SIPPI Documentation

Thomas Mejer Hansen

Sep 26, 2022

Contents:

CHAPTER 1

About SIPPI

SIPPI is a MATLAB toolbox (compatible with GNU Octave) that been been developed in order solve probabilistically formulated inverse problems (Tarantola and Valette, 1982; Tarantola, 2005) where the solution is the a posteriori probability density

$$\rho(\mathbf{m}) \;=\; k\,\rho(\mathbf{m})\,L(g(\mathbf{m})),$$

where

$$g(\mathbf{m})$$

refer to the forward model,

$$\rho(\mathbf{m})$$
$$the a priori model, and$$

$$L(g(\mathbf{m}))$$
$$the likelihood.$$

SIPPI allow sampling the a posteriori probability density (Mosegaard and Tarantola, 1995) in case the forward model is non-linear, and in case using a combination of a number of widely used geostatistical methods to describe a priori information (Hansen el al., 2012).

In order to make use of SIPPI one has to

- Install and setup SIPPI.
- Define the prior model,

  $$\rho(\mathbf{m})$$

  , in form of the `prior` data structure.
- Define the forward model,

  $$g(\mathbf{m})$$

  , in form of the `forward` data structure, and the `sippi_forward.m` m-file.

- Define the data and noise model, i.e. the likelihood

$$L(g(\mathbf{m}))$$

  , in form of the `data` structure.

- Choose a method for sampling the a posteriori probability density (i.e. the solution to the inverse problem).

1.1 Implemented methods and algorithms

A number of different a priori models are available: UNIFORM, GAUSSIAN, FFTMA, CHOLESKY, VISIM, PLURIGAUSSIAN, VORONOI, MPS, SNESIM.

A number of forward solvers is implented: LINEAR (linear forward operator) , TRAVELTIME (ray, fat, eikonal, born), GPR_FW (full waveform modeling).

Three methods exist that allow sampling the a posterior probability density: extended Rejection sampling, extended Metropolis sampling, and linear least squares.

1.1.1 Getting started

The best way to learn to use SIPPI is by going through some examples:

- Lineftting example: A simple low-dimensional inverse problem.
- GPR cross hole tomography: A more complexe inverse problem illustrating most uses of SIPPI.

1.1.2 Referencing

Two manuscripts exist describing SIPPI. Part I, is a general introduction on how to setup and use SIPPI. Part II, is an example of using SIPPI to solve cross hole GPR inverse problems (see example):

> Hansen, T. M., Cordua, K. S., Looms, M. C., & Mosegaard, K. (2013). SIPPI: A Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 1 — Methodology. Computers & Geosciences, 52, 470-480.
> DOI:10.1016/j.cageo.2012.09.004.

> Hansen, T. M., Cordua, K. S., Looms, M. C., & Mosegaard, K. (2013). SIPPI: A Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 2 — Application to crosshole GPR tomography. Computers & Geosciences, 52, 481-492.
> DOI:10.1016/j.cageo.2012.09.001.

The key idea that allow using complex a priori models, referred to as 'sequential Gibbs sampling' is described in detail in

> Hansen, T. M., Cordua, K. S., & Mosegaard, K. (2012). Inverse problems with non-trivial priors: Efficient solution through sequential Gibbs sampling. Computational Geosciences, 16(3), 593-611.
> DOI:
> doi:10.1007/s10596-011-9271-1

References to other manuscript considered/used in SIPPI is listed in the Bibliography.

1.2 Acknowledgement

SIPPI make use of other open software projects such as :

- mGstat : http://mgstat.sourceforge.net
- MPSLib: https://github.com/ergosimulation/mpslib
- VISIM : http://imgp.nbi.ku.dk/visim.php
- Accurate Fast Marching Matlab toolbox : http://www.mathworks.com/matlabcentral/fileexchange/24531-accurate-fast-marching

Codes and theory has been developed by the Inverse Modeling and Geostatistics Project

CHAPTER 2

Install SIPPI

The latest stable version of SIPPI can be downloaded from http://sippi.sourceforge.net as SIPPI.zip. This version includes

MGSTAT, VISIM, SNESIM, and MPSLIB.

—

Unpack ZIPPI.zip somewhere, for example to `c:\Users\tmh\SIPPI`. Then setup the Matlab path to point to the appropriate SIPPI directories by typing:

```
addpath c:\Users\tmh\SIPPI
sippi_set_path
```

For use on Windows, no other setup should be needed.
For use on Linux (Ubuntu 16.10), no other setup should be needed.

For use on OS X, Xcode with gcc libraries but be available to run some of the compiled programs. In addition, the `DYLD_LIBRARY_PATH` must be set to point to the shared libraries neeeded by the compiled programs for OSX. In MATLAB this can be set using:

```
setenv('DYLD_LIBRARY_PATH', '/usr/local/bin');
```

On OSX (and Linux versions that are binary incompatible with Ubuntu 16.10) it may be needed to recompile `MPSlib` using:

```
cd SIPPI/toolboxes/mpslib
./configure
make all
```

2.1 Install SIPPI manually from github

The current development version of SIPPI (less stable and documented) is hosted on github and can be downloaded (including MGSTAT, SNESIM, VISIM and MPS) using git:

```
cd INSTALL_DIR
git clone --recursive https://github.com/cultpenguin/sippi.git SIPPI
```

To update this installation (using mGstat and MSPLIB as submodules) use

```
git pull --recurse-submodules
```

Then add a path to SIPPI in Matlab using

```
addpath INSTALL_DIR/SIPPI
sippi_set_path
```

To download SIPPI, mGstat and MPSlib seperately use:

```
cd INSTALL_DIR
git clone  https://github.com/cultpenguin/sippi.git SIPPI
git clone  https://github.com/cultpenguin/mgstat.git mGstat
git clone  https://github.com/ergosimulation/mpslib.git MPSlib
```

Then add a path to both SIPPI, mGstat and MPS/matlab using

```
addpath INSTALL_DIR/mGstat
addpath INSTALL_DIR/SIPPI
addpath INSTALL_DIR/MPSlib/matlab
sippi_set_path
```

2.1.1 Manual compilation

SIPPI (optionally) make use of standalone programs from
MPS (github),
SNESIM (github) and
VISIM (part of mGstat at github). Pre-compiled slef-contained binaries are available for windows and Linux, but for use on OS-X one may need to manually these programs.

These programs are needed to make use of the MPS, VISIM and SNESIM type *a priori models* (and can hence be ignored if not used).

2.1.1.1 Compiling VISIM

The source code for VISIM is located in `mGstat/visim/visim_src`. The compiled program should be placed in `mGstat/bin` and called `visim`.

VISIM use pre-allocation of memory (set before compilation). This is defined in `visim.inc`. If this file is changed, VISIM must be recomiled.

2.1.1.2 Compiling SNESIM

The source code for SNESIM is available from github.

The compiled program should be placed in `mGstat/bin` and called `snesim.exe` (windows) `snesim_glnxa64.exe` (64 bit Linux) `snesim_maci64.exe` (64 but OS X).

2.1.1.3 Compiling MPS

The source code for MPS is available from *github*. To dowload and compile MPS use for example

```
cd INSTALL_DIR
git clone  https://github.com/ergosimulation/mpslib.git MPSLIB
cd MPS
make all
```

Matlabs path should be updated to include `INSTALL_DIR/MPSLIB/matlab`.

2.1.2 SGeMS (optional)

To make use of the SISIM and SNESIM_STD type prior models, SGeMS needs to be available.

Currently only SGeMS version 2.1 (*download*) for Windows is supported by SIPPI.

Support for SGeMS will be discontinued after relevase v1.5, as the *MPS* library will be used instead.

CHAPTER 3

Setting up SIPPI

This section contains information about how to use and control SIPPI, which requires one to

- Define the *prior model*,

  $$\rho(\mathbf{m})$$

  ,in form of the prior data structure

  This will allow sampling from a variety of geostatistical models, using a unified interface, through ?sippi_prior.m?

- Define the *forward model*,

  $$g(\mathbf{m})$$

  , in form of the forward data structure, and the sippi_forward.m m-file

  This will allow solving a 'forward' problem, such as computhing a geophysical response from some model.

- Define the *data and noise model*,

  $$L(g(\mathbf{m})$$

  ,in form of the prior data structure

  This allow describing data and associated uncertainty.

Once these three data structures has been defined, one can solve the associated inverse problem by sampling the posterior distribution,

$$\sigma(\mathbf{m})$$

3.1 `prior`: The a priori model

A priori information is defined by the `prior` Matlab structure. Any number of different types of a priori models can be defined. For example a 1D uniform prior can be defined in `prior{1}`, and 2D Gaussian prior can be defined in `prior{2}`.

Once a prior data structure has been defined (see examples below), a realization from the prior model can be generated using

```
m=sippi_prior(prior);
```

The realization from the prior can be visualized using

```
sippi_plot_prior(prior,m);
```

A sample (many realizations) from the prior can be visualized using

```
m=sippi_plot_prior_sample(prior);
```

All a priori model types in SIPPI allow to generate a new model in the vicinity of a current model using

```
[m_new,prior]=sippi_prior(prior,m);
```

in such a way that the prior model will be sampled if the process is repeated (see *Sequential Gibbs Sampling*).

3.1.1 Types of a priori models

Six types of a priori models are available, and can be selected by setting the `type` in the `prior` structure using e.q. `prior{1}.type='gaussian'`.

The UNIFORM type prior specifies an uncorrelated ND uniform model.

The GAUSSIAN type prior specifies a 1D generalized Gaussian model.

The FFTMA type prior specifies a 1D-3D Gaussian type a priori model based on the FFT Moving Average method, which is very efficient for unconditional sampling, and for defining a prior Gaussian model with variable/uncertain mean, variance, ranges, and rotation.

The CHOLESKY type prior specifies a 1D-3D Gaussian type a priori model based on Cholesky decomposition of the covariance model.

The VISIM type prior model specifies 1D-3D Gaussian models, utilizing both sequential Gaussian simulation (SGSIM) and direct sequential simulation (DSSIM) that can be conditioned to data of both point- and volume support and linear average data.

The PLURIGAUSSIAN type prior model specifies 1D-3D pluriGaussian. It is a type if truncated Gaussian model that can be used for efficient simulation of categorical values.

The VORONOI type prior defines a number of Voronois cells in a 1D to 3D grid.

The MPS type prior model specifies a 1D-3D multiple-point-based statistical prior model, based on the *MPS* C++ library. Simulation types includes SNESIM (based on a search tree or list), ENESIM, and GENESIM (generalized ENESIM).

The SNESIM type prior model specifies a 1D-3D multiple-point-based statistical prior model based on the SNESIM code from *Stanford/SCRF*.

The SNESIM_STD is similar to the 'SNESIM' type prior, but is based on *SGEMS*.

The following sectionsdocuments the properties of each type of prior model.

Examples of using different types of prior models or combining prior models can be found in the *examples section*.

3.1.1.1 Sequential Gibbs sampling / Conditional Re-sampling

All the available types of prior models allow perturbing one realization of a prior into a new realization of the prior, where the degree of perturbation can be controlled (from a new independent realization to a very small change).

This means that a random walk, with an arbitrary 'step-length' can be performed for any of the a priori types available in SIPPI.

For the a priori types 'FFTMA', 'VISIM', 'CHOLESKY', 'SISIM', 'SNESIM', sequential Gibbs sampling HCM12 is applied. Sequential Gibbs is in essence a type of conditional re-simulation. From a current realization of a prior model, a number of model parameters are discarded and treated as unknown. The unknown model parameters are then re-simulated conditional to the known model parameters.

In order to generate a new realization 'm2' in the vicinity of the realization 'm1' use

```
[m1,prior]=sippi_prior(prior);
[m2,prior]=sippi_prior(prior,m1);
```

If this process is iterated, then a random walk in the space of a priori acceptable models will be perform. Moreover, the collection of realization obtained in this way will represent a sample from prior distribution.

Note that in order to use sequential Gibbs sampling `prior` must be given both as an input variable, and as an (possibly update) output variable.

Controlling sequential Gibbs sampling / Conditional Re-sampling

All properties related to sequential Gibbs sampling can be set in the 'seq_gibbs' structure (which will be avaiable the first time *sippi_prior* is called, or if *sippi_prior_init* is called), for the individual prior models.

The step-length (i.e. the degree of perturbation) is determined by the prior{m}.seq_gibbs.step' parameter.

For the 'uniform' and 'gaussian' type a priori models a step-length closer to 0 zeros imples a 'shorter' step, while a step-length close to 1, implies a 'longer' step-length. A step length of 1, will generate a new independent realization of the prior, while a step length of 0, will return the same realization of the prior

```
prior{m}.seq_gibbs.step=.1;
[m2,prior]=sippi_prior(prior,m1);
```

For the 'FFTMA', 'VISIM', 'CHOLESKY', 'SISIM', and 'SNESIM' type a priori models two types (defined in the `prior{m}.seq_gibbs.type` variable).

The default 'type' is 2, defined as

```
prior{m}.seq_gibbs.step=1;
prior{m}.seq_gibbs.type=2;
```

where the step length defines the percentage of the of model parameters (selected at random) defined in `prior{im}` is conditionally re-sampled. Thus, a step-length closer to 0 zeros imples a 'shorter' step, while a step-length close to 1, implies a 'longer' step-length.

If `prior{m}.seq_gibbs.step=1`, then `prior{m}.seq_gibbs.step` defines the size of a square rectangle/cube which is to be conditionally re-simulated using sequential Gibbs sampling.

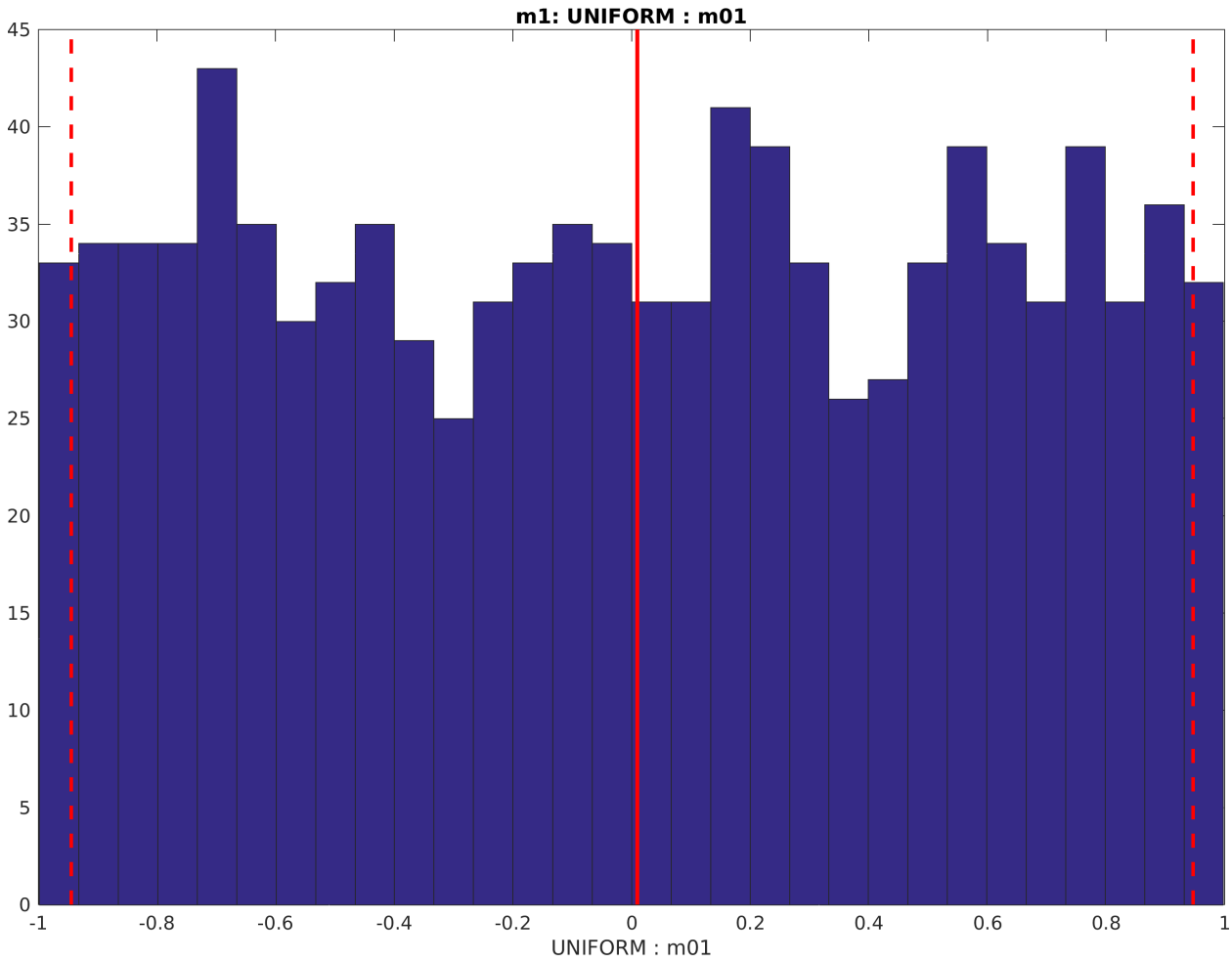
3.1.1.2 Uniform distribution

A uniform prior model can be specified using the 'uniform' type prior model

```
prior{1}.type='uniform';
```

The only parameters needed are the minimum (`min`) and maximum (`max`) values. A 1D uniform distribution between -1 and 1 can be specified as

```
prior{1}.type='uniform';
prior{1}.min=-1;
prior{1}.max=1;
```



Random walk using sequential Gibbs sampling

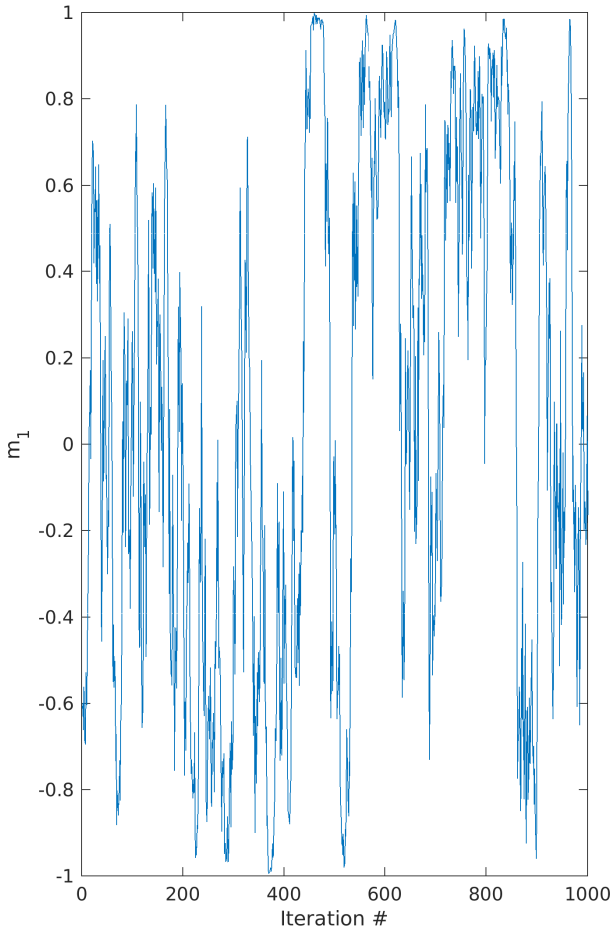
A random walk in the uniform prior (as in any supported prior type) can be obtained using sequential Gibbs sampling:

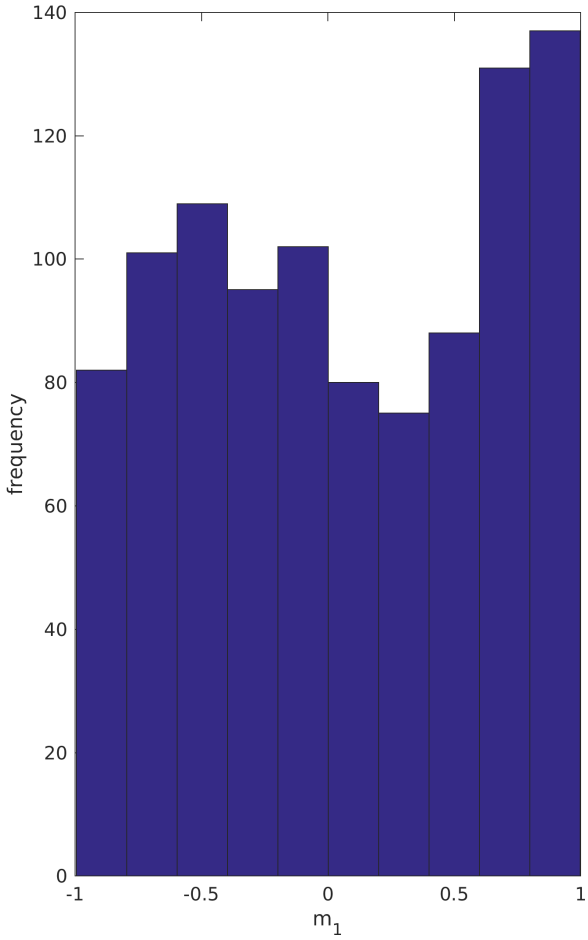
```
%% seq gibbs
prior{1}.seq_gibbs.step=0.2;
N=1000;
m_all=zeros(1,N);
[m,prior]=sippi_prior(prior);
for i=1:1000;
     [m,prior]=sippi_prior(prior,m);
     m_all(i)=m{1};
```

(continues on next page)

(continued from previous page)

```
    subplot(1,2,1);plot(1:i,m_all(1:i));
end
xlabel('Iteration #')
ylabel('m_1')
```





Higher order model

By setting the `x`, `y`, and `z` parameter, a higher order prior (uncorrelated) can be set. For example 3 independent model parameters with a uniform prior distribution between 20 and 50, can be defined as

```
prior{1}.type='uniform';
prior{1}.x=[1 2 3];
prior{1}.min=20;
prior{1}.max=50;
```

Note that using the 'uniform' type priori model, is slightly more computational efficient than using a '*gaussian*' type prior model with a high norm.

3.1.1.3 1D Generalized Gaussian

A 1D generalized Gaussian prior model can be specified using the 'gaussian' type prior model

```
prior{1}.type='gaussian';
```

A simple 1D Gaussian distribution with mean 10, and standard deviation 2, can be specified using

```
ip=1;
prior{ip}.type='gaussian';
prior{ip}.m0=10;
prior{ip}.std=2;
```

The norm of a generalized Gaussian can be set using the 'norm' field. A generalized 1D Gaussian with mean 10, standard deviation of 2, and a norm of 70, can be specified using (The norm is equivalent to the beta factor referenced in *Wikipedia:Generalized_normal_distribution*)

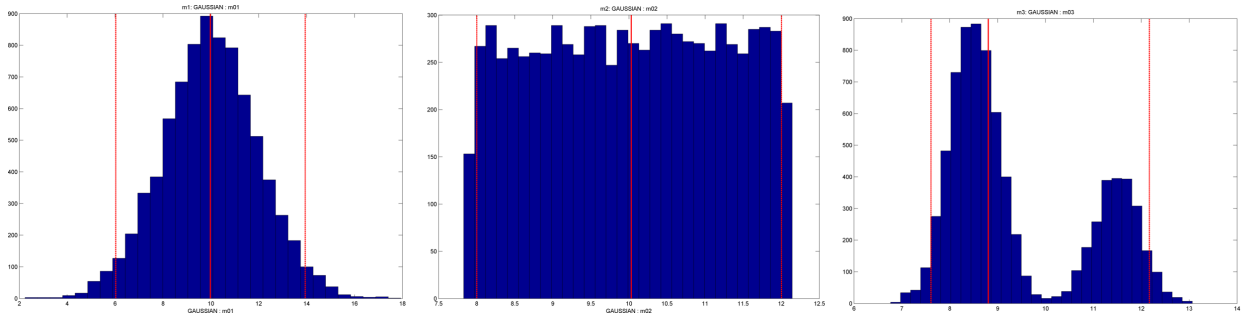
```
ip=2;
prior{ip}.type='gaussian';
prior{ip}.m0=10;
prior{ip}.std=2;
prior{ip}.norm=70;
```

A 1D distribution with an arbitrary shape can be defined by setting `d_target`, which must contain a sample of the distribution that one would like to replicate. For example, to generate a sample from a non-symmetric bimodal distribution, one can use e.g.

```
% Create target distribution
N=10000;
prob_chan=0.3;
d1=randn(1,ceil(N*(1-prob_chan)))*.5+8.5;
d2=randn(1,ceil(N*(prob_chan)))*.5+11.5;
d_target=[d1(:);d2(:)];

% set the target distribution
ip=3;
prior{ip}.type='gaussian';
prior{ip}.d_target=d_target;
```

The following figure shows the 1D histogram of a sample, consisting of 8000 realizations, generated using

```
sippi_plot_prior_sample(prior,1:ip,8000);
```



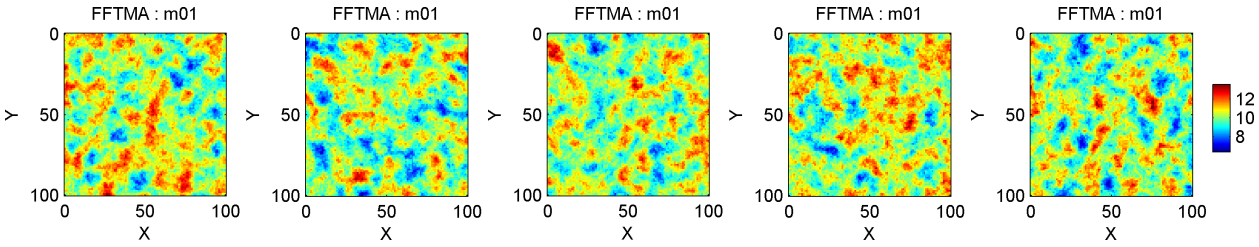
3.1.1.4 FFTMA - 3D Gaussian model

The FFT moving average method provides an efficient approach for computing unconditional realizations of a Gaussian random field.

The mean and the covariance model must be specified in the `m0` and `Cm` fields. The format for describing the covariance model follows 'gstat' notation, and is described in more details in the *mGstat manual*.

A 2D covariance model with mean 10, and a Spherical type covariance model can be defined in a 101x101 size grid (1 unit (e.g., meters) between the cells) using

```
im=1;
prior{im}.type='FFTMA';
prior{im}.x=[0:1:100];
prior{im}.y=[0:1:100];
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10)';
```



Optionally one can translate the output of the Gaussian simulation into an arbitrarily shaped 'target' distribution, using normal score transformation. Note that this transformation will ensure a certain 1D distribution of the model parameters to be reproduced, but will alter the assumed covariance model such that the properties of covariance model are not necessarily reproduced. To ensure that both the covariance model properties and the 1D distribution are reproduced, make use of the VISIM type prior model instead because it utilizes direct sequential simulation.

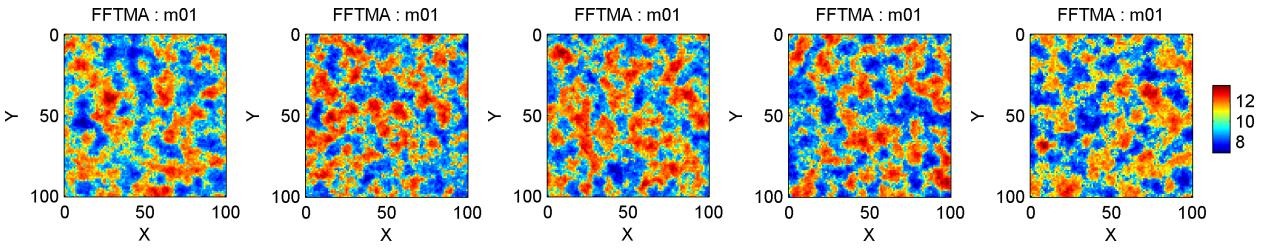
```
im=1;
prior{im}.type='FFTMA';
prior{im}.x=[0:1:100];
prior{im}.y=[0:1:100];
prior{im}.Cm='1 Sph(10)';

% Create target distribution
N=10000;
prob_chan=0.5;
d1=randn(1,ceil(N*(1-prob_chan)))*.5+8.5;
d2=randn(1,ceil(N*(prob_chan)))*.5+11.5;
d_target=[d1(:);d2(:)];
prior{im}.d_target=d_target;
prior{im}.m0=0; % to make sure no trend model is assumed.
```

Alternatively, the normal score transformation can be defined manually such that the tail behavior can be controlled:

```
N=10000;
prob_chan=0.5;
d1=randn(1,ceil(N*(1-prob_chan)))*.5+8.5;
d2=randn(1,ceil(N*(prob_chan)))*.5+11.5;
d_target=[d1(:);d2(:)];
[d_nscore,o_nscore]=nscore(d_target,1,1,min(d_target),max(d_target),0);
prior{im}.o_nscore=o_nscore;
```

3.1.1.5 FFTMA - 3D Gaussian model with variable covariance model properties

The FFTMA method also allows treating the parameters defining the Gaussian model, such as the mean, variance, ranges and angles of rotation as a priori model parameters (that can be inferred as part of inversion, see e.g. *an*



example).

First a prior type defining the Gaussian model must be defined (exactly as listed *above*):

```
im=im+1;
prior{im}.type='FFTMA';
prior{im}.x=[0:.1:10]; % X array
prior{im}.y=[0:.1:20]; % Y array
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10,90,.25)';
```

Now, all parameter such as the mean, variance, ranges and angles of rotations, can be randomized by defining a 1D a priori model type ('uniform' or 'gaussian'), and with a specific 'name' indicating the parameter (see *this example* for a complete list of names), and by assigning the `prior_master` field that points the prior model id for which the parameters should randomized.

For example the range along the direction of maximum continuty can be randomized by defining a prior entry named 'range_1', and settting the `prior_master` to point to the prior with id 1:

```
im=2;
prior{im}.type='uniform';
prior{im}.name='range_1';
prior{im}.min=2;
prior{im}.max=14;
prior{im}.prior_master=1;
I this case the range is randomized following a uniform distribution U[2,14].
```

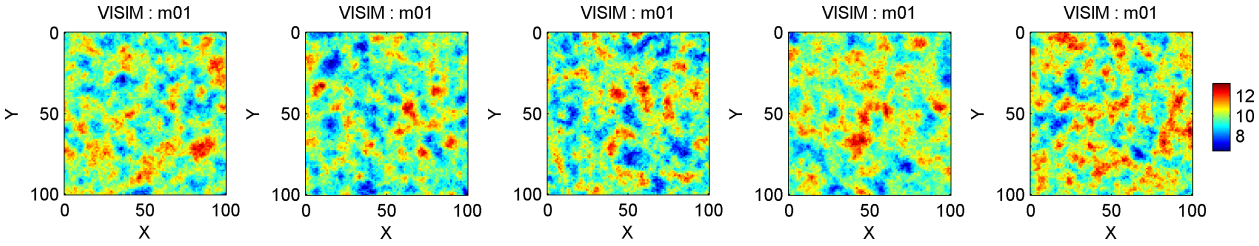
Likewise, the first angle of rotation can be randomized using for example

```
im=3;
prior{im}.type='gaussian';
prior{im}.name='ang_1';
prior{im}.m0=90;
prior{im}.std=10;
prior{im}.prior_master=1;
```

A sample from such a prior type model will thus show variability also in the range and angle of rotation, as seen here

3.1.1.6 VISIM

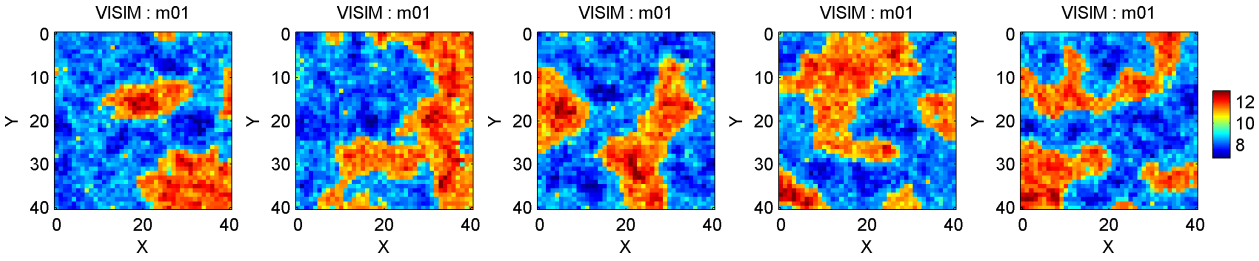
```
im=im+1;
prior{im}.type='VISIM';
prior{im}.x=[0:1:100];
prior{im}.y=[0:1:100];
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10)';
```



As with the FFTMA prior model the VISIM prior can make use of a target distribution. However, if a target distribution is set, the use of the VISIM prior model will utilize direct sequential simulation, which will ensure both histogram and covariance reproduction.

Using a target distribution together with the VISIM prior model is similar to that for the FFTMA prior model. Simply the `type` has to be changed from FFTMA to VISIM:

```
clear all;close all;
im=1;
prior{im}.type='VISIM';
prior{im}.x=[0:1:40];
prior{im}.y=[0:1:40];
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10)';

% Create target distribution
N=10000;
prob_chan=0.5;
d1=randn(1,ceil(N*(1-prob_chan)))*.5+8.5;
d2=randn(1,ceil(N*(prob_chan)))*.5+11.5;
d_target=[d1(:);d2(:)];
prior{im}.d_target=d_target;
```



3.1.1.7 CHOLESKY - 3D Gaussian model

The CHOLESKY type prior utilizes Cholesky decomposition of the covariance in order to generate realizations of a Gaussian random field. The CHOLESKY type prior needs a full description of the covariance model, which will be of size [nxyz*nxyz*nxyz], unlike using the *FFTMA* type prior model that only needs a specification of an isotropic covariance models of size `[1,nxyz]`. Hence, the CHOLESKY type prior is much more demanding on memory, and CPU. However, the CHOLESKY type prior can be used to sample from any covariance model, also non-stationary covariance model.

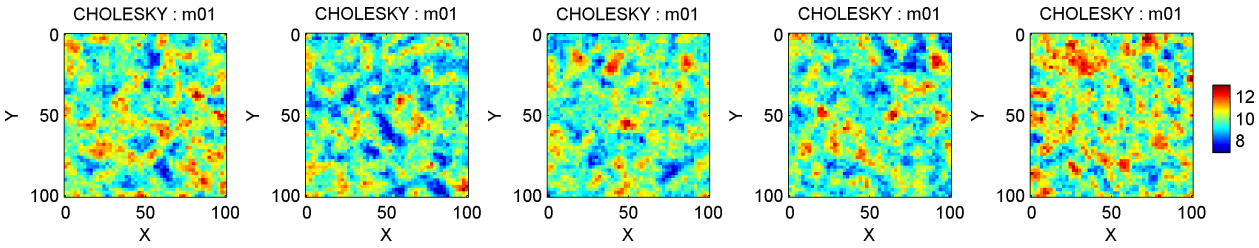
The CHOLESKY model is can be defined almost identically to the *FFTMA* type prior model. As an example:

```
im=1;
prior{im}.type='CHOLESKY';
prior{im}.x=[0:2:100];
```

(continues on next page)

(continued from previous page)

```
prior{im}.y=[0:2:100];
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10)';
```



the use of `d_target` to specify target distributions is also possible, using the same style as for the *FFTMA* type prior.

Be warned that the 'cholesky' type prior model is much more memory demanding than the 'fftma' and 'visim' type prior models, as a full [nxyz*nxyz] covariance model needs to setup (and inverted). Thus, the 'cholesky' type prior is mostly applicable when the number of model parameters (nx*ny*nx) is small.

3.1.1.8 PluriGaussian - ND truncated Gaussian

Plurigaussian simulation is a type og truncated Gaussian simulation. It works by generating a number of realizations of Gassuan models, each with a sepcific choice of covariance model. Using a transformation map, the Gaussian realizations are then converted into disrete units.

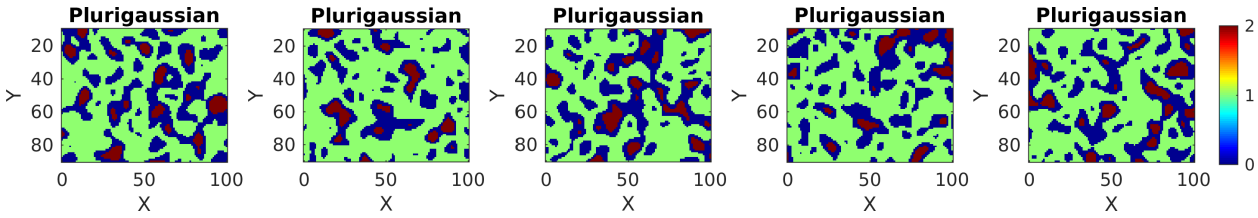
PluriGaussian based on 1 Gaussian

A simple example using 1 Gaussian realization, one must specify one covariance model one plurigaussian transformation map through the two fields `prior{1}.pg_prior{1}.Cm`(or `Cm`) and `prior{1}.pg_map`.

The covariance model is defined as for any other Gaussian based models, and can include anisotropy. In general, the variance (sill) should be 1. Unless set othwerwise, the mean is assumed to be zero.

The values in the transformation map is implicitly assumed to define boundaries along a linear scale from -3 to 3. As there are 7 entries (see below) in the transformation map, each number in the transformation map corresponds to [-3,-2,-1,0,1,2,3] respectively. The figure below show what unit id's any Gaussian realized value will be transformed to.

```
im=im+1;
prior{im}.name='Plurigaussian'; % [optional] specifies name to prior
prior{im}.type='plurigaussian';                % the type of a priori model
prior{im}.x=[0:1:100];                 % specifies the scales of the 1st (X) dimension
prior{im}.y=[10:1:90];                 % specifies the scales of the 2nd (Y) dimension
prior{im}.Cm='1 Gau(10)'; % or next line
prior{im}.pg_prior{1}.Cm=' 1 Gau(10)';
prior{im}.pg_map=[0 0  1 1 0  2 2];
[m,prior]=sippi_prior(prior);          % generate a realization from the prior model
sippi_plot_prior_sample(prior,im,5)
print_mul('prior_example_2d_plurigaussian_1')
figure;
pg_plot(prior{im}.pg_map,prior{im}.pg_limits);
colormap(sippi_colormap);
print_mul('prior_example_2d_plurigaussian_1_pgmap')
```



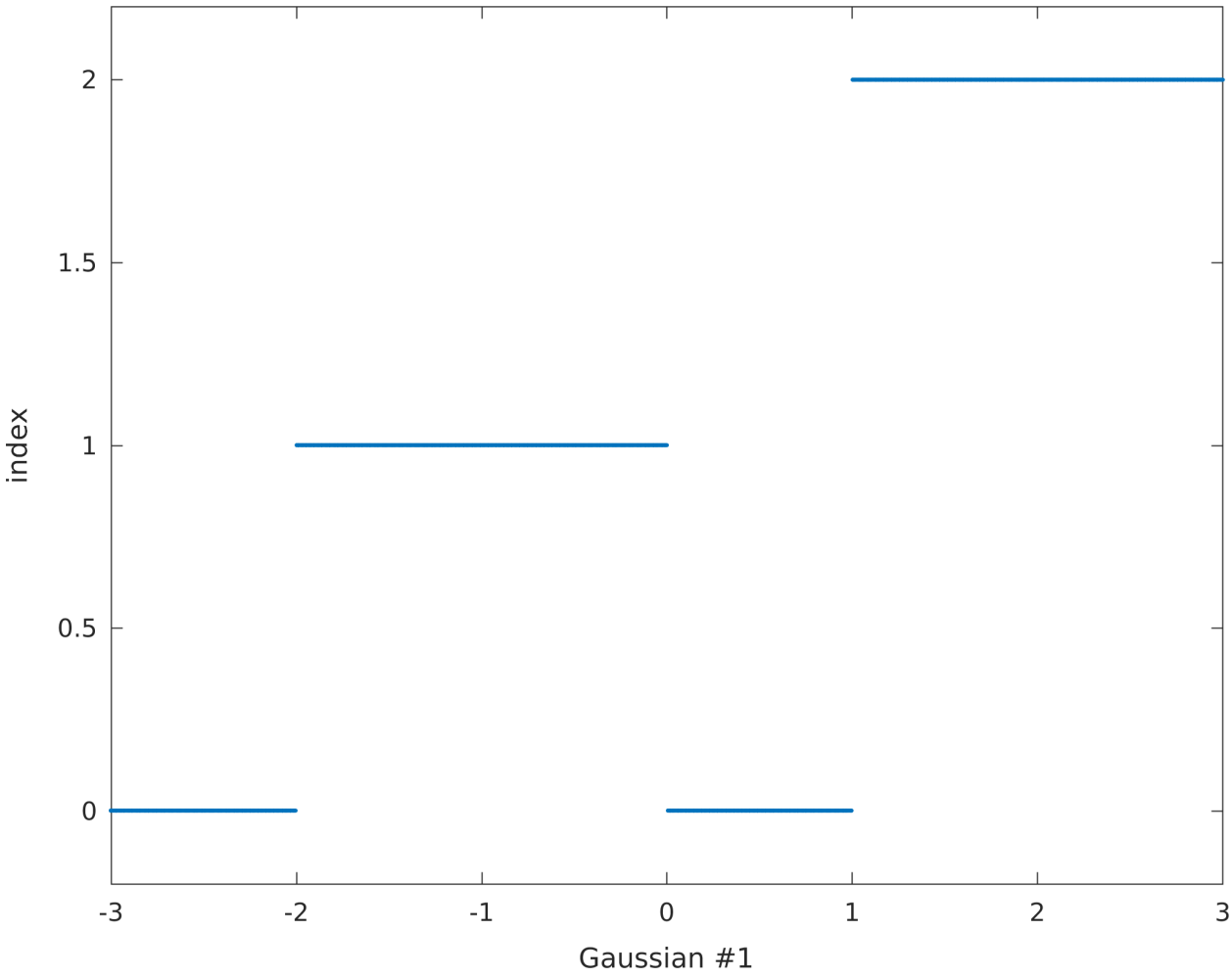
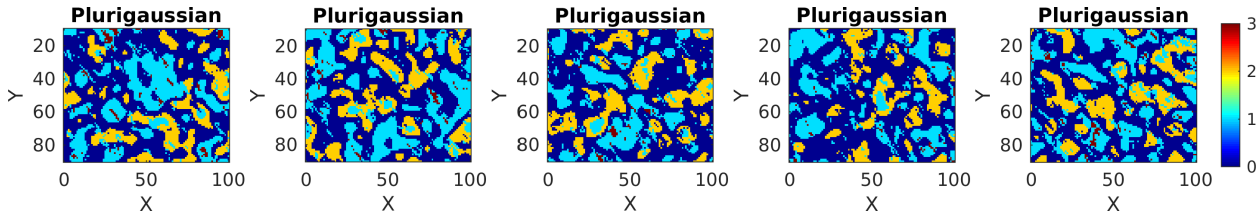


Fig. 1: Plurigaussian transformation map for 1 Gaussian realization

PluriGaussian based on 2 Gaussians

Plurigaussian truncation can be based on more than one Gaussian realization, In the example below, two Gaussian realization are used, and therefore a transformation map needs to be defined. Each dimension of the transformation map corresponds to values of the Gaussian realization between -3 and 3. The transformation maps is visualized below.

```
im=1;
prior{im}.name='Plurigaussian'; % [optional] specifies name to prior
prior{im}.type='plurigaussian';                % the type of a priori model
prior{im}.x=[0:1:100];                 % specifies the scales of the 1st (X) dimension
prior{im}.y=[10:1:90];                 % specifies the scales of the 2nd (Y) dimension
prior{im}.pg_prior{1}.Cm=' 1 Gau(10)';
prior{im}.pg_prior{2}.Cm=' 1 Sph(10,35,.4)';
prior{im}.pg_map=[0 0 0 1 1; 1 2 0 1 1; 1 1 1 3 3];
[m,prior]=sippi_prior(prior);          % generate a realization from the prior model
sippi_plot_prior_sample(prior,im,5)
print_mul('prior_example_2d_plurigaussian_2')

figure;
pg_plot(prior{im}.pg_map,prior{im}.pg_limits);
set(gca,'FontSize',16)
colormap(sippi_colormap);
print_mul('prior_example_2d_plurigaussian_2_pgmap')
```



3.1.1.9 Voronoi

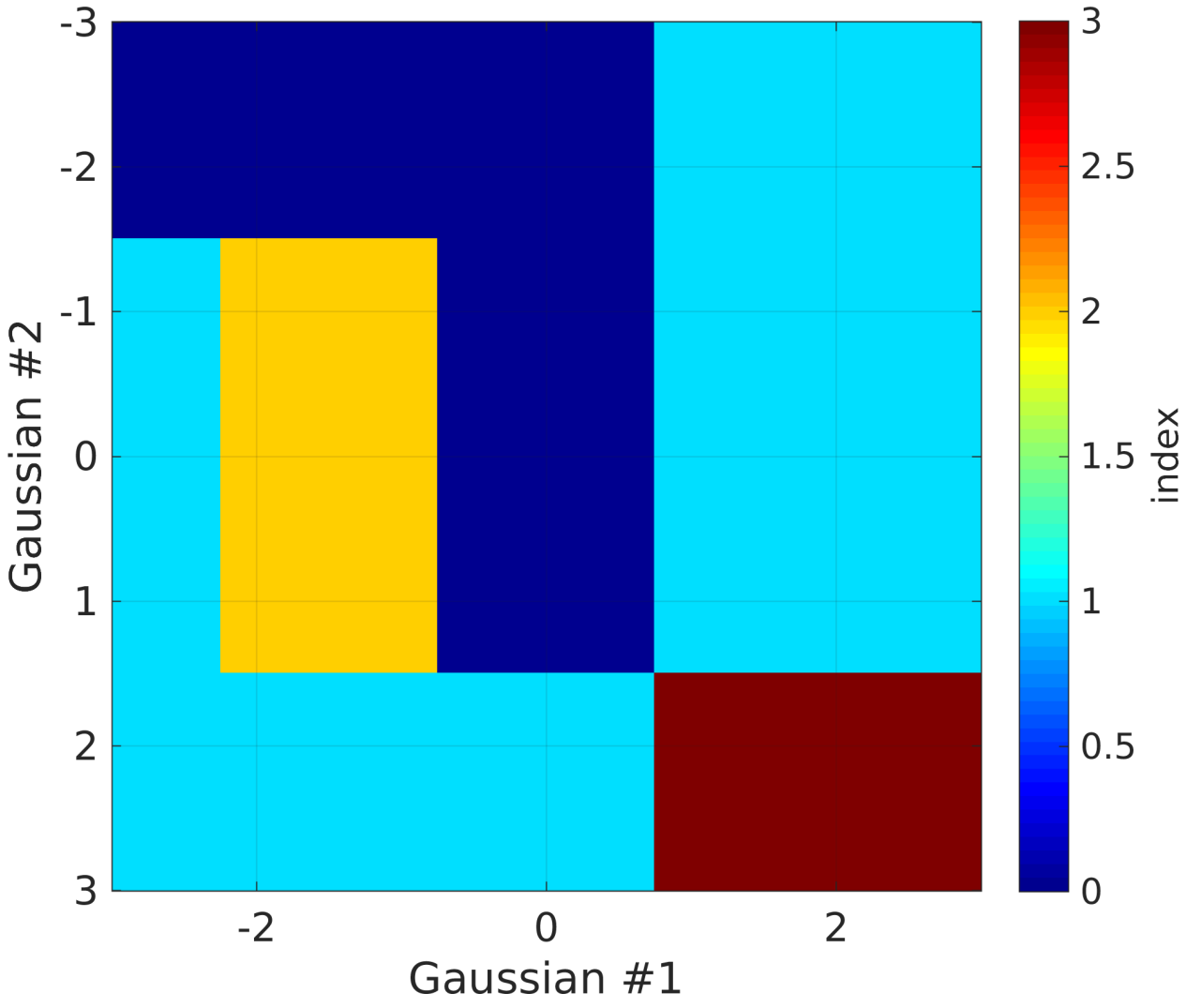
The `voronoi` type defines a prior model defines an a priori model based on a number of Voronoi cells. For example, a 2D model of size (20x20) with 50 randomly located Voronoi cells can be defined using

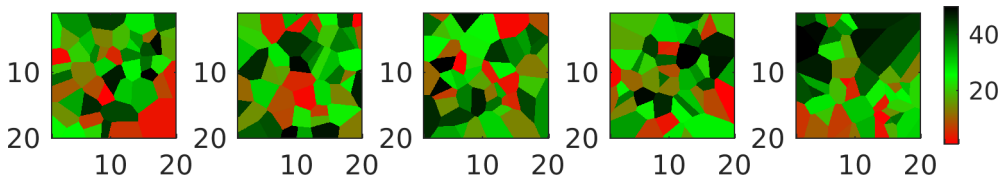
```
cells_N_max=50;
ip=1;
prior{ip}.type='voronoi';
prior{ip}.x=1:.04:20;
prior{ip}.y=1:.04:20;
prior{ip}.cells_N=cells_N_max; % SET NUMBER OF CELLS
prior{ip}.cells_N_min=3;
prior{ip}.cells_N_max=cells_N_max;
sippi_plot_prior_sample(prior);
```

The value of each cell is simply an integer number between 1 and `prior{ip}.cells_N`.

Randomize number of, location of, and value of the Voronois cells

The location and value of the Voronois cells can be randomized by specifying additional, appropriately named a priori types.


Gaussian #2
Gaussian #1
index
-3
-2
-1
0
1
2
3
-2
0
2
0
0.5
1
1.5
2
2.5
3

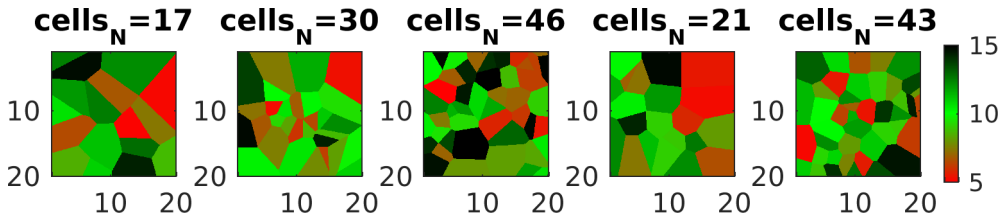

10
20
10
20
40
20

For example to randomize location of the Voronoi cell define a number prior models with names `cells_x`, `cells_y`, and `cells_z` (in 3D), as e.g.:

```
ip=ip+1;
prior{ip}.type='uniform';
prior{ip}.name='cells_x';
prior{ip}.x=[1:cells_N_max];
prior{ip}.min=min(prior{1}.x);
prior{ip}.max=max(prior{1}.x);
prior{ip}.cax=[prior{ip}.min prior{ip}.max];
prior{ip}.prior_master=1;

ip=ip+1;
prior{ip}.type='uniform';
prior{ip}.name='cells_y';
prior{ip}.x=[1:cells_N_max];
prior{ip}.min=min(prior{1}.y);
prior{ip}.max=max(prior{1}.y);
prior{ip}.cax=[prior{ip}.min prior{ip}.max];
prior{ip}.prior_master=1;
```

Finally, to randomize the value of each cell, define a prior with name `cells_value`, as e.g.

```
ip=ip+1;
prior{ip}.type='uniform';
prior{ip}.name='cells_value';
prior{ip}.x=[1:cells_N_max];
prior{ip}.min=5;
prior{ip}.max=15;
prior{ip}.prior_master=1;
prior{1}.cax=[5 15];
```



Random walk using sequential Gibbs sampling

A random walk in the uniform prior (as in any supported prior type) can be obtained using sequential Gibbs sampling:

```
for i=1:500;
    [m,prior]=sippi_prior(prior,m);
    sippi_plot_prior(prior,m);
    drawnow;
end
```

This provides a video as e.g.: {% youtube %}https://www.youtube.com/watch?v=doNhrDIjJpw{% endyoutube %}

The code for the full example can be found here: prior_reals_voronoi.m.

3.1.1.10 MPS

The 'MPS' type prior make use of the *MPS library* for mulitple-point based simulation. For compilation and installation help see *Install SIPPI*. MPS implements the SNESIM (using both a search tree and a list to stor conditional statistics), and the generalized ENESIM algoritm. The type of multiple-point algorithm is define in the method field.

To use the MPS type prior at least the type, dimension, as well as a training image must be provided:

```
ip=1;
prior{ip}.type='mps';
prior{ip}.x=1:1:80;
prior{ip}.y=1:1:80;
```

A trainin imag emust be set in the 'ti' field, as 1D, 2D, or 3D matrix. If not set, the classical training image from Strebelle is used, equivalent to:

```
prior{ip}.ti=channels;
```

More examples of traning images are located in the 'mGstat/ti' folder.

MPS provides three different simulation aglrithms, which canbe chosen in the 'method' field as:

```
prior{ip}.method='mps_snesim_tree';
prior{ip}.method='mps_snesim_list';
prior{ip}.method='mps_genesim';
```

'mps_snesim_tree' is the simulation method selected by default if it is not specified.

options for MPS

All options for the MPS type simulation algorithm are be available in the `prior{ip}.MPS` data structure.

For example, to set the number of used multiple grids, set the `MPS.n_multiple_grids` as

```
ip=1;
prior{ip}.type='mps';
prior{ip}.method='mps_snesim';
prior{ip}.x=0:.1:10;
prior{ip}.y=0:.1:20;
[m,prior]=sippi_prior(prior);
i=0;
for n_mul_grids=[0:1:4];
    prior{ip}.MPS.rseed=1;
    prior{ip}.MPS.n_multiple_grids=n_mul_grids;
    [m,prior]=sippi_prior(prior);
    i=i+1;subplot(1,5,i);
    imagesc(prior{1}.x,prior{1}.y,m{1});axis image
    title(sprintf('NMG = %d',n_mul_grids));
end
```

More details on the use of MPS can be found in the SoftwareX manuscript that describes MPS.

3.1.1.11 SNESIM

The 'SNESIM' type prior model utilizes the SNESIM algorithm, as implemented in Fortran available at *Stanford/SCRF*.

By default a training image (channel structures) from Sebastian Strebelle's PhD theses is used (if no training image is specified). A simple 2D type SNESIM prior model can be defined using the following code:

```
ip=1;
prior{ip}.type='SNESIM';
prior{ip}.x=[0:.1:10]; % X array
prior{ip}.y=[0:.1:20]; % Y array
```

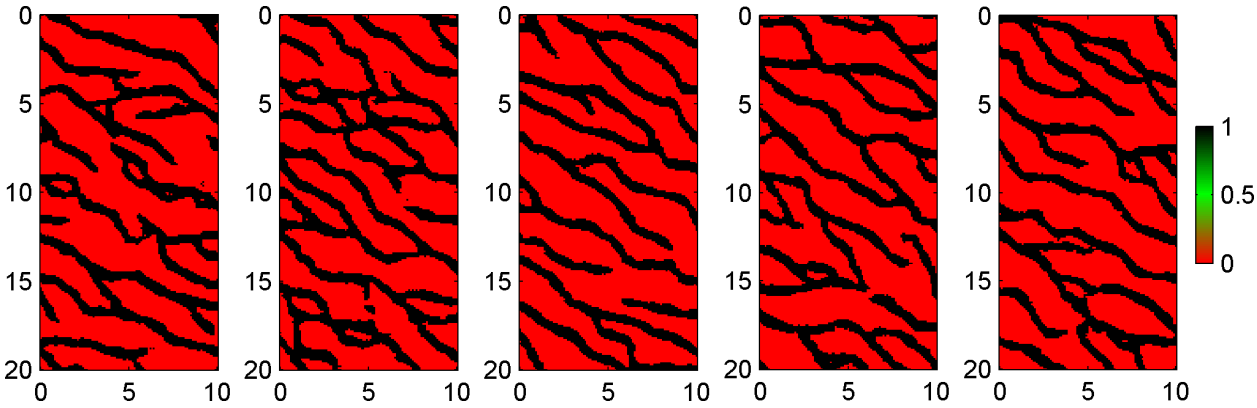
and 5 realizations from this prior can be visualized using

```
for i=1:5;
    m=sippi_prior(prior);
    subplot(1,5,i);
    imagesc(prior{1}.x,prior{1}.y,m{1});axis image
end
```

Note that the training image is always assumed to have the same units as the prior model, so in this case each pixel in the training image is assumed to be separated by a distance '0.1'.

Optionally 'scaling' and 'rotation' of the training image can be set. To scale the training image by 0.7 (i.e., structures will appear 30% smaller) and rotate the training 30 degrees from north use

```
ip=1;
prior{ip}.type='SNESIM';
prior{ip}.x=[0:.1:10]; % X array
prior{ip}.y=[0:.1:20]; % Y array
prior{ip}.scaling=.7;
prior{ip}.rotation=30;
```



Custom training image

A custom training image can be set using the `ti` field, which must be either a 2D or 3D matrix.

```
% create TI from image
EXAMPLE EXAMPLE

% setup the prior
ip=1;
prior{ip}.type='SNESIM';
prior{ip}.x=[0:.1:10]; % X array
prior{ip}.y=[0:.1:20]; % Y array
prior{ip}.ti=ti;
```

Note that the training image MUST consist of integer index values starting from 0 (i.e. '0', '1', '2', ...).

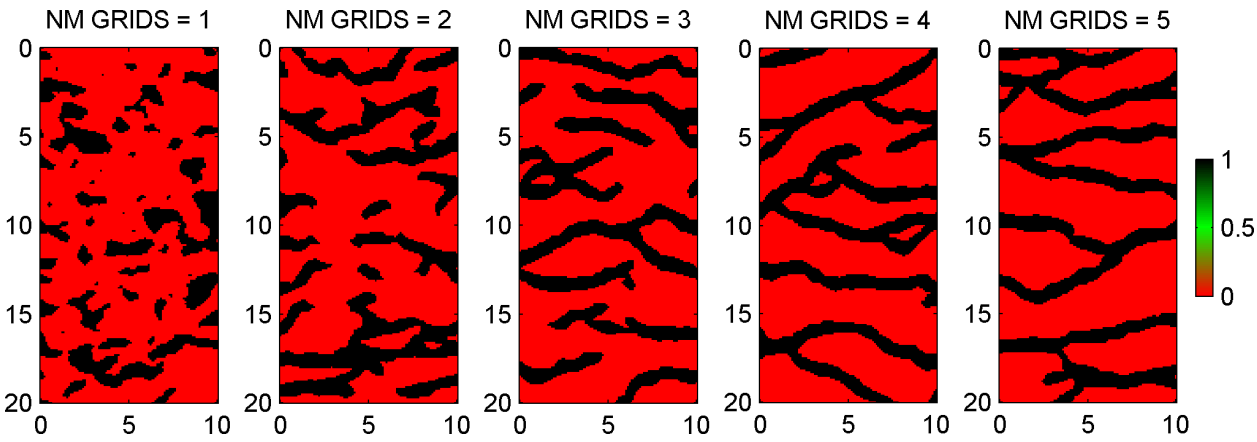
Complete customization

If the `prior` structure is returned from *sippi_prior* using

```
[m,prior]=sippi_prior(prior);
```

then an XML structure `prior{1}.S.XML` will be available. This allows a complete customization of all settings available in SGeMS. For example, the different realizations, using 1, 2, and 3 multiple grids can be obtained using

```
ip=1;
prior{ip}.type='SNESIM';
prior{ip}.x=[0:.1:10]; % X array
prior{ip}.y=[0:.1:20]; % Y array
[m,prior]=sippi_prior(prior);
for i=1:5;
    prior{ip}.S.XML.parameters.Nb_Multigrids_ADVANCED.value=i;
    subplot(1,3,5);
    imagesc(prior{1}.x,prior{1}.y,m{1});axis image
end
```



3.1.1.12 SNESIM_STD

The 'SNESIM_std' type prior model utilizes the SNESIM algorithm, as implemented in *SGeMS*. It can be called using the same options as *sippi_prior_snesim*.

3.2 `data`: Data and data uncertainties/noise

`data` is a Matlab structure that defines any number of data and the associated uncertainty/noise model.

`data{1}` defines the first data set (which must always be defined), and any number of additional data sets can be defined in `data{2}`, `data{3}`, ...

This allows to consider for example seismic data in `data{1}`, and electromagnetic data in `data{2}`.

For each set of data, a Gaussian noise model (both correlated and uncorrelated) can be specified. The noise model for different data types (e.g. `data{1}` and `data{2}` are independent).

Once the noise model has been defined, the log-likelihood related to any model, `m`, with the corresponding *forward response*, `d`, can be computed using

```
[d,forward,prior,data]=sippi_forward(m,forward,prior,data)
logL=sippi_likelihood(data,d)
```

where `d` is the output of *sippi_forward*.

The specification of the noise model can be divided into a description of the *measurement noise* (mandatory) and the *modeling error* (optional).

3.2.1 Gaussian measurement noise

3.2.1.1 Uncorrelated Gaussian measurement noise

To define a set of observed data, [0,1,2], with an associated uncorrelated uncertainty defined by a Gaussian model with mean 0 and standard deviation 2, use

```
data{1}.d_obs=[0 1 2]';
data{1}.d_std=[2 2 2]';
```

which is equivalent to (as the noise model for each data is the same, and independent)

```
data{1}.d_obs=[0 1 2]';
data{1}.d_std=2;
```

One can also choose to define the uncertainty using a variance as opposed to the standard deviation

```
data{1}.d_obs=[0 1 2]';
data{1}.d_var=4;
```

3.2.1.2 Correlated Gaussian measurement noise

Correlated Gaussian measurement uncertainty can be specified using the `Cd` field, as for example

```
data{1}.Cd=[4 1 0 ; 1 4 1 ; 0 1 4];
```

Note that `data{1}.Cd` must be of size [NDxND], where ND is the number of data in `data{1}.d_obs`.

3.2.2 Gaussian modeling error

The modeling error refers to errors caused by using for example an imperfect forward model, see HCM14.

A Gaussian model of the modeling error can be specified by the mean, `dt`, and the covariance, `Ct`.

For example

```
data{1}.dt=[0 0 0];
data{1}.Ct=[4 4 4; 4 4 4; 4 4 4];
```

is equivalent to

```
data{1}.Ct=4
```

which implies a zero mean modeling error with a covariance model where all model parameters has a covariance of 4.

sippi_compute_modelization_forward_error can be used to estimate the modeling error related to using an approximate forward model. See the *tomography example*, for an *example of accounting for correlated modeling errors*, following HCM14.

3.3 `forward`: The forward model:

The specification of the `prior` and `data` is intended to be generic, applicable to any inverse problem considered. The forward problem, on the other hand, is typically specific for each different inverse problem.

In order to make use of SIPPI to sample the posterior distribution of an inverse problem, the solution to the forward problem must be embedded in a Matlab function with the following input and output arguments:

```
[d,forward,prior,data]=sippi_forward(m,forward,prior,data,id)
```

`m` is a realization of the prior model, and `prior` and `data` are the Matlab structures defining the prior and the noise model (see *Prior* and *Data*)

`id` is optional, and can be used to compute the forward response of a subset of the different types of data available (i.e. `data{1}`, `data{2}`,...)

The `forward` variable is a Matlab structure that can contain any information needed to solve the forward problem. Thus, the parameters for the `forward` structure is problem dependent. One option, `forward.forward_function` is though generic, and point to the m-file that implements the forward problem.

The output variable `d` is a Matlab structure of the same size of `data`. Thus, if 4 types of data have been specified, then `d` must also be a structures of size 4.

```
length(data) == length(d);
```

Further, `d{i}` must refer to an array of the same size as `data{i}.d_obs`.

An example of an implementation of the forward problem related to a simple line fitting problem is:

```
function [d,forward,prior,data]=sippi_forward_linefit(m,forward,prior,data);
    d{1}=forward.x*m{2}+m{1};
```

This implementation requires that the 'x'-locations, for which the y-values of the straight line is to be computed, is specified through `forward.x`. Say some y-data has been observed at locations x=[1,5,8], with the values [2,4,9], and a standard deviation of 1 specifying the uncertainty, the forward structure must be set as

```
forward.forward_function='sippi_forward_linefit';
forward.x=[1,5,8];
```

while the data structure will be

```
data{1}.d_obs=[2 4 9]
data{1}.d_std=1;
```

This implementation also requires that the prior model consists of two 1D prior types, such that

```
m=sippi_prior(prior)
```

returns the intercept in `m{1}` and the gradient in `m{2}`.

An example of computing the forward response using an intercept of 0, and a gradients of 2 is then

```
m{1}=0;
m{2}=2;
d=sippi_forward(m,forward)
```

and the corresponding log-likelihood of m, can be computed using

```
logL=sippi_likelihood(data,d);
```

[see more details and examples related to polynomial line fitting at *polynomial line fitting*].

The *Examples* section contains more example of implementation of different forward problems.

3.3.1 Validating `prior`, `data`, and `forward`

A simple way to test the validity of `prior`, `data`, and `forward` is to test if the following sequence can be evaluated without errors:

```
% Generate a realization, m, of the prior model
m=sippi_prior(prior);
% Compute the forward response
d=sippi_forward(m,forward,prior,data);
% Evaluate the log-likelihood of m
logL=sippi_likelihood(data,d);
```

3.3.1.1 sippi_forward_linear.m

3.3.1.2 sippi_forward_traveltime.m

CHAPTER 4

Sampling the posterior

Once the *prior*, *data*, and *forward* data structures have been defined, the associated a posteriori probability can be sampled using [the extended rejection sampler](/chapSampling/chapSampling_rejection.md) and [the extended Metropolis sampler](/chapSampling/chapSampling_metropolis.md).

If the inverse problem is linear and Gaussian it can be solved using [linear least squares](/chapSampling/linear-least-squares.md) inversion.

4.1 The rejection sampler: : `sippi_rejection.m`

The rejection sampler provides a simple, and also in many cases inefficient, approach to sample the posterior distribution.

At each iteration of the rejection sample an independent realization, m_pro, of the prior is generated, and the model is accepted as a realization of the posterior with probability Pacc = L(m_pro)/L_max. It can be initiated using

```
options.mcmc.nite=400000; % Number of iteration, defaults to 1000
options.mcmc.i_plot=500; % Number of iteration between visual updates, defaults to 500
options=sippi_rejection(data,prior,forward,options);
```

By default the rejection sampler is run assuming a maximum likelihood of 1 (i.e. L_max = 1). If L_max is known, then it can be set using in the `options.Lmax` or `options.logLmax` fields

```
options.mcmc.Lmax=1e-9;
options=sippi_rejection(data,prior,forward,options);
```

or

```
options.mcmc.logLmax=log(1e-9);
options=sippi_rejection(data,prior,forward,options);
```

Alternatively, L_max can be automatically adjusted to reflect the maximum likelihood found while running the rejection sampler using

```
options.mcmc.adaptive_rejection=1
options=sippi_rejection(data,prior,forward,options);
```

An alternative to rejection sampling, also utilizing independent realizations of the prior, that does not require one to set L_max is the *independent extended metropolis sampler*, which may be computatinoally superior to the rejection sampler,

4.2 The extended Metropolis sampler: `sippi_metropolis.m`

The extended Metropolis algorithm is in general a much more efficient algorithm (compared to the rejection sampler for sampling the a posteriori probability

The extended Metropolis sampler can be run using

```
options.mcmc.nite=40000;     % number of iterations, default nite=30000
options.mcmc.i_sample=50;    % save the current model for every 50 iterations, default,
↪ i_sample=500
options.mcmc.i_plot=1000;    % plot progress of the Metropolis sampler for every 100␣
↪iterations
                             % default i_plot=50;
options.txt='case_line_fit'; % descriptive name appended to output folder name,␣
↪default txt='';

[options,data,prior,forward,m_current]=sippi_metropolis(data,prior,forward,options)
```

One can choose to accept all steps in the Metropolis sampler, which willresult in an algorithm sampling the prior model, using

```
options.mcmc.accept_all=1; % default [0]
```

One can choose to accept models that lead to an improvement in the likelihood, which results in an optimization like algorithm using

```
options.mcmc.accept_only_improvements=1; % default [0]
```

See *sippi_metropolis* for more details.

4.2.1 Controlling the step length

One optionally, as part of running the *extended Metropolis sampler*, automatically update the 'step'-length of the *sequential Gibbs sampler* in order to ensure a specific approximate acceptance ratio of the Metropolis sampler. See CHM12 for details.

The default parameters for adjusting the step length, as given below, are set in the '*prior.seq_gibbs*' structure.
These parameters will be set the first time 'sippi_prior' is called with the 'prior' structure as output.The default parameters.

```
prior{m}.seq_gibbs.step_min=0;
prior{m}.seq_gibbs.step_max=1;
prior{m}.seq_gibbs.i_update_step=50
```

(continues on next page)

(continued from previous page)

```
prior{m}.seq_gibbs.i_update_step_max=1000
prior{m}.seq_gibbs.n_update_history=50
prior{m}.seq_gibbs.P_target=0.3000
```

By default, adjustment of the step length, in order to achieve an acceptance ratio of 0.3 ('prior{m}.seq_gibbs.P_target'), will be performed for every 50 ('prior{m}.seq_gibbs.i_update_step') iterations, using the acceptance ratio observed in the last 50 ('prior{m}.seq_gibbs.i_update_history') iterations.

Adjustment of the step length will be performed only in the first 1000 ('prior{m}.seq_gibbs.i_update_step_max') iterations.

In order to disable automatic adjustment of the step length simply set

```
prior{m}.seq_gibbs.i_update_step_max=0; % disable automatic step length
```

4.2.2 Controlling how to perturb the prior (when multiple priors exist)

When more than one prior structure is defined (e.g. `prior{1}`, `prior{2}`,...) the perturbation strategy can be controlled by the `options.mcmc.pert_strategy` field.

`options.mcmc.pert_strategy.perturb_all = 0;`: [default] a random chosen prior is chosen for perturbation. The other prior types are ignored.

`options.mcmc.pert_strategy.perturb_all = 1;` Perturb all prior model at each iteration.

`options.mcmc.pert_strategy.perturb_all = 2;` Perturb a random selection of all prior at each iteration.

Further, when `options.mcmc.pert_strategy.perturb_all = 0;` the freqyency with which a prior is perturbed can be set using `options.mcmc.pert_strategy.i_pert` and `options.mcmc.pert_strategy.i_pert_freq`.

For example to perturb only `prior{1}` and `prior{3}`, while perturbing `prior{1}` 4 times more freqeuntly than`prior{3}` use

```
options.mcmc.pert_strategy.i_pert = [1,3]; % only perturb prior 1 and 3
options.mcmc.pert_strategy.i_pert_freq = [2 8]; % perturb prior 3 80% of
                                                % the time and prior 1 20% % of the␣
↪time
```

By default the probability of perturbing a specific prior is uniform. In case of 3 prior types this is set using:

```
options.mcmc.pert_strategy.i_pert = [1,2,3]; % only perturb prior 1 and 3
options.mcmc.pert_strategy.i_pert_freq = [1,1,1]; % same probability of perturbing␣
↪all prior types
```

4.2.3 The independent extended Metropolis sampler

The 'independent' extended Metropolis sampler, in which each proposed model is independant of the previously visited model, can be chosen by forcing the 'step'-length to be 1 (i.e. leading to independant samples from the prior), using e.g.

```
% force independent prior sampling
for ip=1:length(prior);
    prior{ip}.seq_gibbs.step=1;
```

(continues on next page)

(continued from previous page)

```
    prior{ip}.seq_gibbs.i_update_step_max=0;
end
% run 'independent' extended Metropolis sampling
[options,data,prior,forward,m_current]=sippi_metropolis(data,prior,forward,options)
```

4.2.4 Annealing schedule

Simulated annealing like behavior can be controlled in the `options.mcmc.anneal` structure. By default annealing is disabled.

Annealing consist of setting the temperature (similar to scaling the noise). A temperature foes not affect the exploration. For temperatures larger than 1, the acceptance ratio increases (the exploration of the Metropolis sampler increases). For temperatures below 1, the acceptance ratio decreases (and hence the exploration of the Metropolis sampler).

The temperature is set to `options.mcmc.anneal.T_begin` at any iteration before `options.mcmc.anneal.i_begin`. The temperature is set to `options.mcmc.anneal.T_end` at any iteration after `options.mcmc.anneal.i_end`.

In between iteration number `options.mcmc.anneal.i_start` and `options.mcmc.anneal.i_end` the temperature changes following either an exponential decay (`options.mcmc.anneal.type='exp'`), or simple linear interpolation (`options.mcmc.anneal.type='linear'`).

An annealing schedule can be used allow a Metropolis sampler that allow exploration of more of the model space in the beginning of the chain. Recall though that the posterior is not sampled until (at least) the annealing has been ended at iteration, `options.mcmc.anneal.i_end`, if the `options.mcmc.anneal.T_end=1`. This can potentially help not to get trapped in a local minimum.

To use this type of annealing, where the annealing stops after 10000 iterations, after which the algorithm performs like a regular Metropolis sampler, use for example

```
options.mcmc.anneal.i_begin=1; % default, iteration number when annealing begins
options.mcmc.anneal.i_end=10000; %  iteration number when annealing stops
```

which is equivalent to

```
options.mcmc.anneal.i_begin=1; % default, iteration number when annealing begins
options.mcmc.anneal.i_end=10000; %  iteration number when annealing stops
options.mcmc.anneal.T_begin=5; % start temperature
options.mcmc.anneal.T_end=1; % end temperature
```

4.2.5 Parallel tempering

Parallel tempering is implemented according to S13. It is an extension of the Metropolis algorithm, that start a number of parallel chains of Metropolis sampling algorithms. Each chain is run with a different temperature, and the state of each chain is allowed jump between chains according to some rules that ensure the correct probability density is sampled. This allow the sampling algorithm to better handle a posterior distribution with multiple, disconnected, areas of high probability.

The following three setting enable parallel tempering.

```
% TEMPERING
options.mcmc.n_chains=3; % set number of chains (def=1, no multiple chains)
```

(continues on next page)

(continued from previous page)

```
options.mcmc.T=[1 2 3];      % set temperature of chains [1:n_chains]
options.mcmc.chain_frequency_jump=0.1; % probability allowing a jump between two␣
↪chains
```

`options.mcmc.n_chains` defines the number of chains. If not set only one chain is used, and the no parallel tempering is performed.

`options.mcmc.T` defines the temperature of each chain. A temperature of '1', which is the default, implies no tempering. A higher temperature
allow a chain to be more exploratory.

`options.mcmc.chain_frequency_jump` defines the frequency with which a jump from one chain to another is suggested. A value of one means that a
jump is proposed at each iteration, while a value of 0.1 (default) means that a jump is only proposed with 10 percentage probability (on average
one in 10 iterations).

4.3 Linear Least Squares inversion: `sippi_least_squares.m`

If the prior is defined using a pure (no histogram reproduction) Gaussian type prior model, a Gaussian likelihood/noise for the data, and a linear forward model, then the a posteriori probability density will also be Gaussian.

In this case the Gaussian a posterior probability density can be directly estimated using Linear Least Squares inversion (see e.g. Tarantola and Valette (1982) or Tarantola (2005)), which is available through `sippi_least_squares.m`, which can be called using

```
[m_est,Cm_est,m_reals,,options,data,prior,forward]=sippi_least_squares(data,prior,
↪forward,options);
```

To compute posterior mean and covariance only use e.g.

```
[m_est,Cm_est]=sippi_least_squares(data,prior,forward);
```

A number of realizations from the posterior distribution can also be computed using

```
[m_est,Cm_est,m_reals,options]=sippi_least_squares(data,prior,forward);
```

In this case the computed realizations, as well as all computed data, will be stored in the folder `options.txt`, similar to when using sippi_metropolis.m and sippi_rejection. Some figures analyzing the posterior distrbibution can then be generated using e.g. `sippi_plot_posterior.m`.

`options.lsq` contains all the operators that is used for the least squares inversion (`d0`,`Cd`,`m0`,`Cm`,`G`).

CHAPTER 5

Examples

Perhaps the simplest approach to learn using SIPPI is by trying out (and alter) an example.

A priori models

Examples of different types of a priori models

Line fitting

Probabilistic polynomial regression

Cross-hole GPR tomographic inversion

An example of 2D/3D inversion of crosshole traveltime data obtained from GPR measurements from Stevns, Denmark. (The case examined in Hansen et al. (2013b)).

Covariance model inference

An example of inference of covariance model parameters, from either direct observations of model parameters, of from linear average observations (such as data related to an linear inverse problem).

See also:

Hansen, T. M., Cordua, K. S., & Mosegaard, K. (2015). A General Probabilistic Approach for Inference of Gaussian Model Parameters from Noisy Data of Point and Volume Support. Mathematical Geosciences, 47(7), 843-865. PDF.

5.1 Examples of A priori models

5.1.1 Multiple 1D Gaussian prior model

A prior model consisting of three independent 1D distributions (a Gaussian, Laplace, and Uniform distribution) can be defined using

```
ip=1;
prior{ip}.type='GAUSSIAN';
prior{ip}.name='Gaussian';
```

(continues on next page)

(continued from previous page)

```
prior{ip}.m0=10;
prior{ip}.std=2;

ip=2;
prior{ip}.type='GAUSSIAN';
prior{ip}.name='Laplace';
prior{ip}.m0=10;
prior{ip}.std=2;
prior{ip}.norm=1;

ip=3;
prior{ip}.type='GAUSSIAN';
prior{ip}.name='Uniform';
prior{ip}.m0=10;
prior{ip}.std=2;
prior{ip}.norm=60;

m=sippi_prior(prior);

m =

    [14.3082]    [9.4436]    [10.8294]
```

1D histograms of a sample (consisting of 1000 realizations) of the prior models can be visualized using ...

```
sippi_plot_prior_sample(prior);
```

5.1.2 ## Multivariate Gaussian prior with unknown covariance model properties.

The *FFT-MA* type a priori model allow separation of properties of the covariance model (covariance parameters, such as range, and anisotropy ratio) and the random component of a Gaussian model. This allow one to define a Gaussian a priori model, where the covariance parameters can be treated as unknown variables.

In order to treat the covariance parameters as unknowns, one must define one a priori model of type `FFTMA`, and then a number of 1D `GAUSSIAN` type a priori models, one for each covariance parameter. Each gaussian type prior model must have a descriptive name, corresponding to the covariance parameter that is should describe:

```
prior{im}.type='gaussian';
prior{im}.name='m_0';     % to define a prior for the mean
prior{im}.name='sill';    % to define a prior for sill (variance)
prior{im}.name='range_1'; % to define a prior for the range parameter 1
prior{im}.name='range_2'; % to define a prior for the range parameter 2
prior{im}.name='range_3'; % to define a prior for the range parameter 3
prior{im}.name='ang_1';   % to define a prior for the first angle of rotation
prior{im}.name='ang_2';   % to define a prior for the second angle of rotation
prior{im}.name='ang_3';   % to define a prior for the third angle of rotation
prior{im}.name='nu';      % to define a prior for the shape parameter, nu
              %   (only applies when the Matern type Covariance model is used)
```

A very simple example of a prior model defining a 1D Spherical type covariance model with a range between 5 and 15 meters, can be defined using:

```
im=1;
prior{im}.type='FFTMA';
prior{im}.x=[0:.1:10]; % X array
```

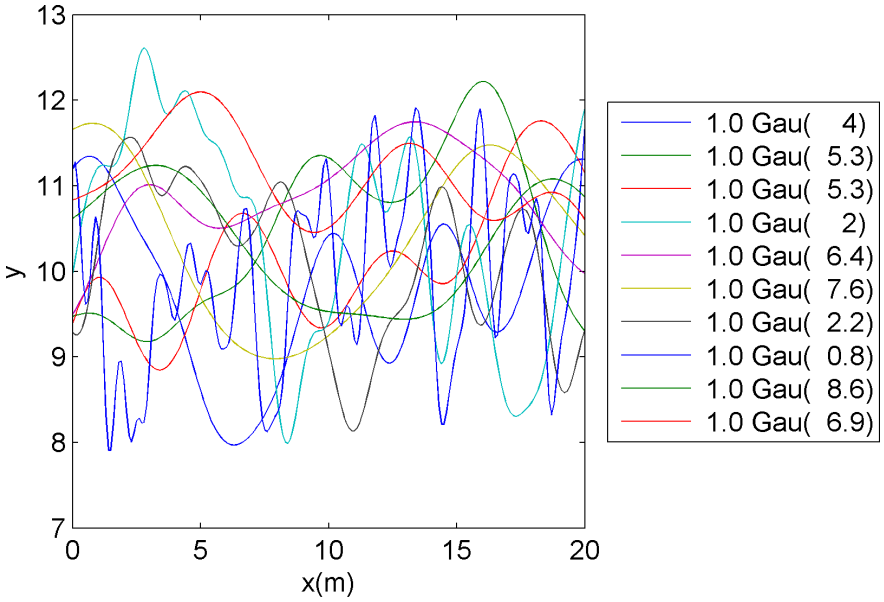
(continues on next page)

(continued from previous page)

```
prior{im}.m0=10;
prior{im}.Va='1 Sph(10)';
prior{im}.fftma_options.constant_C=0;


im=2;
prior{im}.type='gaussian';
prior{im}.name='range_1';
prior{im}.m0=10;
prior{im}.std=5
prior{im}.norm=80;
prior{im}.prior_master=1; % -- NOTE, set this to the FFT-MA type prior for which this␣
↪prior type
                          % should describe the range
```

Note that the the field `prior_master` must be set to point the to the FFT-MA type a priori model (through its id/number) for which it should define a covariance parameter (in this case the range).

10 independent realizations of this type of a priori model are shown in the following figure



Such a prior, as all prior models available in SIPPI, works with *sequential Gibbs sampling*, allowing a random walk in the space of a prior acceptable models, that will sample the prior model. An example of such a random walk can be performed using

```
prior{1}.seq_gibbs.step=.005;
prior{2}.seq_gibbs.step=0.1;
clear m_real;
for i=1:150;
    [m,prior]=sippi_prior(prior,m);
    m_real(:,i)=m{1};
end
```

An example of such a set of 150 dependent realization of the prior can be seen below

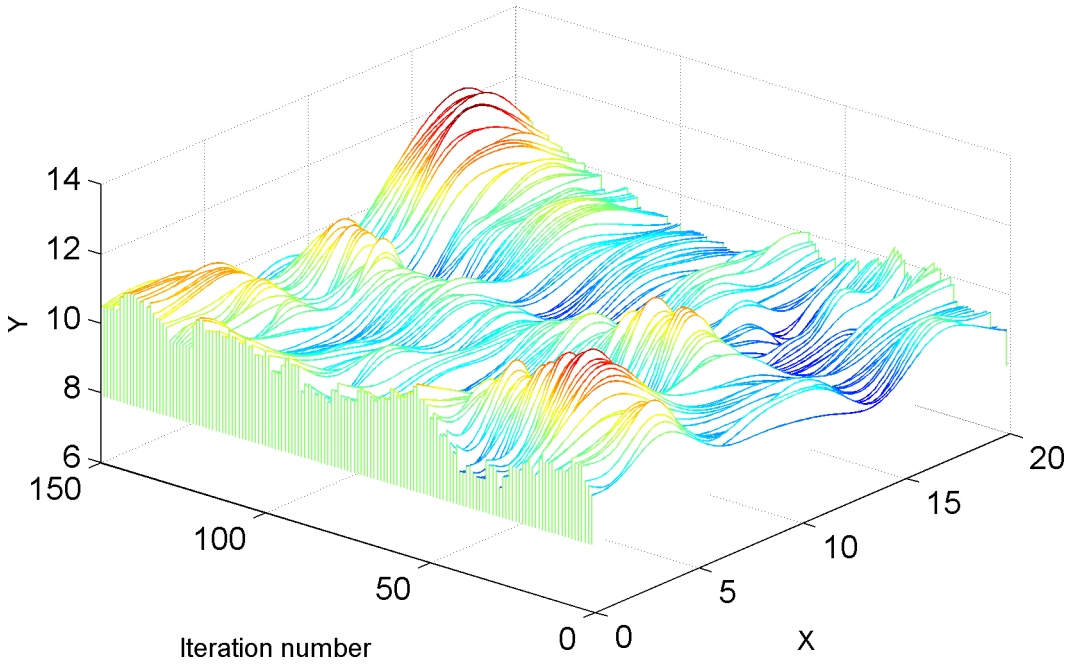


Fig. 1: Unknown

5.1.3 Simulating the cover of Joy Division's Unknown Pleasers

SIPPI can be used to simulate something similar to the iconic cover of Joy Divisions's Unknown Pleasures.

```
%% Setup two prior structuresx=linspace(0.1,.9,101);d_target=rand(1,1000);
nl=60; % number of lines

ip=1;
prior{ip}.type='FFTMA';
prior{ip}.x=x;
prior{ip}.y=1:1:nl;
prior{ip}.m0=0;
prior{ip}.Va='0.1 Sph(.1,90,0.001)';
prior{ip}.d_target=d_target/20;

ip=2;
prior{ip}.type='FFTMA';
prior{ip}.x=x;
prior{ip}.y=1:1:nl;
prior{ip}.m0=0;
prior{ip}.Va='.99 Gau(.06,90,0.001) + .01 Sph(.1,90,0.001)';
prior{ip}.d_target=3*d_target;

%% Compute a padding matrix, such that prior{2} is only used 'in the middle'␣
↪pad=zeros(size(x));
x1=0.3;ix1=find(x>=x1);
ix1=min(ix1);x2=0.4;
ix2=find(x>=x2);ix2=ix2(1);xx1=ix1:1:ix2;
fadein=sin(interp1([ix1 ix2],[0 pi/2],xx1));
pad(find(x>x1&x<(1-x2)))=1;
pad(xx1)=fadein;
```

(continues on next page)

(continued from previous page)

```
xx2=fliplr(length(x)-xx1);
pad(xx2)=fliplr(fadein);

% add a small fading from left to rightlinpad=linspace(1.3,0.0,length(x));pad=pad.
↪*linpad;

pad=repmat(pad,nl,1);

%% Generate first realization.[m,prior]=sippi_prior(prior);
mm=m{1}+m{2}.*pad;
```

mm will now contain one realizations that when visualized should look similar to the original album cover. A movie of a random walk in this 'prior' model obtained using sequential Gibbs sampling can now be performed, and will render something similar to this video: {% youtube %}https://www.youtube.com/watch?v=La0uESBYLEA{% endyoutube %}

The full source is available at SIPPI/examples/prior_tests/unknown_pleasures.m

5.2 Probilistic covariance/semivariogram indeference

This chapter documents how to use SIPPI to infer properties of a covariance/semivariogram model from noisy data (both data of point support and linear average data can be considered).

To perform probabilistic inference of covariance model parameters one must

1. define the data and associated uncertainty (if any),
2. define a prior model of the covariance model parameters that one wish to infer information about, and
3. define the linear forward operator (only applicable if data are not of point support).

The methodology is published in Hansen et al. (2015).

5.2.1 Specification of covariance model parameters

The following covariance model properties can be defined, that allow defining an isotropic or an-isotropic covariance model in 1D, 2D, or 3D:

```
type                % covariance model type (1->Sph, 2->Exp, 3->Gau)
m0                  % the mean
sill                % the variance
nugget_fraction     % percentage of the variance assigned to a Nugget
range_1             % range in primary direction
range_2             % range in secondary direction
range_3             % range in tertiary direction
ang_1               % first angle of rotation
ang_2               % second angle of rotation
ang_3               % third angle of rotation
```

Inference of a full 1D covariance model requires defining [type,sill,nugget_fraction,range_1].

Inference of a full 2D covariance model requires defining [type,sill,nugget_fraction,range_1,range_2,ang_1].

Inference of a full 3D covariance model requires defining [type,sill,nugget_fraction,range_1,range_2,range_3,ang_1,ang_2,ang_3].

In order to define which of the covariance model parameters to infer information about, simply define a prior structure for any of these parameters, as 1D type SIPPI prior model.

For example, to simple infer information about the range in the primary direction, with a priori distribution of the range as U[0,3] use

```
forward.Cm='1 Sph(10)';

im=1;
prior{im}.type='uniform';
prior{im}.name='range_1'; % the 'name' field is used to identify the covariance model␣
↪parameter!
prior{im}.min=0;
prior{im}.max=3;
```

In this case an `range_1` refers to the isotropic range in the covariance model defined in the `forward.Cm` field

If, instead

```
forward.Cm='1 Sph(10,90,.25)';
```

then `range_1` would refer to the range in the direction of maximum continuity (90 degrees from North). `range_2` will in this case be fixed.

As described above, the covariance model type can be considered as a unknown parameter, that can be inferred during inversion. This may pose some problems as discussed in HCM15.

To infer the covariance model type, a prior 1D structure should be defined as e.g.

```
im=1;
prior{im}.type='uniform';
prior{im}.name='type'; %
prior{im}.min=0;
prior{im}.max=3;
```

Any value between 0 and 1 defines a spherical type covariance. Any value between 1 and 2 defines an exponential type covariance. Any value between 3 and 3 defines a Gaussian type covariance.

Thus no prior should be defined for the 'type' prior that can provide values below 0, and above 3. In the case above, all three covariance model types has the sane a priori probability.

A detailed description of how to parameterize the inverse covariance model parameter problem, can be found in *sippi_forward_covariance_inference*.

5.2.2 Inferring a 2D covariance model from the Jura data set - Example of point support

The Jura data set (see Goovaerts, 1997) contains a number observations of different properties in 2D. Below is an example of how to infer properties of a 2D covariance model from this data set.

A Matlab script implementing the steps below can be found here: *jura_covariance_inference.m*

5.2.2.1 Load the Jura data

Firs the Jura data is loaded.

```
% jura_covariance_inference
%
% Example of inferring properties of a Gaussian model from point data
%
```

(continues on next page)

(continued from previous page)

```
%% LOAD THE JURA DATA
clear all;close all
[d_prediction,d_transect,d_validation,h_prediction,h_transect,h_validation,x,y,pos_
↪est]=jura;
ix=1;
iy=2;
id=6;

% get the position of the data
pos_known=[d_prediction(:,[ix iy])];

% perform normal score transformation of tha original data
[d,o_nscore]=nscore(d_prediction(:,id));
h_tit=h_prediction{id};
```

5.2.2.2 Setting up SIPPI for covariance parameter inference

First a SIPPI 'prior' data structure is setup do infer covariance model parameters for a 2D an-isotropic covariance model. That is, the `range_1`, `range_2`, `ang_1`, and `nugget_fraction` are defined using

```
im=0;
% A close to uniform distribution of the range, U[0;3].
im=im+1;
prior{im}.type='uniform';
prior{im}.name='range_1';
prior{im}.min=0;
prior{im}.max=3;

im=im+1;
prior{im}.type='uniform';
prior{im}.name='range_2';
prior{im}.min=0;
prior{im}.max=3;

im=im+1;
prior{im}.type='uniform';
prior{im}.name='ang_1';
prior{im}.min=0;
prior{im}.max=90;

im=im+1;
prior{im}.type='uniform';
prior{im}.name='nugget_fraction';
prior{im}.min=0;
prior{im}.max=1;
```

Thus the a priori information consists of uniform distributions of ranges between 0 and 3, rotation between 0 and 90, and a nugget fraction between 0 and 1 is.

Then the data structure is set up, using the Jura data selected above, while assuming a Gaussian measurement uncertainty with a standard deviation of 0.1 times the standard deviation of the data:

```
%% DATA
data{1}.d_obs=d; % observed data
```

(continues on next page)

(continued from previous page)

```
data{1}.d_std=0.1*std(d);.4; % uncertainty of observed data (in form of standard␣
↪deviation of the noise)
```

Finally the forward structure is setup such that `sippi_forward_covariance_inference` allow inference of covariance model parameters.

In the `forward` structure the location of the point data needs to be given in the `pos_known` field, and the initial mean and covariance needs to be set. Also, the name of the forward function used (in this case *sippi_forward_covariance_inference*) must be set. Use e.g.:

```
%% FORWARD
forward.forward_function='sippi_forward_covariance_inference';
forward.point_support=1;
forward.pos_known=pos_known;

% initial choice of N(m0,Cm), mean and sill are 0, and 1, due
% due to normal score
forward.m0=mean(d);
forward.Cm=sprintf('%3.1f Sph(2)',var(d));
```

Now, SIPPI is set up for inference of covariance model parameters. Use for example the Metropolis sampler to sample the a posterior distribution over the covariance model parameters using:

```
options.mcmc.nite=100000;
options.mcmc.i_plot=1000;
options.mcmc.i_sample=25;
options.txt=name;
[options,data,prior,forward,m_current]=sippi_metropolis(data,prior,forward,options)

sippi_plot_prior(options.txt);
sippi_plot_posterior(options.txt);
```

Sampling the posterior provides the following 2D marginal distributions Note how several areas of high density scatter points (i.e. areas with high posterior probability) can be found.

5.3 Polynomial line fitting

An simple of application of SIPPI is to perform linefitting, as a probabilistic inverse problem.

Here follows simple polynomial (of order 0, 1 or 2) line-fitting is considered. Example m-files and data can be found in the `SIPPI/examples/case_linefit` folder.

First, the forward problem is defined. Then examples of stochastic inversion using SIPPI is demonstrated using a a synthetic data set.

5.4 The challenge

Assume that some observed data `d_obs` are available, as function of a corresponding set of model parameters `x`. Assume alse that the observed data are contaminated by Gaussian noise, with mean 0, and standard devaiation 10.

```
::
```

cd SIPPI/examples/case_linefit load sippi_linefit_data errorbar(x,d_obs,d_std)

or

::

x=[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]; d_obs = [-36.5 -10.8 -24.0 -22.5 -16.1 -10.7 -9.2 -1.6 -6.4 -0.7 -18.5]; d_std = ones(size(d_obs)).*10; errorbar(x,d_obs,d_std)

The problem is now to infer information about the 3 parameters defining a 2nd order polynomium, given the data above.

5.5 Solving the problem using SIPPI

In roder to use SIPPI, the `data`, `prior`, and `forward` data strutures must be setup, and the a way of solving the forward problem must be defined.

5.5.1 The data

The observed data, as well as the associated uncertainty id defined in the `sippi` structure

::

data{1}.d_obs=d_obs; data{1}.d_std=d_std;

5.5.2 The prior

A prior distributions of tree parameters, reflecting coefficient of the polynomium, can be defined as for example;

::

%% setup the prior model % the intercept im=1; prior{im}.type='gaussian'; prior{im}.name='intercept'; prior{im}.m0=0; prior{im}.std=30;

% 1st order, the gradient im=2; prior{im}.type='gaussian'; prior{im}.name='gradient'; prior{im}.m0=0; prior{im}.std=4; prior{im}.norm=80;

% 2nd order im=3; prior{im}.type='uniform'; prior{im}.name='2nd'; prior{im}.min=-.3; prior{im}.max=.3;

Note that each model parameter is associated with it ownm independent, distritution. Here two Gaussian (N(0,30^2), and N(0,4^2)) as well as a uniform (U[-0.3 0.3]) is assumed.

A sample of the prior can be generated using

::

m=sippi_prior(prior)

m =

1×3 cell array

[-19.4704] [3.1968] [-0.1655]

5.5.3 The forward problem

Finally, a way to solvge the forward problem must be implemented, that takes as input, at least `m` (a realization of the pror) and `forward` (a matlab structure that contains any informaion neede to solve the forward problem), and that produces an output `d`, where `d{1}` is of exactly the same size and type as ´data{1}.d_obs´.

One way to implement this as the m-file `[sippi_forward_linefit.m](#sippi_forward_linefit):`

```
::
```

```
% sippi_forward_linefit Line fit forward solver for SIPPI % % [d,forward,prior,data]=sippi_forward_linefit(m,forward,prior,data); % function [d,forward,prior,data]=sippi_forward_linefit(m,forward,prior,data);

if length(m)==1; d{1}=forward.x.*0 + m{1};

elseif length(m)==2; d{1}=forward.x*m{2}+m{1};

else d{1}=forward.x.^2*m{3}+forward.x*m{2}+m{1};

end
```

Here `forward.x` must be an array of the x-locations, for which the d-values will be computed.

Note that the prior must be defined such that `prior{1}` refer to the intercept, `prior{2}` to the gradient, and `prior{3}` to the 2nd order polynomial coefficient.

If only one prior type is defined then the forward response will just be a constant, and if two prior types are defined, then the forward response will be a straight line.

Having implemented the m-file that solves the forward problem in the style reqiured by SIPPI, the forward can be setup using

```
::
```

```
%% Setup the forward model in the 'forward' structure forward.x=x forward.forward_function='sippi_forward_linefit';
```

5.5.4 Evaluate the prior, data, and forward

A simple way find problems related to how `prior`, `data`, `forward`, and `sippi_forward_linefit` has been setup, correctly is to test wether he follwing three lines can be executed without errors:

```
::
```

```
m=sippi_prior(prior); d=sippi_forward(m,forward); logL=sipppi_likelihood(d,data);
```

5.6 Sampling the a posterior distribution

Information about the model parameters can be inferred by running the `extended Metropolis sampler <#metropolis>__` or the `rejection sampler <#rejection>__` using

5.6.1 Using the Metropolis sampler

The `extended Metropolis sampler <chapSampling_metropolis.md>__` can be setup and run using:

::

options.mcmc.nite=40000; % Run for 40000 iterations options.mcmc.i_sample=100; % Save every 100th visited model to disc options.mcmc.i_plot=5000; % Plot the progress information for every 2500 iterations options.txt='case_line_fit_2nd_order'; % descriptive name for the output folder

[options]=sippi_metropolis(data,prior,forward,options);

Generic statistics about the posterior can be plotted using.

::

% plot posterior statistics, such as 1D and 2D marginals from the prior and posterior distributions sippi_plot_prior_sample(options.txt); sippi_plot_posterior(options.txt); 20140521_1644_sippi_metropolis_case_line_fit_2nd_order_m1_3_posterior_sample.png

The figure below show the prior and posterior distribution of the 3 model parameters, as well as the reference values (used to generate the synthetic data set, in green)

The figure below plots forward response related to the obtained sample of the posterior distribution over the model parameters (gray), as well as the observed data (black), and the noise free reference data obtained from the reference set of model parameters. |image0|

5.6.2 Using the rejection sampler

In a similar manner the `rejection sampler <chapSampling_rejection.md>`__ can be setup and run using

::

options.mcmc.adaptive_rejection=1; % automatically adjust the normalizing likelihood options.mcmc.nite=100000; options=sippi_rejection(data,prior,forward,options);

5.7 Cross hole tomography

SIPPI includes a publically available cross hole GPR data from Arrenaes that is free to be used.

SIPPI also includes the implementation of multiple methods for computing the travel time delay between a set of sources and receivers. This allows SIPPI to work on for example cross hole tomographic forward and inverse problems.

This examples is probably the best way to learnthe capabilities of SIPPI.

This section contains examples for setting up and running a cross hole tomographic inversion using SIPPI using the reference data from Arrenæs, different types of a priori and forward models.

ArrenaesData.md

The following section contains a few different examples, and many more are located in [SIPPI/examples/case_tomography/].

5.7.1 Reference data set from Arrenæs

A 2D/3D data set of recorded travel time data from a cross hole Georadar experiment is available in the 'data/crosshole' folder.

4 Boreholes were drilled, AM1, AM2, AM3, and AM4 at the locations shown below

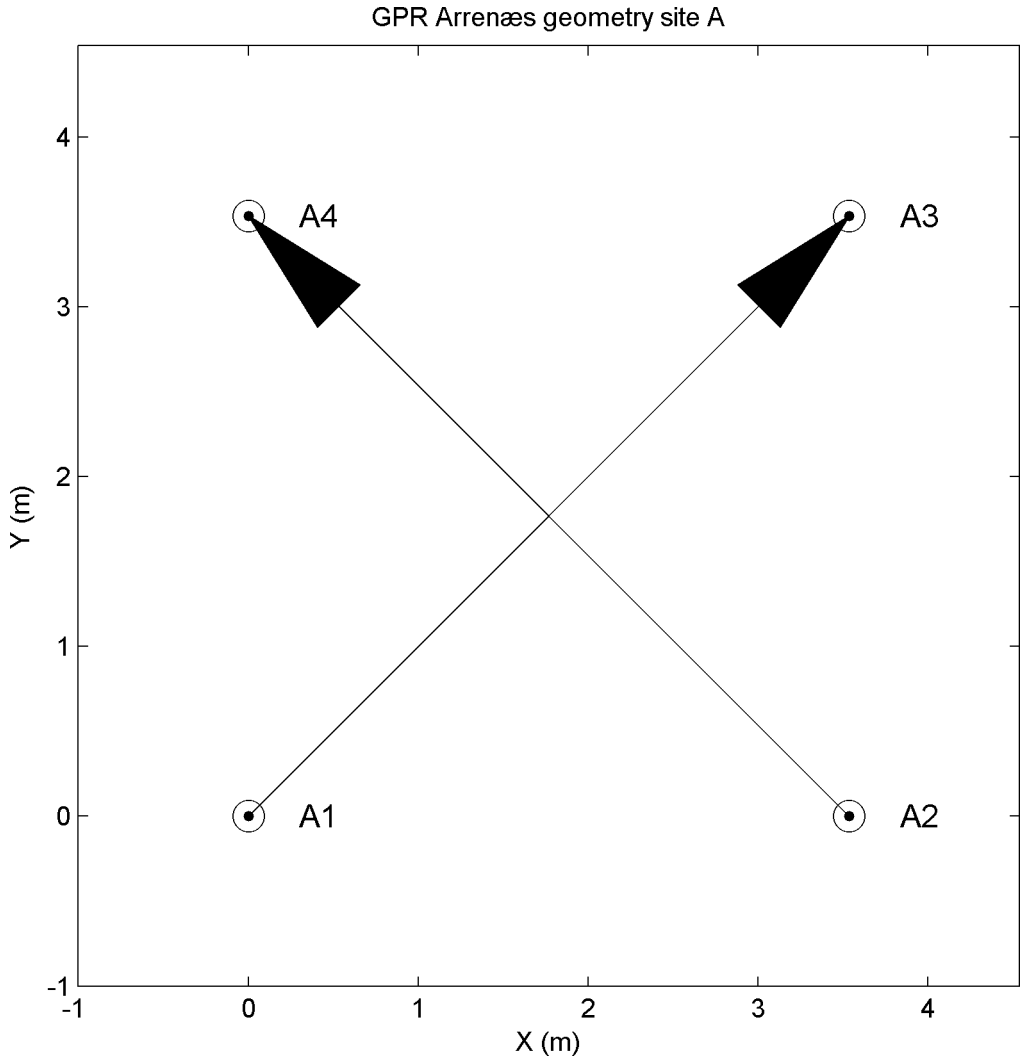


Fig. 2: Location of boreholes AM1, AM2, AM3, and AM4 at Arrenæs.

Travel time data were collected between boreholes AM1 and AM3, and AM2 and AM4 respectively, in a depth interval between 1m and 12m. The travel times for each of the two 2D data sets are available in the *AM13_data.mat* and *AM24_data.mat* files. All the data have been combined in the 3D data set available in *AM1234_data.mat*.

All mat-files contains the following variable

```
S --> [ndata,ndim] each row contains the position of the source
R --> [ndata,ndim] each row contains the position of the receiver
d_obs --> [ndata,1] each row contains the observed travel time in milliseconds
d_std --> [ndata,1] each row contains the standard deviation of the uncertainty of␣
↪the observed travel time in milliseconds
```

All data are also available as ASCII formatted EAS files in `AM13_data.eas`, `AM24_data.eas`, and `AM1234_data.eas`.

The following 3 Figures show the ray coverage (using straight rays) for each of the AM13, AM24, and AM1234 data sets. The color of each ray indicates the average velocity along the ray computed using v_av = raylength/d_obs. AM13 ray coverageAM24 ray coverageAM1234 ray coverage.

5.7.2 Cross hole travel time delay computation: The forward problem

A number of different methods for solving the problem of computing the first arrival travel time of a seismic or electromagnetic wave traveling between a source in one borehole and a receiver in another borehole has been implemented in the m-file 'sippi_forward_traveltime'.

```
[d,forward,prior,data]=sippi_forward_traveltime(m,forward,prior,data,id,im)
```

In order to use this m-file to describe the forward problem specify the 'forward_function' field in the `forward` structure using

```
forward.forward_function='sippi_forward_traveltime';
```

In order to use `sippi_forward_traveltime`, the location of the sources and receivers must be specified in the `forward.S` and `forward.R`. The number of columns reflect the number of data, and the number of rows reflect whether data are 2D (2 columns) or 3D (3 columns):

```
forward.S %  [ndata,ndim]
forward.R %  [ndata,ndim]
```

Using for example the data from Arrenæs, the forward geometry can be set up using

```
D=load('AM13_data.mat');
forward.sources=D.S;
forward.receivers=D.R;
```

In addition the method used to compute the travel times must also be given (see below).

In order to use the geometry from the AM13 reference data, and the Eikonal solution to the wave-equation, the `forward` structure can be defined using

```
D=load('AM13_data.mat');
forward.forward_function='sippi_forward_traveltime';
forward.sources=D.S;
forward.receivers=D.R;
forward.type='eikonal';
```

5.7.2.1 Ray type forward model (high frequency approximation)

Ray type models are based on an assumption that the wave propagating between the source and the receiver has infinitely high frequency. Therefore the travel time delay is due to the velocity along a ray connecting the source and receiver.

The linear so-called straight ray approximation, which assumes that the travel time for a wave traveling between a source and a receiver is due to the travel time delay along a straight line connecting the source and receiver, can be chosen using

```
forward.type='ray';
forward.linear=1;
```

The corresponding so-called bended-ray approximation, where the travel time delay is due to the travel time delay along the fast ray path connecting a source and a receiver, can be chosen using

```
forward.type='ray';
forward.linear=0;
```

When sippi_forward_traveltime has been called once, the associated forward mapping operator is stored in 'forward.G' such the the forward problem can simply be solved by calling e.g. 'd{1}=forward.G*m{1}'

5.7.2.2 Fat Ray type forward model (finite frequency approximation)

Fat type model assume that the wave propagating between the source and the receiver has finite high frequency. This means that the travel time is sensitive to an area around the raypath, typically defined using the 1st Fresnel zone.

A linear fat ray kernel can be chosen using

```
forward.type='fat';
forward.linear=1;
forward.freq=0.1;
```

and the corresponding non-linear fat kernel using

```
bforward.type='fat';
forward.linear=0;
forward.freq=0.1;
```

Note that the center frequency of the propagating wave must also be provided in the 'forward.freq' field. The smaller the frequency, the 'fatter' the ray kernel.

For 'fat' type forward models we rely on the method described by Jensen, J. M., Jacobsen, B. H., and Christensen-Dalsgaard, J. (2000). Sensitivity kernels for time-distance inversion. Solar Physics, 192(1), 231-239

5.7.2.3 Born type forward model (finite frequency approximation)

Using the Born approximation, considering only first order scattering, can be chosen using

```
forward.type='born';
forward.linear=1;
forward.freq=0.1;
```

For a velocity field with small spatial variability one can compute 'born' type kernels (using 'forward.linear=0', but as the spatial variability increases this is not possible.

For the 'born' type forward model we make use if the method described by Buursink, M. L., Johnson, T. C., Routh, P. S., and Knoll, M. D. (2008). Crosshole radar velocity tomography with finite-frequency Fresnel volume sensitivities. Geophysical Journal International, 172(1), 1-17.

5.7.2.4 The eikonal equation (high frequency approximation)

The eikonal solution to the wave-equation is a high frequency approximation, such as the one given above.

However, it is computationally more efficient to solve the eikonal equation directly, that to used the 'forward.type='ray';' type forward model.

To choose the eikonal solver to compute travel times use

```
forward.type='eikonal';
```

The Accurate Fast Marching Matlab toolbox : http://www.mathworks.com/matlabcentral/fileexchange/24531-accurate-fast-marching is used to solve the Eikonal equation.

5.7.3 Cross hole traveltime inversion of GPR data obtained from Arrenæs: Inversion of cross hole GPR data from Arrenaes

In the following a simple 2D Gaussian a priori model is defined, and SIPPI is used to sample the corresponding a posteriori distribution. (A example script is avalable at examples/case_tomography/sippi_AM13_metropolis_gaussian.m

5.7.3.1 Setting up the data structure

For a more detailed decsription of the available data see the section Arrenæs Data.

The data from Arrenæse can be loaded, and set up in a data structure appropriate for SIPPI using

```
% Load the data
D=load('AM13_data.mat');

%% Setup SIPPI 'data' strcuture
id=1;
data{id}.d_obs=D.d_obs;
data{id}.d_std=D.d_std;
data{id}.dt=0; % Mean modelization error
data{id}.Ct=1; % Covariance describing modelization error
```

Note that only the `d_obs` and `d_std` needs to be defined in the `data` structure. This will define an uncorrelated Gaussian noise model.

A correlated noise, for example describing modeling errrors, can be decsribed in the `dt` and `Ct` variables. In the above example a Gaussian modelization error, N(dt,Ct) is defined. The total uncertainty model is then comprised on both the observation uncertainties, and the modeling error.

Here the use of a correlated modeling error is introduced to because we will make use of a forward model, the eikonal solver, that we know will systematically provide faster travel times than can be obtained from the earth. In reality the wave travelling between bore holes never has infinitely high frequency as assumed by using the eikonal solver. The eikonal solver provides the fast travel time along a ray connecting the source and receiver. Therefore we introduce a modelization error, that will allow all the travel times to be biased with the same travel time.

5.7.3.2 Setting up the prior model

The a priori model is defined using the `prior` data structure. Here a 2D Gaussian type a priori model in a 7x13 m grid (grid cell size .25m) using the FFTMA type a priori model. The a priori mean is 0.145 m/ns, and the covariance function a Spherical type covariance model with a range of 6m, and a sill(variance) of 0.0003 m^2/ns^2.

```
%% SETUP PRIOR
im=1;
prior{im}.type='FFTMA';
prior{im}.m0=0.145;
prior{im}.Va='.0003 Sph(6)';
prior{im}.x=[-1:.15:6];
prior{im}.y=[0:.15:13];
```

The VISIM or CHOLESKY type a priori models can also be used simply by substituting 'FFTMA' type with 'VISIM' or 'CHOLESKY' above.

5.7.3.3 Setting up the forward structure

The m-file `sippi_forward_traveltime.m` has been implemented that allow computation of `sippi_forward_traveltime.m`, and the location of the sources and receivers needs to be set

```
forward.forward_function='sippi_forward_traveltime';
forward.sources=D.S;
forward.receivers=D.R;
forward.type='ray_2d';
```

`forward.type` defines the type of forward model to use. Here a simple, linear forward models based on 2D raytracing is chosen. See the secion Traveltime Forward Modeling for more details on how to control the forward modeling.

In order to visualize the Source-Receiver setup, and linear forward operator (for the linear kernels only) run the following

```
sippi_plot_traveltime_kernel(forward,prior);
```

5.7.3.4 Testing the setup

As the `prior`, `data`, and `forward` have been defined, one can in principle initiate an inversion. However, it is advised to perform a few test before applying the inversion.

First, one should check that independent realization of the prior model resemble the a priori knowledge. A sample from the prior model can be generated and visualized calling `sippi_plot_prior_sample`

```
sippi_plot_prior_sample(prior);
```

which provides the following figure The one can check that the forward solver, and the computation of the likelihood wors as expected using

```
% generate a realization from the prior
m=sippi_prior(prior);
% Compute the forward response related to the realization of the prior model␣
↪generated above
[d]=sippi_forward(m,forward,prior,data);
% Compute the likelihood
[logL,L,data]=sippi_likelihood(d,data);
% plot the forward response and compare it to the observed data
sippi_plot_data(d,data);
```

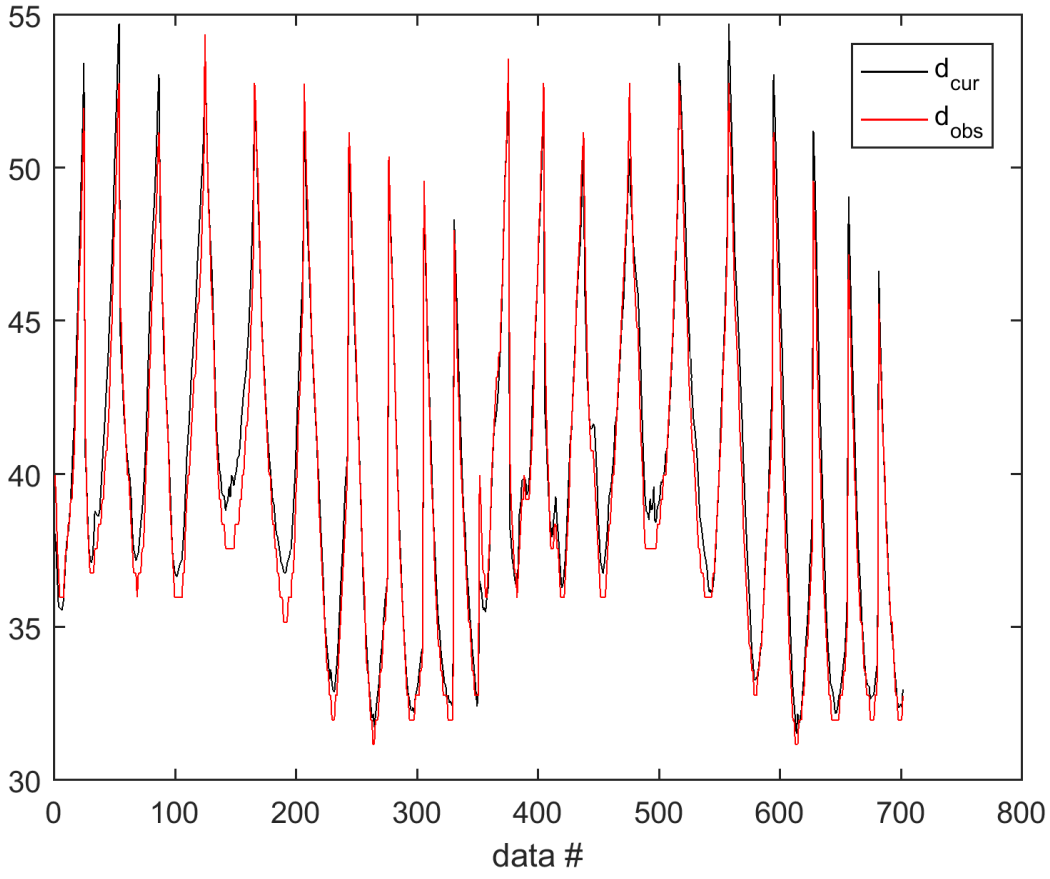
which produce a figure of the forward response of a realization of the prior, compared to observed data in `data{1}.d_obs`:

5.7.3.5 Sampling the a posterior distribution using the extended Metropolis algorithm

The *extended Metropolis sampler* can now be run using sippi_metropolis.

```
options=sippi_metropolis(data,prior,forward);
```

In practice the user will have to set a few options, controlling the behavior of the algorithm. In the following example the number of iterations is set to 500000; the current model is saved to disc for every 2500 iterations. The step-lengt, data fit, log-likelihood and current model is shown for every 1000 iterations:


55
50
45
40
35
30
0
100
200
300
400
500
600
700
800
data #
d cur
d obs

```
options.mcmc.nite=500000; % optional, default:nite=30000
options.mcmc.i_sample=2500; % optional, default:i_sample=500;
options.mcmc.i_plot=5000; % optional, default:i_plot=50;
options=sippi_metropolis(data,prior,forward,options);
```

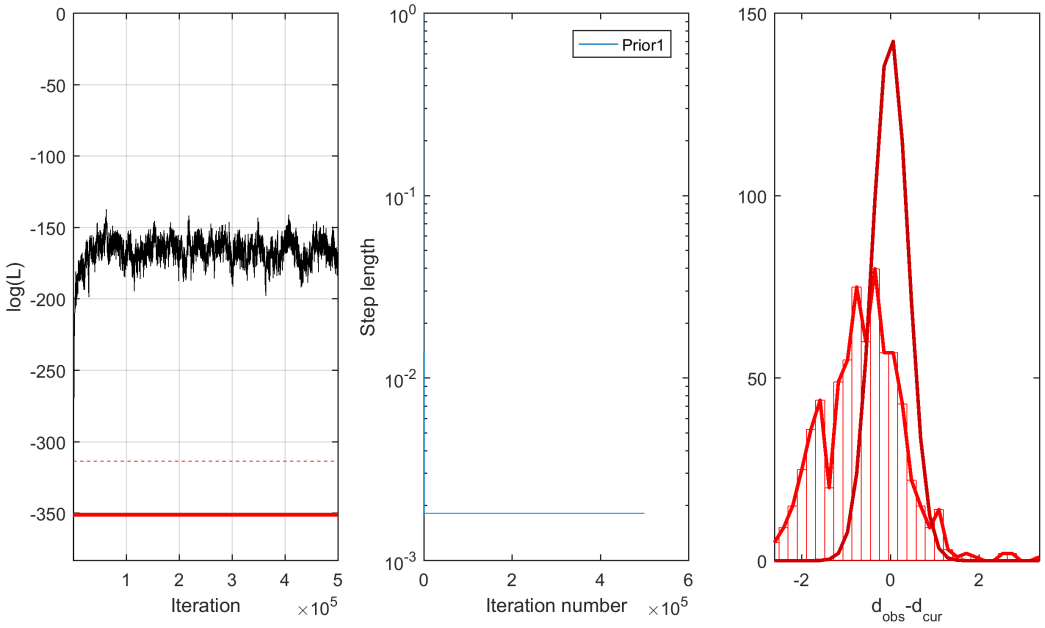
By default the *'step'-length* for sequential Gibbs sampling is adjusted (to obtain an average acceptance ratio of 30%) for every 50 iterations until iteration number 1000.

An output folder will be generated with a filename formatted using 'YYYYMMDD-HHMM', followed by a automatic description. In the above case the output folder could be name `20140701_1450_sippi_metropolis_eikonal`. The actual folder name is returned in `options.txt`.

One can define a description for the folder name by setting `options.txt` before running sippi_metropolis.

The folder contains one mat file, with the same name as the folder name, and N ASCII files (where N=length(prior); one for each a priori type) which contains the models saved to disc. They also have the same name as the folder name, appended with '_m1.asc', '_m2.asc', and so forth.

As the sampling is performed, progress the progress in likelihood, step-lengt and data mistfit is shown as e.q.



Posterior statistics

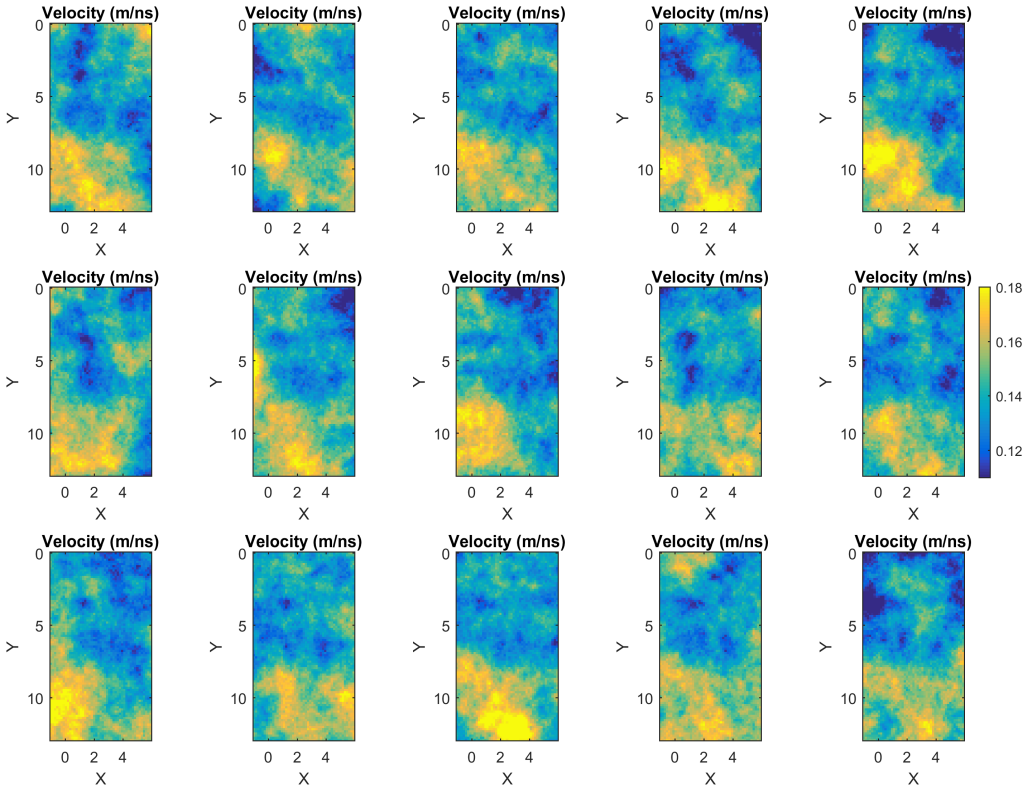
A number of plots can be generated automatically after the sampling has ended, using `sippi_plot_posterior.m`. One can either be located in the folder from which `sippi_metropolis` was run and do:

```
sippi_plot_posterior(options.txt);
```

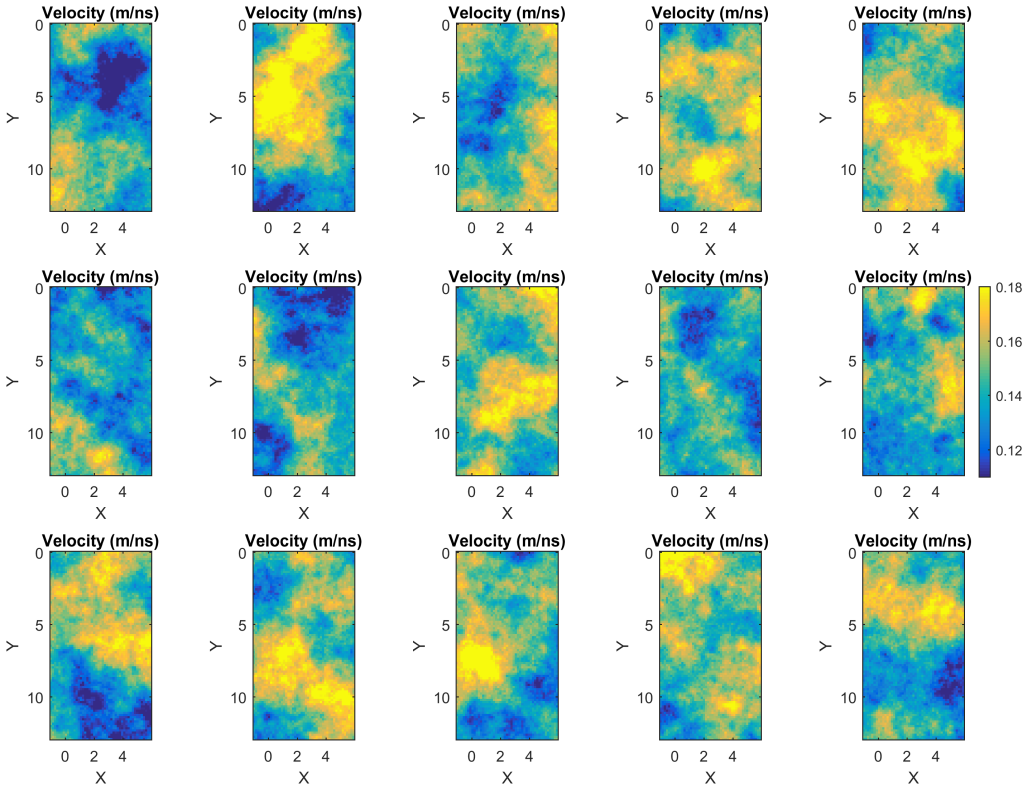
or one can go to the folder created by `sippi_metropolis` and do

```
cd 20140701_1450_sippi_metropolis_eikonal
sippi_plot_posterior;
```

This will visualize for example a sample (consisting of 15 realizations) from the posterior:


Velocity (m/ns)
Y
X
0
5
10
0 2 4
0.18
0.16
0.14
0.12

which should be compared the a similar sample of the prior distribution:



The point-wise mean and standard deviation (E-types) are also shown:

Also a movie of 200 (if that many has been created) posterior realizations is generated: {% youtube %}https://www.youtube.com/watch?v=wyLFYxHAkck{% endyoutube %}

5.7.4 AM13 Gaussian with bimodal velocity distribution

A Matlab script for the following example is avalable at sippi_AM13_metropolis_bimodal.m.

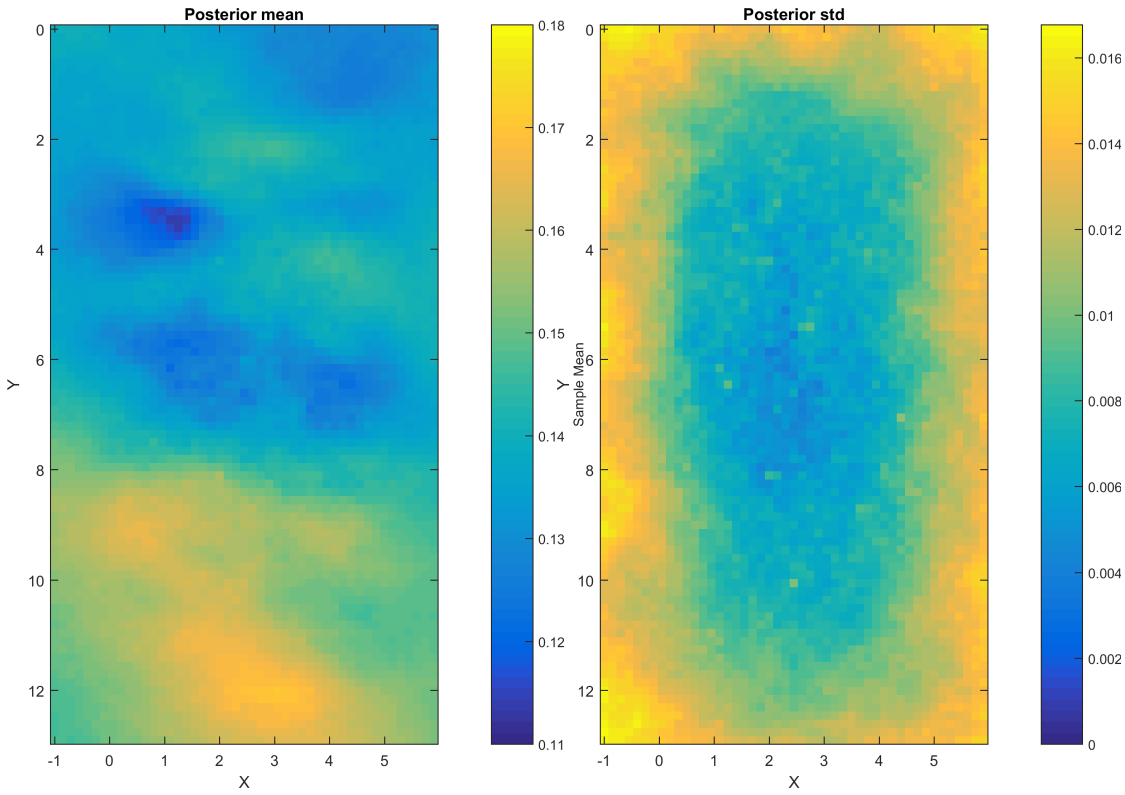
The *GAUSSIAN* and *FFTMA*a prior types implicitly assume a normal distribution of the model parameter.

It is however possible to change the Gaussian distribution to any shaped distribution, using a normal score transform. Note that when this is done the given semivariogram model for the *FFTMA* a priori model will not be reproduced. If this is a concern, then the *VISIM* type a priori model should be used.

The data and forward structures is identical to the one described in the *previous* example.

```
%% Load the travel time data set from ARRENAES
clear all;close all
D=load('AM13_data.mat');
options.txt='AM13';

%% SETUP DATA
id=1;
data{id}.d_obs=D.d_obs;
```

(continues on next page)


Posterior mean
Posterior std
Y
X
Sample Mean

(continued from previous page)

```
data{id}.d_std=D.d_std;
data{id}.Ct=D.Ct+1; % Covariance describing modeling error

% SETUP THE FORWARD MODEL USED IN INVERSION
forward.forward_function='sippi_forward_traveltime';
forward.sources=D.S;
forward.receivers=D.R;
forward.type='fat';forward.linear=1;forward.freq=0.1;
```

The desired distribution (the 'target' distribution) must be provided as a sample of the target distribution, in the `data{id}.d_target` distribution.

```
%% SETUP PRIOR
im=1;
prior{im}.type='FFTMA';
prior{im}.name='Velocity (m/ns)';
prior{im}.m0=0.145;
prior{im}.Va='.0003 Sph(6)';
dx=0.15;
prior{im}.x=[-1:dx:6];
prior{im}.y=[0:dx:13];
prior{im}.cax=[.1 .18];

% SET TARGET
N=1000;
prob_chan=0.5;
dd=.014*2;
d1=randn(1,ceil(N*(1-prob_chan)))*.01+0.145-dd; %0.1125;
d2=randn(1,ceil(N*(prob_chan)))*.01+0.145+dd; %0.155;
d_target=[d1(:);d2(:)];
prior{im}.d_target=d_target;
```

5 realizations from the corresponding a priori model looks like Figure *figure_title* compares the distribution from one realization of both prior models considered above.

As for the examples above, the a posteriori distribution can be sampled using e.g.

```
options.mcmc.nite=500000; % optional, default:nite=30000
options.mcmc.i_sample=500; % optional, default:i_sample=500;
options.mcmc.i_plot=1000; % optional, default:i_plot=50;
options=sippi_metropolis(data,prior,forward,options);

% plot posterior statistics
sippi_plot_posterior(options.txt);
```

5.7.5 AM13 Gaussian, Linear least squares tomography

A Matlab script for the following example is available at sippi_AM13_least_squares.m.

sippi_least_squares.m allow solving a linear inverse problem with Gaussian prior and noise model. The tomographic problem can be considered linear in case any of the linear forward models are chosen, and the prior parameterized in slowness.

Load the data

```
clear all;close all
D=load('AM13_data.mat');
txt='AM13';
```

Define a Gaussian noise model using e.g.:

```
%% THE DATA
id=1;
data{id}.d_obs=D.d_obs;
data{id}.d_std=D.d_std;
```

Define a Gaussiain type prior model, using(for example) the FFTMA method, using slowness (inverse velocity):

```
im=1;
prior{im}.type='FFTMA';
prior{im}.name='Slowness (ns/m)';
prior{im}.m0=7.0035;
prior{im}.Va='0.7728 Exp(6)';
prior{im}.x=[-1:dx:6];
prior{im}.y=[0:dx:13];
prior{im}.cax=1./[.18 .1];
```

Finally, define a linear forward model

```
forward.forward_function='sippi_forward_traveltime';
forward.type='ray';forward.linear=1;
% forward.type='fat';forward.linear=1; % alternative forward model
% forward.type='born';forward.linear=1; % alternative forward model
forward.sources=D.S;
forward.receivers=D.R;
forward.is_slowness=1; % USE SLOWNESS PARAMETERIZATION
```

The above represents a linear Gaussian inverse problem. This can be solved using sampling methods, or it can be solved using linear least squares inversion.

```
[m_est,Cm_est,m_reals,options]=sippi_least_squares(data,prior,forward,options);
```

5.8 Cross hole GPR tomopgraphy using Neural Networks

The example below requires the SIPPI toolbox.

```
% grl_nn
clear all;close all

createTrainingSet=1;
createReferenceModel=1;
useTargetDist=0;
Ntrain=40000;

Nr_modeling=6000; %size of sample for modeling error

TrainSizes=[1000 5000 10000 20000 40000];

splitData=3;
```

(continues on next page)

(continued from previous page)

```
epochs=30000;
hiddenLayerSize=80;

%
%Ntrain=2500;
%splitData=0;
%epochs=10000;
%hiddenLayerSize=20;
%TrainSizes=[1000 2500];


% mcmc
nite=2000000;;

%% LOAD DATA AND CONFIGURATION
D=load('AM13_data.mat');

% REVERSE S and R for fd forward
D2=D;
D.S(352:end,:)=D2.R(352:end,:);
D.R(352:end,:)=D2.S(352:end,:);
D.S(:,1)=D.S(:,1)+1;
D.R(:,1)=D.R(:,1)+1;
clear D2
i_use=1:1:size(D.S,1);
id=1;
data{id}.d_obs=D.d_obs(i_use);
data{id}.d_std=D.d_std(i_use).*0+0.1;


%% B: Define FD
%forward_fd;
forward.forward_function='sippi_forward_traveltime';
forward.sources=D.S(i_use,:);
forward.receivers=D.R(i_use,:);
forward.type='fd';
%forward.type='fat';forward.linear=1;forward.freq=0.1;
%forward.type='eikonal';
%forward.m0=prior{1}.m0;

forward.fd.t=1e-7;
forward.fd.addpar.Tg=100*10^6;
forward.fd.dx_fwd=0.1;

%% A: Define prior model
im=1;
dx=0.2;
prior{im}.type='FFTMA';
prior{im}.name='Velocity (m/ns)';
prior{im}.m0=0.1431;
prior{im}.Va='.000215 Sph(6)';
prior{im}.x=[-.5:dx:6.5];
prior{im}.x=[0:dx:7];
prior{im}.y=[0:dx:13];
prior{im}.cax=[-1 1].*.03+prior{im}.m0;

if useTargetDist==1;
```

(continues on next page)

(continued from previous page)

```
    d_target=[randn(1,100)*.003-0.01 randn(1,100)*.003+0.01]+prior{im}.m0;
    prior{im}.d_target=d_target;
    prior{im}.m0=0; %% MAKE SURE sippi_forward_traveltime tests for a non-zero␣
↪velocity
end


%% C: Make Reference model
if createReferenceModel==1
    rng(1);
    % Reference mo
    [m_ref,prior]=sippi_prior(prior);
    NM=prod(size(m_ref{1}));
    % reference data
    [d_ref,forward]=sippi_forward(m_ref,forward,prior);

    % data
    data{1}.d_ref=d_ref{1};
    data{1}.d_noise=randn(size(d_ref{1})).*data{1}.d_std;
    data{1}.d_obs=data{1}.d_ref+data{1}.d_noise;



    % compute reference data in m0
    m_ref0=m_ref;
    m_ref0{1}=m_ref0{1}.*0+prior{1}.m0;
    disp('computing reference forward')
    [d_ref0,forward0]=sippi_forward(m_ref0,forward,prior);
    d0=d_ref0{1};

    save grl_ReferenceModel
else
    load grl_ReferenceModel
end

%% D: Create M-D training data set for forward model
if createTrainingSet==1
    ATTS=zeros(length(m_ref{1}(:)),Ntrain);
    DATA=zeros(length(d_ref{1}(:)),Ntrain);

    iplot=1;
    t0=now;
    for i=1:Ntrain;

        if ((i/iplot)==round(i/iplot)&&(i>1));progress_txt(i,Ntrain,time_loop_end(t0,
↪i-1,Ntrain));end
        m=sippi_prior(prior);
        try
            d=sippi_forward(m,forward,prior,data);

            ATTS(:,i)=m{1}(:);
            DATA(:,i)=d{1}(:)-d0;
        catch
            disp(sprintf('Something went wrong.'));
            i=i-1;
        end
    end
```

(continues on next page)

(continued from previous page)

```
    save(sprintf('grl_%s_NM%d_NT%d',forward.type,NM,Ntrain));
else
    load grl_eikonal_NM2376_NT300
    %load grl_eikonal_NM2376_NT1000.mat
end


%% E: SETUP FORWARD MODELS

if ~exist('splitData');
    splitData=0; % SPLIT DATA FOR NN
end
if ~exist('epochs');
    epochs=100000; %
end
if ~exist('hiddenLayerSize');
    hiddenLayerSize=80;
end

forward_nn.forward_function='sippi_forward_mynn';
forward_nn.sources=forward.sources;
forward_nn.receivers=forward.receivers;
forward_nn.ATTS=ATTS;
forward_nn.DATA=DATA;
forward_nn.d0=d0;
clear DATA ATTS
forward_nn.splitData=splitData;
forward_nn.epochs=epochs;;
forward_nn.hiddenLayerSize=hiddenLayerSize;
forward_nn.max_nm=1e+10;
txt=sprintf('grl_NM%d_DX%d_%s_NT%d_SD%d_NH%d',NM,dx*100,forward.type,Ntrain,forward_
↪nn.splitData,forward_nn.hiddenLayerSize);
forward_nn.mfunc_string=txt;
disp(sprintf('%s: setting name ''%s'''',mfilename,txt));

% setup all forward models
i_forward=0;
if ~exist('TrainSizes');
    TrainSizes=[100 200];
end
for n_use=TrainSizes;
    if n_use<= size(forward_nn.ATTS,2);
        i_forward=i_forward+1;
        f_mul{i_forward}=forward_nn;
        f_mul{i_forward}.ATTS=forward_nn.ATTS(:,1:n_use);
        f_mul{i_forward}.DATA=forward_nn.DATA(:,1:n_use);
        txt_use=sprintf('grl_NM%d_DX%d_%s_NT%d_SD%d_NH%d',NM,dx*100,forward.type,n_
↪use,forward_nn.splitData,forward_nn.hiddenLayerSize);
        f_mul{i_forward}.mfunc_string=txt_use;
    end
end

% eikonal
i_forward=i_forward+1;
f_mul{i_forward}=forward;
f_mul{i_forward}.type='eikonal';
```

(continues on next page)

(continued from previous page)

```
% ray_2d
i_forward=i_forward+1;
f_mul{i_forward}=forward;
f_mul{i_forward}.type='ray_2d';
f_mul{i_forward}.linear=1;
f_mul{i_forward}.freq=0.1;
f_mul{i_forward}.r=2;
f_mul{i_forward}.normalize_vertical=0;


%% F: EVALUATE forward models once to setup NN and Linear operators
for i=1:length(f_mul);
    t1=now;
    [d_mul{i},f_mul{i}]=sippi_forward(m_ref,f_mul{i},prior);
    t2=now;
    time_mul{i}=(t2-t1)*3600*24;

end

save(sprintf('%s_forward',txt))

%% G: Estimate modeling errors
if ~exist('Nr_modeling');
    Nr_modeling=6000;
end
[Ct,dt,dd,d_full,d_app]=sippi_compute_modelization_forward_error(forward,f_mul,prior,
↪Nr_modeling);

% H: Setup one data structure per forward model, with the correct modeling error
for i=1:length(f_mul);

    %s=sum(abs(dd{5}));
    %ii=find(s<180);

    data_mul{i}=data;
    data_mul{i}{1}.dt=dt{i};
    data_mul{i}{1}.Ct=Ct{i};
end

save(sprintf('%s_modelerr',txt))


%% I: Perform probabilistic inversion using extended Metropolis sampling
if ~exist('nite');
    nite=1000000; %
end
options.mcmc.m_ref=m_ref;
options.mcmc.nite=nite;   % [1] : Number if iterations
options.mcmc.i_sample=ceil(options.mcmc.nite/1000); % : Number of iterations between␣
↪saving model to disk
options.mcmc.i_plot=100000;  % [1]: Number of iterations between updating plots
options.mcmc.i_save_workspace=100000;  % [1]: Number of iterations between

i_burnin=options.mcmc.nite/30;
prior{1}.seq_gibbs.i_update_step_max=i_burnin;

options.mcmc.anneal.i_begin=1; % default, iteration number when annealing begins
```

(continues on next page)

(continued from previous page)

```
options.mcmc.anneal.i_end=ceil(i_burnin/2); %  iteration number when annealing stops
options.mcmc.anneal.T_begin=5; % Start temperature for annealing
options.mcmc.anneal.T_end=1; % End temperature for annealing

%options.mcmc.n_chains=2; % set number of chains (def=1)
%options.mcmc.T=[1 1.1 1.2];      % set temperature of chains [1:n_chains]

% RUN MCMC

rseed=1;
for i=1:(length(f_mul));
    rng(rseed);
    if isfield(f_mul{i},'mfunc');
        options.txt=f_mul{i}.mfunc_string;
    else
        try
            %options.txt=sprintf('grl_NM%d_DX%d_%s_SD%d_NH%d',NM,dx*100,f_mul{i}.type,
↪forward_nn.splitData,forward_nn.hiddenLayerSize);
            options.txt=sprintf('grl_NM%d_DX%d_%s_SD%d_NH%d',NM,dx*100,forward_nn.
↪type,forward_nn.splitData,forward_nn.hiddenLayerSize);
        catch
            options.txt=txt;
        end
    end
    t_start=now;
    [o]=sippi_metropolis(data_mul{i},prior,f_mul{i},options);
    options_out{i}.txt=o.txt;
    %sippi_plot_posterior_sample(options_out{i}.txt);
    sim_minutes(i)=(now-t_start)*60*24;
end

save(sprintf('%s_inverted',txt))
```

CHAPTER 6

Bibliography

Hansen, T. M., Cordua, K. S., & Mosegaard, K. (2012). Inverse problems with non-trivial priors: Efficient solution through sequential Gibbs sampling. Computational Geosciences, 16(3), 593-611.
DOI:10.1007/s10596-011-9271-1.

Hansen, T. M., Cordua, K. S., Looms, M. C., & Mosegaard, K. (2013a). SIPPI: A Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 1 — Methodology. Computers & Geosciences, 52, 470-480.
DOI:10.1016/j.cageo.2012.09.004.

Hansen, T. M., Cordua, K. S., Looms, M. C., & Mosegaard, K. (2013b). SIPPI: A Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 2 — Application to crosshole GPR tomography. Computers & Geosciences, 52, 481-492.
DOI:10.1016/j.cageo.2012.09.001.

Hansen, T.M., Cordua. K.S., and Mosegaard, K. (2015). A general probabilistic approach for inference of Gaussian model parameters from noisy data of point and volume support. Mathematical Geosciences 47(7), pp 843-865.
DOI:10.1007/s11004-014-9567-5.

Hansen, T.M., Cordua, K. S., Jacobsen, B. J., and Mosegaard, K. (2015). Accounting for imperfect forward modeling in geophysical inverse problems - exemplified for cross hole tomography.
Geophsyics, 79(3) H1-H21, 2014.
DOI:10.1190/geo2013-0215.1.

Hansen, T. M., Vu, L. T., and Bach. T. (2016). MPSLIB: A C++ class for sequential simulation of multiple-point statistical models. Software X, vol 5, pp 127–133.
DOI:10.1016/j.softx.2016.07.001.

Sambridge, M. (2014). A parallel tempering algorithm for probabilistic sampling and multimodal optimization. Geophysical Journal International 196(1).
DOI:10.1093/gji/ggt342.

Tarantola, A., and Valette, B. (1982). Inverse problems= quest for information. J. geophys 50(3), 150-170.
PDF

Tarantola, A. (2005). Inverse problem theory and methods for model parameter estimation. SIAM.
PDF.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search