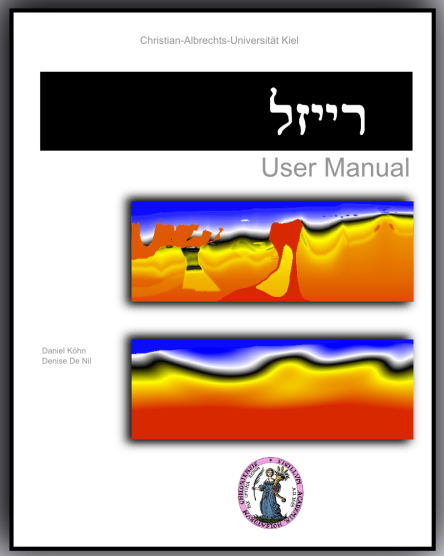
Introduction to frequency domain modelling and FWI of georadar data with GERMAINE

Daniel Köhn, Denise De Nil, Wolfgang Rabbel

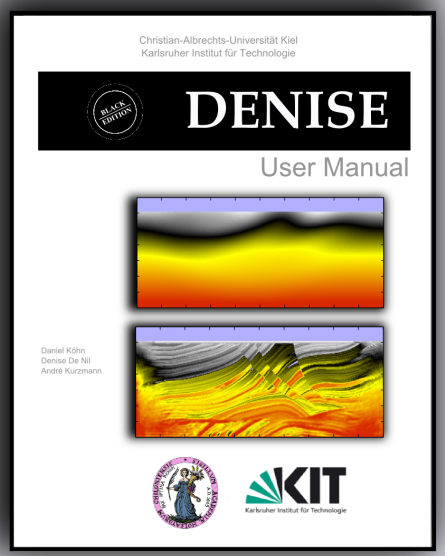
September 27, 2017

Overview of (seismic) inversion codes

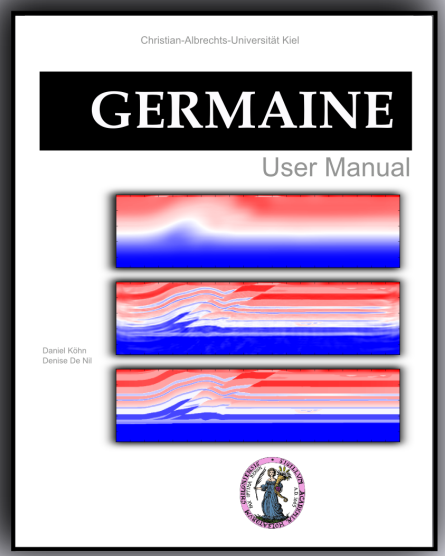
Software packages



2D first arrival traveltime tomograpy



2D time-domain full waveform inversion



2D frequency-domain full waveform inversion

- **Forward modelling:** Cartesian finite differences
- **Inverse problem:** gradient-based local optimization
- **MPI parallelization:** shots and domain decomposition
- **Code repository:** https://github.com/daniel-koehn/

Introduction to FDFD modelling and FWI of georadar data with GERMAINE

1. GERMAINE installation
2. The forward problem
   - Modelling example: homogeneous full space model
   - Modelling example: 2-cross model
3. Full Waveform Inversion
   - GERMAINE FWI example: 2-cross model

GERMAINE installation

GERMAINE installation requirements

- **C-compiler**: gcc, Intel icc
- **MPI**: OpenMPI, Intel-MPI
- **Optimized BLAS/LAPACK library**, e.g. OpenBLAS or Intel MKL
- **Suite Sparse**:
  http://faculty.cse.tamu.edu/davis/suitesparse.html
- **Git** (optional)
- **Seismic Unix**: http://www.cwp.mines.edu/cwpcodes/
- **Numpy, SciPy, Matplotlib**: https://www.scipy.org/
  e.g. using Anaconda https://www.anaconda.com/download/

GERMAINE installation ...

- ... on the NEC HPC-Linux cluster of RZ Kiel

  Click here

- ... Desktop computer at Department of Geophysics

  Click here

Installation of GERMAINE TE test problem

Get the benchmark model ...

1. Clone TE-mode model and parameter files from:

   https://github.com/daniel-koehn/DENISE-Benchmark

   by typing:

   ```
   git clone https://github.com/daniel-koehn/DENISE-Benchmark.git
   ```

2. Copy model and parameter files for the TE-mode model examples from DENISE-Benchmark to GERMAINE

   ```
   cp DENISE-Benchmark/2_cross_TE_model/input_files/GERMAINE* GERMAINE/par/
   cp DENISE-Benchmark/2_cross_TE_model/start/2_cross_TE_* GERMAINE/par/start/
   cp DENISE-Benchmark/2_cross_TE_model/receiver/receiver_2_cross_TE.dat GERMAINE/par/receiver/
   cp DENISE-Benchmark/2_cross_TE_model/source/source_2_cross_TE.dat GERMAINE/par/source/
   ```

The forward problem

The forward problem

Acoustic frequency domain forward problem

$$\nabla^2 \mathrm{P} + \frac{\omega^2}{\mathrm{c}^2} \mathrm{P} = \mathrm{f}$$

with

- P the pressure wavefield $[\mathrm{Pa}]$,
- f the source term $[\mathrm{Pa/m^2}]$
- c the P-wave velocity $[\mathrm{m/s}]$,
- ω the angular frequency $[\mathrm{rad/s}]$.

TE-mode frequency domain forward problem

$$\nabla^2 \mathrm{E_y} + \mu_0 \epsilon_\mathrm{e} \omega^2 \mathrm{E_y} = -\mathrm{i}\omega\mu_0 \mathrm{J_{s_y}}$$

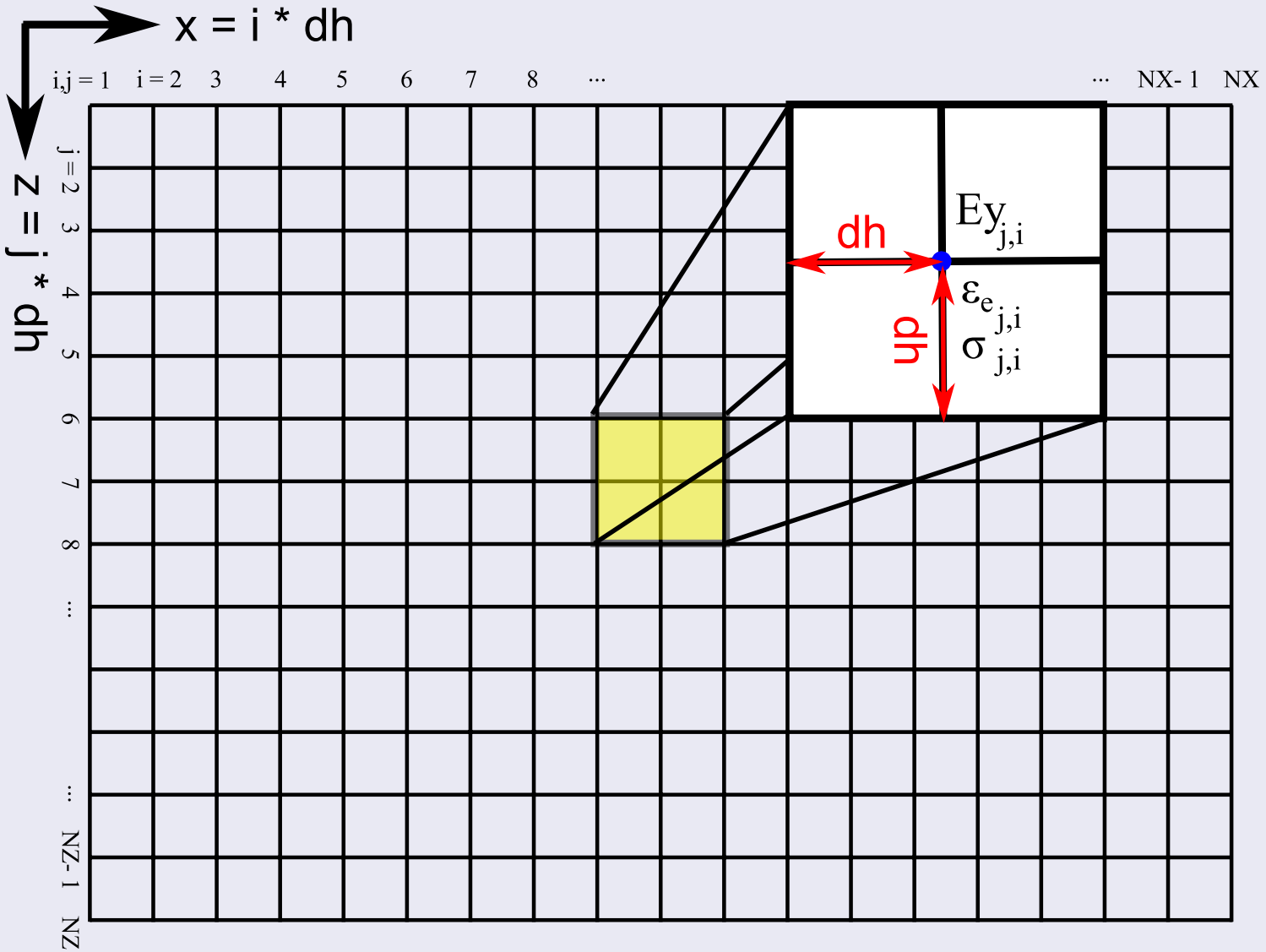
$$\epsilon_\mathrm{e} = \epsilon + \frac{\mathrm{i}\sigma}{\omega}$$

with

- $\mathrm{E_y}$ the electric wavefield $[\mathrm{V/m}]$,
- $\mathrm{J_{s_y}}$ the source current $[\mathrm{A/m^2}]$
- $\mu_0 = 4\pi \cdot 10^{-7}$ the free space magnetic permeability $[\mathrm{H/m}]$
- ϵ the dielectric permittivity $[\mathrm{F/m}]$,
- σ the electrical conductivity $[\mathrm{S/m}]$,
- ω the angular frequency $[\mathrm{rad/s}]$.

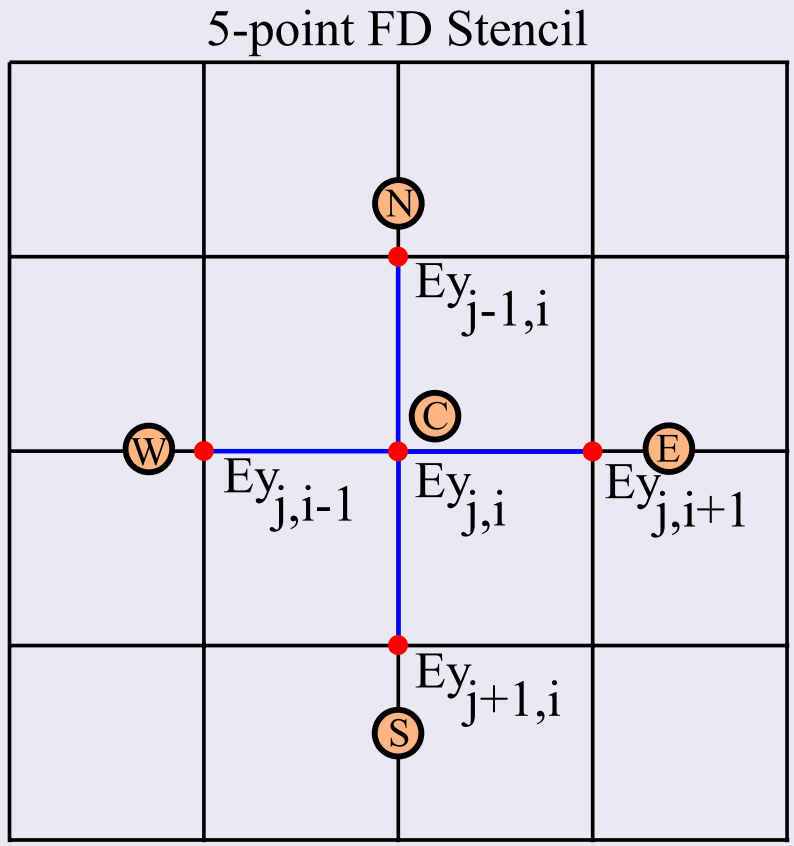
Numerical solution of TE-mode equation

Discretization of ϵ_e, σ and E_y



Numerical solution of TE-mode equation

Approximation of Laplace operator by finite-differences



$$\nabla^2 \mathrm{Ey}(\mathbf{x}, \mathrm{t}) \approx \frac{\mathrm{Ey_{j,i-1}} + \mathrm{Ey_{j,i+1}} + \mathrm{Ey_{j-1,i}} + \mathrm{Ey_{j+1,i}} - 4\mathrm{Ey_{j,i}}}{\mathrm{dh}^2}$$

with dh = spatial grid point distance

Numerical solution of TE-mode equation

Discretization of Helmholtz equation

Which leads to the discretized wave equation:

$$\frac{\mathrm{Ey_{j-1,i}} + \mathrm{Ey_{j,i-1}}}{\mathrm{dh}^2} + \left(\mu_0 \epsilon_\mathrm{e} \omega^2 - \frac{4}{\mathrm{dh}^2}\right) \mathrm{Ey_{j,i}} + \frac{\mathrm{Ey_{j,i+1}} + \mathrm{Ey_{j+1,i}}}{\mathrm{dh}^2} = -\mathrm{i}\omega\mu_0 \mathrm{Jsy_{j,i}}.$$

This can be written as matrix equation

$$\mathrm{A}\,\mathrm{x} = \mathrm{b}$$

where

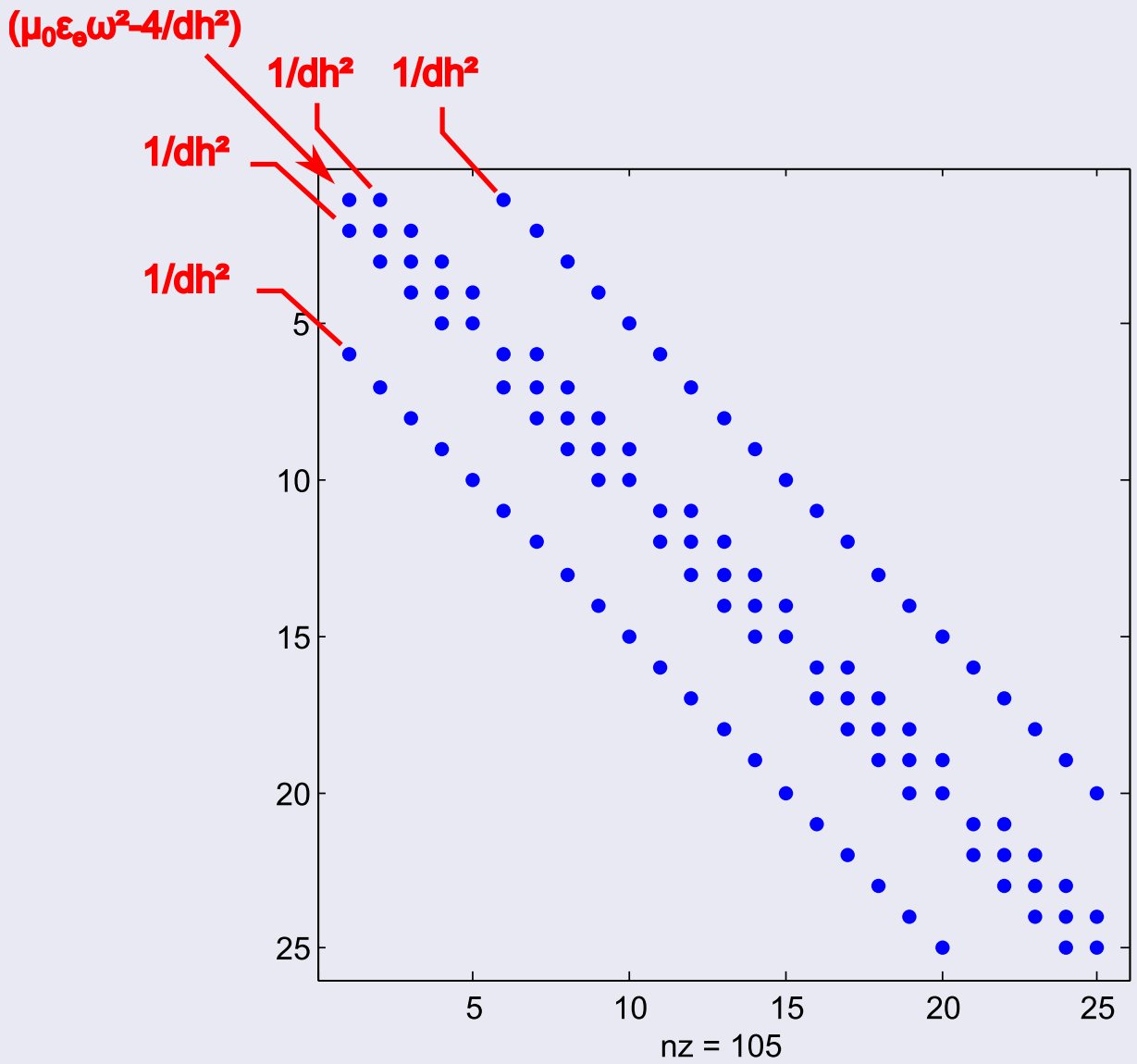
A = impedance matrix

x = solution vector

b = source vector

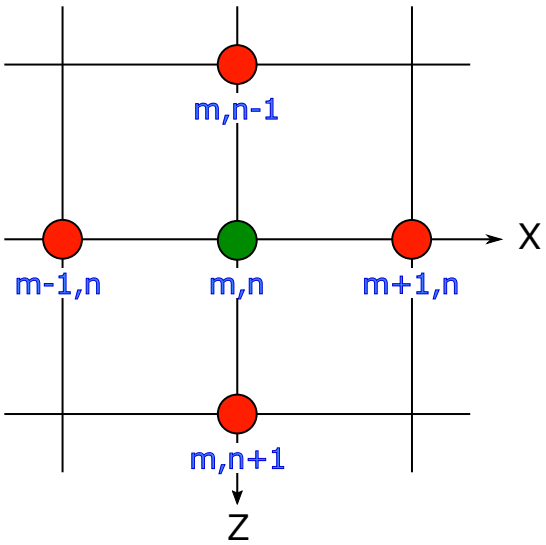
Numerical solution of TE-mode equation

Structure of matrix A for a 5 x 5 gridpoint model

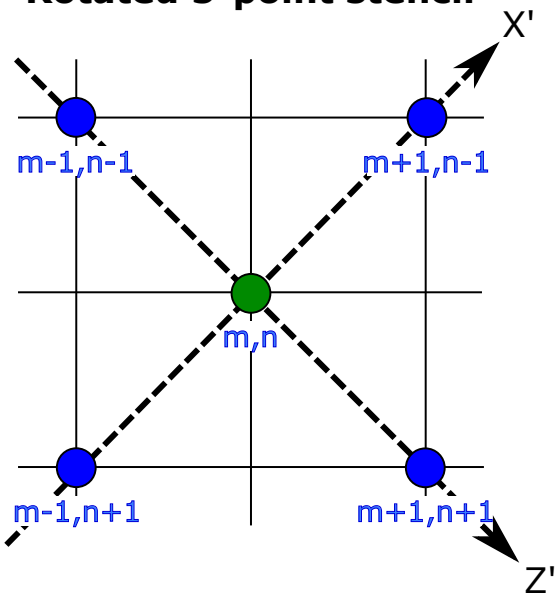


Numerical solution of TE-mode equation

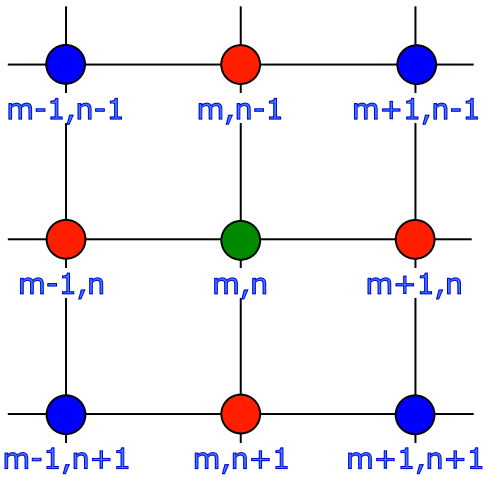
Classical 5-point stencil



Rotated 5-point stencil



Mixed-grid 9-point stencil



adapted from [Jo et al., 1996]

- Minimizing numerical anisotropy and grid dispersion by using mixed-grid 9-point finite-difference scheme with mass-lumping: [Hustedt et al., 2004] [Operto et al., 2009]

Numerical solution of TE-mode equation

Perfectly Matched Layers (PMLs)

Introduction to Uni-axial (UPMLs) and Stretched-Coordinate (SC-PMLs):

- Lecture notes (lecture 9):
  http://emlab.utep.edu/ee5390cem.htm
- Full lecture available on YouTube:
  https://www.youtube.com/watch?v=w_NnRZlNuAA

Basic ideas of PMLs

1. Introduce artificial attenuation of EM-waves in a boundary frame around the model domain
   Problem: new artificial impedance contrast
2. Reduce material parameters in boundary frame to match impedances
   Problem: damping effect depends on incidence angle, frequency and polarization of EM wave
3. Introduce artificial anisotropy in boundary frame which leads to the UPML formulation
4. Move PML parameters from RHS to LHS of Maxwell equations and associate them with the curl operator. The PML parameters are now "stretching" the coordinates (SC-PMLs)

Numerical solution of TE-mode equation

Solution of the problem Ax=b using a direct solver

1. Decompose A into an upper U and lower L triangular matrix: $A = LU$
2. Solve the problem Ly = b by forward substitution
3. Solve the problem Ux = y by backward substitution

In GERMAINE we use the direct solver UMFPACK (Unsymmetric MultiFrontal PACKage)

http://faculty.cse.tamu.edu/davis/suitesparse.html

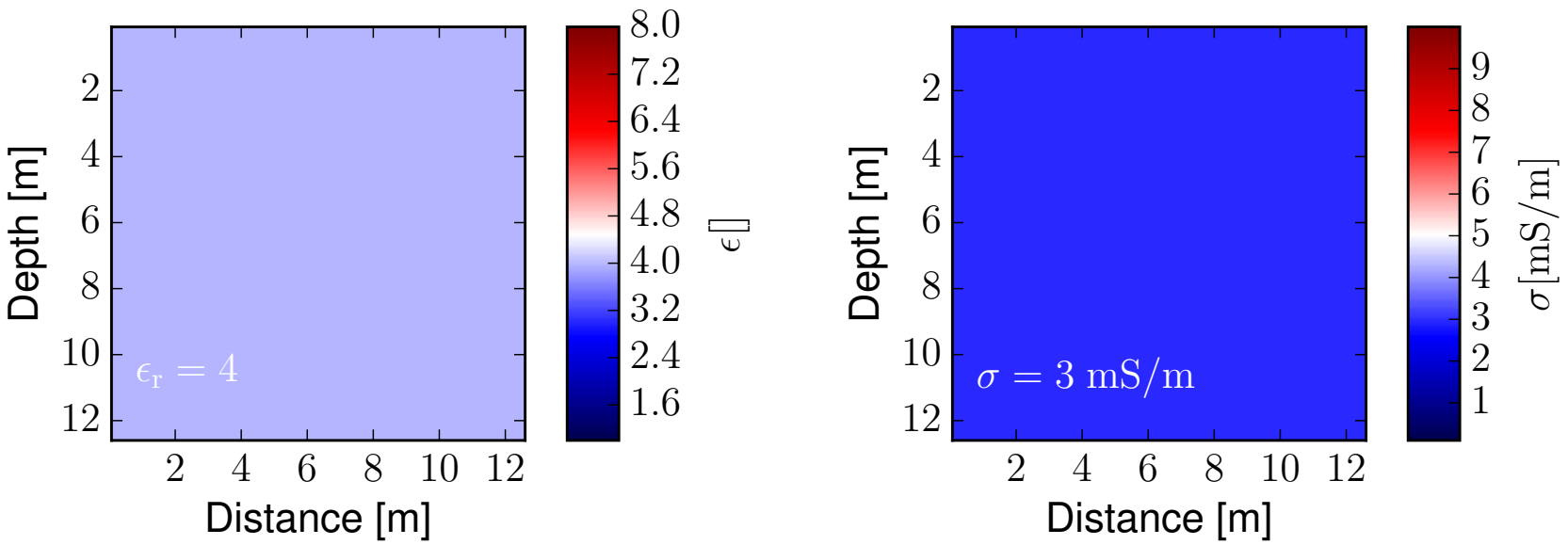
Future implementation of parallel direct solvers

- The currently implemented shot-frequency parallelization is not really efficient. Especially in case of small problems the LU-factorization is the computationally most demanding part
- UMFPACK will be replaced by a parallel solver like MUMPS or PARADISO in a future release

Modelling example: homogeneous full space model

Modelling example: homogeneous full space model

Homogeneous model [Meles et al., 2011, Lavoué et al., 2014]

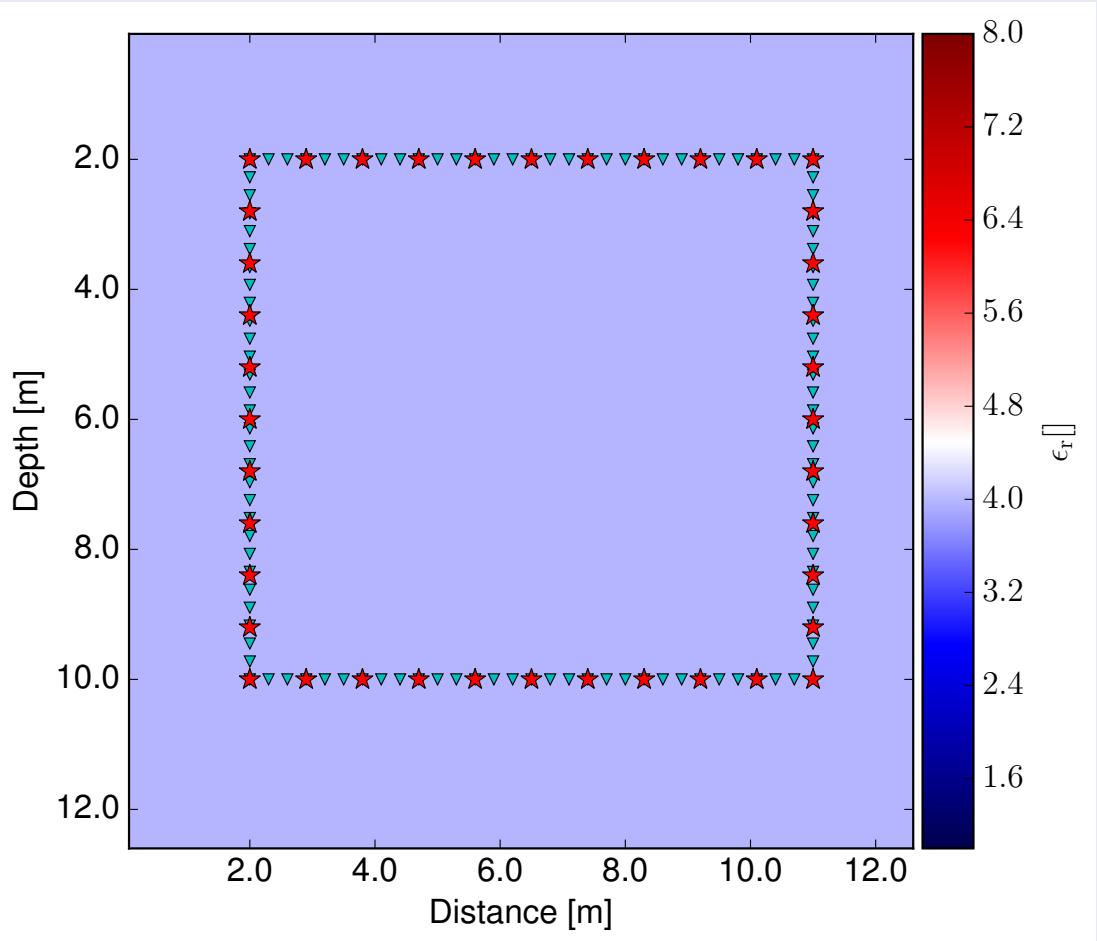


Model discretization:

- NX $= 180$ gridpoints $\times$ NZ $= 180$ gridpoints
- $+$ 10 gridpoints PMLs on each side
- DH $= 7$ cm

Modelling example: homogeneous full space model

Perfect illumination acquisition geometry



- 40 shots and 117 receivers
- Modelling of frequencies between 50 MHz - 200 MHz

Modelling example: homogeneous full space model

GERMAINE structure

- **bin**
  This directory contains the executable program GERMAINE
- **include**
  This directory contains header files and global parameters
- **jobs**
  This directory contains Batch-scripts to submit GERMAINE modelling/inversion runs on HPCs with PBS-batch system
- **par**
  Parameter files for GERMAINE modelling/inversion and Jupyter notebooks for visualization
- **src**
  This directory contains the complete source codes

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **GERMAINE Modus:**

```
#------------------- GERMAINE modus ----------------------------
forward_modelling_(yes=0)_FDFWI_(yes=1)_RTM_(yes=2)_(INVMAT) = 0
#
```

Frequency domain forward modelling	INVMAT = 0
Frequency domain FWI	INVMAT = 1
Reverse Time Migration	INVMAT = 2

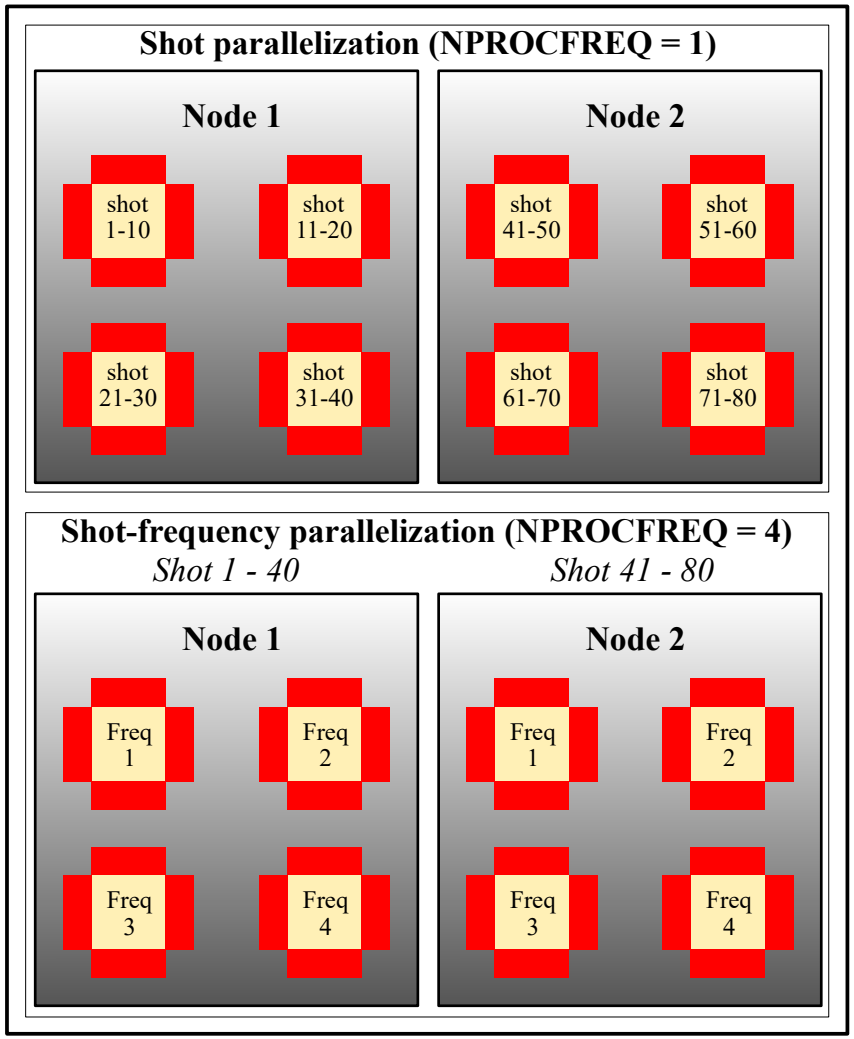
- **GERMAINE Physics:**

```
# ---------- GERMAINE Physics ----------
(2D-AC=1;2D-SH=2;2D-TE=4)_(PHYSICS) = 4
#
```

2D acoustic	PHYSICS = 1
2D elastic SH	PHYSICS = 2
2D TE mode	PHYSICS = 4

GERMAINE parameter file GERMAINE_2_cross_TE.inp

```
#-------------- MPI Parallelization -----------------------------------
number_of_MPI_processes_for_frequency_parallelization_(NPROCFREQ) = 1
```



Shot parallelization

- Assume we want to model FD wavefields for 1 frequency and 80 shots on two CPUs with 4 cores, you can distribute the shots over the 8 cores by setting NPROCFREQ = 1

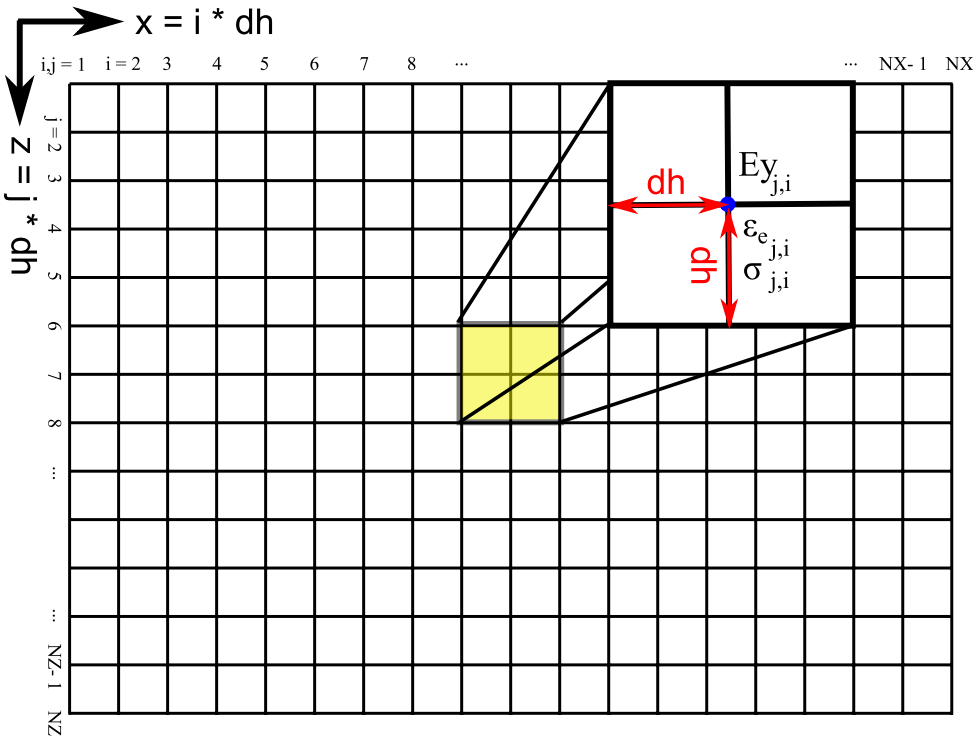
Shot-frequencey parallelization

- If you want to model FD wavefields for 4 frequencies simultaneously, you can assign frequencies to specific cores by setting NPROCFREQ = 4. In this case the shots are distributed over nodes 1 and 2. If you have more frequencies than cores, the frequencies are distributed over the cores.

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Spatial model discretization**

```
#-------------------- 2-D Grid -----------------
number_of_gridpoints_in_x-direction_(NX) = 180
number_of_gridpoints_in_y-direction_(NZ) = 180
distance_between_gridpoints(in_m)_(DH) = 0.07
```



GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Source geometry and wavelet**

```
#--------------------Source----------------------
# File with source positions:
SOURCE_FILE = ./source/source_2_cross_TE.dat
# Read time-domain source wavelet from SU file:
READ_WAVELET = 0
WAVELET_NAME = start/wavelet.su
#
```

Source positions are defined in SOURCE_FILE

An external time-domain wavelet can be read from a SU file (READ_WAVELET=1). Frequency domain data for a given frequency of the wavelet is calculated by DFT.

If READ_WAVELET=0 a spike wavelet is assumed.

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Model file**

```
#-------------------- Model ----------------------
read_model_parameters_from_MFILE(yes=1)(READMOD) = 1
MFILE = start/2_cross_TE_true
#
```

If READMOD=1, the permittivity ϵ [F/m] and conductivity σ [S/m] grids are read from external binary files. MFILE defines the basic file name that is expanded by the following extensions: Permittivity: ".eps", Conductivity: ".sig". For this example:

```
start/2_cross_TE_true.eps
start/2_cross_TE_true.sig
```

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Free Surface boundary condition**

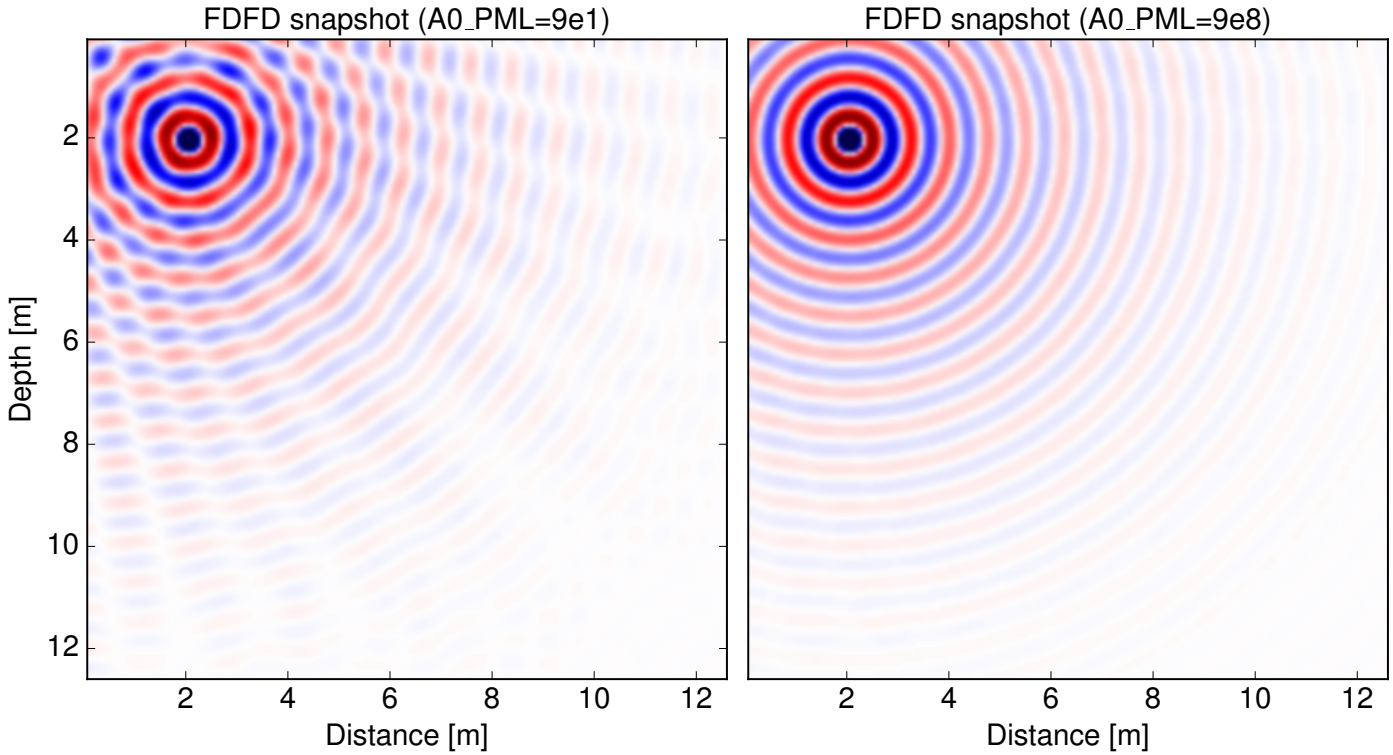
```
#-----------Free Surface-------------
free_surface_(yes=1)(FREE_SURF) = 0
#
```

Not implemented for TE-mode.

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **PML boundary frame**

```
#-------------- Define PML parameters ----------------
width_of_absorbing_frame_(in_gridpoints)_(NPML) = 10
damping_parameter_(A0_PML) = 9e8
#
```



PML layers defined outside of the modelling domain.

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Snapshots**

```
#----------------------Snapshots-------------------
output_of_Ey_field_(SNAP)(yes>0) = 0
increment_x-direction_(IDX) = 1
increment_y-direction_(IDY) = 1
data-format_(SNAP_FORMAT)(ASCII(2);BINARY(3)) = 3
basic_filename_(SNAP_FILE) = ./snap/2_cross_TE
#
```

If SNAP=1, the real part of the monochromatic FD wavefield $\mathrm{E_y}$ will be written to file SNAP_FILE for each shot. This requires that only one stage and frequency is defined in the GERMAINE workflow file. Details of GERMAINE workflow file

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Receiver**

```
#----------------------- Receiver -----------------------------------
output_of_seismograms_(SEISMO) = 1
file_with_receiver_positions_(REC_FILE) = ./receiver/receiver_2_cross_TE
read_receiver_positions_from_multiple_files_(yes=1)_(READ_REC) = 0
reference_point_for_receiver_coordinate_system_(REFREC) = 0.0 , 0.0
#
```

Read receiver positions from ASCII file REC_FILE.

By setting READ_REC = 1 different receiver positions for each shot can be read from separate receiver files:

REC_FILE_shot_1.dat, REC_FILE_shot_2.dat ...

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **FD seismograms**

```
#-------------------- FD pressure seismograms  ----------------------
Outputfile_for_pressure_seismograms_(PICKS_FILE) = ./seis/2_cross_TE
#
```

Write frequency domain data for all shots at the receiver positions to the IEEE little endian binary file PICKS_FILE for each stage defined in the GERMAINE workflow file. For details regarding the used data structure I refer to the output in src/AC/write_seis_AC.c:

```
FILE *fp;
sprintf(pickfile_char,"%s_p_stage_%d.bin",PICKS_FILE,nstage);
fp=fopen(pickfile_char,"wb");

for(k=1;k<=NF;k++){
    for(j=1;j<=nshots;j++){
        for(i=1;i<=ntr;i++){

            index =  i + ntr * (j-1) + ntr * nshots * (k-1);
            tmp = precr_vec[index];
            tmp1 = preci_vec[index];

            fwrite(&tmp, sizeof(float), 1, fp);
            fwrite(&tmp1, sizeof(float), 1, fp);

        }
    }
}
fclose(fp);
```

GERMAINE parameter file GERMAINE_2_cross_TE.inp

- **Forward modelling log file**

```
# each PE is printing log-information to LOG_FILE.MYID
log-file_for_information_about_progress_of_program_(LOG_FILE) = log/GERMAINE.log
info_of_processing_element_zero_to_stdout_(yes=1/no=0)_(LOG) = 1
```

Write details of forward modelling to log-file LOG_FILE.

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

Workflow file GERMAINE_workflow_2_cross_TE.inp

- **GERMAINE workflow file (forward modelling)**

```
PRO     FC_low      FC_high     NF   ...
0.0     50e6        200e6       1    ...
0.0     60e6        200e6       1    ...
...
```

Only columns 2-4 are important for FD forward modelling:

2nd column = lowest frequency modelled with FDFD [Hz]
3rd column = highest frequency modelled with FDFD [Hz]
4th column = number of frequencies NF modelled between FC_low and FC_high

If NF > 1 a frequency sample interval

$$df = \frac{FC_{high} - FC_{low}}{NF - 1}$$

between FC_low and FC_high is assumed.

Workflow file GERMAINE_workflow_2_cross_TE.inp

- **GERMAINE workflow file (forward modelling)**

```
PRO     FC_low     FC_high     NF   ...
0.0     100e6      200e6       1    ...
```

If you want to model monochromatic FD wavefield snapshots, the workflow file has to contain only 1 line and 1 frequency NF=1.

For the example above, FD snapshots are calculated for a frequency of 100 MHz.

Workflow file GERMAINE_workflow_2_cross_TE.inp

- **GERMAINE workflow file (forward modelling)**

```
PRO     FC_low     FC_high     NF     ...
0.0     50e6       200e6       1000   ...
```

To produce time-domain data from frequency-domain data you can model a whole bunch of frequencies in 1 stage.

To reduce computation time, I recommend to use the GERMAINE frequency parallelization by setting NPROCFREC in the GERMAINE parameter file to the maximum number of cores available on your computer.

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

Receiver file receiver_2_cross_TE.dat

- **Receiver positions**

```
2.00    2.00
2.30    2.00
2.60    2.00
2.90    2.00
3.20    2.00
3.50    2.00
3.80    2.00
4.10    2.00
...
```

line n = receiver coordinate of receiver no. n
1st column = receiver x-coordinate [m]
2nd column = receiver z-coordinate [m]

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

Source file source_2_cross_TE.dat

- **Source positions**

```
40
2.00    0.00    2.00    0.00    10.00    1.00    0.00    3.00
2.90    0.00    2.00    0.00    10.00    1.00    0.00    3.00
3.80    0.00    2.00    0.00    10.00    1.00    0.00    3.00
4.70    0.00    2.00    0.00    10.00    1.00    0.00    3.00
5.60    0.00    2.00    0.00    10.00    1.00    0.00    3.00
6.50    0.00    2.00    0.00    10.00    1.00    0.00    3.00
7.40    0.00    2.00    0.00    10.00    1.00    0.00    3.00
...
```

1st line = number of sources
line n+1 = source coordinate of receiver no. n
1st column = source x-coordinate [m]
3rd column = source z-coordinate [m]
Other columns = can be ignored

Modelling example: homogeneous full space model

GERMAINE modelling/FWI files

- **par/GERMAINE_2_cross_TE.inp**
  Defines modelling and fixed FWI parameters
- **par/GERMAINE_workflow_2_cross_TE.inp**
  Definition of frequencies and variable FWI parameters
- **par/receiver/receiver_2_cross_TE.dat**
  Definition of receiver positions
- **par/source/source_2_cross_TE.dat**
  Definition of source positions
- **par/start/2_cross_TE_***
  Model files discretized on Cartesian grid

Creation of model files

- **Create model files in Matlab**

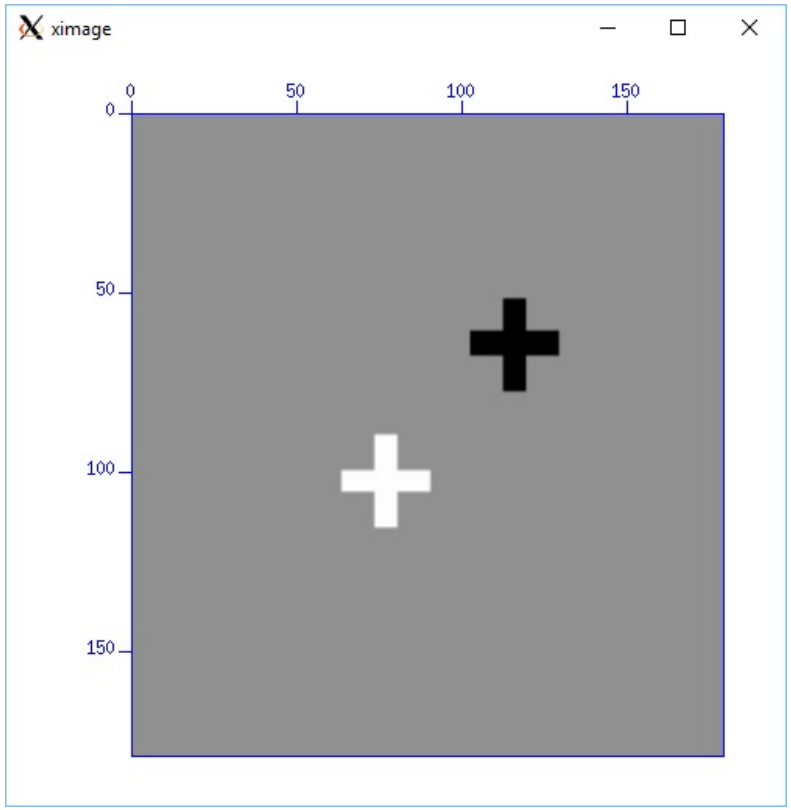
```
% loop over model grid
for i=1:NX
   for j=1:NZ

      % create your fancy epsilon and sigma models here ...
      eps_model(j,i) = ...
      sig_model(j,i) = ...

   end
end

% write epsilon model to binary file
file1=['my_epsilon_model.eps'];
fid1=fopen(file1,'w','ieee-le');
fwrite(fid1,eps_model,'float')
fclose(fid1);
```

- **Create model files in Python**

  Example Jupyter notebook for model generation

Creation of model files

- **Quick model check with Seismic Unix**




```
ximage n1=180 < start/2_cross_TE_true.eps
```

with n1 = NZ

Reading model files from GERMAINE

- **Reading model files in Matlab**

```
% load GERMAINE_epsilon_model
file=['GERMAINE_epsilon_model.eps'];
disp([' loading file ' file]);
fid=fopen(file,'r','ieee-le');
eps_model=fread(fid,[NZ,NX],'float');
fclose(fid);

% plot model
imagesc(eps_model);
```

- **Reading model files in Python**

  Example Jupyter notebook to read GERMAINE models

Running GERMAINE for the homogeneous model ...

Modelling example: homogeneous full space model

Running GERMAINE for the homogeneous model ...

1. In GERMAINE/par/GERMAINE_2_cross_TE.inp set the following parameters for a forward modelling run of the homogeneous model

```
forward_modelling_(yes=0)_FDFWI_(yes=1)_RTM_(yes=2)_(INVMAT) = 0
MFILE = start/2_cross_TE_init
output_of_P_field_(SNAP)(yes>0) = 1
```

2. Copy GERMAINE/par/GERMAINE_workflow_2_cross_TE.inp to GERMAINE_workflow_2_cross_TE_1.inp and remove every line except the first two:

```
PRO   FC_low    FC_high   NF  ...
0.0   50e6      200e6     1   ...
```

3. Run GERMAINE forward modelling on e.g. 4 cores of your CPU from the /par directory:

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE.inp GERMAINE_workflow_2_cross_TE_1.inp
```

Modelling example: homogeneous full space model

4. For each shot defined in source/source_2_cross_TE.dat the real and imaginary parts of the pressure wavefield at the receiver positions defined in receiver/receiver_2_cross_TE.dat are written to

   seis/incl2_p_stage_[fwi_stage].bin

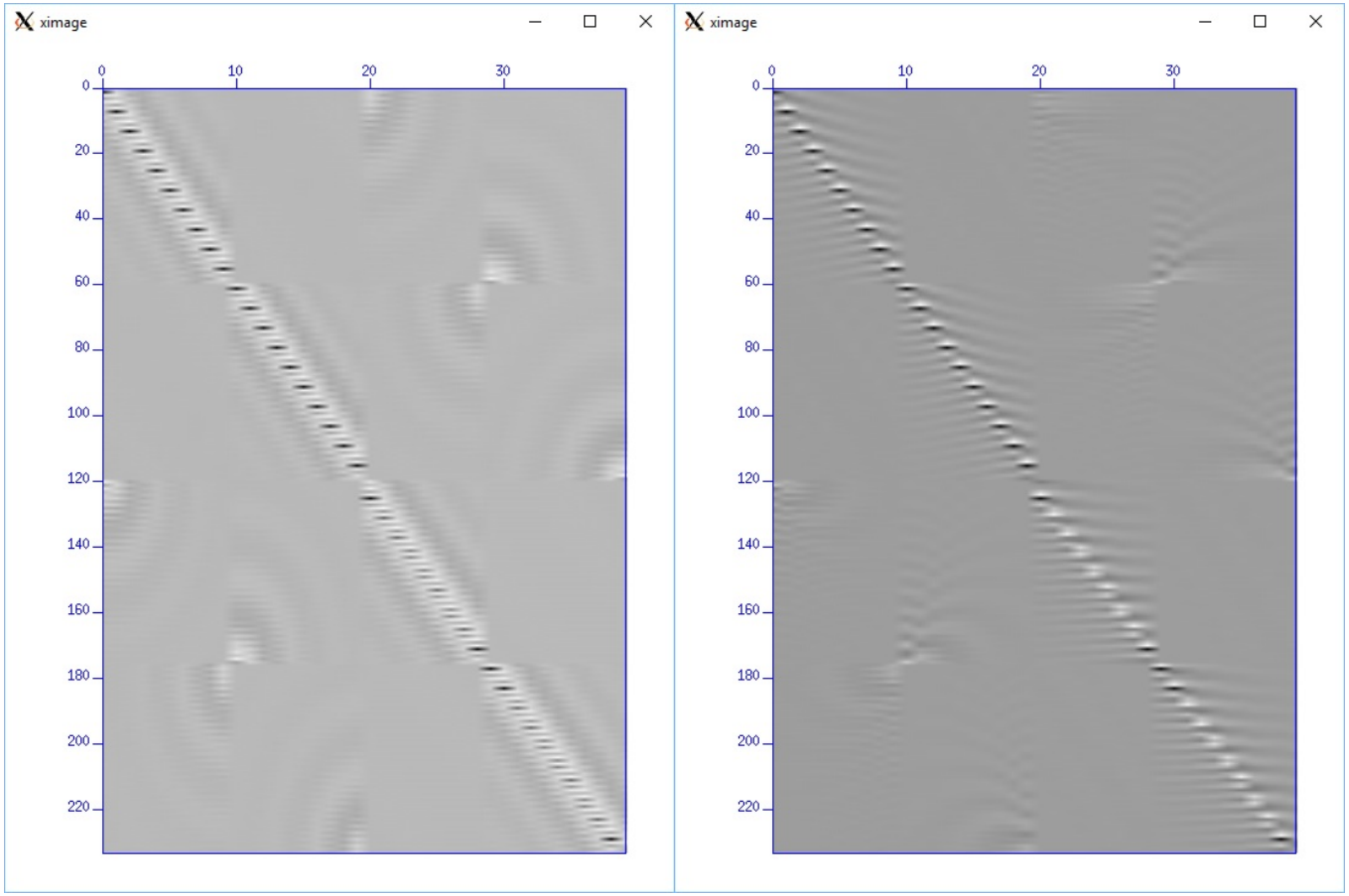
   for each frequency (group) defined in the workflow file

   GERMAINE_workflow_2_cross_TE_1.inp

Modelling example: homogeneous full space model

GERMAINE quick and dirty FD data visualization ...

5. ... with Seismic Unix:




```
ximage n1=234 <  seis/2_cross_TE_p_stage_1.bin
```

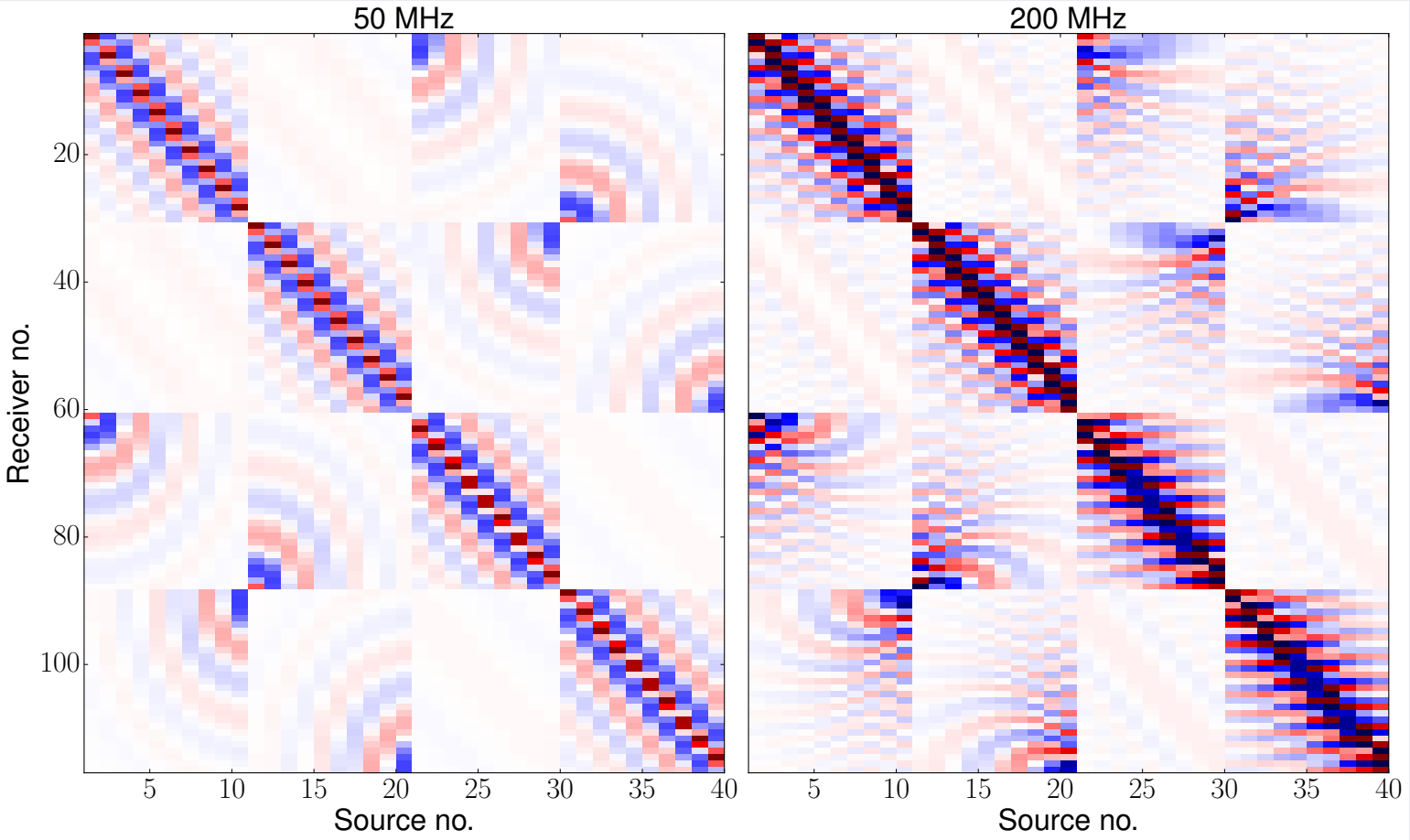
with n1 = 2 * number of receivers

Modelling example: homogeneous full space model

GERMAINE publication-ready FD data visualization ...

5. ... with Matplotlib:

Frequency domain data $\Re(\mathrm{E_y})$

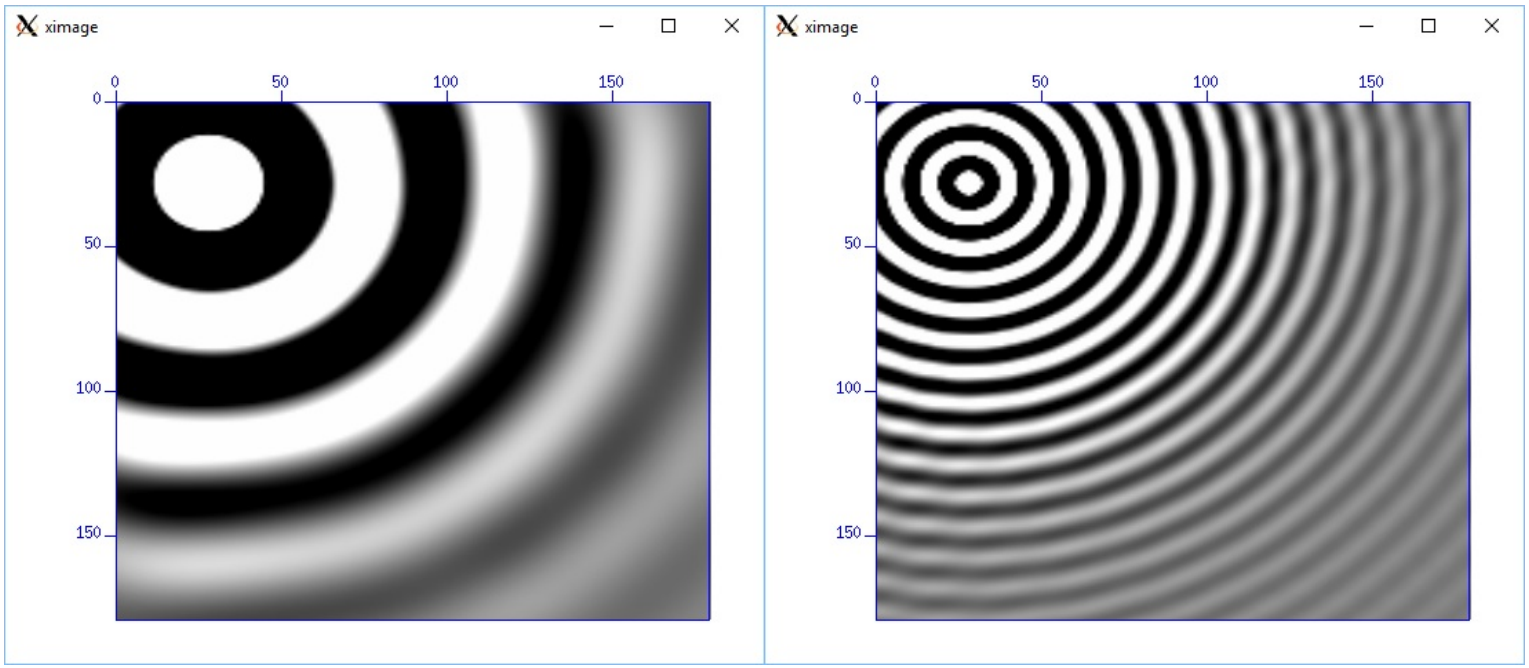


Jupyter notebook for frequency domain data visualization

Modelling example: homogeneous full space model

GERMAINE quick and dirty FD wavefield visualization ...

6. ... with Seismic Unix:




```
ximage n1=180 clip=8e0 < snap/2_cross_TE_shot_1.p
```

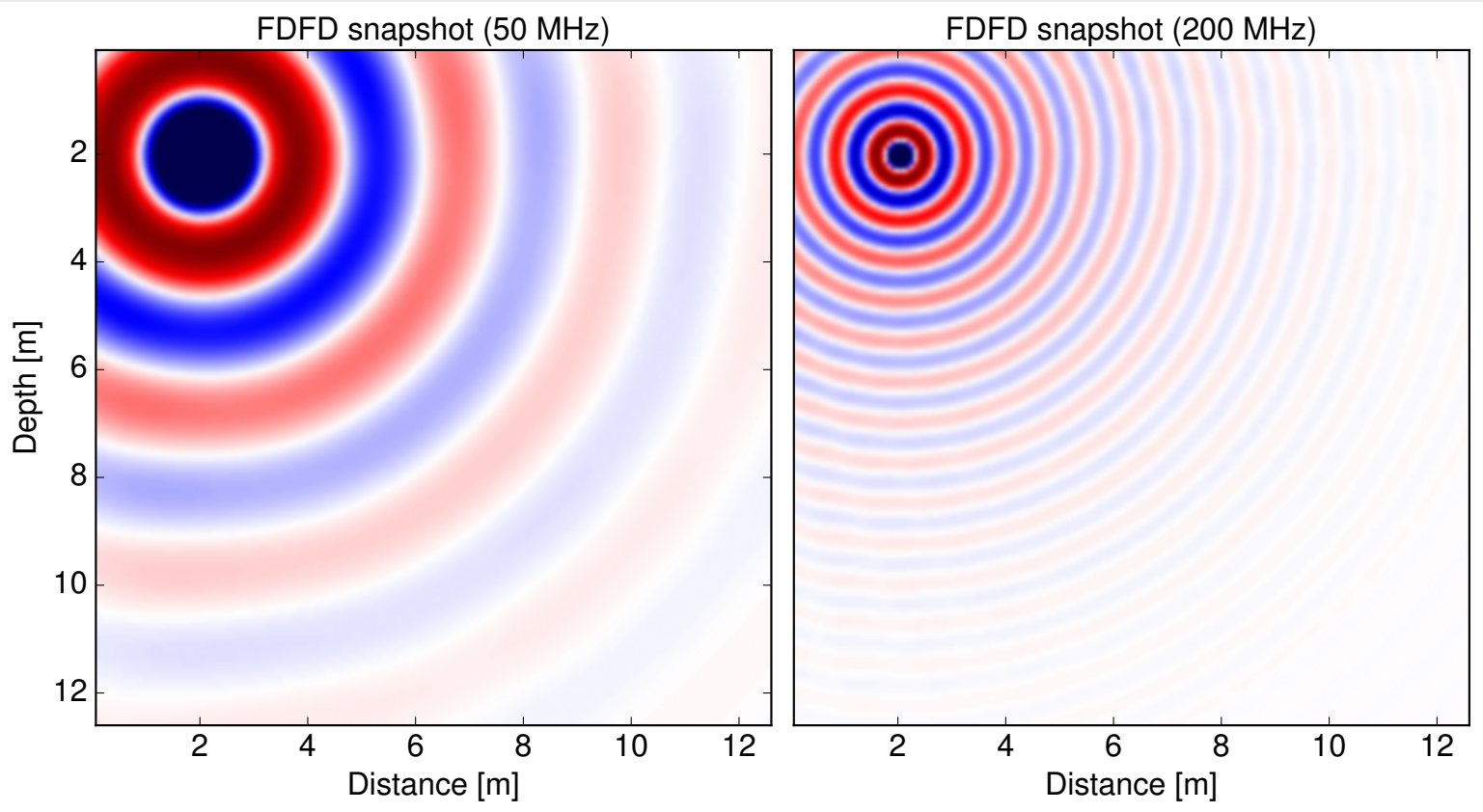
with n1 = NZ

Modelling example: homogeneous full space model

GERMAINE publication-ready FD wavefield visualization ...

6. ... with Matplotlib:

Frequency domain data $\Re(\mathrm{E_y})$



Jupyter notebook for frequency domain wavefield visualization

Modelling example: homogeneous full space model

Creating TD radargrams from FD data

1. Create a Ricker source wavelet with a centre frequency of 100 MHz using this Jupyter notebook

2. Add SU header to the wavelet.bin file

   ```
   suaddhead n1=1001 < wavelet.bin | sushw key=dt a=250 > tmp.su
   ```

   where dt denotes the sampling rate [1e-12 s]

3. If you use a SU version which supports only XDR SU format, you additionally have to apply:

   ```
   suswapbytes < tmp.su > wavelet.su
   ```

Creating TD radargrams from FD data

4. Move wavelet.su to the par/start directory

```
mv par/visu/wavelet.su par/start/
```

5. Activate the option to read external source wavelets in GERMAINE_2_cross_TE.inp:

```
# Read time-domain source wavelet from SU file:
READ_WAVELET = 1
WAVELET_NAME = start/wavelet.su
```

6. Modify the GERMAINE_workflow_2_cross_TE_1.inp to model 400 frequencies:

```
PRO   FC_low   FC_high   NF
0.0   1e6      200e6     400 ...
```

Modelling example: homogeneous full space model

Creating TD radargrams from FD data

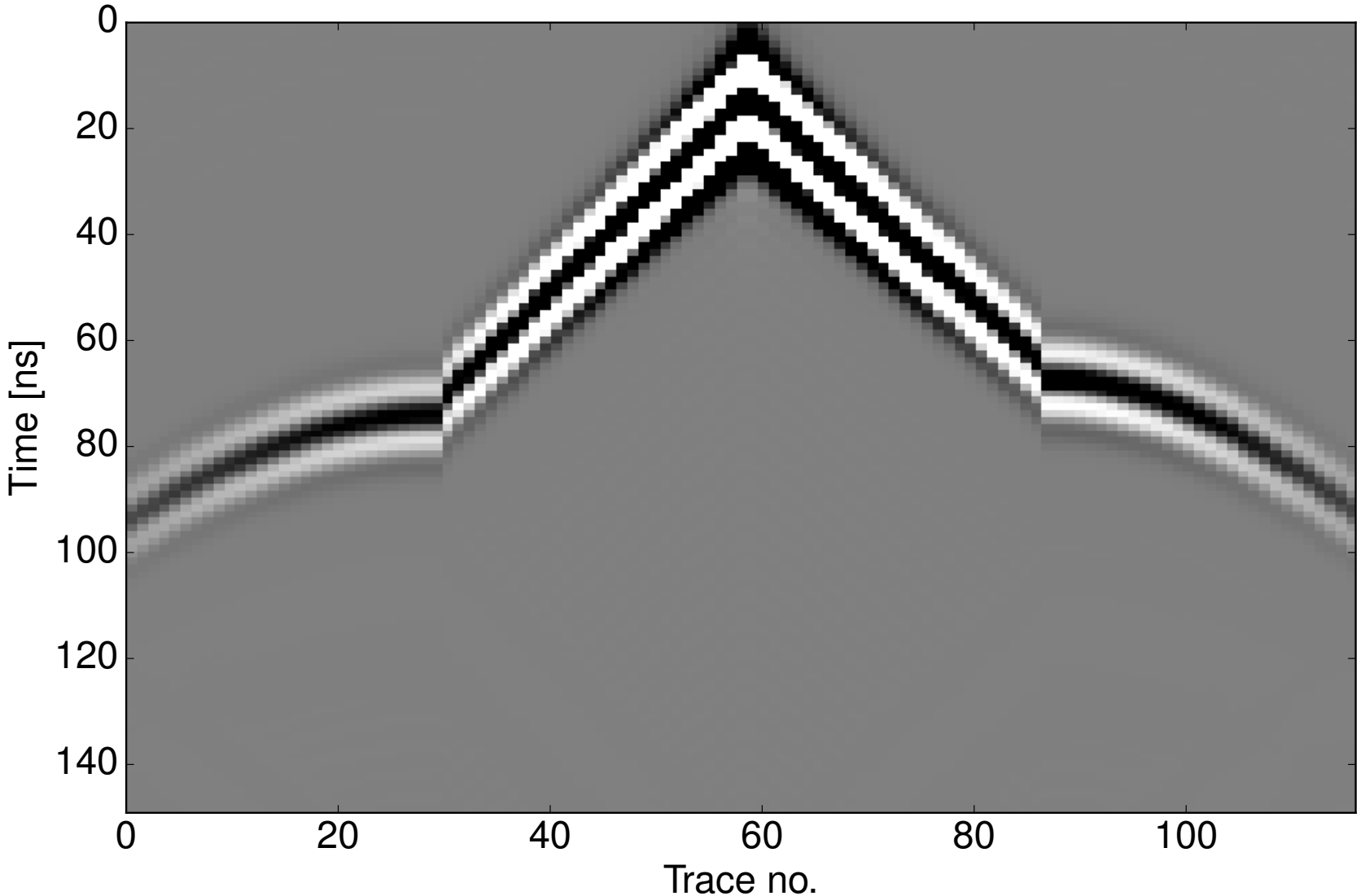
7. Run GERMAINE to create FD data

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE.inp GERMAINE_workflow_2_cross_TE_1.inp
```

8. Calculate IFFT and visualization of the resulting TD radargrams with this Jupyter notebook should lead to TD data like this ...

Modelling example: homogeneous full space model

Time-domain radargrams $\mathrm{E_y}$



Modelling example: homogeneous full space model

Analytical solution for 2D homogeneous medium

$$\mathrm{E_y^{analytic}(x,z,\omega)} = \frac{\omega\mu_0 \mathrm{J_{sy}}}{4}\mathrm{H_0^{(1)}(kr)}$$

$$\mathrm{k} = \omega\sqrt{\mu_0\epsilon_e}$$

$$\epsilon_e = \epsilon + \frac{\mathrm{i}\sigma}{\omega}$$

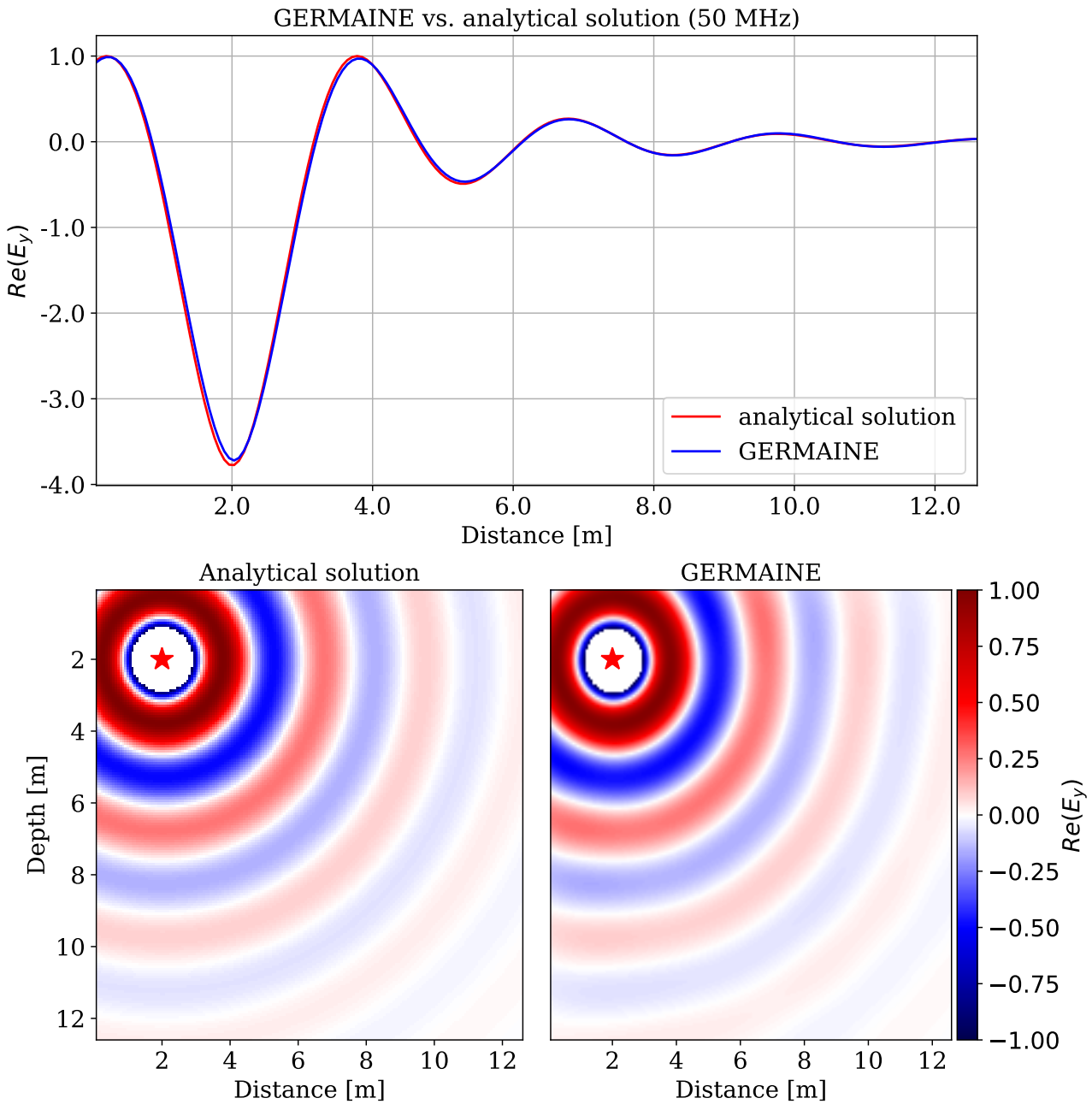
$$\mathrm{r} = \sqrt{(\mathrm{x}-\mathrm{x_s})^2 + (\mathrm{z}-\mathrm{z_s})^2}$$

with

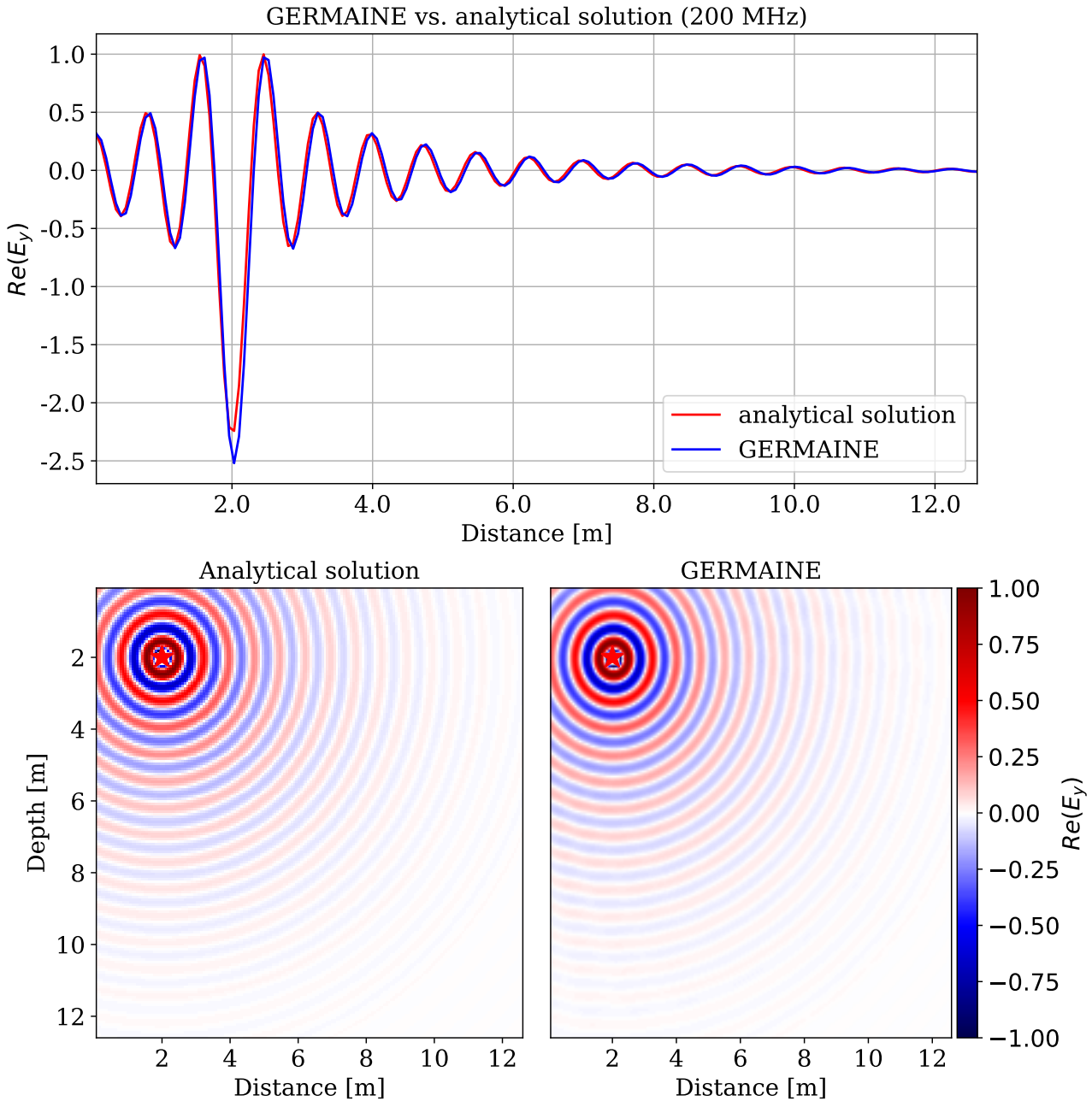
- $\mathrm{E_y}$ the electric wavefield,
- $\mathrm{J_{sy}} = \delta(\mathrm{x}-\mathrm{x_s})\delta(\mathrm{z}-\mathrm{z_s})$ the source current
- $\mathrm{x_s}, \mathrm{z_s}$ the source position
- ω the angular frequency $[\mathrm{rad/s}]$.
- $\epsilon = 3.54 \cdot 10^{-11}\ \mathrm{F/m}$,
- $\sigma = 3\ \mathrm{mS/m}$,
- $\mu_0 = 4\pi \cdot 10^{-7}\ \mathrm{H/m}$
- $\mathrm{H_0^{(1)}}$ the Hankel function of first kind and order zero

Jupyter notebook to compute analytical solution

GERMAINE vs. analytical solution (50 MHz)

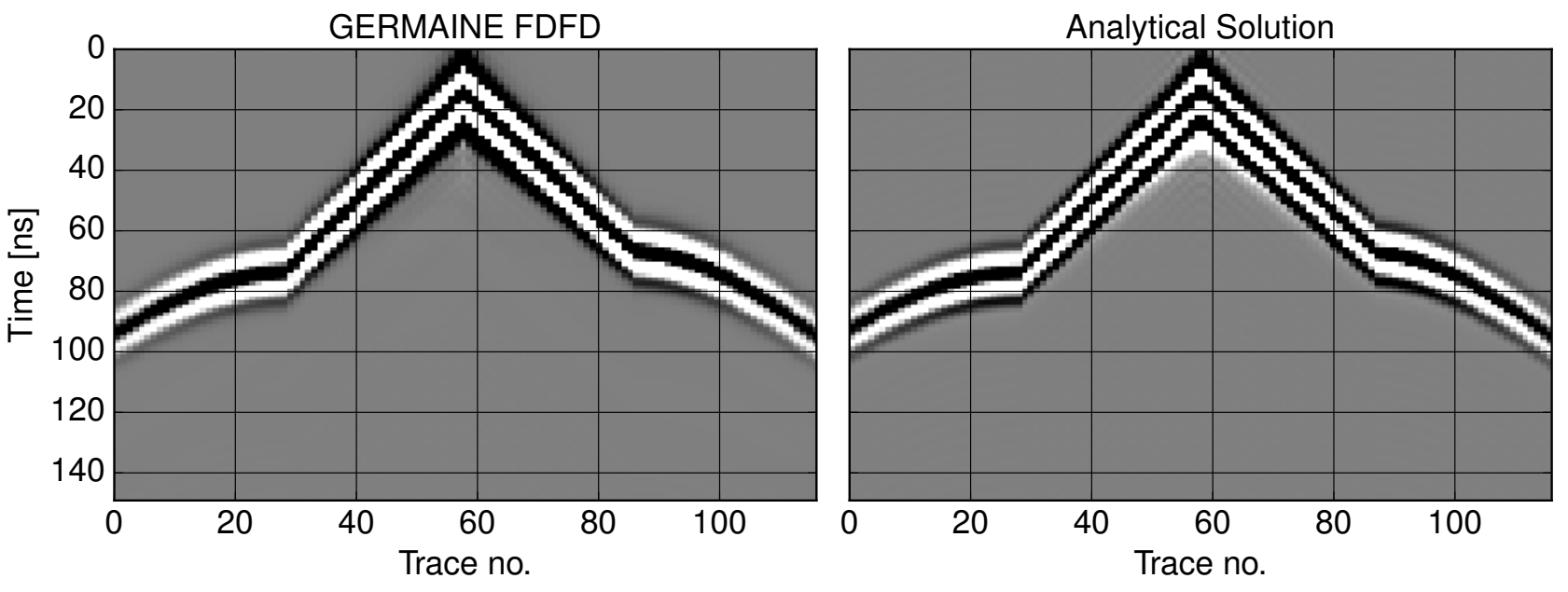


GERMAINE vs. analytical solution (200 MHz)



GERMAINE vs. analytical solution

Comparison of time-domain radargrams



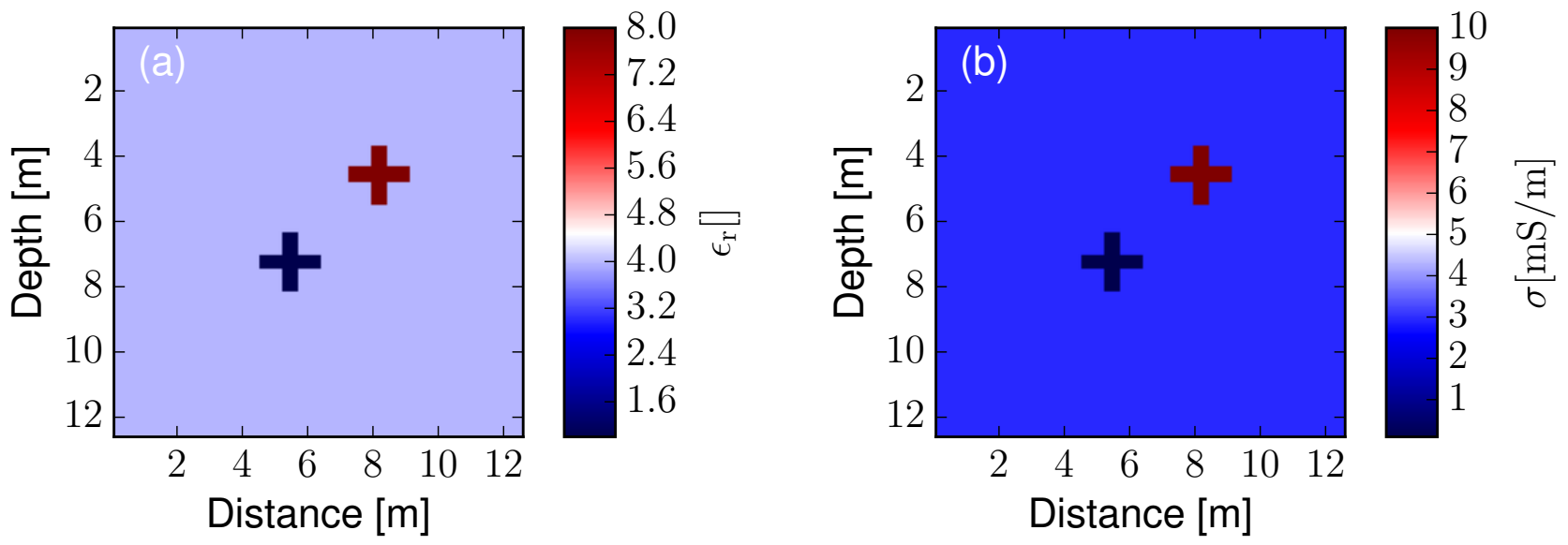
Jupyter notebook to calculate TD analytical solution at receiver positions

Jupyter notebook to compare FDFD solution with analytical solution in TD

Modelling example: 2-cross model

Modelling example: 2-cross model

2-cross model [Meles et al., 2011, Lavoué et al., 2014]

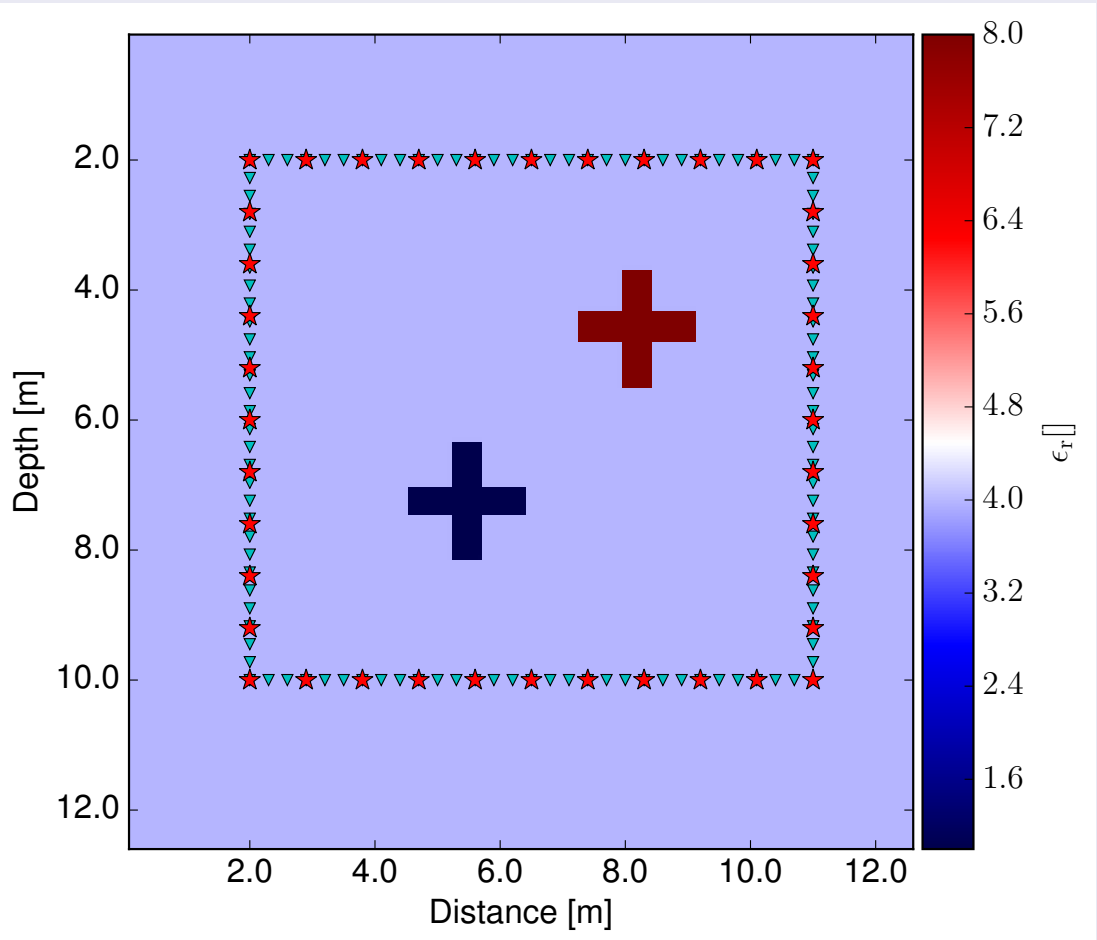


Model discretization:

- $\mathrm{NX} = 180$ gridpoints $\times$ $\mathrm{NZ} = 180$ gridpoints
- $+$ 10 gridpoints PMLs on each side
- DH $= 7$ cm

Modelling example: 2-cross model

Perfect illumination acquisition geometry



- 40 shots and 117 receivers
- 7 frequencies $[50, 60, 70, 80, 100, 150, 200]$ MHz

Modelling example: 2-cross model

Running GERMAINE for the 2-cross model ...

1. In GERMAINE/par/GERMAINE_2_cross_TE.inp set the following parameters for a forward modelling run of the 2-cross model

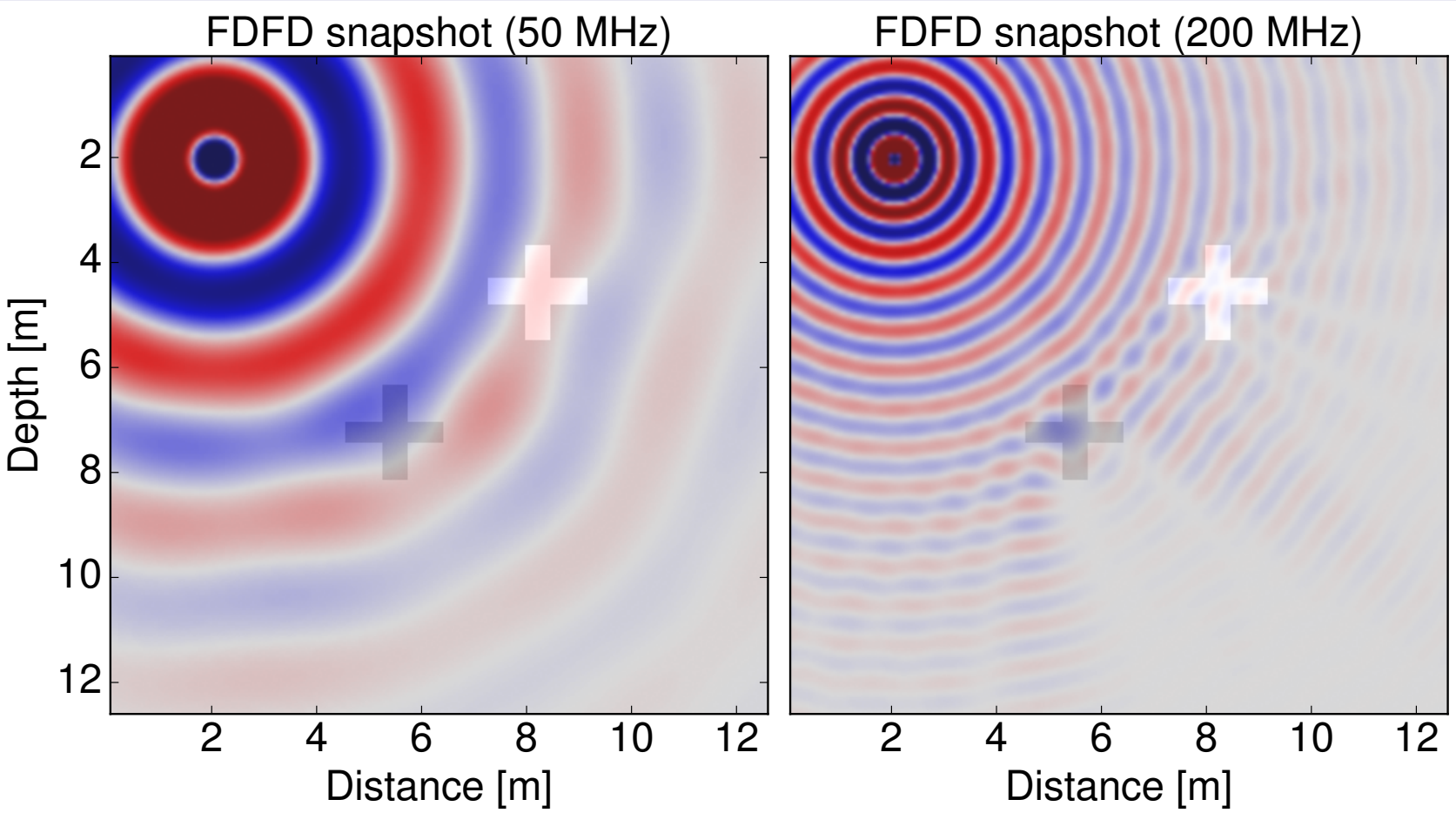
```
forward_modelling_(yes=0)_FDFWI_(yes=1)_RTM_(yes=2)_(INVMAT) = 0
MFILE = start/2_cross_TE_true
output_of_P_field_(SNAP)(yes>0) = 1
```

2. Run GERMAINE forward modelling on e.g. 4 cores of your CPU from the /par directory:

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE.inp GERMAINE_workflow_2_cross_TE_1.inp
```

Modelling example: 2-cross model

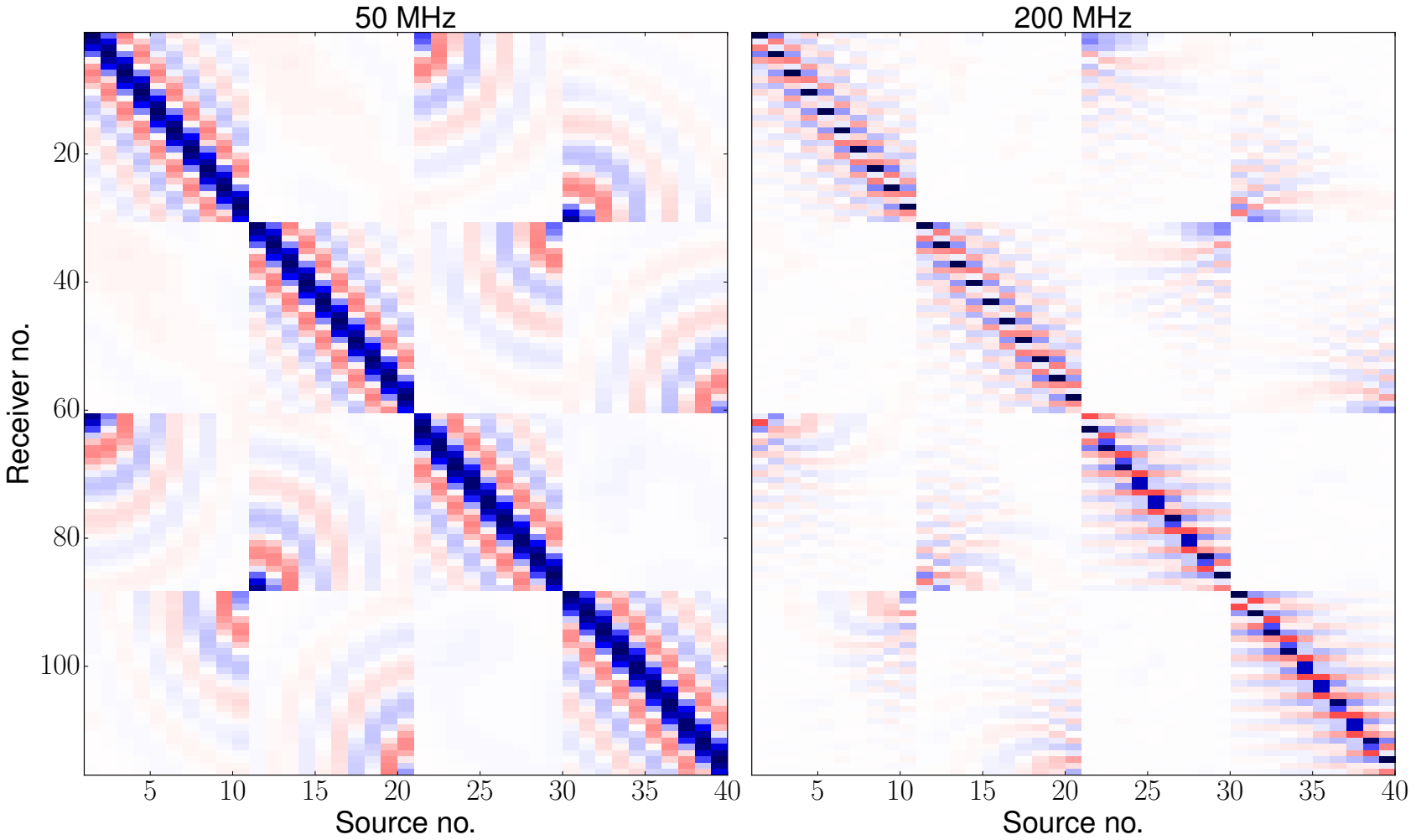
Frequency domain snapshots $\Re(\mathrm{E_y})$



Jupyter notebook for frequency domain wavefield visualization

Modelling example: 2-cross model

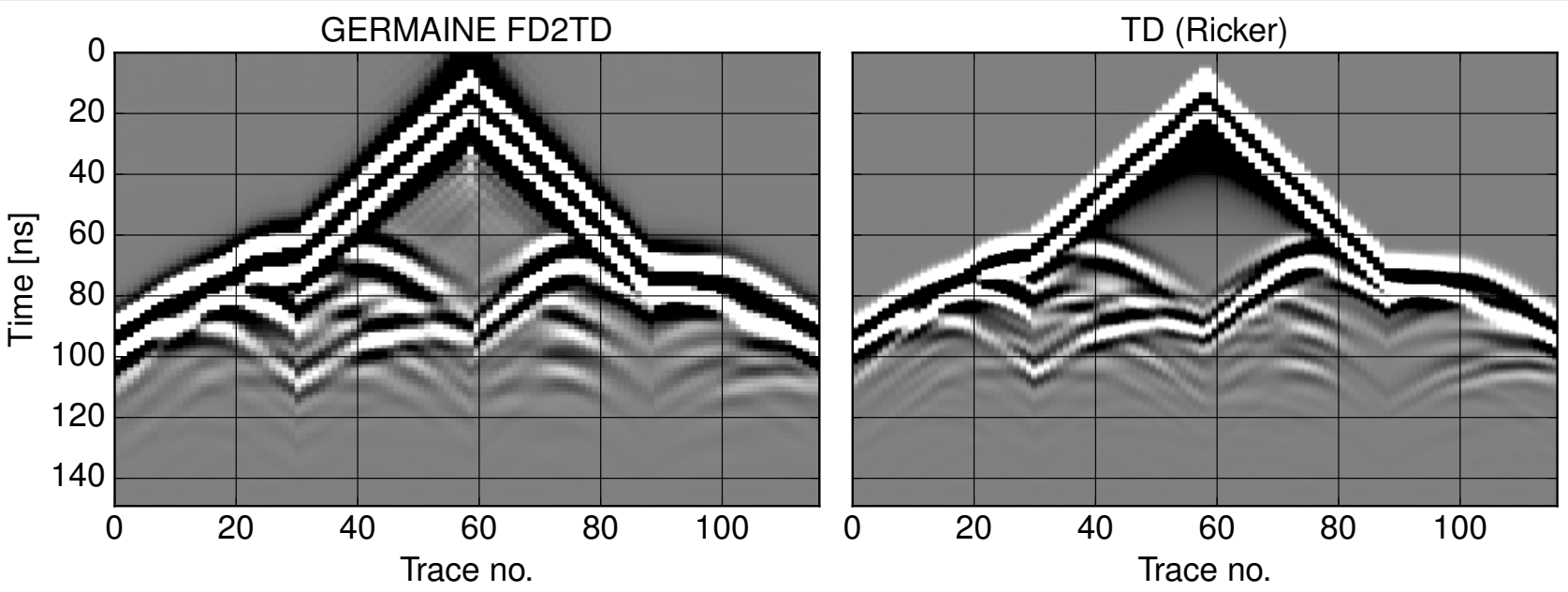
Frequency domain data $\Re(\mathrm{E_y})$



Jupyter notebook for frequency domain data visualization

Modelling example: 2-cross model

GERAMINE FDFD vs. TD modelling [Irving and Knight, 2006]

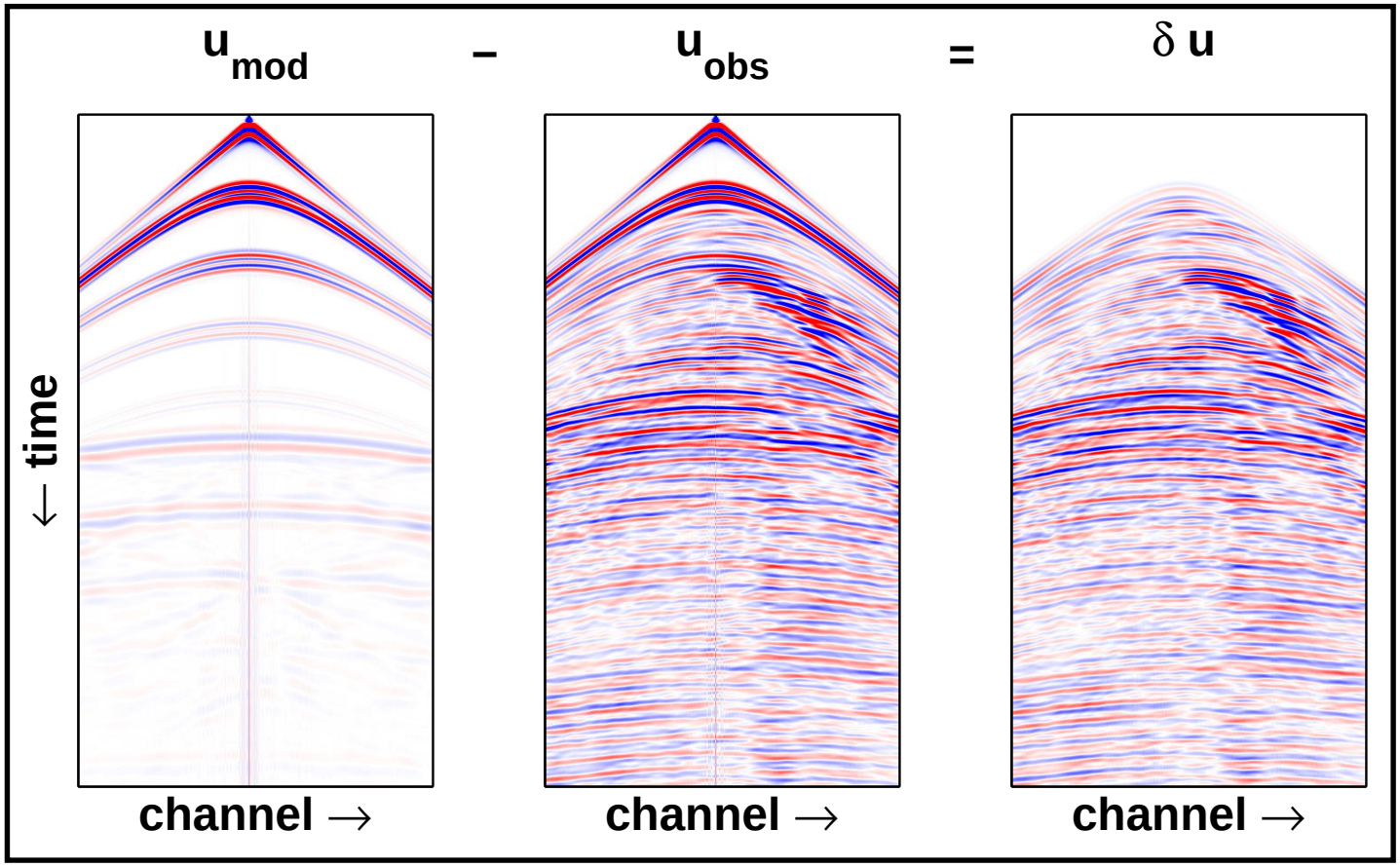


Jupyter notebook for FD + TD data comparison

Full Waveform Inversion

Full waveform inversion

1. Define objective function:

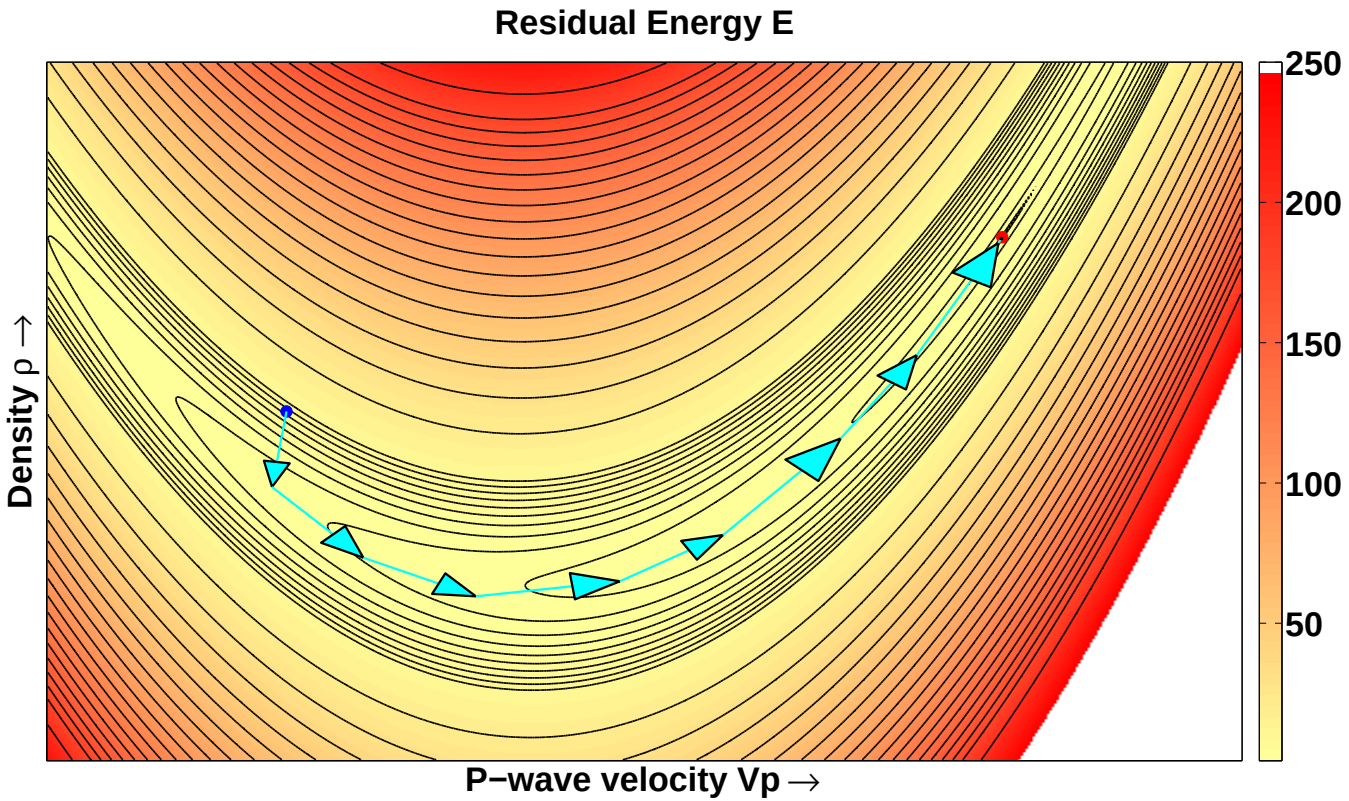


$$\mathrm{E} = \frac{1}{2}\delta\mathbf{r}^{\dagger}\delta\mathbf{r} \text{ with } \delta\mathbf{r} = \mathbf{u}^{\mathbf{mod}} - \mathbf{u}^{\mathbf{obs}}$$

$\mathbf{u}^{\mathbf{mod}}, \mathbf{u}^{\mathbf{obs}}$ modelled and field data (time/frequency domain)

Full waveform inversion

2. Minimize objective function by **Newton** optimization method



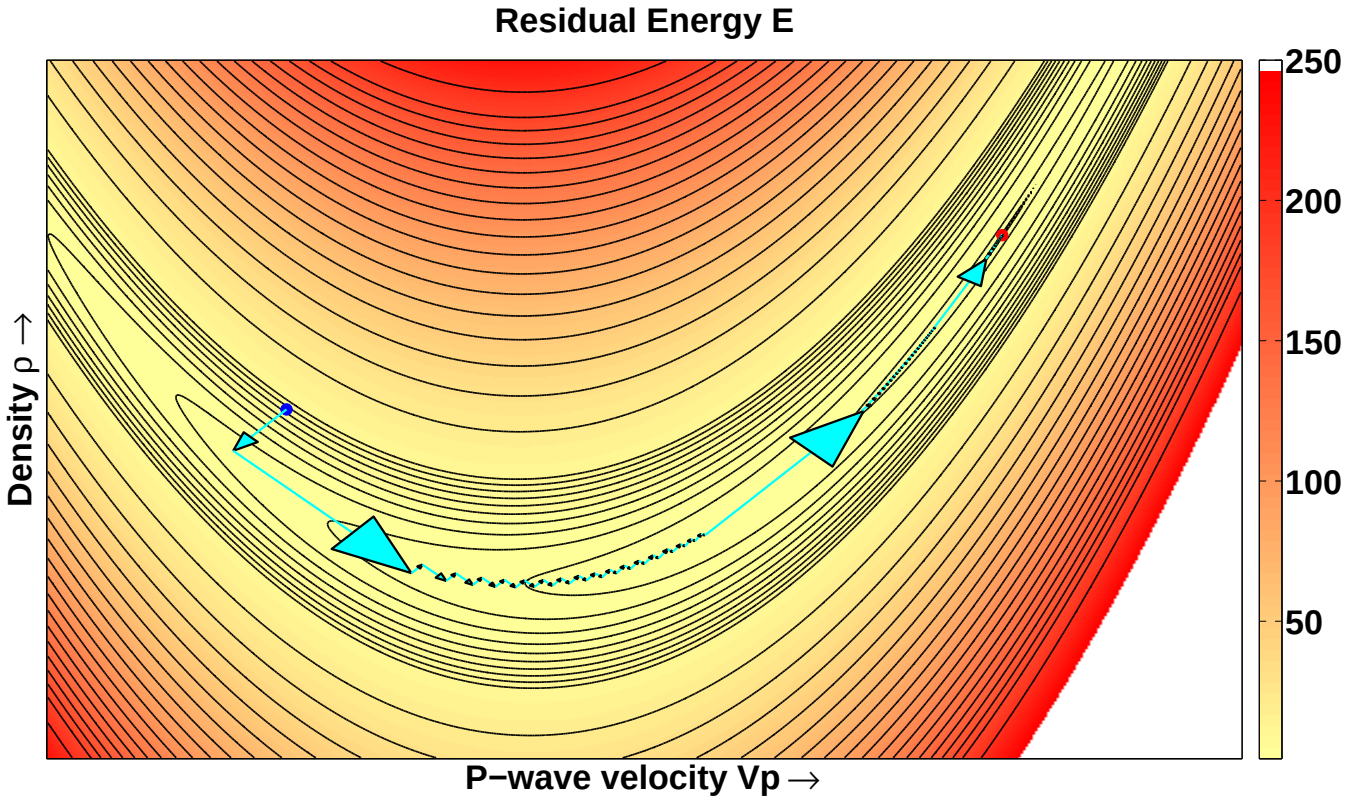
for the model parameters $\mathbf{m} = [\sigma, \epsilon]$

$$\mathbf{m}^{n+1} = \mathbf{m}^{n} - \mu^{n} \left(\mathbf{H}^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{n}$$

with gradient $\partial \mathbf{E}/\partial \mathbf{m}$, Hessian $\mathbf{H}$ and μ step-length

Full waveform inversion

2. Minimize objective function by **Steepest Descent** method



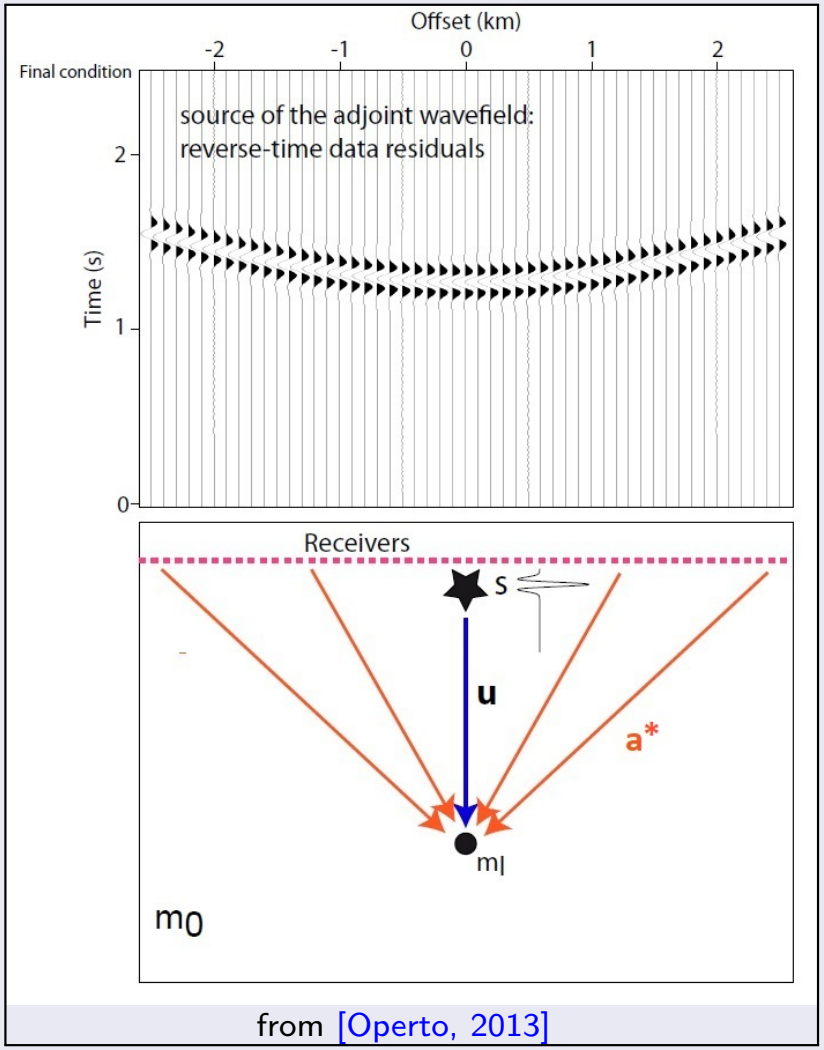
for the model parameters $\mathbf{m} = [\sigma, \epsilon]$

$$\mathbf{m}^{n+1} = \mathbf{m}^{n} - \mu^{n} \left(\frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{n}$$

with gradient $\partial \mathbf{E} / \partial \mathbf{m}$ and μ step-length

Full waveform inversion

Gradient computation



from [Operto, 2013]

- Efficient gradient calculation by the adjoint-state method:

$$\frac{\partial E}{\partial m_l} = -\Re\left\{ \mathbf{u}^t \left(\frac{\partial \mathbf{A}}{\partial m_l} \right)^t \mathbf{a}^* \right\}$$

with

u the forward wavefield

a the adjoint wavefield

A the impedance matrix

$\mathrm{m_l}$ the material parameter at grid point l

[Brossier and Virieux, 2011]

Basic FWI parameters in GERMAINE_2_cross_TE.inp

- **Maximum number of iterations per FWI stage:**

```
#-----------------------------------------
# GERMAINE inversion parameters
# -----------------------------------------
number_of_TDFWI_iterations_(ITERMAX) = 1000
```

- **Location of measured frequency-domain data:**

```
measured_FD_data_(DATA_DIR) = seis/2_cross_TE_true/2_cross_TE
```

- **Location of FWI results:**

```
# ----- Output of inverted models ----- #
output_of_models_(INV_MODELFILE) = model/modelTest
```

Basic FWI parameters in GERMAINE_workflow_2_cross_TE.inp

- You can define a simple FWI workflow similar to [Lavoué et al., 2014]

```
PRO     FC_low     FC_high     NF   ...
0.0     50e6       200e6       1    ...
0.0     60e6       200e6       1    ...
0.0     70e6       200e6       1    ...
0.0     80e6       200e6       1    ...
0.0     100e6      200e6       1    ...
0.0     150e6      200e6       1    ...
0.0     200e6      200e6       1    ...
```

which sequentially inverts the 50 MHz, 60 MHz, 70 MHz, 80 MHz, 100 MHZ, 150 MHz and 200 MHz data, respectively.

GERMAINE FWI example: 2-cross model

GERMAINE FWI example: 2-cross model

1. First, we have to create some synthetic field data for the 2-cross model. In GERMAINE/par/GERMAINE_2_cross_TE.inp set the following parameters for a forward modelling run of the 2-cross model

```
forward_modelling_(yes=0)_FDFWI_(yes=1)_RTM_(yes=2)_(INVMAT) = 0
MFILE = start/2_cross_TE_true
output_of_P_field_(SNAP)(yes>0) = 0
```

2. Run GERMAINE forward modelling on e.g. 4 cores of your CPU from the /par directory:

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE.inp GERMAINE_workflow_2_cross_TE.inp
```

3. In GERMAINE/par/seis generate the directory 2_cross_TE_true

```
mkdir 2_cross_TE_true
```

4. Move the FDFD data of the true model from /seis to /seis/2_cross_TE_true

```
mv 2_cross_TE_p_stage_* 2_cross_TE_true/
```

GERMAINE FWI example: 2-cross model

5. To run the TE FDFD FWI change the following parameters in GERMAINE/par/GERMAINE_2_cross_TE.inp

```
forward_modelling_(yes=0)_FDFWI_(yes=1)_RTM_(yes=2)_(INVMAT) = 1
MFILE = start/2_cross_TE_init
```

6. Start the 2D TE FDFD FWI with:

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE.inp GERMAINE_workflow_2_cross_TE.inp
```

As defined in GERMAINE_workflow_2_cross_TE.inp the FWI is based on a sequential inversion workflow using the 50 MHz, 60 MHz, 70 MHz, 80 MHz, 100 MHz, 150 MHz and 200 MHz FD-data, respectively. The permittivity and conductivity models of the current iteration are saved in:

model/modelTest_eps.bin

model/modelTest_sig.bin

GERMAINE FWI example: 2-cross model

7. Depending on the clock speed of your CPU and number of cores wait approximately a few hours ...

   The intermediate results after the FWI of each stage are saved separatly in

   model/modelTest_eps_stage_1.bin
   model/modelTest_sig_stage_1.bin
   ...
   model/modelTest_eps_stage_7.bin
   model/modelTest_sig_stage_7.bin

GERMAINE FWI example: 2-cross model

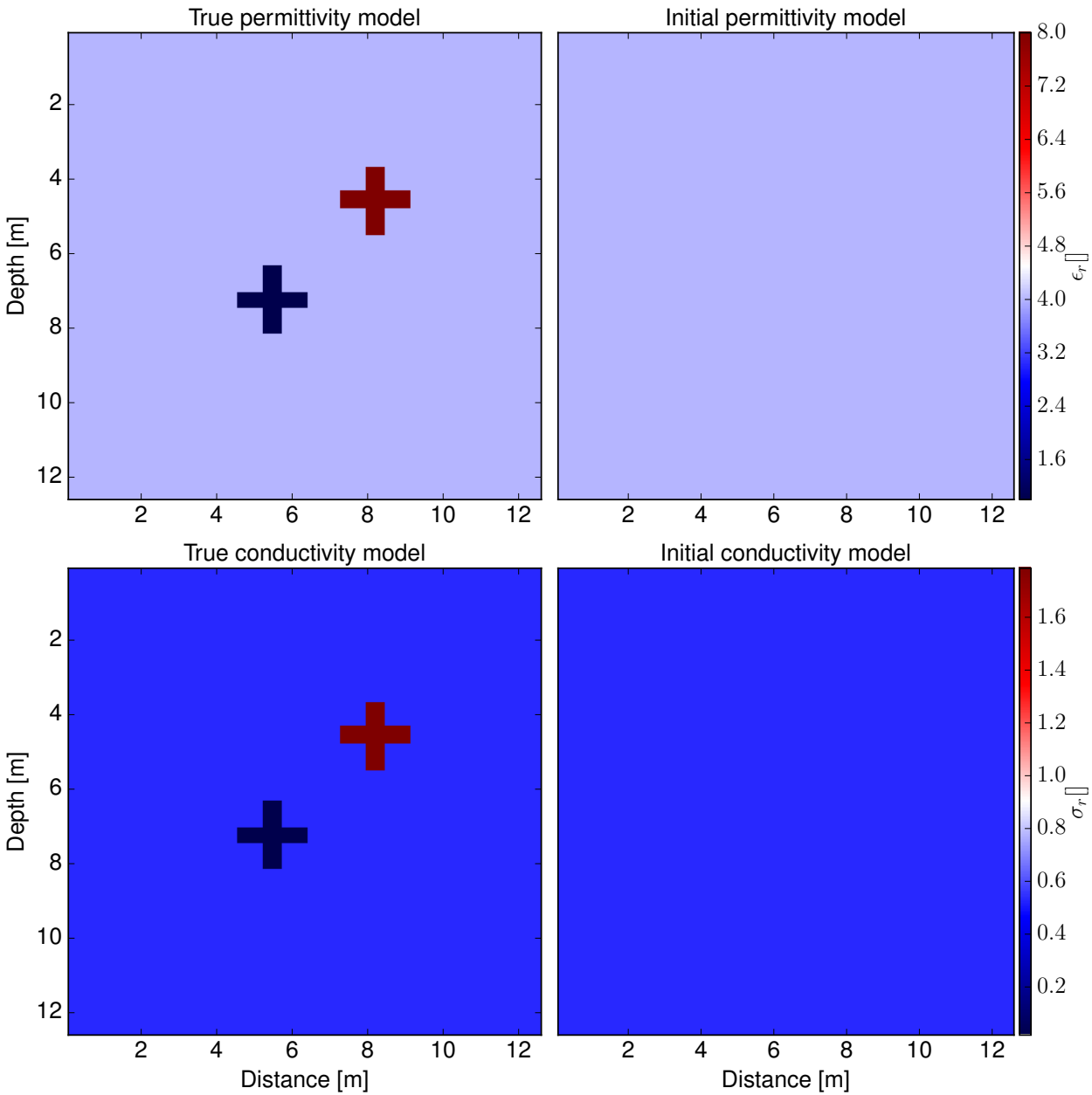
8. You can visualize the results with Seismic Unix

```
ximage n1=180 < model/modelTest_eps_stage_7.bin
```

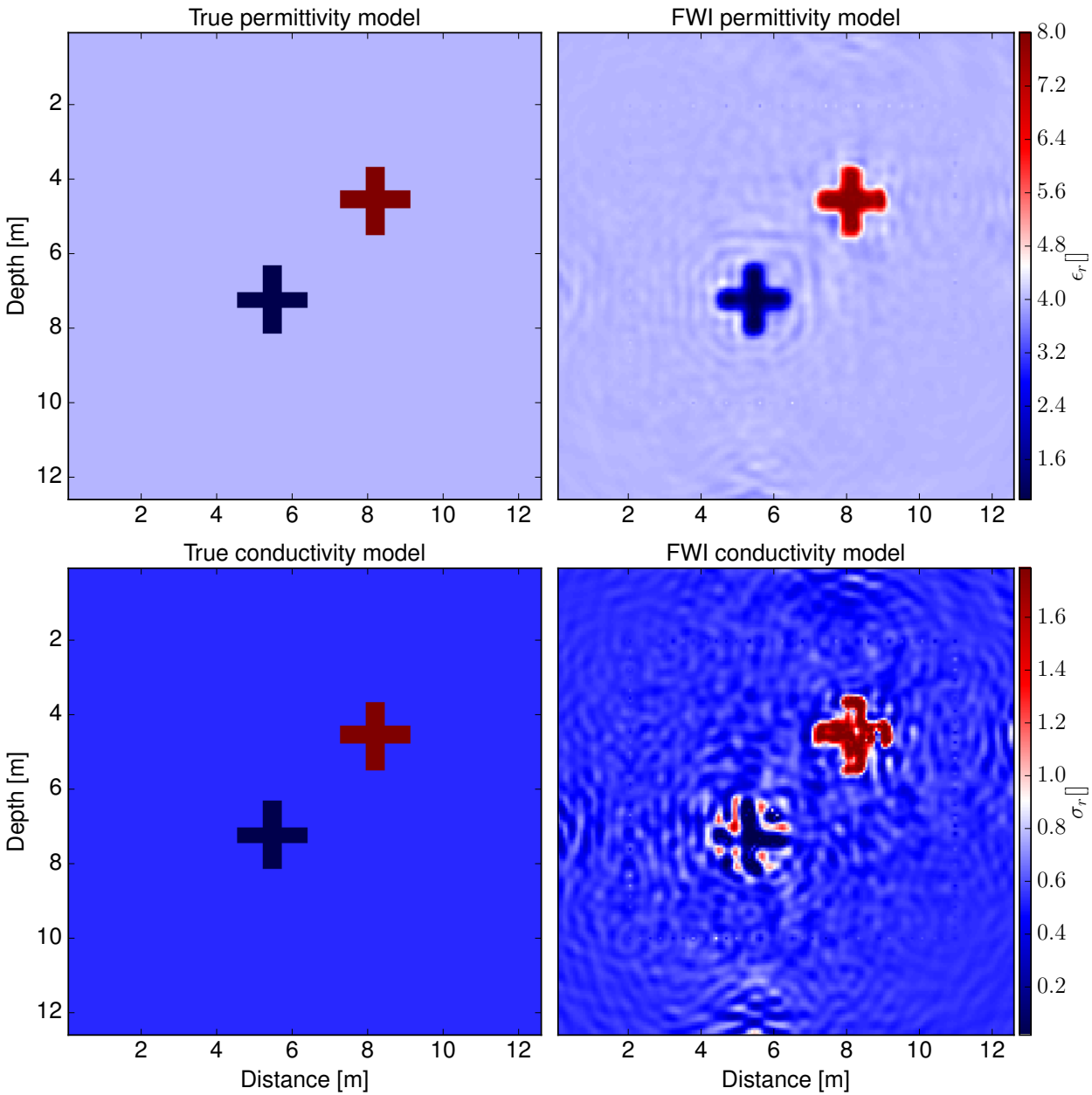
or use this Jupyter notebook

Jupyter notebook for FWI result visualization

FWI: initial model



FWI: result with optimized inversion parameters



GERMAINE FWI example: 2-cross model

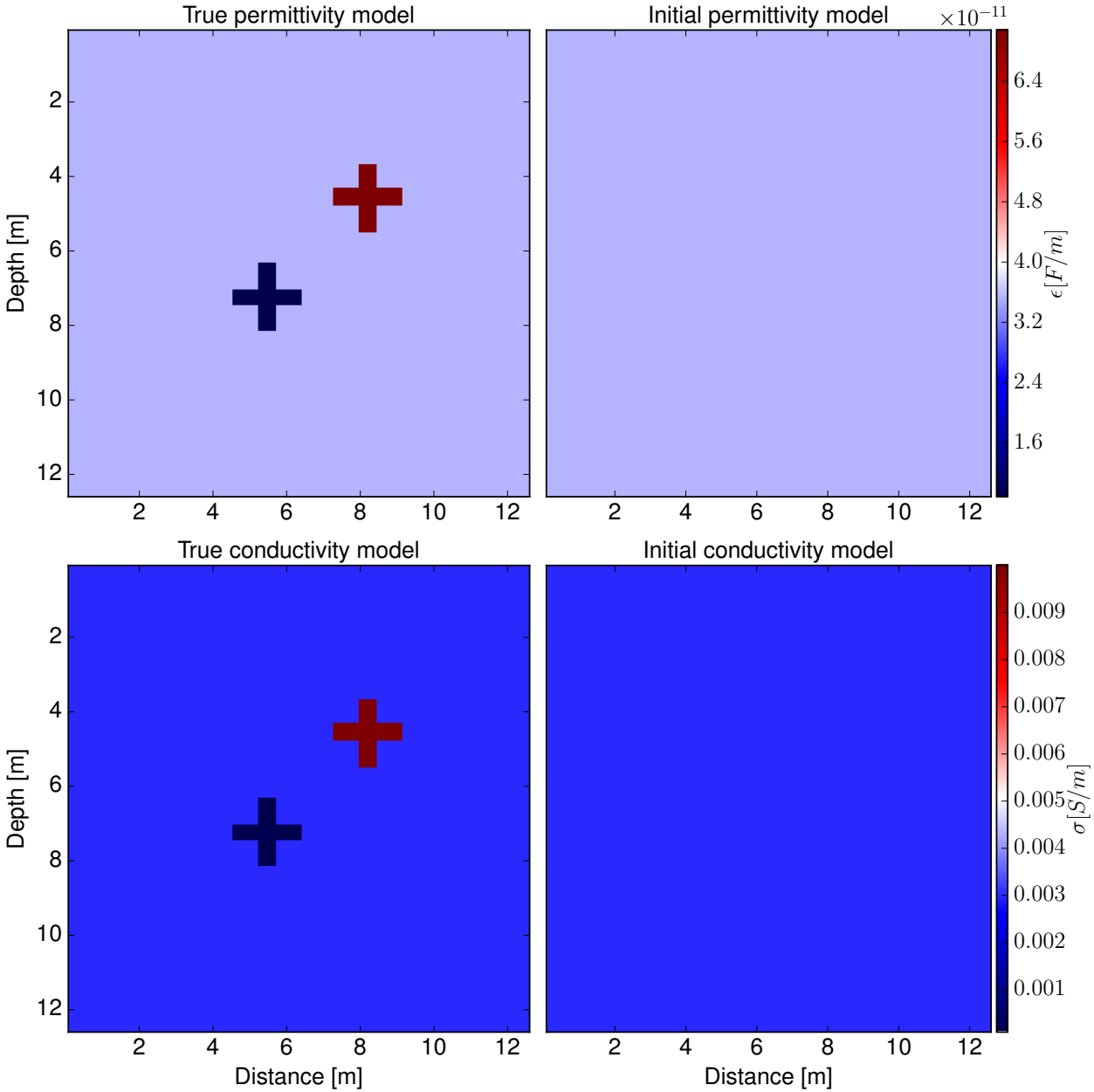
The FWI result might suggest, that multi-parameter FWI is a very easy task. However, its success depends on a lot of parameter tuning.

To discuss the influence of the different parameters on the FWI result, we start with a very simple, naive, non-optimized inversion based on steepest descent optimization:

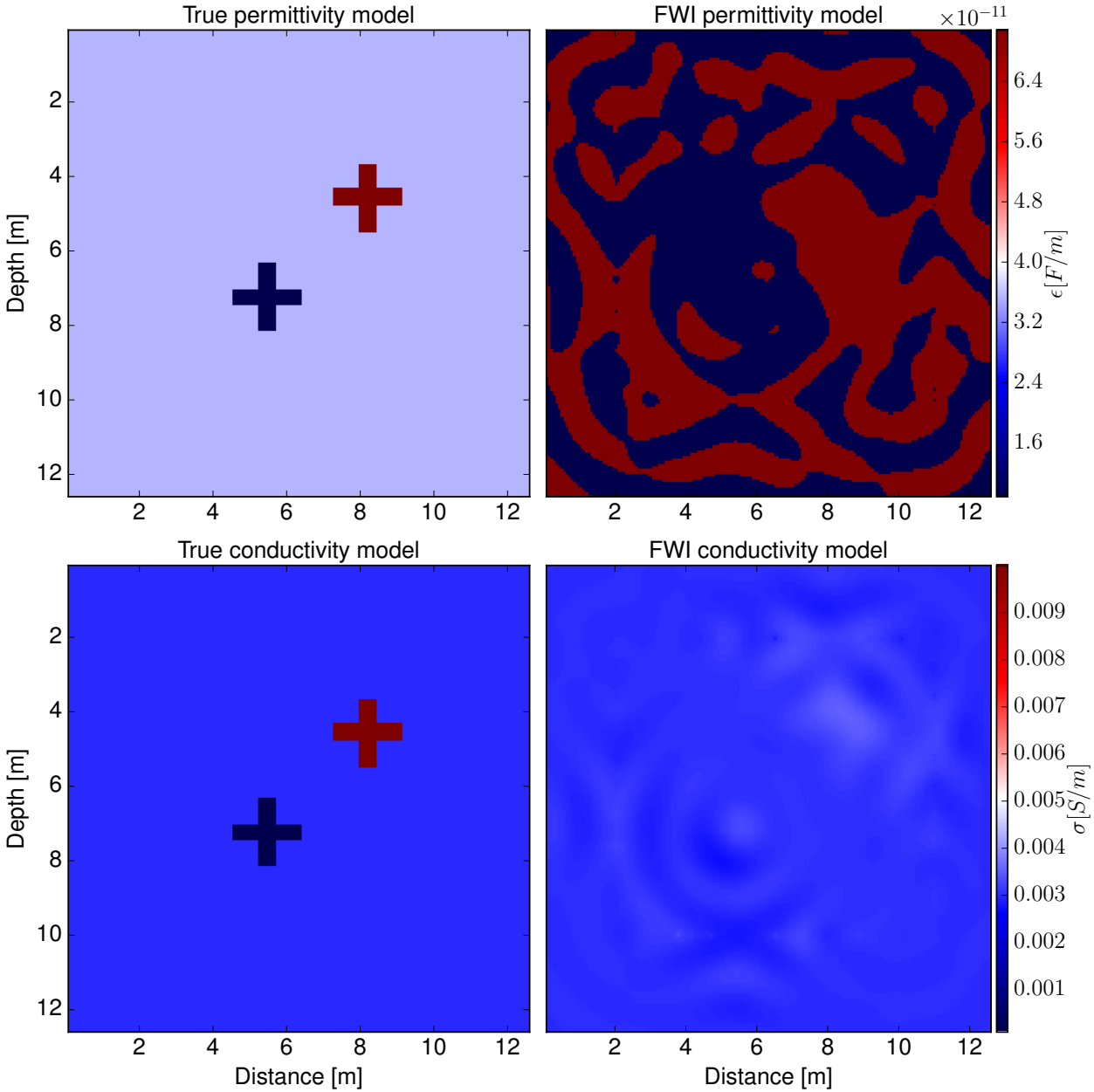
Run the naive 2D TE FDFD FWI with:

```
mpirun -np 4 ../bin/germaine GERMAINE_2_cross_TE_naive.inp GERMAINE_workflow_2_cross_TE_naive.inp
```

FWI: initial model



FWI: inversion result with steepest descent method



FWI of normalized material parameters

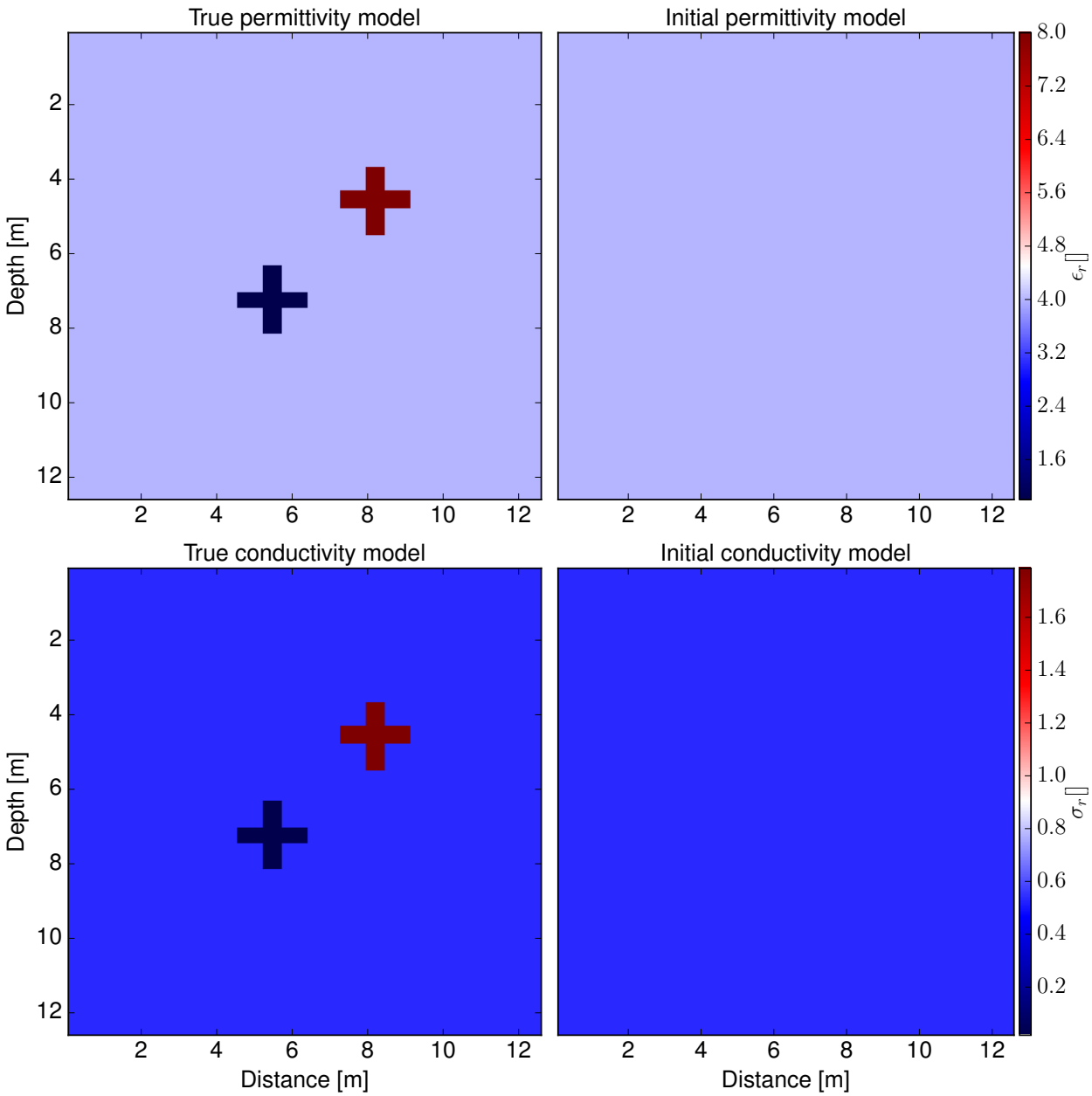
- Permittivity and conductivity models vary on different scales
  $\rightarrow$ FWI fails to converge

- **Idea:**
  Invert normalized material parameters for FWI
  $\epsilon_\mathrm{r} = \epsilon / \epsilon_0$,
  $\sigma_\mathrm{r} = \sigma / \sigma_0$
  with $\epsilon_0 = 8.85 \cdot 10^{-12}\ \mathrm{F/m}$ and $\sigma_0 = 5.6 \cdot 10^{-3}\ \mathrm{S/m}$

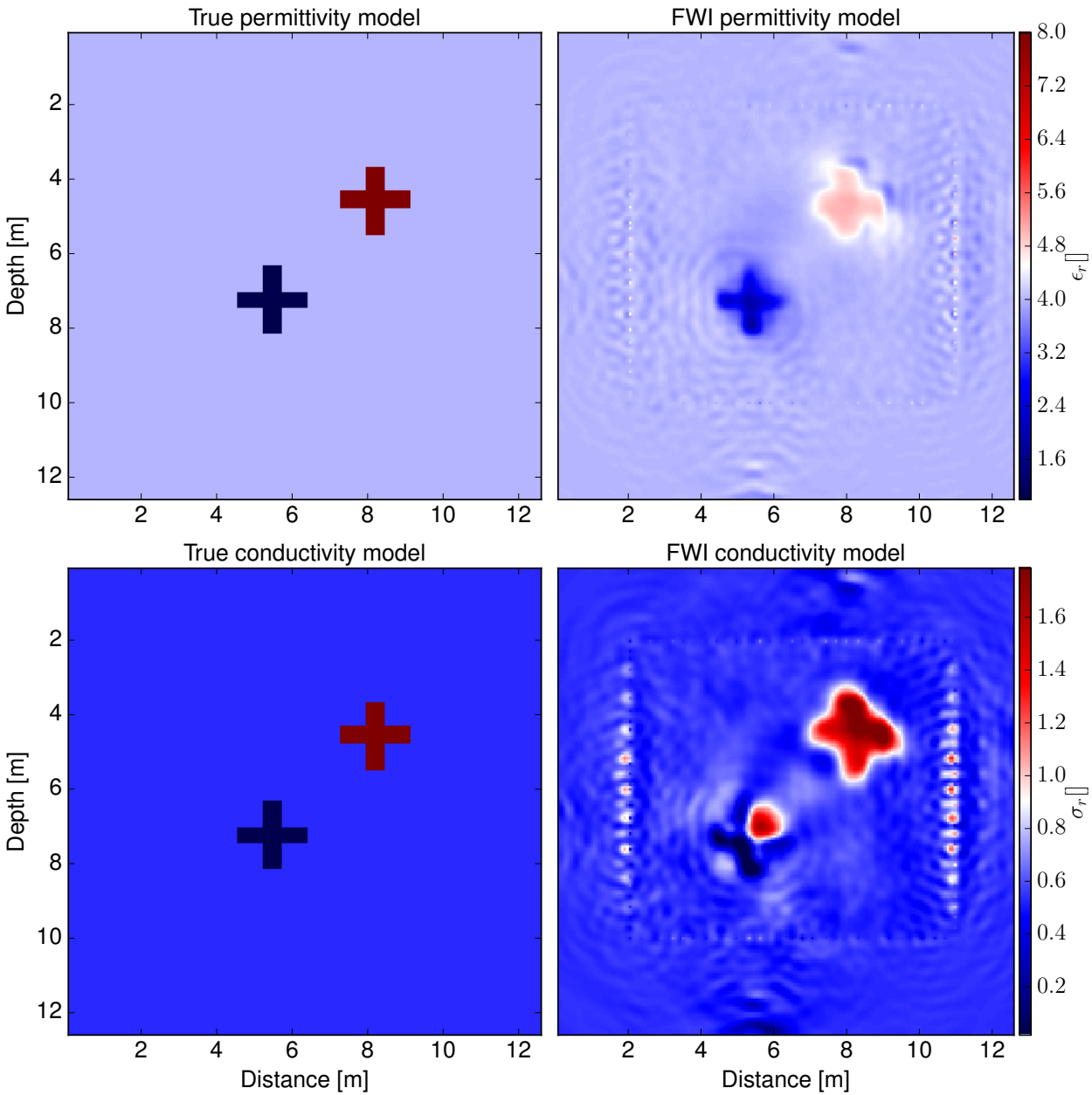
- Definition in GERMAINE parameter file:

```
# -- Normalization factor for material parameters --
norm_sigma_(MAT1_NORM) = 5.6e-3
norm_epsilon_(MAT2_NORM) = 8.85e-12
```

FWI: initial normalized parameters

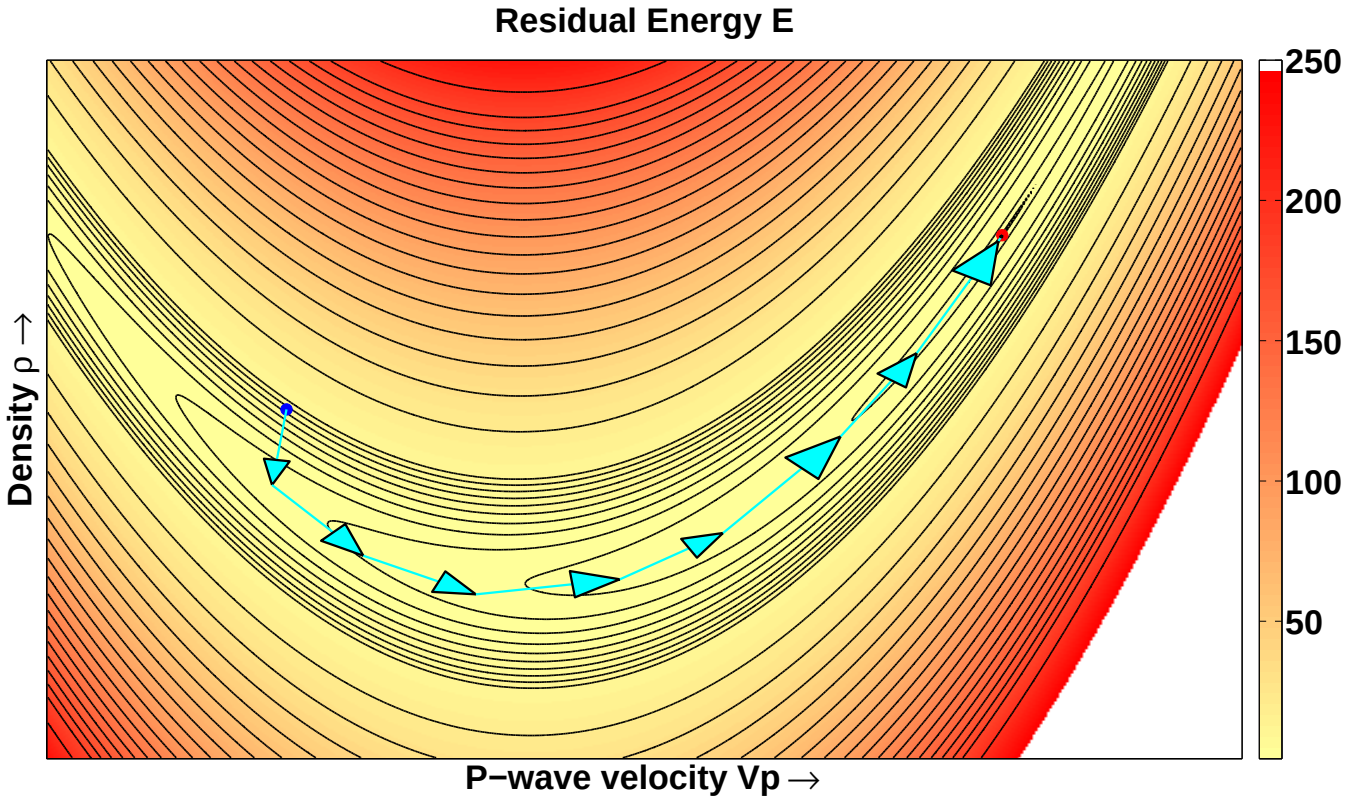


FWI results using normalized parameters



Preconditioning gradient

2. Minimize objective function by **Newton** optimization method



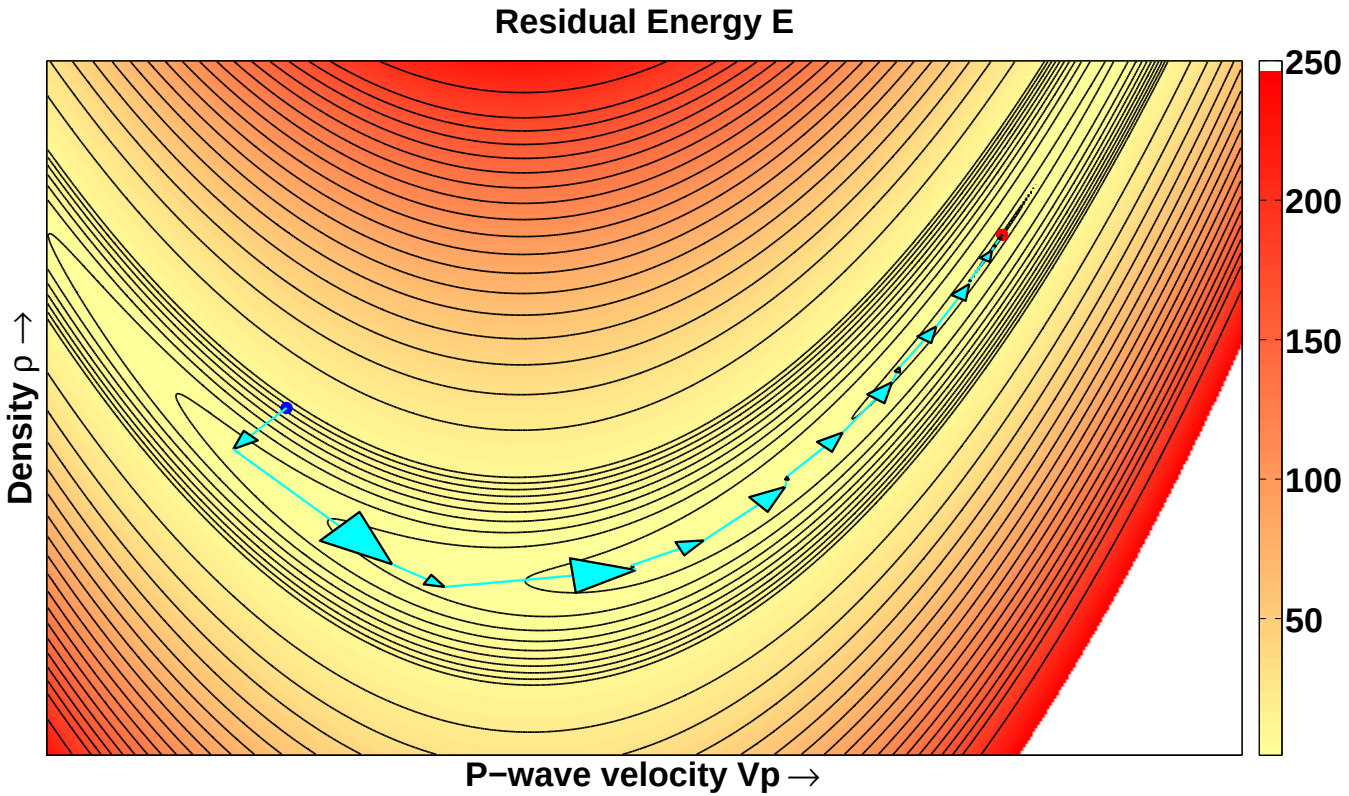
for the model parameters $\mathbf{m} = [\sigma, \epsilon]$

$$\mathbf{m}^{n+1} = \mathbf{m}^{n} - \mu^{n} \left(\mathbf{H}^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{n}$$

with gradient $\partial \mathbf{E}/\partial \mathbf{m}$, Hessian $\mathbf{H}$ and μ step-length

Preconditioning gradient

2. Minimize objective function by **Preconditoned Gradient**

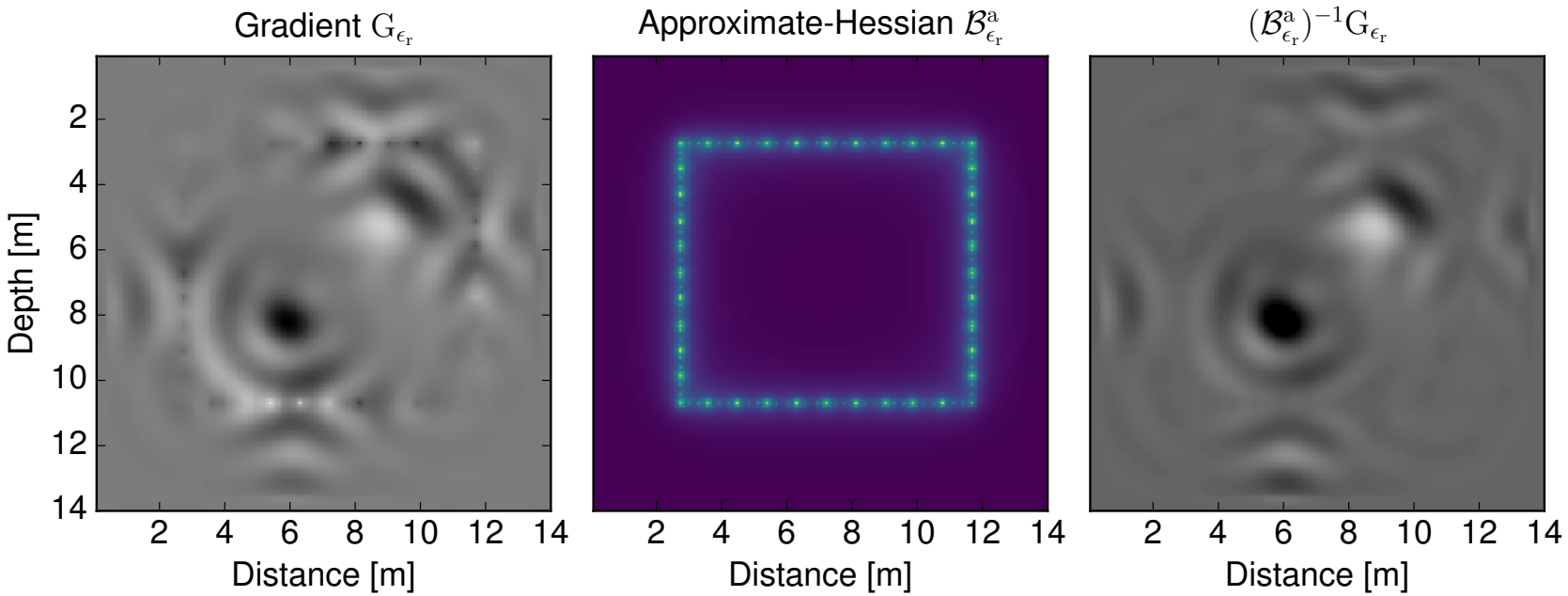


for the model parameters $\mathbf{m} = [\sigma, \epsilon]$

$$\mathbf{m}^{n+1} = \mathbf{m}^{n} - \mu^{n} \left(\mathcal{B}^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{n}$$

with gradient $\partial \mathbf{E} / \partial \mathbf{m}$, Hessian approximation $\mathcal{B}$

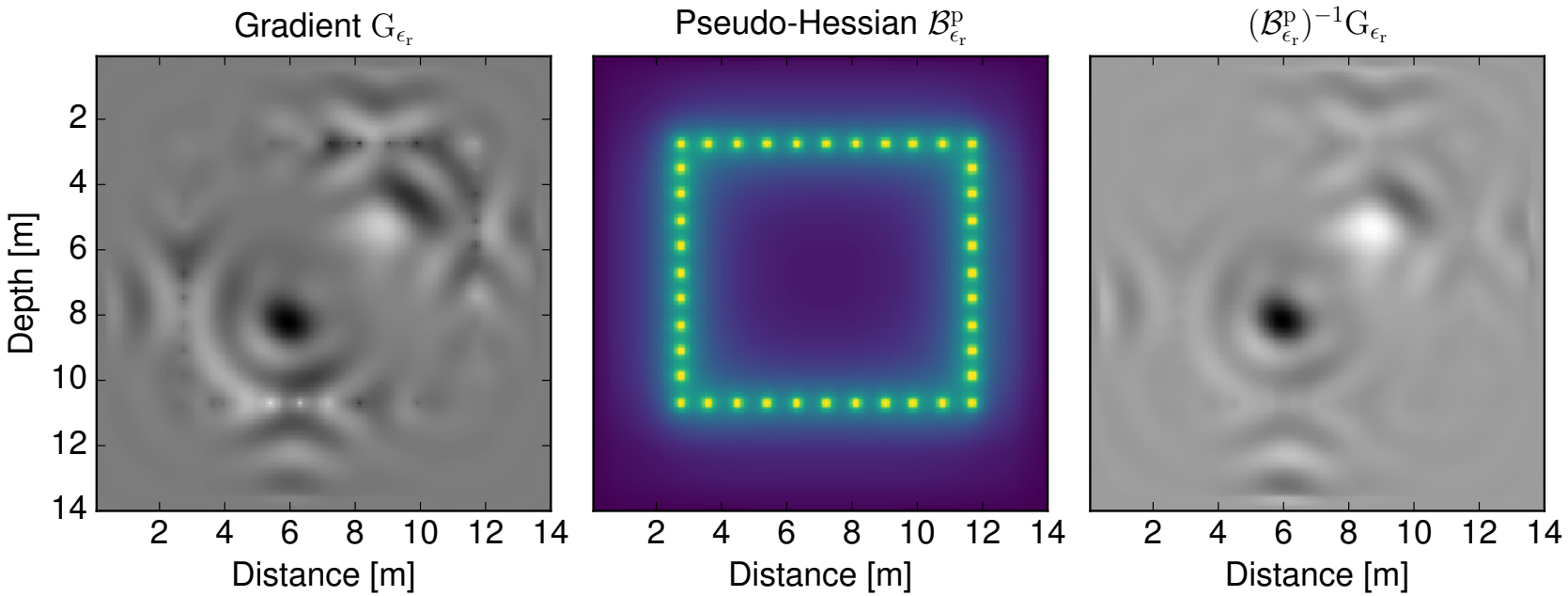
Preconditioning gradient with Approximate-Hessian



$$\mathcal{B}^{\mathrm{a}}_{\mathrm{kk}} = \sum_{\mathrm{s}} \sum_{\mathrm{r}} \left(\mathrm{u}^{\dagger}_{\mathrm{s}} \left(\frac{\partial \mathrm{A}}{\partial \mathrm{m}_{\mathrm{k}}} \right)^{\dagger} \mathrm{A}^{-1} \mathrm{R}^{\mathrm{t}}_{\mathrm{r}} \right) \left(\mathrm{R}_{\mathrm{r}} \mathrm{A}^{-1} \left(\frac{\partial \mathrm{A}}{\partial \mathrm{m}_{\mathrm{k}}} \right) \mathrm{u}_{\mathrm{s}} \right)$$

$$\mathbf{m}^{\mathrm{n}+1} = \mathbf{m}^{\mathrm{n}} - \mu^{\mathrm{n}} \left((\mathcal{B}^{\mathbf{a}} + \{\epsilon_{\mathrm{hess}} \max(|\mathcal{B}^{\mathbf{a}}|)\}\mathbf{I})^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{\mathrm{n}}$$

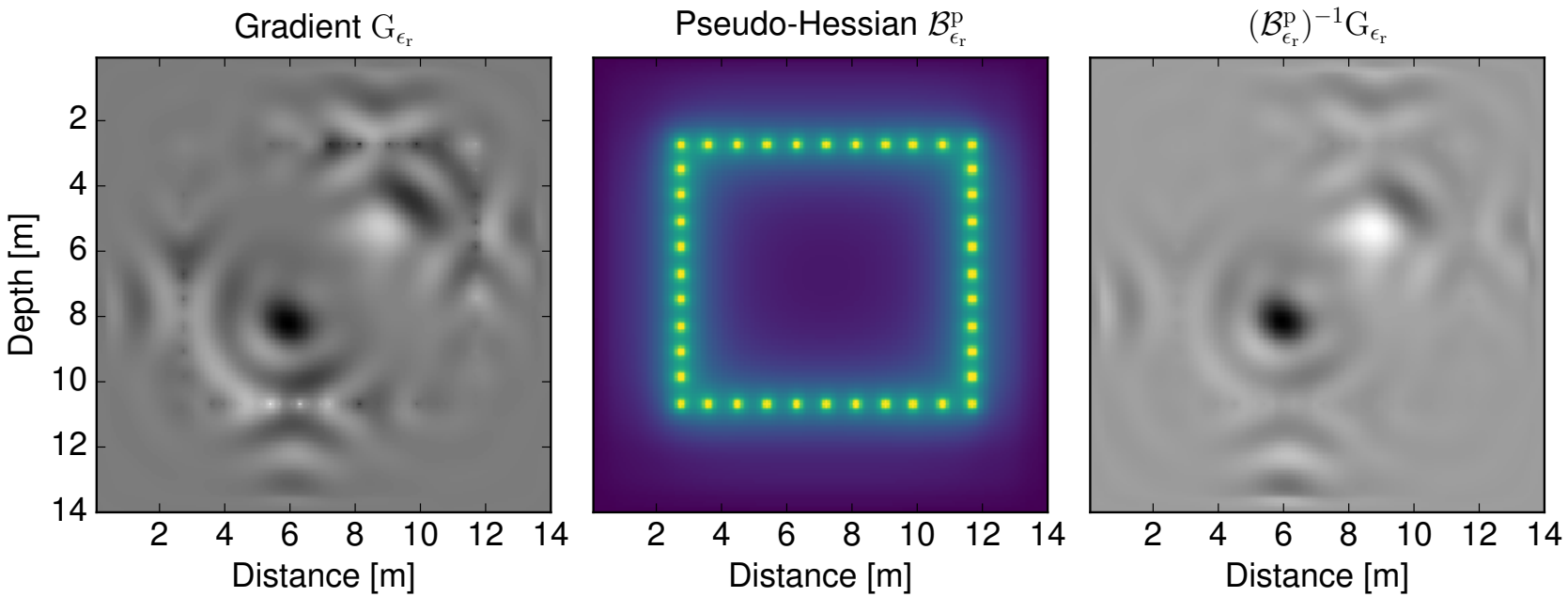
Preconditioning gradient with Pseudo-Hessian



$$\mathcal{B}^{\mathrm{p}}_{\mathrm{kk}} = \sum_{\mathrm{s}} \left(\left(\frac{\partial \mathrm{A}}{\partial \mathrm{m_k}} \right) \mathrm{u_s} \right)^{\dagger} \left(\left(\frac{\partial \mathrm{A}}{\partial \mathrm{m_k}} \right) \mathrm{u_s} \right)$$

$$\mathbf{m}^{\mathrm{n+1}} = \mathbf{m}^{\mathrm{n}} - \mu^{\mathrm{n}} \left((\mathcal{B}^{\mathbf{p}} + \{\epsilon_{\mathrm{hess}} \max(|\mathcal{B}^{\mathbf{p}}|)\}\mathbf{I})^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{\mathrm{n}}$$

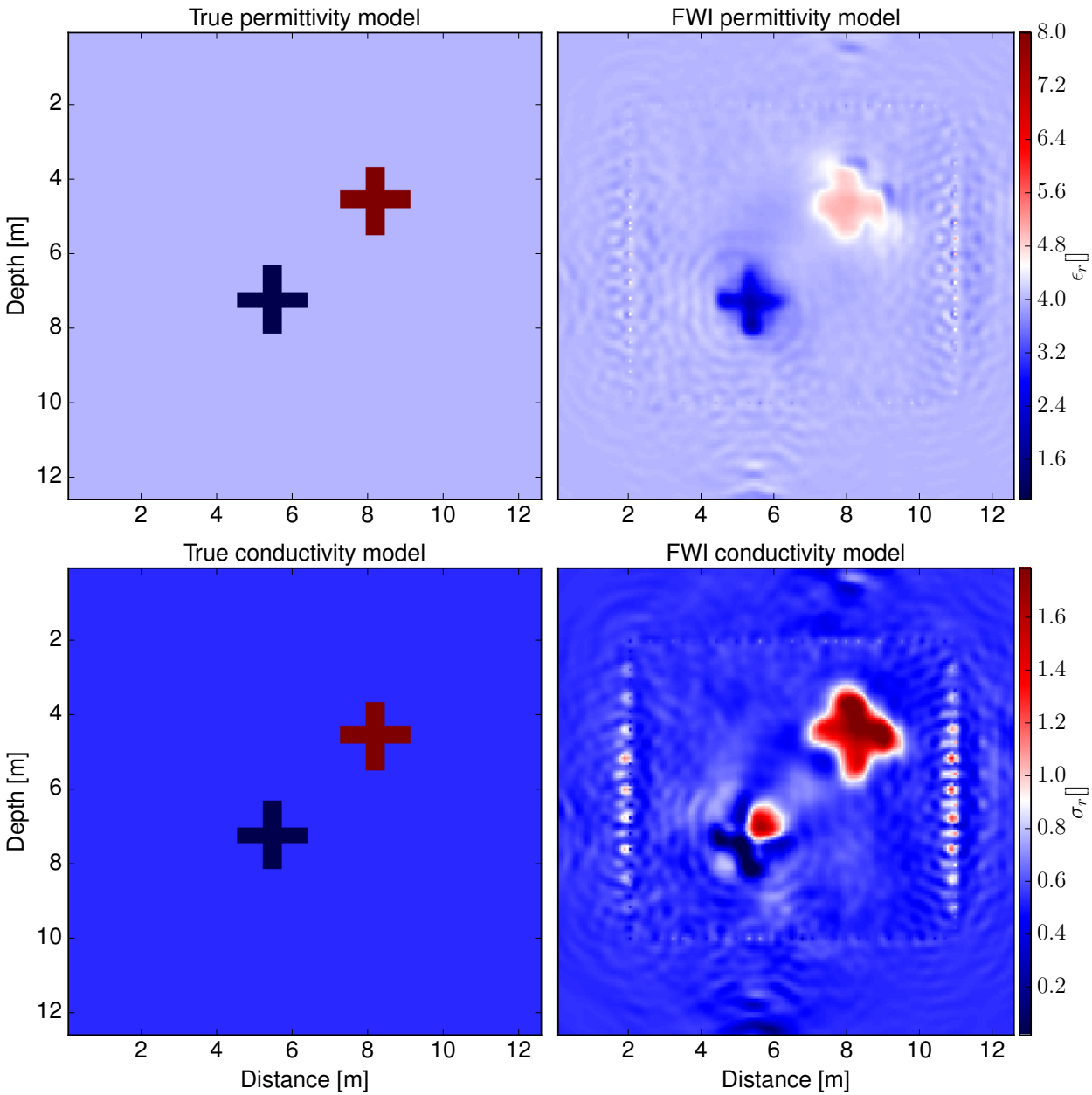
Preconditioning gradient with Pseudo-Hessian



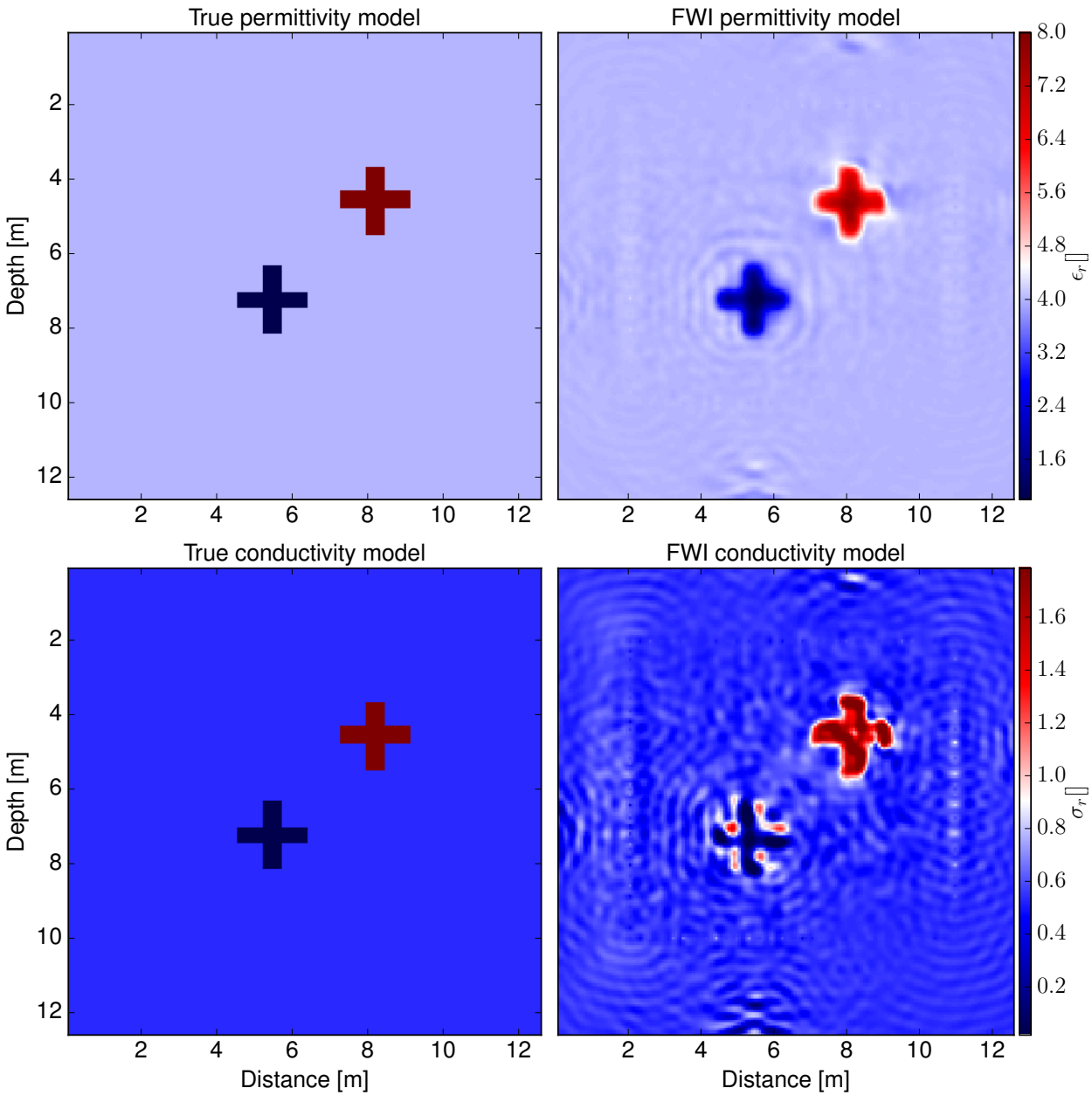
- Definition in GERMAINE parameter file:

```
HESSIAN_(no=0/app_Hessian=1/pseudo_Hessian=2) = 2
Regularization_term_for_inverse_Hessian_(EPS_HESS) = 0.01
```

FWI results using gradient method



FWI results using gradient with Pseudo-Hessian



Quasi-Newton l-BFGS optimization

So far we used the very inefficient steepest descent optimization ...

$$\mathbf{m}_{n+1} = \mathbf{m}_n + \mu_n \mathbf{\Delta m}_n$$

with the search direction:

$$\mathbf{\Delta m}_n = -\left(\frac{\partial \mathbf{E}}{\partial \mathbf{m}}\right)_n$$

or preconditioned search direction:

$$\mathbf{\Delta m}_n = -\left(\mathcal{B}^{-1}\frac{\partial \mathbf{E}}{\partial \mathbf{m}}\right)_n$$

Improve Hessian approximation by l-BFGS optimization

Quasi-Newton l-BFGS algorithm

[Nocedal and Wright, 2006, Métivier and Brossier, 2016]

At iteration step n:

1. Compute $\mathrm{g_n} = \left(\frac{\partial \mathrm{E}}{\partial \mathrm{m}}\right)_n$
2. Compute and store $s_n = m_{n+1} - m_n$
   Compute and store $y_n = g_{n+1} - g_n$
3. $q = -g_n$
4. for i = n-1 to n-l do
   $\rho_i = \frac{1}{y_i^T s_i}$
   $\alpha_i = \rho_i s_i^T q$
   $q = q - \alpha_i y_i$
   end for
5. Compute $H_n^0 = \frac{s_{n-1}^T y_{n-1}}{y_{n-1}^T y_{n-1}}$
6. Compute $z = H_n^0 q$
7. for i = n-l to n-1 do
   $\beta_i = \rho_i y_i^T z$
   $z = z + s_i(\alpha_i - \beta_i)$
   end for
8. $\Delta m_n = z$
9. Update model $m_{n+1} = m_n + \mu_n \Delta m_n$

Quasi-Newton PI-BFGS optimization

So far we used the very inefficient steepest descent optimization ...

$$\mathbf{m}_{n+1} = \mathbf{m}_n + \mu_n \mathbf{\Delta m}_n$$

with the search direction:

$$\mathbf{\Delta m}_n = -\left(\frac{\partial \mathbf{E}}{\partial \mathbf{m}}\right)_n$$

or preconditioned search direction:

$$\mathbf{\Delta m}_n = -\left(\mathcal{B}^{-1}\frac{\partial \mathbf{E}}{\partial \mathbf{m}}\right)_n$$

Improve Hessian approximation by PI-BFGS optimization

Preconditioned Quasi-Newton PI-BFGS algorithm

[Nocedal and Wright, 2006, Métivier and Brossier, 2016]

At iteration step n:

1. Compute $g_n = \left(\frac{\partial E}{\partial m}\right)_n$
2. Compute and store $s_n = m_{n+1} - m_n$
   Compute and store $y_n = g_{n+1} - g_n$
3. $q = -g_n$
4. for i = n-1 to n-l do
   $\rho_i = \frac{1}{y_i^T s_i}$
   $\alpha_i = \rho_i s_i^T q$
   $q = q - \alpha_i y_i$
   end for
5. Compute $H_n^0 = (\mathcal{B}_n)^{-1}$
6. Compute $z = H_n^0 q$
7. for i = n-l to n-1 do
   $\beta_i = \rho_i y_i^T z$
   $z = z + s_i(\alpha_i - \beta_i)$
   end for
8. $\Delta m_n = z$
9. Update model $m_{n+1} = m_n + \mu_n \Delta m_n$

Quasi-Newton l-BFGS optimization

- Definition in GERMAINE parameter file

```
# ----- Optimization-Method ------ #
gradient_method_(PCG=1/LBFGS=2/Descent=3)_(GRAD_METHOD) = 2
PCG_BETA_(Fletcher_Reeves=1/Polak_Ribiere=2/Hestenes_Stiefel=3/Dai_Yuan=4) = 4
save_(NLBFGS)_updates_during_LBFGS_optimization = 5
```

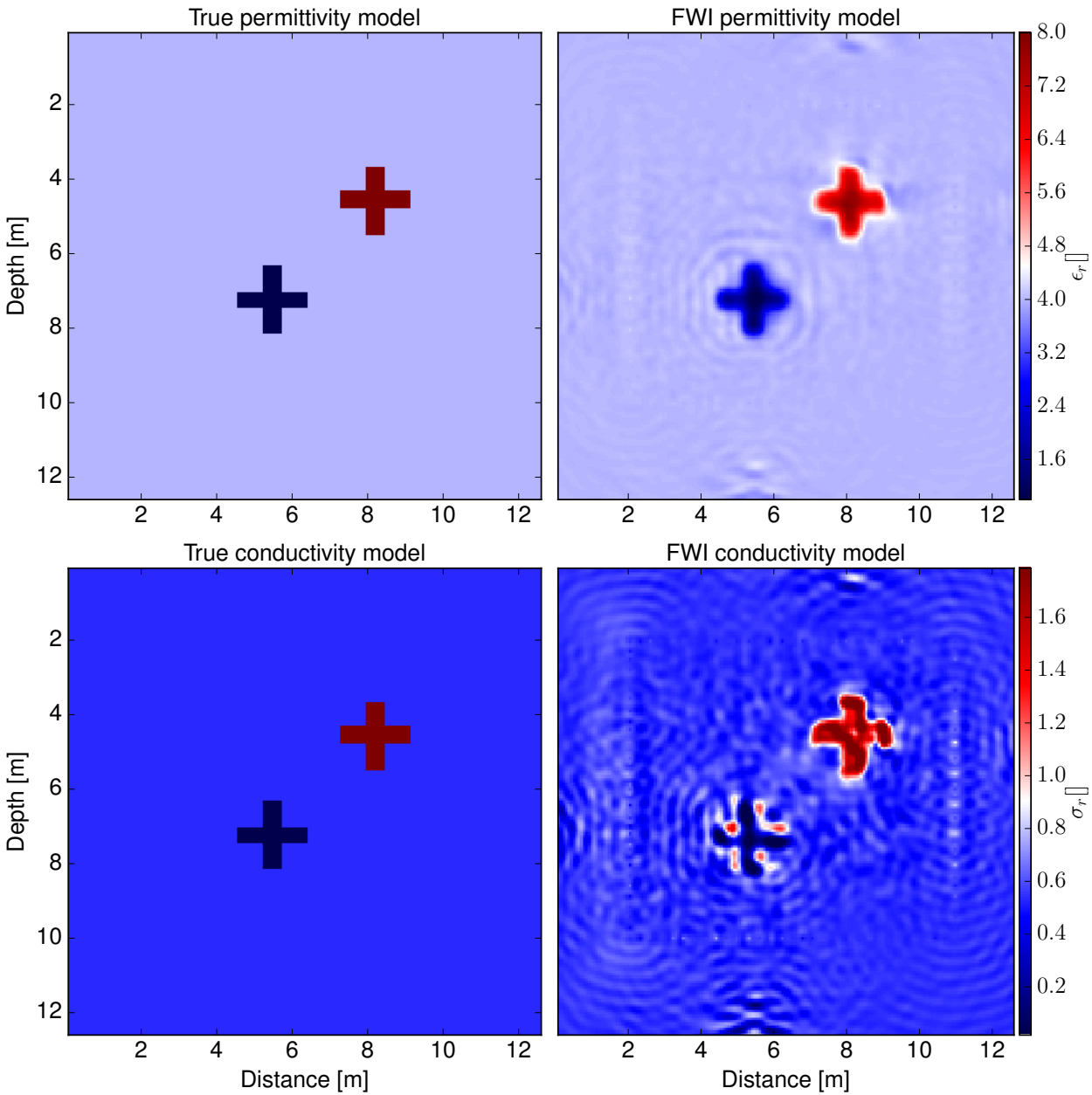
- NLBFGS defines how many $\mathrm{s_n}$ and $\mathrm{y_n}$ values are stored during l-BFGS optimization

- Preconditioned quasi-Newton Pl-BFGS optimization is used if the Approximate-Hessian (HESSIAN=1) or Pseudo-Hessian (HESSIAN=2) is activated

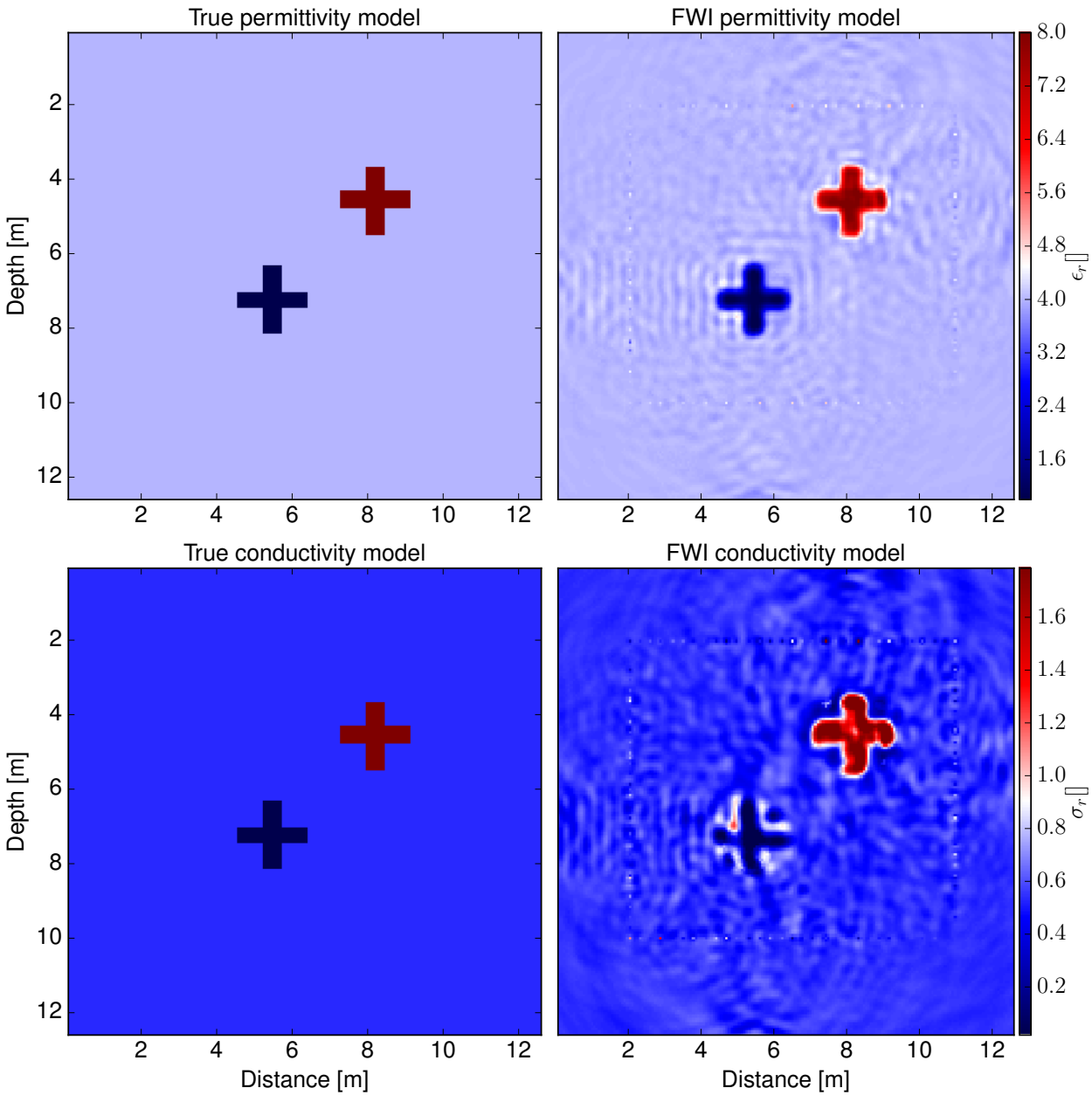
```
HESSIAN_(no=0/app_Hessian=1/pseudo_Hessian=2) = 2
Regularization_term_for_inverse_Hessian_(EPS_HESS) = 0.01
```

otherwise l-BFGS optimization is applied.

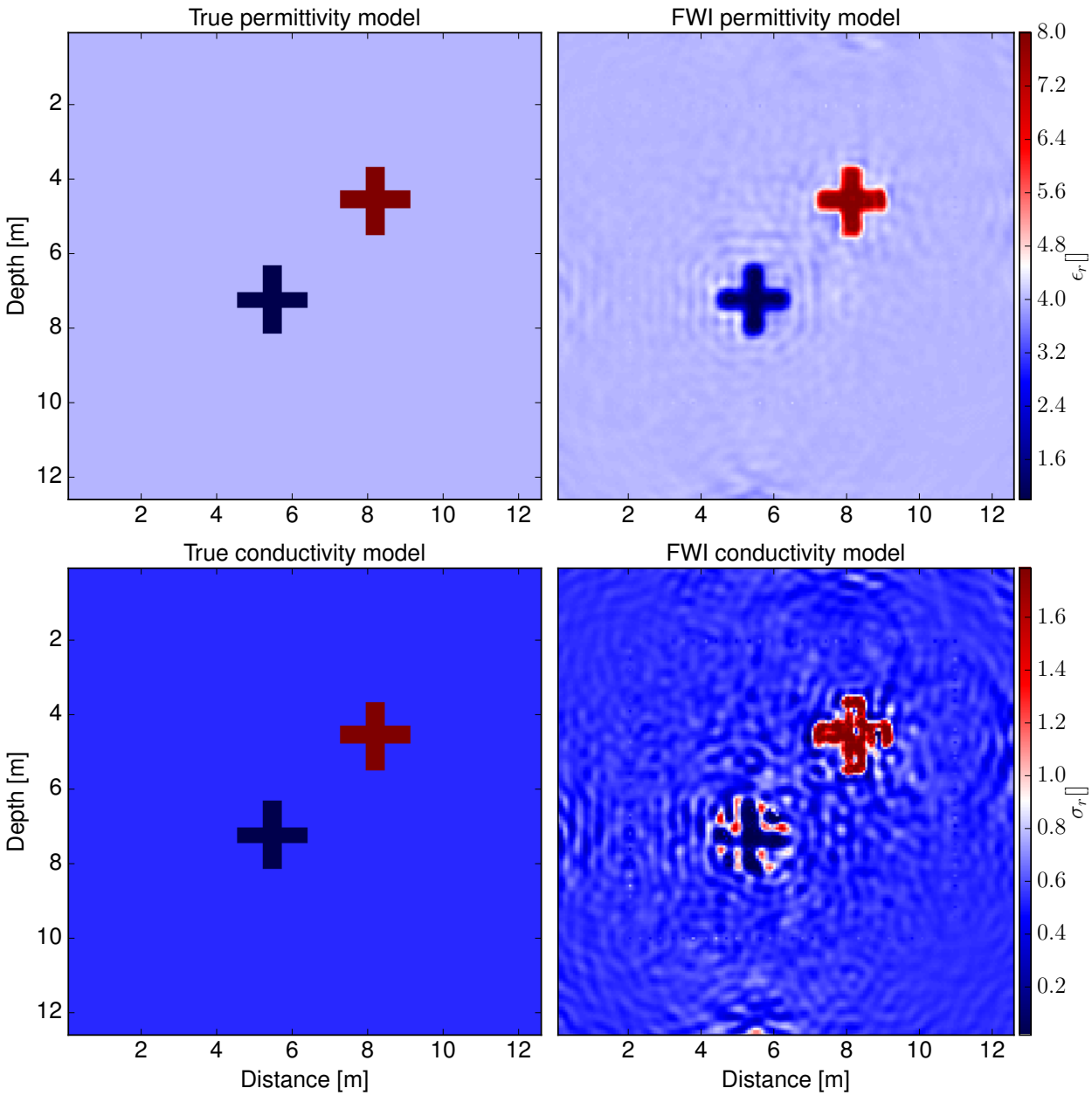
FWI results using gradient with Pseudo-Hessian



FWI results using l-BFGS



FWI results using PI-BFGS with Pseudo-Hessian



FWI parameter scaling

- To decrease the conductivity update a parameter scaling factor β_σ can be introduced, which scales the normalized conductivity [Lavoué et al., 2014]:

  $\tilde{\sigma}_\mathrm{r} = \sigma_\mathrm{r} / \beta_\sigma$

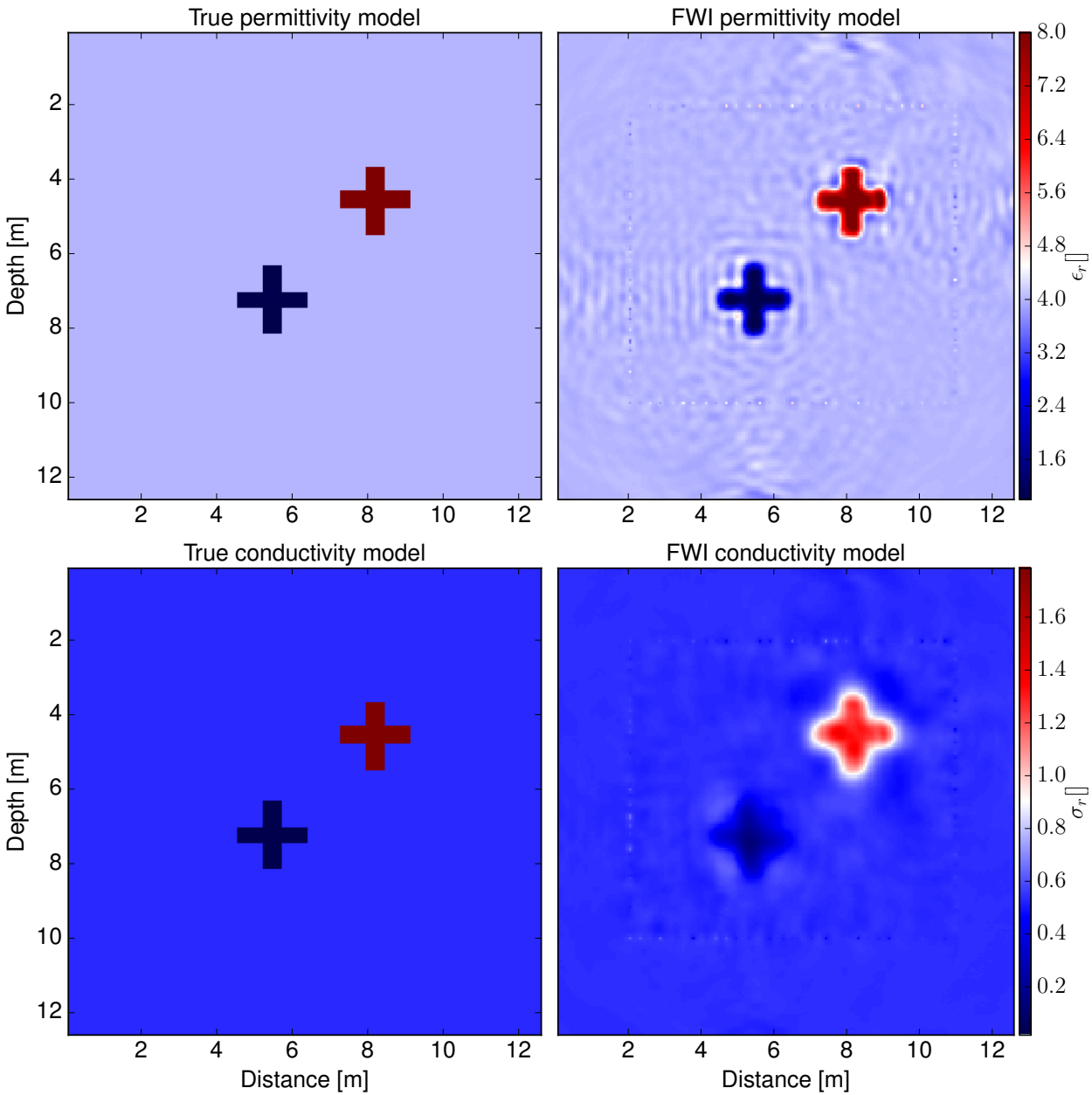
- Definition in GERMAINE workflow file:

```
...   MISFIT  BETA_SIGMA  BETA_EPSILON  ...
...   1       0.25        1.0           ...
...   1       0.25        1.0           ...
...   1       0.25        1.0           ...
```

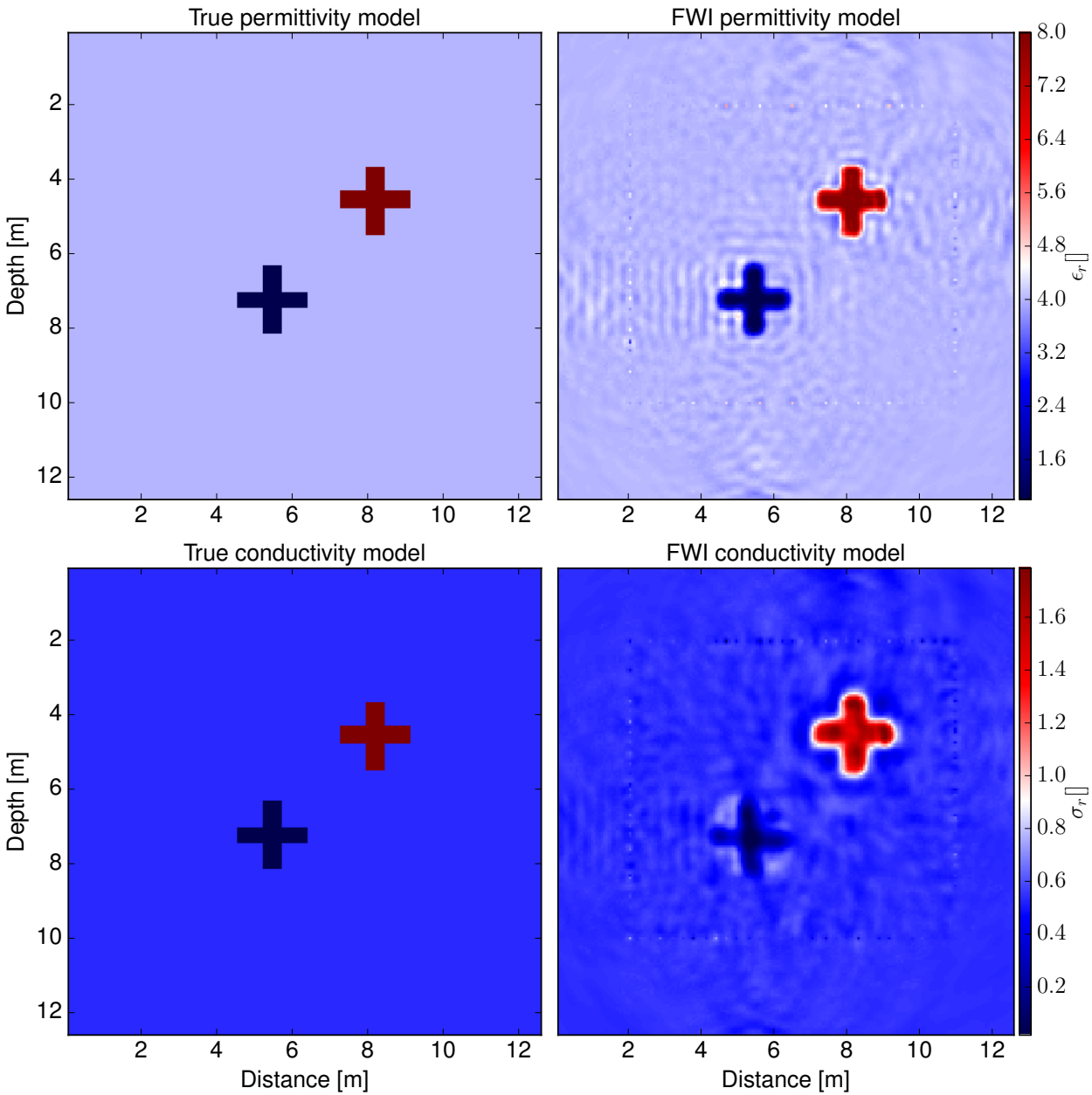
  11^{th} column $= \beta_\sigma$ []
  12^{th} column $= \beta_\epsilon$ []

- Seem to have nearly no effect on the FWI result when using PI-BFGS instead of I-BFGS optimization

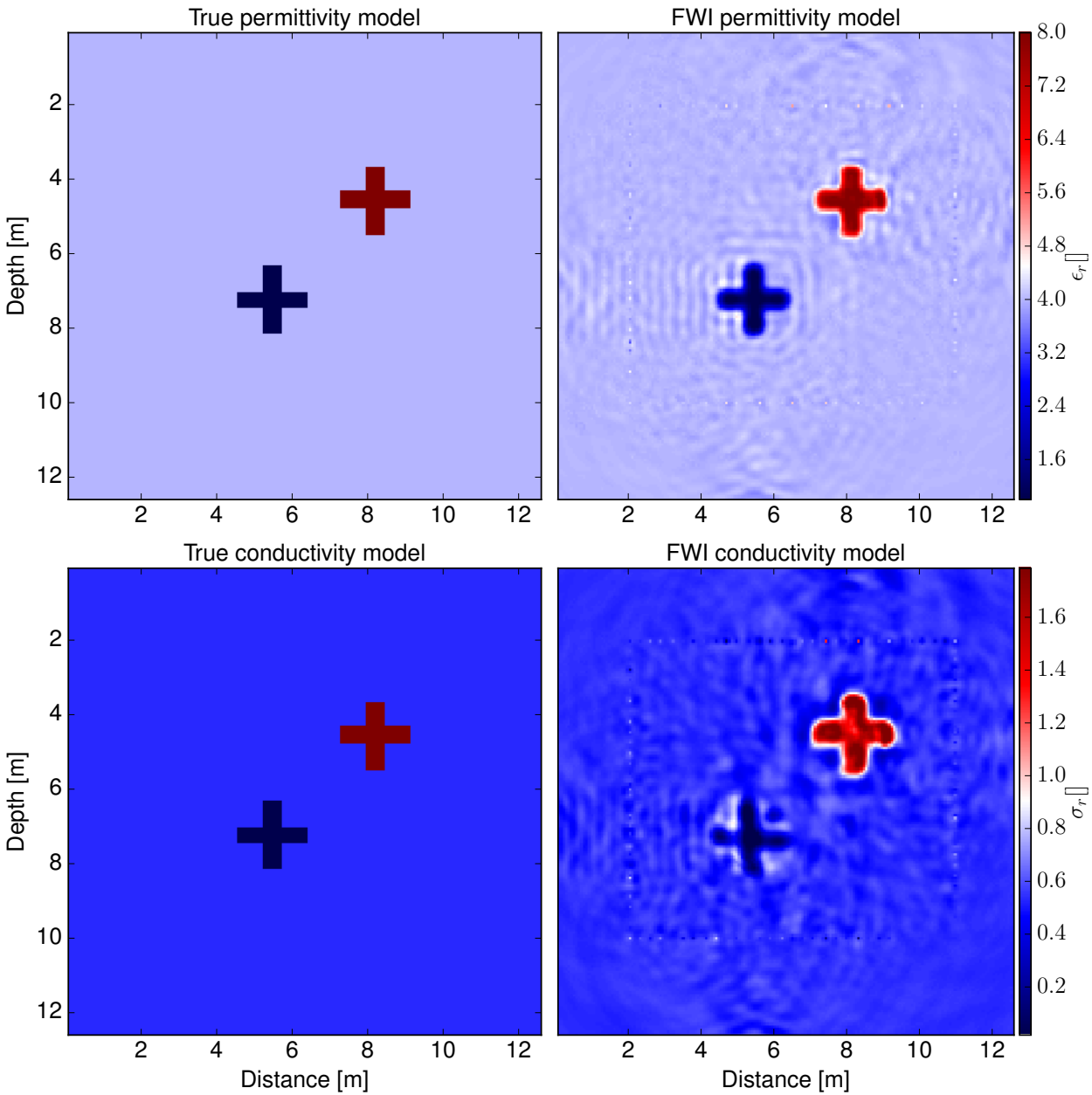
I-BFGS FWI results with parameter scaling $\beta_\sigma = 0.25$



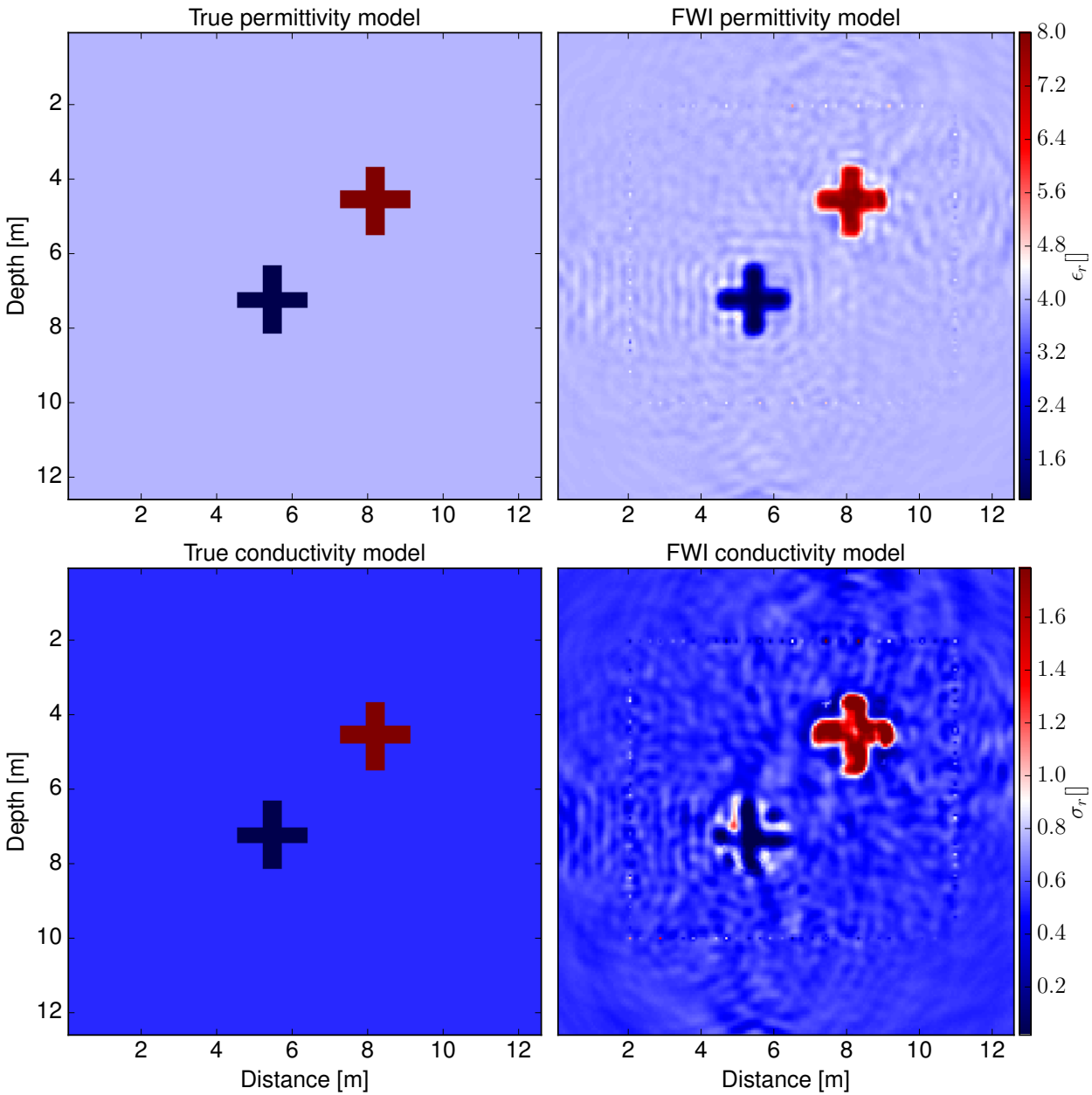
I-BFGS FWI results with parameter scaling $\beta_\sigma = 0.5$



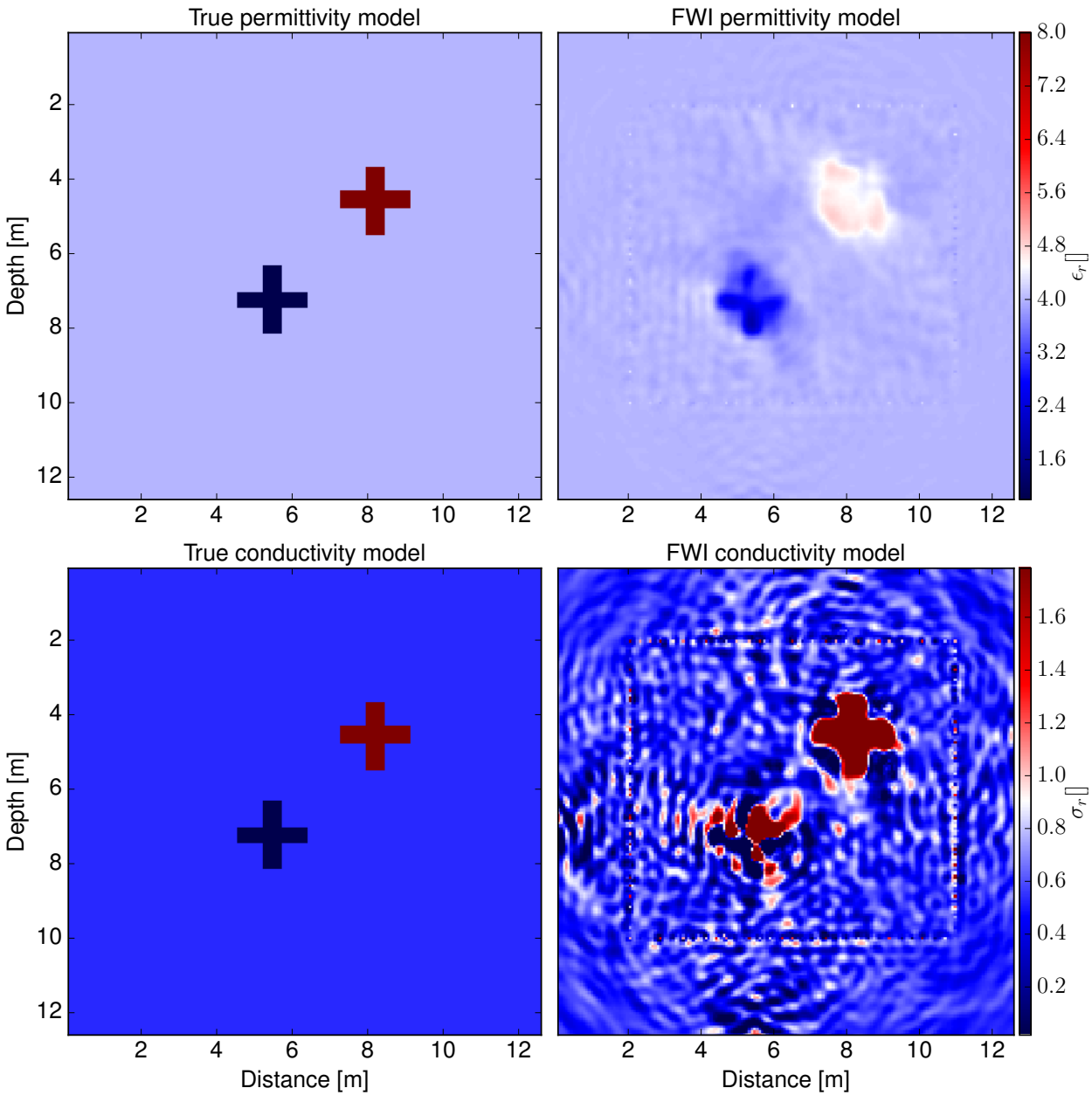
I-BFGS FWI results with parameter scaling $\beta_\sigma = 0.75$



I-BFGS FWI results with parameter scaling $\beta_\sigma = 1.0$



I-BFGS FWI results with parameter scaling $\beta_\sigma = 4.0$



Bound constraints

- Restrict the updated parameters

$$\mathbf{m}^{n+1} = \mathbf{m}^{n} - \mu^{n} \left(\mathcal{B}^{-1} \frac{\partial \mathbf{E}}{\partial \mathbf{m}} \right)^{n}$$

to a bounded part of the parameter space [Métivier and Brossier, 2016]:

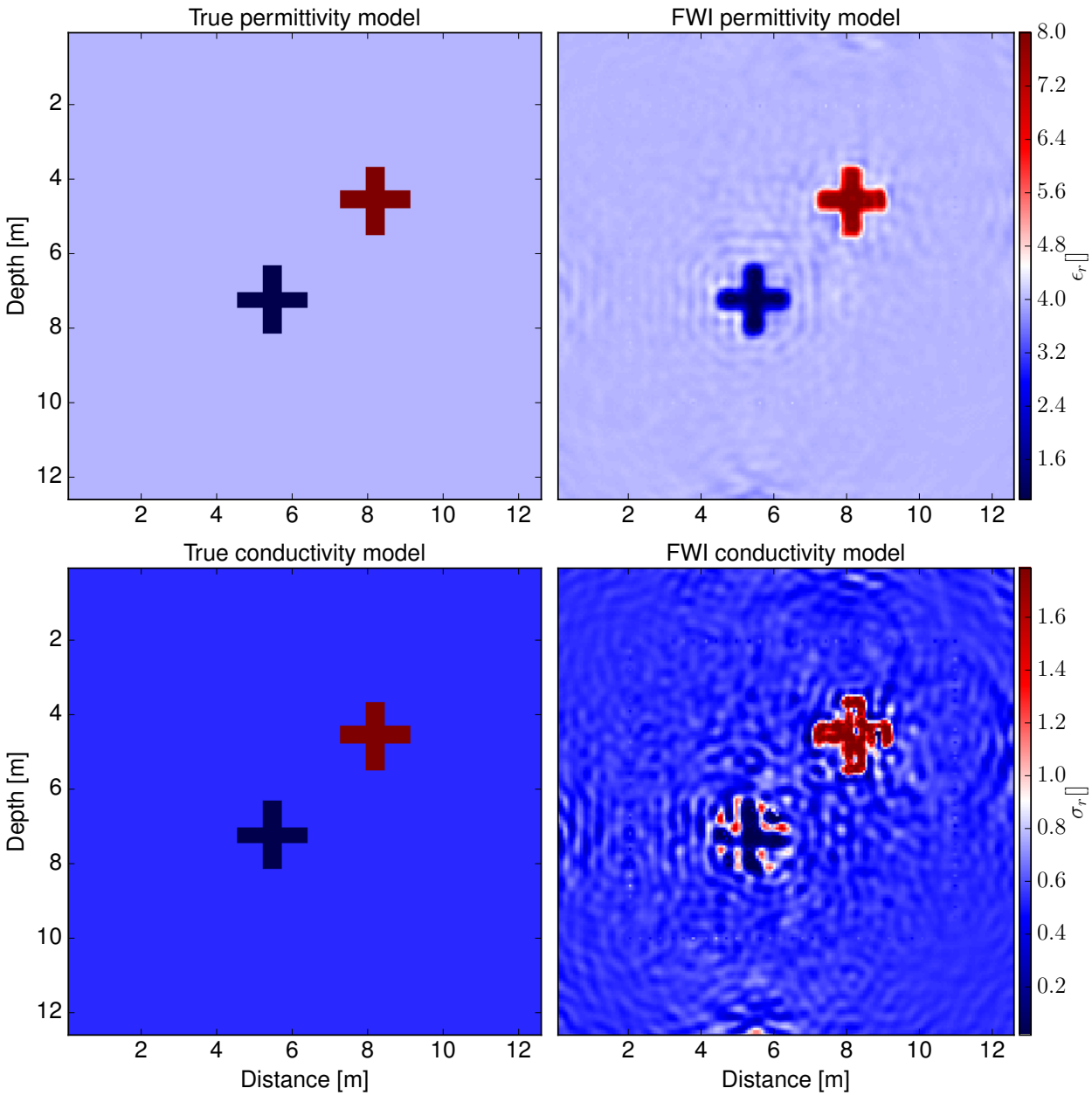
$$m_i^{n+1} = \begin{cases} m_i^{n+1} & \text{if } \mathrm{MAT\_LOW} \leq m_i^{n+1} \leq \mathrm{MAT\_UP}, \\ \mathrm{MAT\_LOW} & \text{if } m_i^{n+1} < \mathrm{MAT\_LOW}, \\ \mathrm{MAT\_UP} & \text{if } m_i^{n+1} > \mathrm{MAT\_UP}, \end{cases}$$

where $\mathrm{MAT\_LOW}$, $\mathrm{MAT\_UP}$ denote the lower and upper bounds for the non-normalized material parameters, respectively.

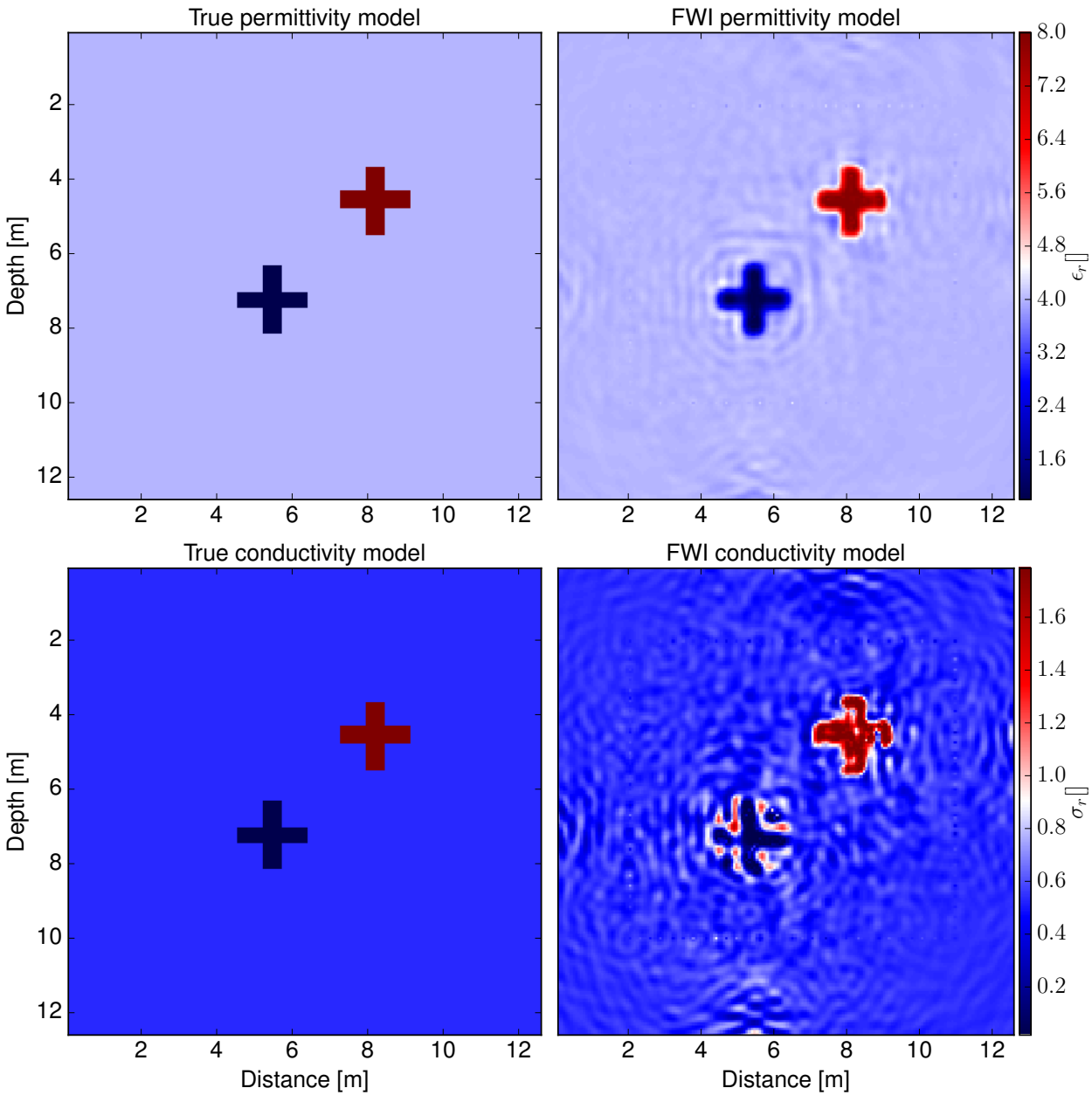
- Definition in GERMAINE parameter file:

```
# ----- Upper and lower bound for model parameters ----- #
limits_for_sigma_(MAT1_LOW,MAT1_UP) = 0.0001, 0.01
limits_for_epsilon_(MAT2_LOW,MAT2_UP) = 8.86e-12, 7.0e-11
```

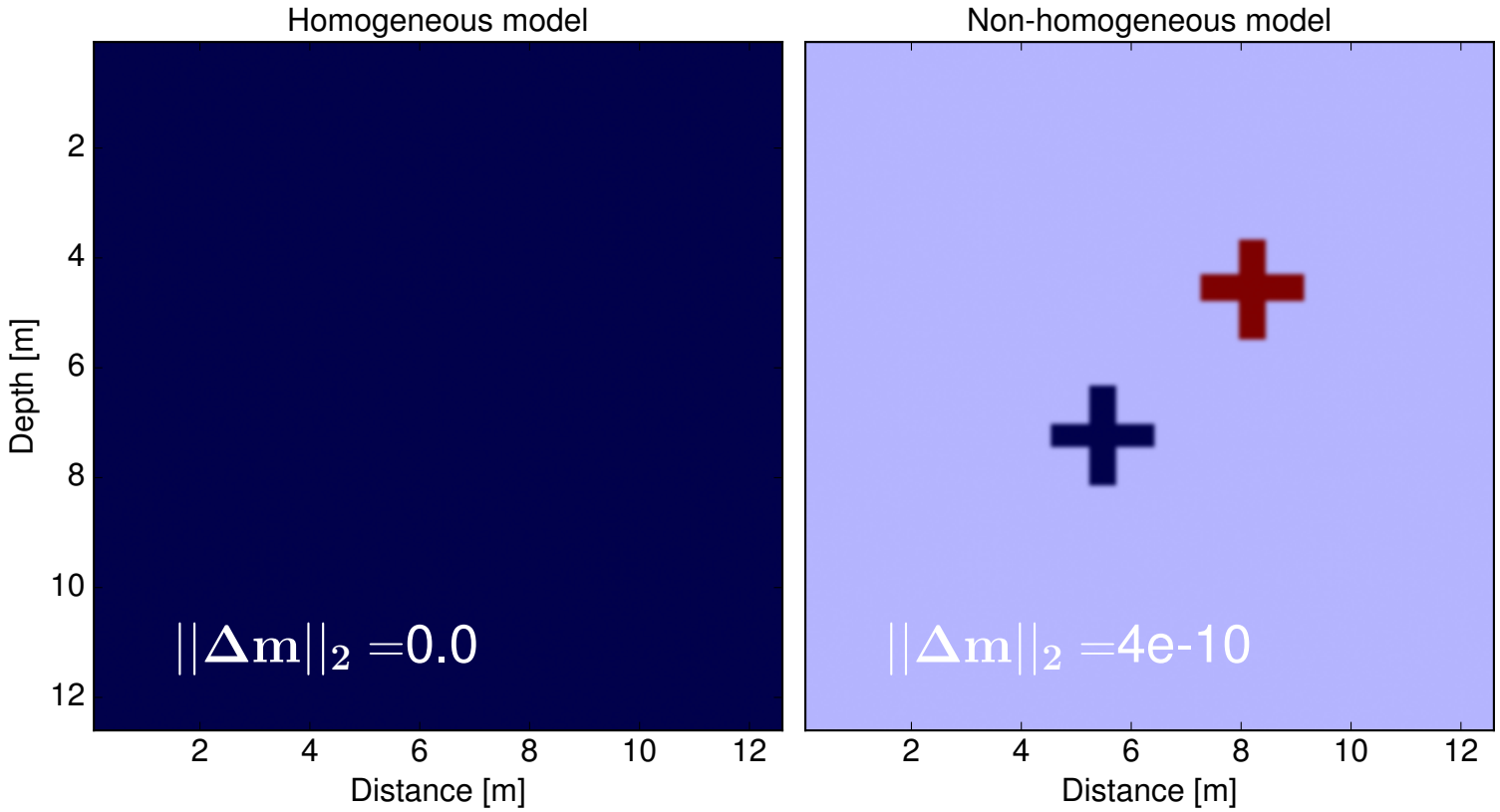
FWI using Pl-BFGS without bound constraints



FWI using PI-BFGS with bound constraints



Tikhonov regularization



Define a measure for the model roughness, e.g.:

$$||\Delta \mathbf{m}||_2,$$

where Δ denotes the Laplace operator.

Tikhonov regularization

- Minimze model roughness in FWI result by adding the Tikhonov regularization term to the objective function:

$$\mathrm{E} = \frac{1}{2}\delta\mathbf{r}^{\dagger}\delta\mathbf{r} + \lambda_{\epsilon}||\Delta\epsilon_{\mathbf{r}}||_2 + \lambda_{\sigma}||\Delta\sigma_{\mathbf{r}}||_2,$$

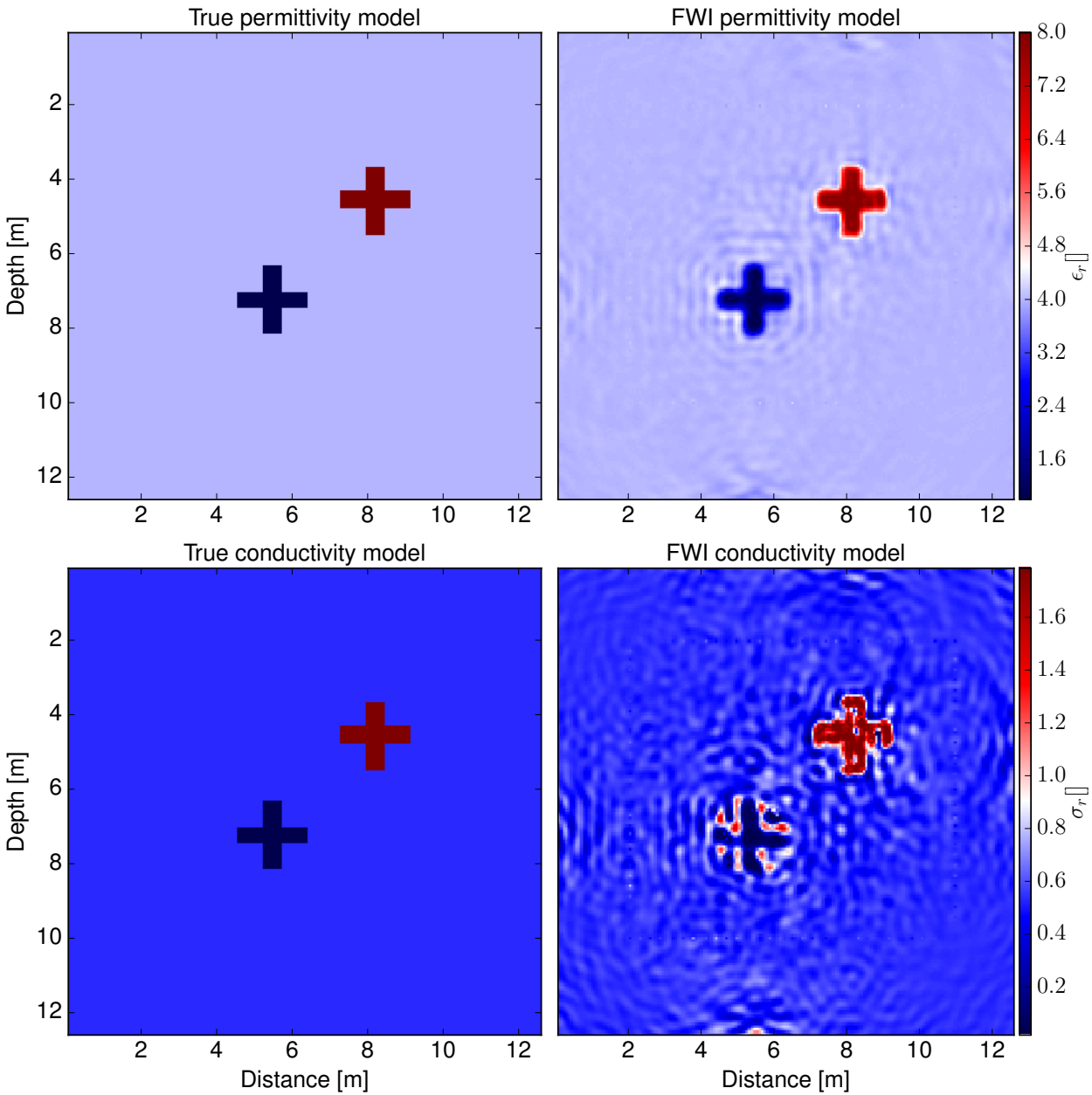
  where the hyperparameter λ balances the contribution of the regularization term with respect to the data misfit term.

- Flexible definition in GERMAINE workflow file for permittivity and conductivity model:

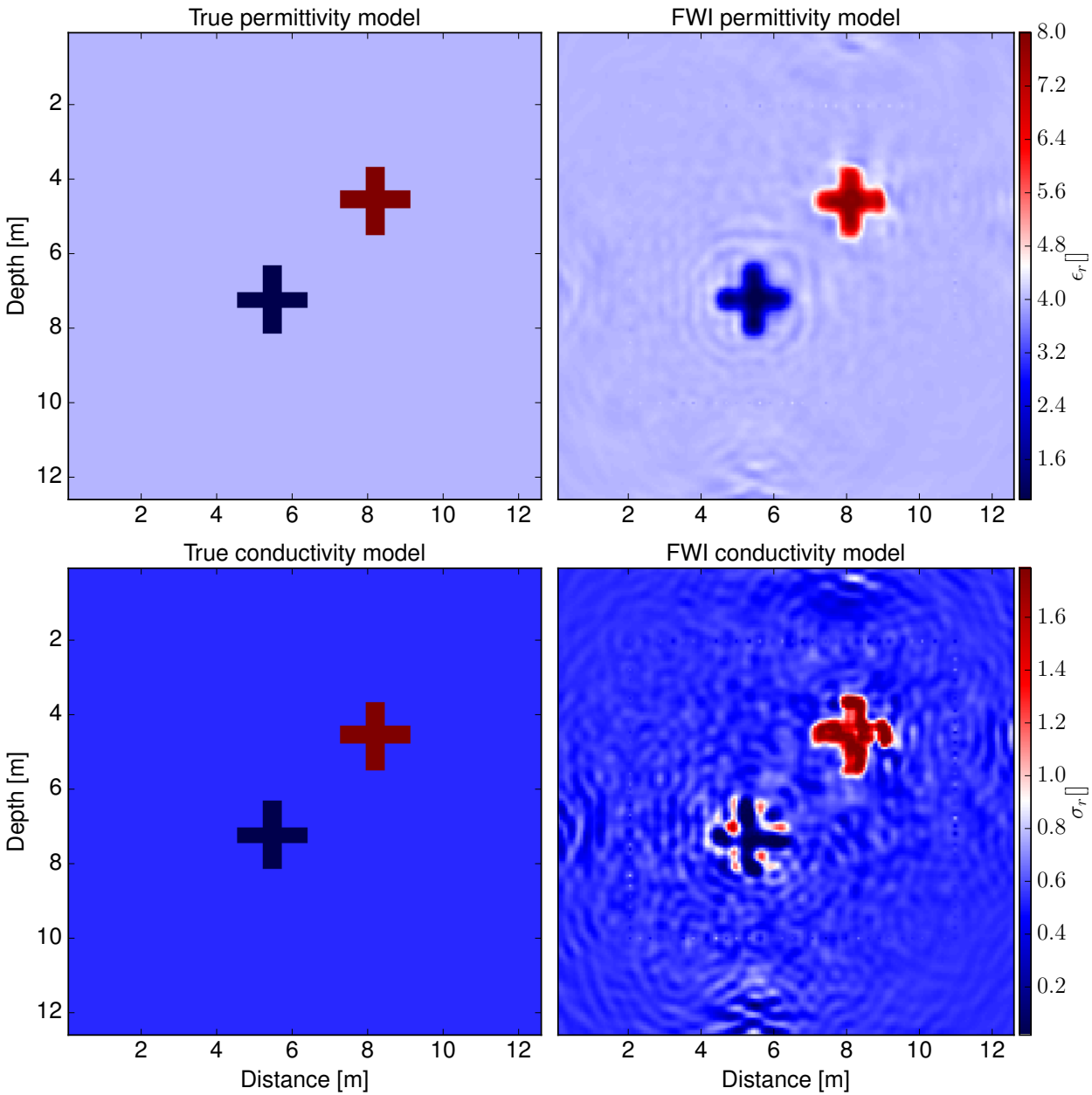
```
... LAMBDA_SIGMA     LAMBDA_EPSILON
... 1e-4             0.0
... 0.001            1e-4
```

13$^{\mathrm{th}}$ column = hyperparameter λ_{σ}
14$^{\mathrm{th}}$ column = hyperparameter λ_{ϵ}

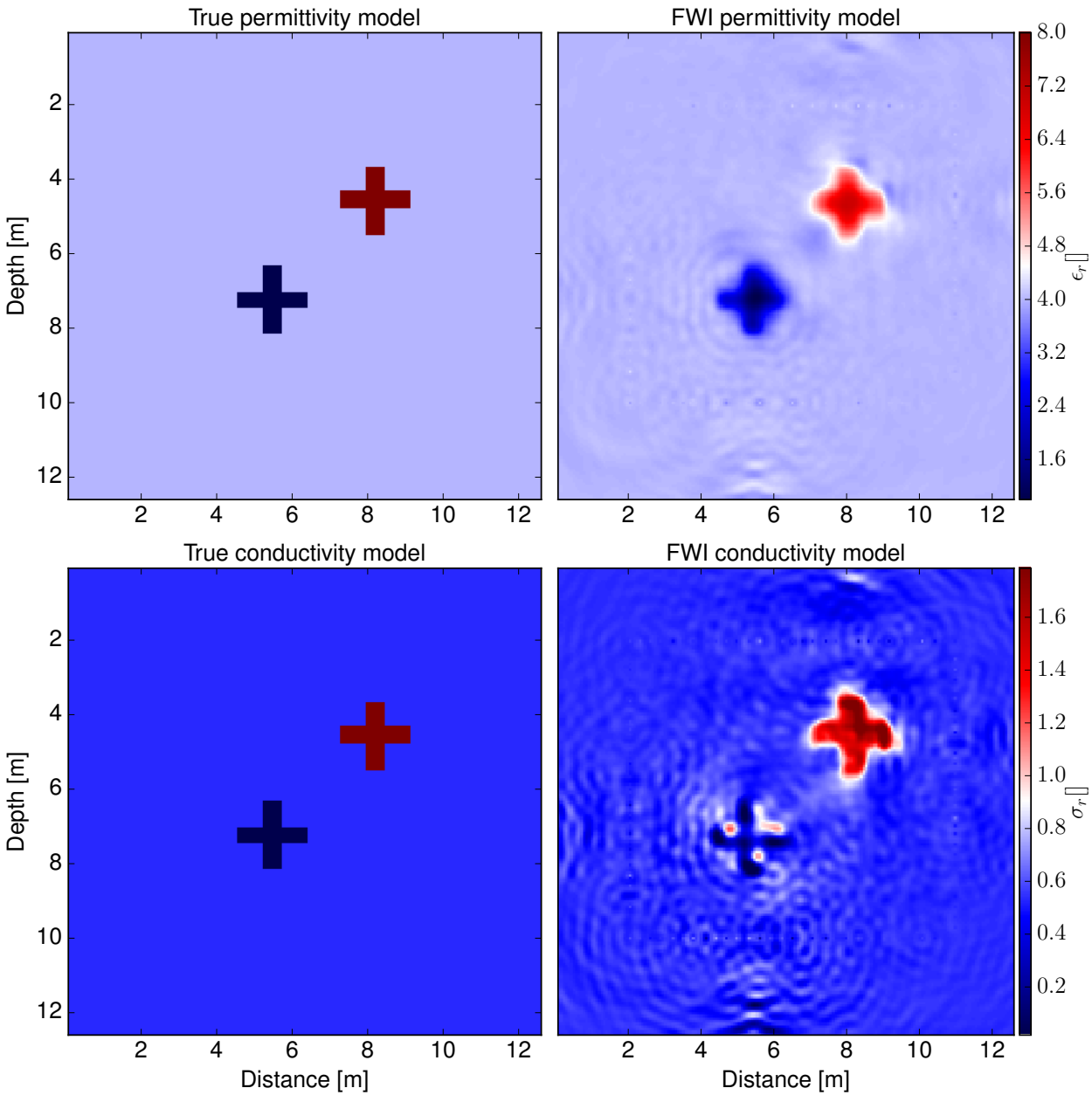
Tikhonov regularization: PI-BFGS ($\lambda_\epsilon = \lambda_\sigma = 0.0$)



Tikhonov regularization: PI-BFGS ($\lambda_\epsilon = \lambda_\sigma = 0.0001$)



Tikhonov regularization: PI-BFGS ($\lambda_\epsilon = \lambda_\sigma = 0.001$)



References I

- Brossier, R. and Virieux, J. (2011).
  Lecture notes on full waveform inversion.
  available at https://seiscope2.obs.ujf-grenoble.fr/701-Lectures-2011.
- Hustedt, B., Operto, S., and Virieux, J. (2004).
  Mixed-grid and staggered-grid finite-difference methods for frequency-domain acoustic wave modelling.
  Geophysical Journal International, 157(3):1269–1296.
- Irving, J. and Knight, R. (2006).
  Numerical modeling of ground-penetrating radar in 2-D using MATLAB.
  Computers & Geosciences, 32(9):1247 – 1258.
- Jo, C., Shin, C., and Suh, J. H. (1996).
  An optimal 9-point, finite-difference, frequency-space, 2–D scalar wave extrapolator.
  Geophysics, 61(2):529–537.
- Lavoué, F., Brossier, R., Métivier, L., Garambois, S., and Virieux, J. (2014).
  Two-dimensional permittivity and conductivity imaging by full waveform inversion of multioffset GPR data: a frequency-domain quasi-Newton approach.
  Geophysical Journal International, 197(1):248–268.
- Meles, G., Greenhalgh, S., van der Kruk, J., Green, A., and Maurer, H. (2011).
  Taming the non-linearity problem in GPR full-waveform inversion for high contrast media.
  Journal of Applied Geophysics, 73(2):174 – 186.
- Métivier, L. and Brossier, R. (2016).
  The SEISCOPE optimization toolbox: A large-scale nonlinear optimization library based on reverse communication.
  Geophysics, 81(2):F11–F25.
- Nocedal, J. and Wright, S. (2006).
  Numerical Optimization.
  Springer, New York.

References II

- Operto, S. (2013).
  A guided tour of FWI: from theory to practice.
  available at https://www.geoazur.net/PERSO/operto/LECTURES/fwi.pdf.
- Operto, S., Virieux, J., Ribodetti, A., and Anderson, J. E. (2009).
  Finite-difference frequency-domain modeling of viscoacoustic wave propagation in 2D tilted transversely isotropic (TTI) media.
  Geophysics, 74(5):T75–T95.

Appendix

GERMAINE installation on the NEC HPC-Linux cluster

GERMAINE installation ...

... on the NEC HPC-Linux cluster at Kiel university

- 172 nodes with 2 Intel Xeon Gold 6130 (Skylake-Sp) CPUs (32 cores, clock speed 2.1 GHz), 192 GB DDR4 RAM
- + additional FAT nodes

More info:

https://www.rz.uni-kiel.de/de/angebote/hiperf/nec-linux-cluster

GERMAINE installation on the NEC HPC-Linux cluster

1. Clone GERMAINE source code from:
   https://github.com/daniel-koehn/GERMAINE
   to the $WORK directory by typing:

   ```
   git clone https://github.com/daniel-koehn/GERMAINE.git
   ```

2. Modify the Makefile in GERMAINE/src by out commenting:

   ```
   # On Linux NEC-cluster with Intel-MPI
   CC=mpiicc
   LFLAGS=-lm -ftz -lstdc++ -lumfpack
   CFLAGS=-O3 -xAVX -fno-fnalias -restrict
   SFLAGS=-L$//sfs/fs2/work-sh2/sungw331/SuiteSparse/lib
   IFLAGS=-I$//sfs/fs2/work-sh2/sungw331/SuiteSparse/include -I./../include
   ```

   Do not forget to comment the unnecessary compiler options

GERMAINE installation on the NEC HPC-Linux cluster

3. Add the following lines to .bashrc file in the $HOME-directory:

```
# SuiteSparse directory
export SuiteSparse_DIR='/sfs/fs2/work-sh2/sungw331/SuiteSparse'
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SuiteSparse_DIR/lib
```

4. Read .bashrc in the running shell:

```
. .bashrc
```

5. In GERMAINE/src compile the source code with

```
make GERMAINE
```

Click here to continue ...

GERMAINE installation on a desktop computer at Department of Geophysics

GERMAINE installation on a CAU Geophysics computer

1. Clone GERMAINE source code from:
   https://github.com/daniel-koehn/GERMAINE
   to the HOME directory by typing:

   ```
   git clone https://github.com/daniel-koehn/GERMAINE.git
   ```

2. Modify the Makefile in GERMAINE/src by out commenting:

   ```
   # On Desktop computer @ Geophysics (CAU Kiel) with Debian, OpenMPI and gcc 4.8.2
   CC=mpicc
   LFLAGS=-lm -lstdc++ -lumfpack
   CFLAGS=-O3 -w -fno-stack-protector -D_FORTIFY_SOURCE=0
   SFLAGS=-L$//home/dkoehn/libs/SuiteSparse/lib
   IFLAGS=-I$//home/dkoehn/libs/SuiteSparse/include -I./../include
   ```

   Do not forget to comment the unnecessary compiler options

GERMAINE installation on a CAU Geophysics computer

3. Add the following lines to .bashrc file in your HOME-directory:

```
export OpenBLAS_DIR='/home/dkoehn/libs/OpenBLAS'
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OpenBLAS_DIR/lib

export LD_LIBRARY_PATH=/home/dkoehn/libs/SuiteSparse/lib:$LD_LIBRARY_PATH
```

4. Read .bashrc in the running shell:

```
. .bashrc
```

5. In GERMAINE/src compile the source code with

```
make GERMAINE
```

Click here to continue ...

SuiteSparse installation on NEC HPC-Linux cluster

SuiteSparse installation on the NEC HPC-Linux cluster

1. Download SuiteSparse v4.5.5 source code from: http://faculty.cse.tamu.edu/davis/SuiteSparse/SuiteSparse-4.5.5.tar.gz
2. Copy TAR-archive to $WORK directory on NEC-cluster
3. Extract source code:

   ```
   tar -zxvf SuiteSparse-4.5.5.tar.gz
   ```

4. Load Cmake, Intel compiler and MPI modules, which include the MKL library:

   ```
   module load cmake3.9.1 intel17.0.4 intelmpi17.0.4
   ```

SuiteSparse installation on the NEC HPC-Linux cluster

5. In the SuiteSparse directory type:

```
make config
make
```

6. Add the following lines to .bashrc in $HOME directory:

```
# SuiteSparse directory
export SuiteSparse_DIR='/sfs/fs2/work-sh2/sungw331/SuiteSparse'
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SuiteSparse_DIR/lib
```

where SuiteSparse_DIR has to be adapted to your path.