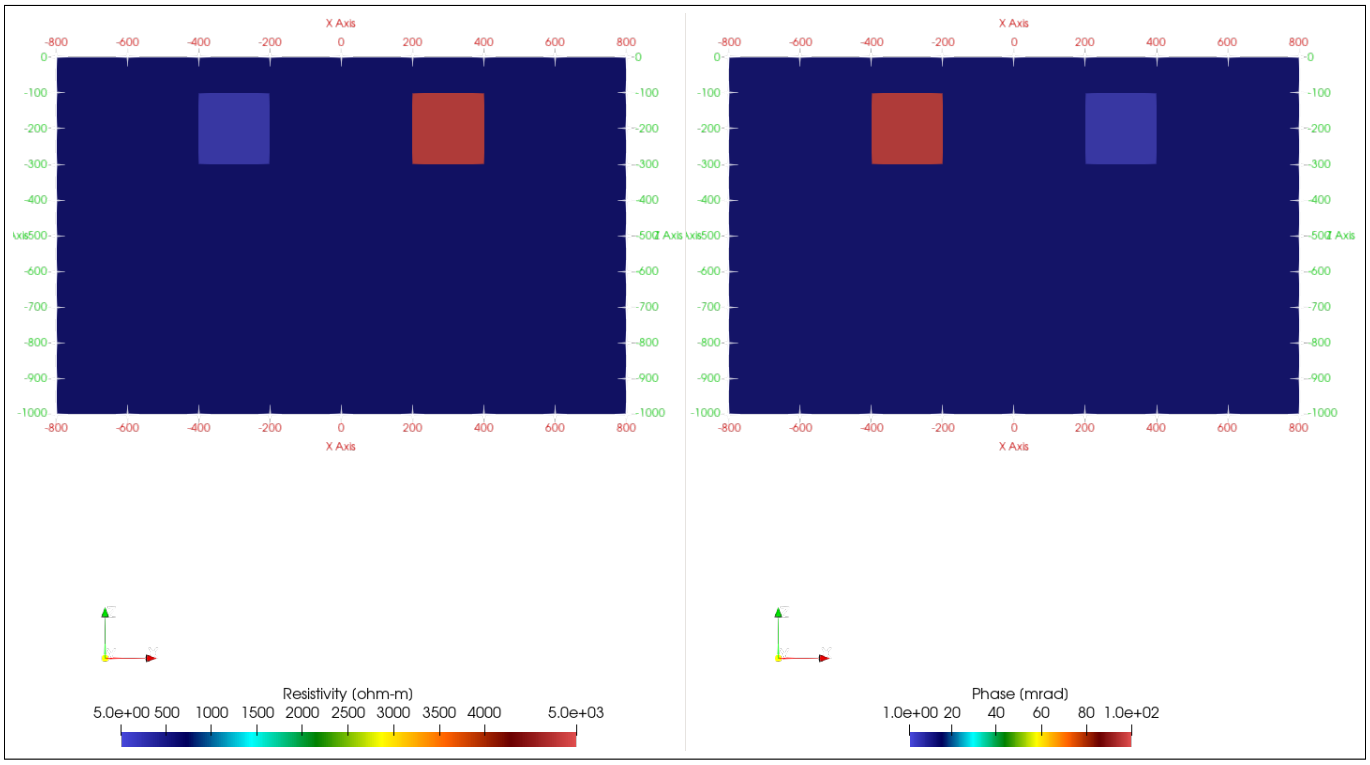

Figure 6: Synthetic model section view.

## 7.1 Preliminary Setup

It is strongly recommended to compile E4D, TetGen, and Triangle using the Intel oneAPI compilers (see Section **??** for detailed instructions). To enable support for oneAPI, it is necessary to configure the environment variables accordingly.

Begin by opening a terminal in the intended working directory and sourcing the oneAPI environment setup script:

```
source /opt/intel/oneapi/setvars.sh
```

Next, configure the MPI communication fabric by setting the following environment variable:

```
export I_MPI_FABRICS=shm
```

Finally, add the directory containing the required executables to your system's PATH:

```
export PATH="$PATH:../E4D-HR/bin"
```

## 7.2 Forward SIP1

### 7.2.1 Input Files

In the `/two_blocks/forward_SIP1/` directory, you will find two key files:

- `e4d.inp`
- `two_blocks.cfg`

The `e4d.inp` file instructs E4D to run in the specified mode, which in this case is SIP mesh generation, or `SIP1`. The `two_blocks.cfg` file contains the necessary information to build the mesh, which we will examine closely in this section.

Examining `two_blocks.cfg` line by line, we observe the following standard options:

- `mesh_qual = 1.3`: Ensures that highly irregularly shaped tetrahedral elements are avoided, as these are commonly associated with poor numerical accuracy.
- `mesh_vol_max = 1e18`: While this value may seem excessively large, it is acceptable since small element sizes are not required far away from potential sources.
- `tetgen_flag = 1`: Instructs TetGen to build the mesh.
- `tetgen_path = tetgen`: Specifies the path to the TetGen executable, assuming it has been added to the system's PATH environment variable as described in Section 7.1.
- `triangle_path = triangle`: Specifies the path to the Triangle executable, assuming it has been added to the system's PATH environment variable as described in Section 7.1.

Moving on to the control point definitions, there are 124 points defined in the following order:

- The 4 surface boundary points.
- The 8 vertices of zone 1, which is a 1600 m × 1200 m × 2000 m rectangular prism.

- The 8 vertices of zone 2, which is a 200 m × 200 m × 200 m cube.
- The 8 vertices of zone 3, which is also a 200 m × 200 m × 200 m cube.
- 48 electrode locations.
- 48 electrode refinement points, located 5 m below each electrode to enforce smaller element sizes, thereby improving numerical accuracy during modelling.

Next, we define 17 facets in the `two_blocks.cfg` file. The first 5 facets correspond to the 4 vertical faces and the lower face of the boundary that encloses zone 1. It's important to note that the upper face is not specified because all vertices of the upper face have `flag = 1`, allowing E4D to automatically handle this case.

The following 6 facets correspond to the 6 faces of the first block (zone 2), listed in the following order: upper, vertical, and lower. In this instance, all 6 faces are specified because the block is entirely below the surface.

The final 6 facets correspond to the 6 faces of the second block (zone 3), listed in the same order: upper, vertical, and lower.

Continuing on, we observe that there are no defined holes in this mesh, which is typical for most configurations.

Next, we define 4 zones within the mesh: the background, the first block, the second block, and the remainder of the mesh.

Each zone is defined by specifying a point that lies within it. We also specify the maximum element size for each zone. Notably, zone 4 (the remainder of the mesh) has a larger maximum element size because elements in this zone are located far from potential sources, and therefore, do not require as much refinement. Additionally, we define the real and imaginary conductivity for each zone, which will be used in the modelling process.

Finally, we set the following parameters:

- `vis_flag = 0`: Disables visualisation output.
- `vis_loc = none`: Specifies no visualisation location.
- `translate_flag = 0`: Disables any translation of the mesh.

Understanding the concept of zones in E4D is crucial, especially when it comes to defining facets. It is recommended to take the time to thoroughly examine the `two_blocks.cfg` file.

### 7.2.2 Generating the Mesh with the SIP1 mode

To begin the mesh generation process, ensure that your working directory contains the following files:

- `e4d.inp`
- `two_blocks.cfg`

Verify that all steps outlined in Section 7.1 have been completed within the current terminal environment.

To execute the process, use the following command:

```
mpirun -np n e4d
```

Here, `n` represents the number of processors you wish to utilise for parallel execution of E4D.

Upon successful execution, the following files should be present in the mesh subdirectory in your working directory:

- `two_blocks.1.node`
- `two_blocks.1.ele`
- `two_blocks.1.edge`
- `two_blocks.1.face`
- `two_blocks.1.neigh`
- `two_blocks.trn`
- `two_blocks.sig`

## 7.3 Forward Model

### 7.3.1 Input Files

In the `/two_blocks/forward_SIP1/` directory, you will find three key files:

- `e4d.inp`
- `dummy.srv`
- `e4d.out`

The file to pay particular attention to is `dummy.srv`. This file contains the locations of each electrode, followed by the electrode configuration for each measurement that will be used to calculate the forward response. You might wonder why observation and error values are included in this file, even though the forward model has not yet been calculated. These are placeholder or dummy values, which are required by the process but have no effect on the forward modelling itself. Familiarise yourself with this file format—more details can be found in Section 5.3.2—as the `srv` file format is what you will need to convert your survey data into when inverting real field data.

The `e4d.out` file contains standard settings and does not include any additional options to export the potential distribution.

### 7.3.2 Calculating a Forward Model with the IP2 mode

To initiate the forward modelling process, ensure that your working directory contains the following files:

- `e4d.inp`
- `dummy.srv`
- `e4d.out`

Additionally, include the mesh files generated in the previous section:

- `two_blocks.1.node`

- `two_blocks.1.ele`
- `two_blocks.1.edge`
- `two_blocks.1.face`
- `two_blocks.1.neigh`
- `two_blocks.trn`
- `two_blocks.sig`

Verify that all steps outlined in Section 7.1 have been completed within the current terminal environment.

To execute the process, use the following command:

```
mpirun -np n e4d
```

Here, `n` represents the number of processors you wish to utilise for parallel execution of E4D.

Upon successful execution, the following file should be present in your working directory:

- `two_blocks.sig.srv`

## 7.4 Inversion Mesh

Now, we need to create a mesh specifically for the inversion. You might wonder why an additional mesh is necessary when we already have one from the forward modelling process. The reason is that the inversion mesh should not contain any a priori subsurface information. For this example, we will use a simple half-space model.

In the `/inversion_SIP1/inversion_mesh.cfg` directory, you will find two key files:

- `e4d.inp`
- `inversion_mesh.cfg`

You will notice that the `inversion_mesh.cfg` file is largely similar to `/forward_SIP1/two_blocks.cfg`, particularly in that it includes control points at the electrode locations, along with the additional refinement points. However, the key difference is that we do not define control points for the two blocks, nor do we define their facets. Consequently, this inversion mesh consists of just two zones: 1600 m × 1200 m × 2000 m rectangular zone and the rest of the mesh.

### 7.4.1 Generating the Mesh with the SIP1 mode

To generate the mesh, ensure that your working directory contains the following files:

- `e4d.inp`
- `inversion_mesh.cfg`

Upon successful completion, you should have the following files (in the mesh subdirectory):

- `inversion_mesh.1.node`

- `inversion_mesh.1.ele`
- `inversion_mesh.1.edge`
- `inversion_mesh.1.face`
- `inversion_mesh.1.neigh`
- `inversion_mesh.trn`
- `inversion_mesh.sig`

## 7.5 Inversion

### 7.5.1 Artificial Noise

An important step when inverting synthetic data is to introduce artificial noise to simulate real-world conditions. In this tutorial, a random noise component ranging from $0.0\%$ to $5.0\%$ has been added to each measurement, specifically to the `dcr_obs` and `ip_obs` fields. Additionally, the standard deviations, `dcr_sd` and `ip_sd`, have been set to $5.0\%$ of their respective observation values. This noise has already been incorporated into the `two_blocks_noise.srv` file. However, if you wish to experiment, you can manually edit the `two_blocks.sig.srv` file from Section 7.3.

### 7.5.2 Input Files for Inversion

In the `/two_blocks/inversion_IP2/` directory, you will find five essential files:

- `e4d.inp`
- `two_blocks_noise.srv`
- `e4d.out`
- `e4d_dc.inv`
- `e4d_ip.inv`

Particular attention should be given to the `inv` files. These files implement four constraints: the first two constrain the smoothness of the conductivity and phase spatial distribution, while the third and fourth enforce minimum and maximum value constraints. The smoothness constraints are applied using structural metric 2 and weighting function 1 with a large mean and a small standard deviation of 10.0 and 0.001, respectively. This essentially transforms the weighting function into a step function that only deactivates for very steep gradients. The minimum and maximum constraints use conductivity and phase values of $0.00001$ S/m and $20$ S/m, and $1.0 \times 10^{-9}$ radians and $0.2$ radians, respectively. These values are not commonly encountered in practice, meaning these constraints primarily serve to guide the inversion away from extremely unstable results.

### 7.5.3 Running the Inversion in SIP3 Mode

To begin the inversion process, ensure that your working directory contains the following files:

- `e4d.inp`
- `two_blocks_noise.srv`
- `e4d.out`

Additionally, include the mesh files generated in the previous section (in the mesh subdirectory):

- `inversion_mesh.1.node`
- `inversion_mesh.1.ele`
- `inversion_mesh.1.edge`
- `inversion_mesh.1.face`
- `inversion_mesh.1.neigh`
- `inversion_mesh.trn`
- `inversion_mesh.sig`

Make sure that all steps outlined in Section 7.1 have been completed within the current terminal environment.

To execute the inversion, use the following command:

```
mpirun -np n e4d
```

Here, `n` represents the number of processors you wish to use for parallel execution of E4D.

Once the inversion has converged, the following files should be present in your working directory:

- `sigma.#`
- `sigmai.#`

The file `sigma.#` contains two columns representing the real and imaginary conductivity of each element in the mesh at iteration `#`. However, at this stage, the phase has not yet been updated for each element.

The file `sigmai.#` also contains two columns representing the real and imaginary conductivity of each element in the mesh at iteration `#`, but here, the phase has been inverted, reflecting the updates made during the inversion process.

For detailed information about each iteration, refer to the `e4d.log` file, which provides comprehensive logs of the inversion process.

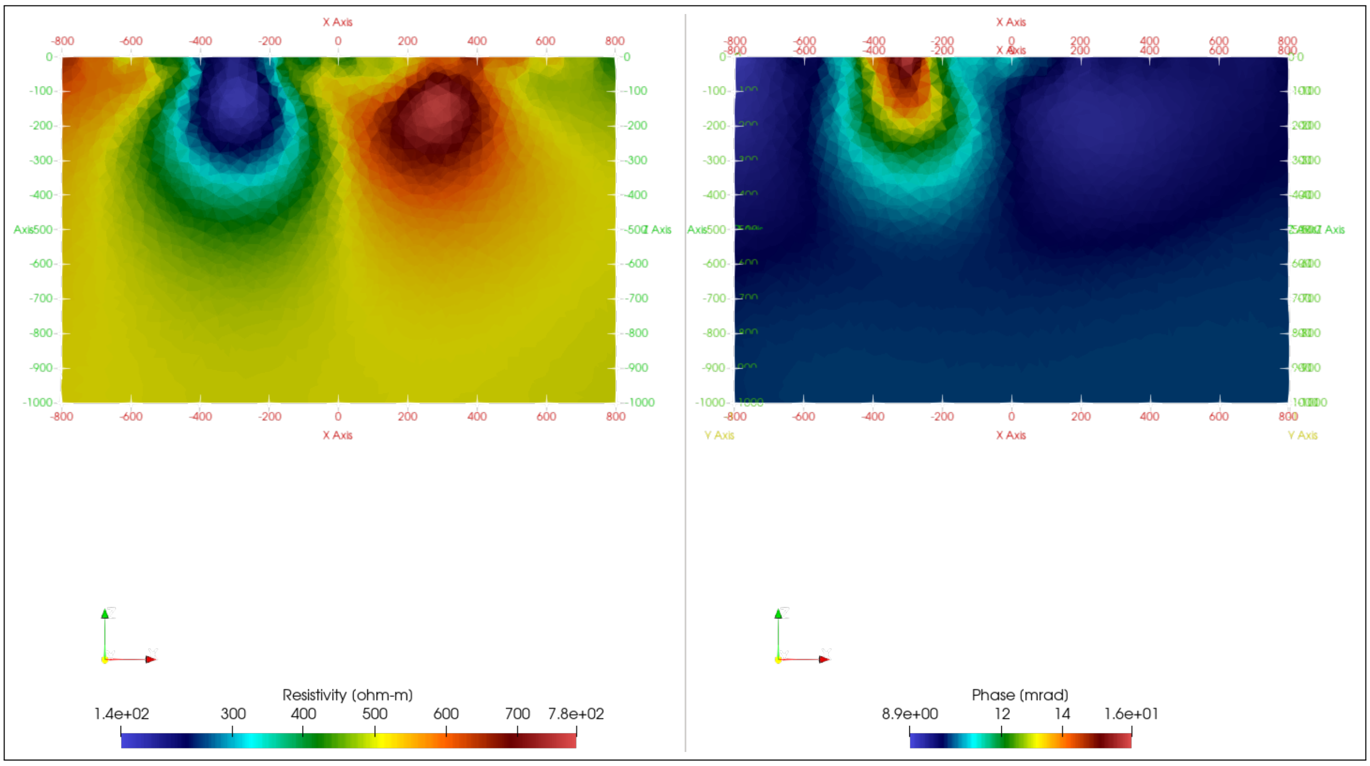

Figure 7: Inversion model section view.

## References

[1] Satish Balay, Shrirang Abhyankar, Mark Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William Gropp, et al. Petsc users manual. 2019.

[2] Timothy C Johnson and Jonathan Thomle. 3-d decoupled inversion of complex conductivity data in the real number domain. *Geophysical Journal International*, 212(1):284–296, January 2017.

[3] Carsten Rücker, Thomas Günther, and Klaus Spitzer. Three-dimensional modelling and inversion of dc resistivity data incorporating topography—i. modelling. *Geophysical Journal International*, 166(2):495–505, August 2006.

[4] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2), February 2015.