SIPPI

Ed. version 1.0

Contents

List of Figures

About

SIPPI is a Matlab toolbox (compatible with GNU Octave) that allows Sampling the solution of non-linear Inverse Problems with realistic a Priori Information.

In order to make use of SIPPI one has to

- Install and setup SIPPI
- Define the prior model, in form of the prior data structure
- Define the forward model, in form of the forward data structure, and the sippi_forward.m m-file
- Define the data and noise model, in form of the prior data structure
- Choose a method for sampling the a posteriori probability density (i.e., solution) of the inverse problem.

Details about the implementation and the methods implemented in SIPPI can be found in [HCM12], [CHM12], [HCLM13a], [HCLM13b] and, [HCM14].

This version of the documentation was compiled on Jun 30, 2014 , and refer to SIPPI version 1.00.

Chapter 1

Installation

1.1 SIPPI

Download the latest version of SIPPI from http://sippi.sourceforge.net.

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

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

1.1.1 SGeMS (optional)

To make use of the SISIM and SNESIM type priori models SGeMS needs to be available.

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

Chapter 2

Setting up SIPPI

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

- Define the prior model, in form of the prior data structure
- Define the forward model, in form of the forward data structure, and the sippi_forward.m m-file
- Define the data and noise model, in form of the prior data structure

[For examples of how to apply SIPPI for different problems, see the section with examples].

2.1 `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, 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);
sippi_plot_prior(prior,m);
```

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

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

Each prior type is defined by setting a number field in the `prior` Matlab structure. For example, an decsriptive name (which is can be optionallyt set) decsribing the prior can be set in the `name` field, e.g.

```
prior{1}.name='My Prior';
```

2.1.1 Types of a priori models

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

The FFTMA specifies 1D-3D Gaussian models using efficient unconditional sampling,

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 SNESIM type prior model specifies a 1D-3D multiple-point-based statistical prior model, which relies on training images from where the conditional dependencies of the spatial variables are obtained (i.e., learned). This type of prior model requires SGEMS to be installed.

The following section documents 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.

2.1.1.1 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 devation 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;
```

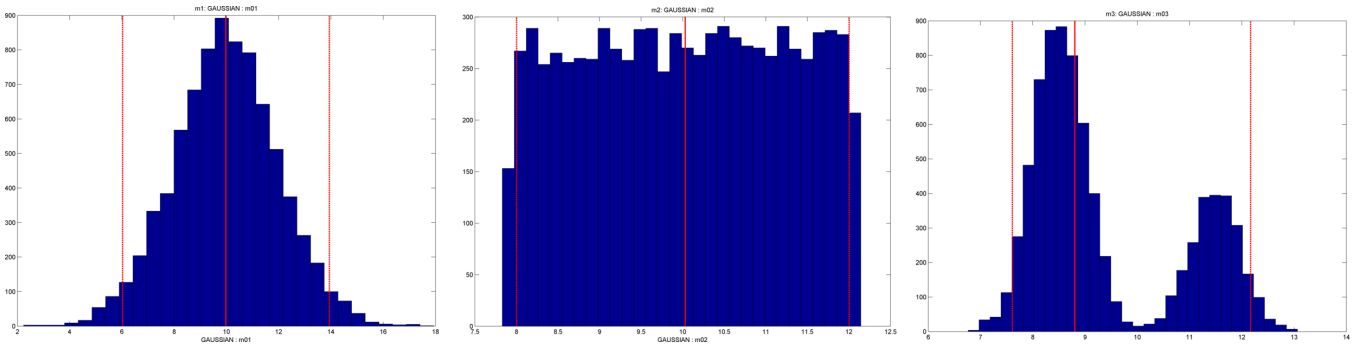
A 1D distribution with an arbitrary shape can be defined by setting `d_target`, which must contain a sample of the distribution that one would like to replicate. For example, to generate a sample from a non-symmetric bimodal distribution, one can use e.g.

```
% 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);
```



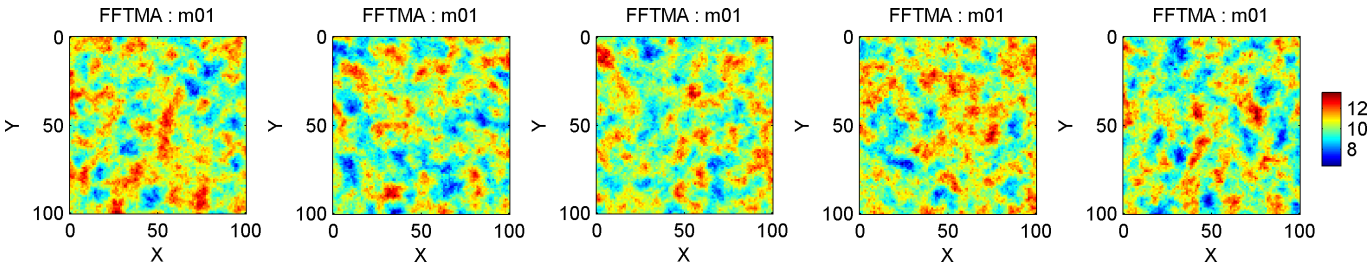
2.1.1.2 FFTMA - 3D Gaussian model

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.

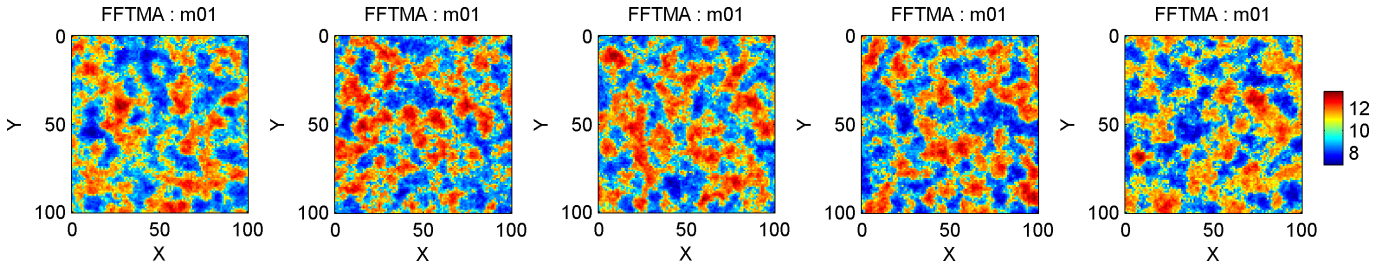
```
im=1;
prior{im}.type='FFTMA';
prior{im}.x=[0:1:100];
prior{im}.y=[0:1:100];
prior{im}.m0=10;
prior{im}.Cm='1 Sph(10)';

% 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;
```

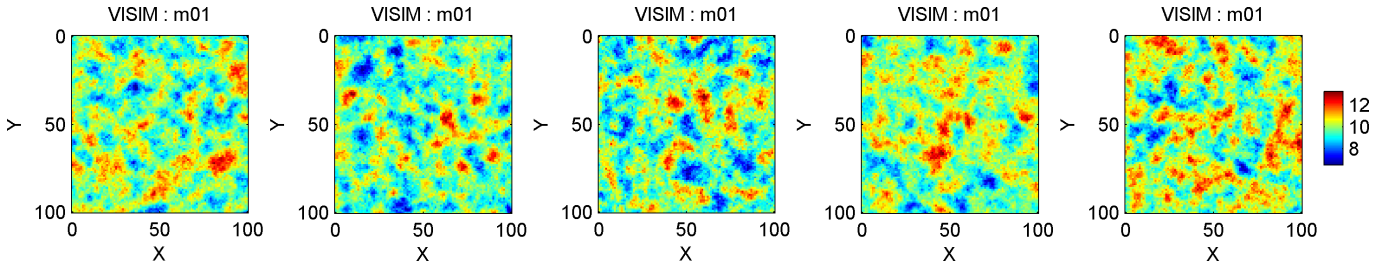
Alternatively, the normal score transformation can be defined manually such that the tail behaviour can be controlled:

```
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;
```



2.1.1.3 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)';
```



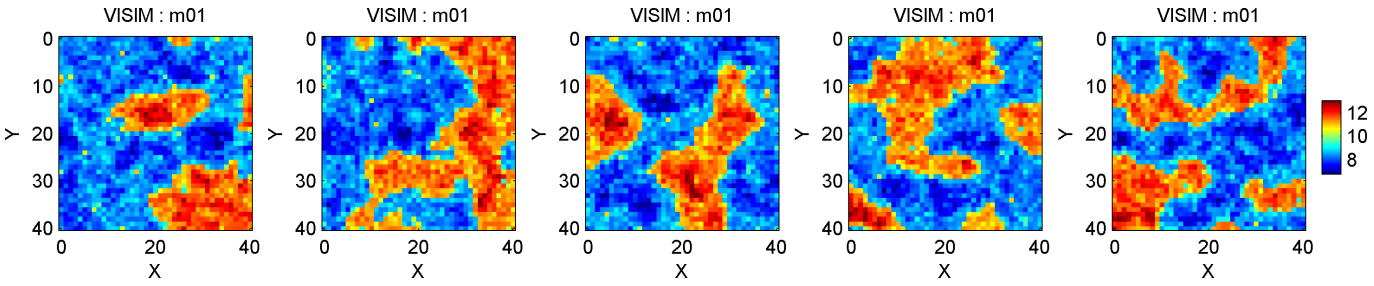
As with the FFTMA prior model the VISIM prior can make use of a target distribution. However, if a target distribution is set, the use of the VISIM prior model will utilize direct sequential simulation, which will ensure both histogram and covariance reproduction.

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;
```



2.1.1.4 SNESIM

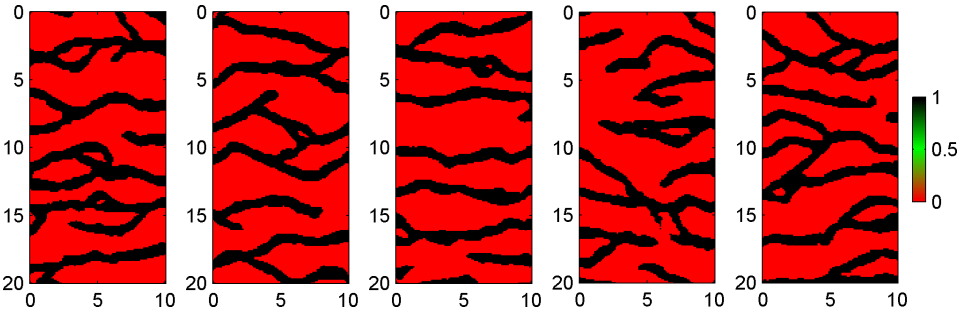
The 'SNESIM' type prior model utilizes the SNESIM algorithm, as implemented in SGeMS. As opposed to the Gaussian prior models defined above, the SNESIM prior model infer spatial statistics from a training image, which should be a 2D/3D stationary image.

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

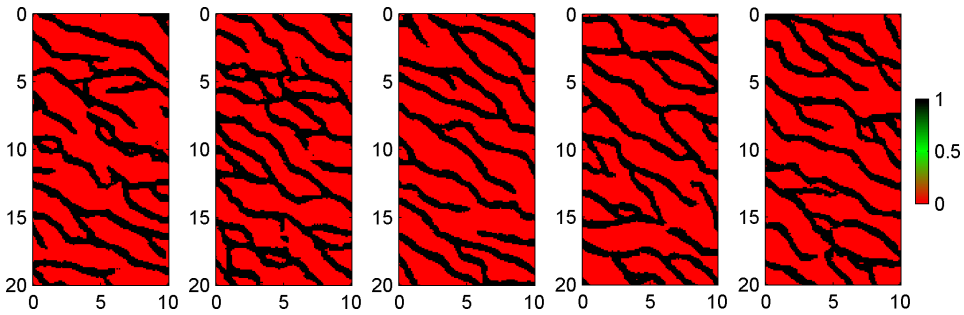
```
for i=1:5;
    m=sippi_prior(prior);
    subplot(1,5,i);
    imagesc(prior{1}.x,prior{1}.y,m{1});axis image
end
```



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 seperated 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;
```



2.1.1.4.1 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', ...).

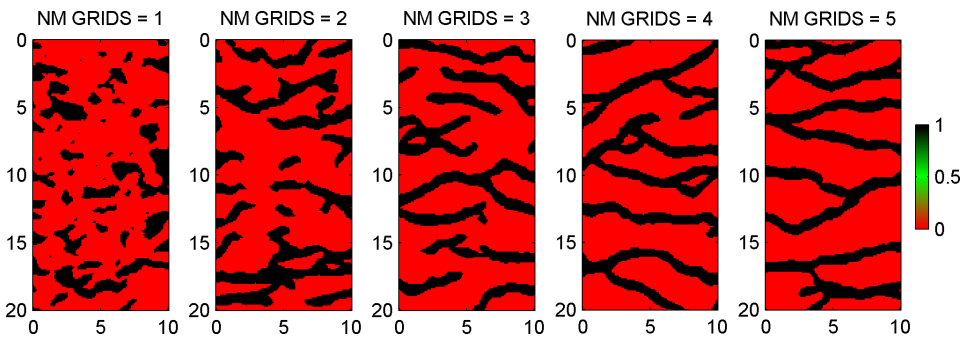
2.1.1.4.2 Complete customization

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
```



2.1.2 Sampling the prior

Once the prior data structure has been defined/modified, a sample from the prior distribution can be generated using

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

'm' is a Matlab data structure of the same size as the 'prior' data structure. Thus, if two prior distributions have been defined in 'prior{1}' and 'prior{2}', then 'm{1}' will hold a realization of 'prior{1}', and 'm{2}' will hold a realization of 'prior{2}'.

Each time 'm=sippi_prior(prior)' is called, a new independent realization of the prior will be generated.

2.1.3 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 prior, where the degree of perturbation can be controlled (from a new independent realization to a very small change). To do this, we make use of sequential Gibbs sampling [HCM12]. 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=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.

2.1.3.1 Controlling sequential Gibbs sampling / Conditional Resampling

All properties related to sequential Gibbs sampling can be set in the 'seq_gibbs' structure, for the individual prior models. The following two parameters determine how the a current model is perturbed

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

2.2 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).

2.2.1 Gaussian measurement noise

2.2.1.1 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;
```

2.2.1.2 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 the number of data in `data{1}.d_obs`.

2.2.2 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.

See the tomography example, for an example of accounting for correlated modeling errors.

2.3 `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 dependant. 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.

2.4 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);
```

Chapter 3

The a posteriori distribution

3.1 Sampling the a posteriori probability density

Once the `prior`, `data`, and `forward` data structures have been defined, the associated a posteriori probability can be sampled using the rejection sampler and the extended Metropolis sampler.

3.1.1 The rejection sampler

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

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 independant realizations of the prior, that does not require one to set L_max is the independant extended metropolis sampler, which may be computatinoally superior to the rejection sampler,

3.1.2 The extended Metropolis sampler

The extended Metropolis algorithm is in general a mcuh more efficient algroirthm 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 foldername, 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 will result in an algorthm 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.

3.1.2.1 Controling 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 Metropolios 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_min=1;
prior{m}.seq_gibbs.i_update_step=50
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 automatiuc adjustment of the step length simply set

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

3.1.2.2 The independent extended Metropolis sampler

The 'independent' extended Metropolis sampler, in which each proposed model is independant of the previsouly 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 indepedant prior sampling
for ip=1:length(prior);
  prior{ip}.seq_gibbs.step=1;
  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)
```

3.1.2.3 Annealing schedule

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

Annealing consist of multiplying the the noise level using an exponentially decerasing noise factor from `options.mcmc.anneal.fac_begin` to `options.mcmc.anneal.fac_end`, from iteration number `options.mcmc.anneal.i_begin` to `options.mcmc.anneal.i_end`.

The annealing schedule can be used start a Metropolis sampler that allow to explore more of the model space in the beginning. 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.fac_end=1`. This can potentially help not to get trapped in a local minima.

To use this type of annealing, where the annealing stops after 10000 iterations, after which the algorothm 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.fac_begin=20; % default, noise is scaled by fac_begin at iteration  ↩
    i_begin
options.mcmc.anneal.fac_end=1; % default, noise is scaled by fac_end at iteration i_end
```

3.2 Simulated Annealing

Simulated annealing type optimization can be setup using an annealing schedule that is enable to the entire run og the Metropolis sampler, and that ends by a noise scaling factor less than 1. This can be obtained using e.g.

```
options.mcmc.anneal.i_begin=1; % default, iteration number when annealing begins
options.mcmc.anneal.i_end=options.mcmc.nite; %  iteration number when annealing stops
options.mcmc.anneal.fac_begin=20; % default, noise is scaled by fac_begin at iteration  ↩
    i_begin
options.mcmc.anneal.fac_end=0.01; % 1/100 of the noise level
```

Chapter 4

Examples

SIPPI can be used as a convenient approach for unconditional an conditional simulation.

In order to use SIPPI to solve inverse problems, one must provide the solution to the forward problem. Essentially this amounts to implementing a Matlab function that solves the forward problem in using a specific input/output format. If a solution to the forward problem already exist, this can be quite easily done simply using a Matlab wrapper function.

A few implementations of solutions to forward problems are included as examples as part of SIPPI. These will be demonstrated in the following

4.1 Examples of A priori models

4.1.1 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';
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);
```

4.1.2 Multivariate Gaussian prior with unknown covariance model properties.

The FFT-MA type a priori model allow seperation of properties of the covariance model (covariance parameters, such as range, and anisotropy ratio) and the random compoent 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 patemeter, nu
        %   (only applies when the Mater type Covariance model is used)
```

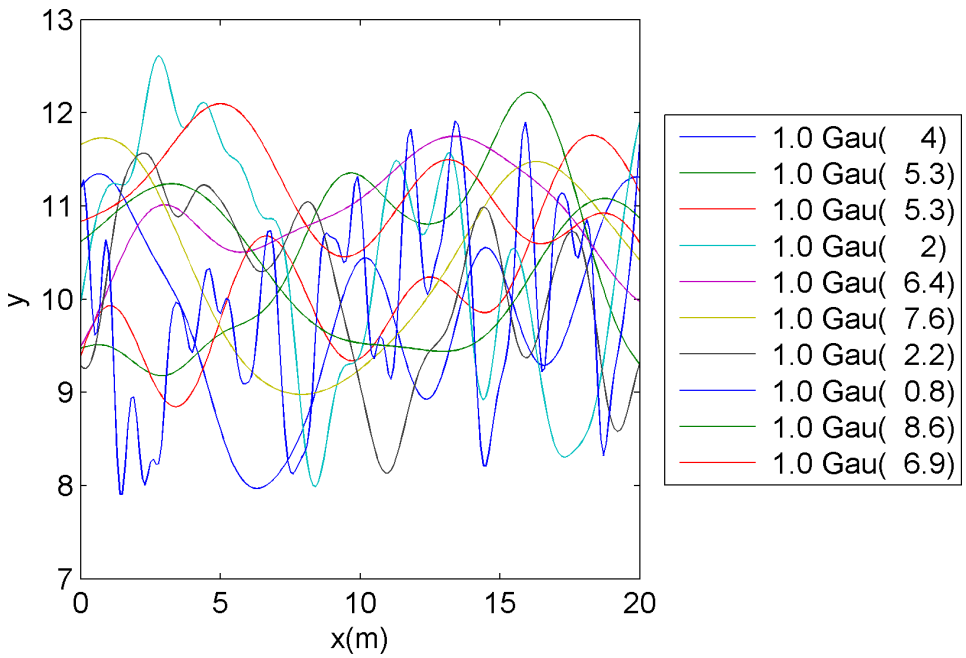
A very simple example of a prior model defining a 1D Sperical type covariance model with a range between 5 and 15 meters, can be defined using:

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

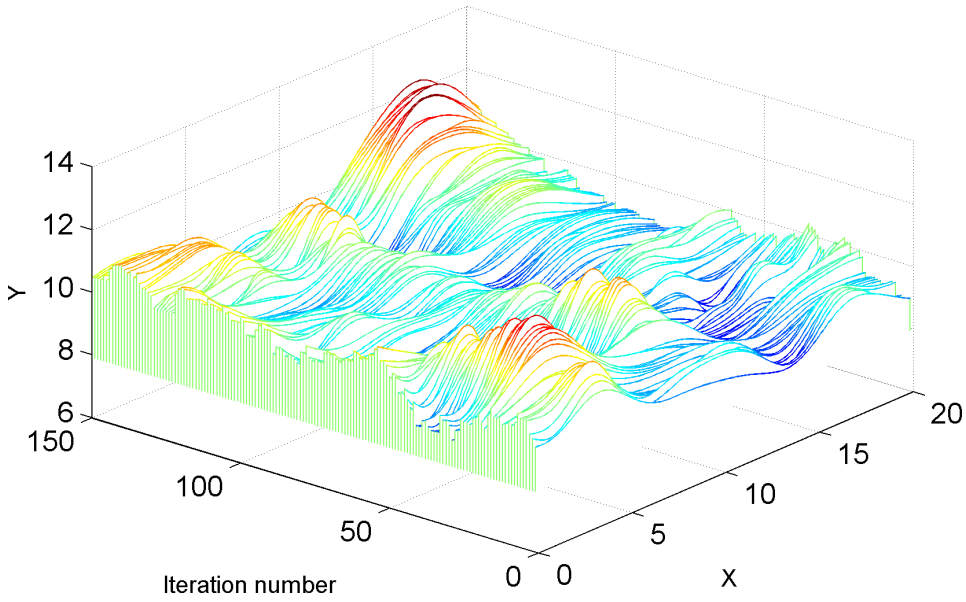
10 independent realizations of this type of a priori model are shown in the following figure



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



4.2 Polynomial line fitting

Here follows simple polynomial (of order 0, 1 or 2) line-fitting is considered. Example m-files 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.

4.2.1 The forward problem

The forward problem consists of computing the y-value as a function of the x-position of the data, and the polynomial coefficients determining the line. sippi_forward_linefit.m:

```
% 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*m{1};
elseif length(m)==2;
    d{1}=forward.x*m{1}+m{2};
else
    d{1}=forward.x.^2*m{1}+forward.x*m{2}+m{3};
end
```

the `forward.x` must be an array of the x-locations, for which the y-values of the corresponding line will be evaluated.

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.

4.2.2 Reference data, data, forward

A reference data set can be computed using

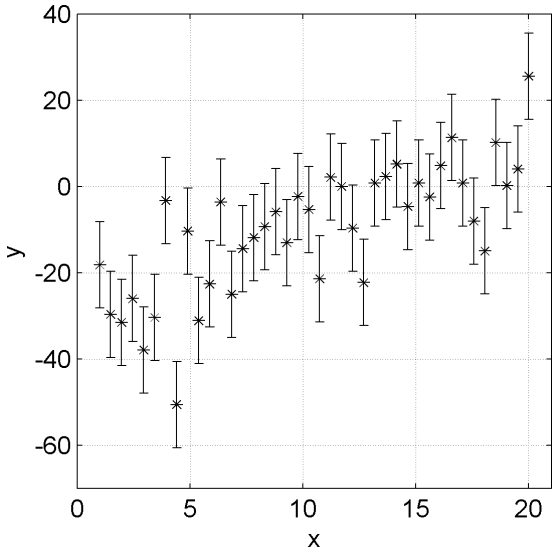
```
clear all;close all;
rand('seed',1);randn('seed',1);

%% Select reference model
m_ref{1}=-30;
m_ref{2}=2;
m_ref{3}=0;

%% Setup the forward model in the 'forward' structure
nd=40;
forward.x=linspace(1,20,nd);
forward.forward_function='sippi_forward_linefit';

%% Compute a reference set of observed data
d=sippi_forward(m_ref,forward);
d_obs=d{1};
d_std=10;
d_obs=d_obs+randn(size(d_obs)).*d_std;

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



4.2.3 The prior model

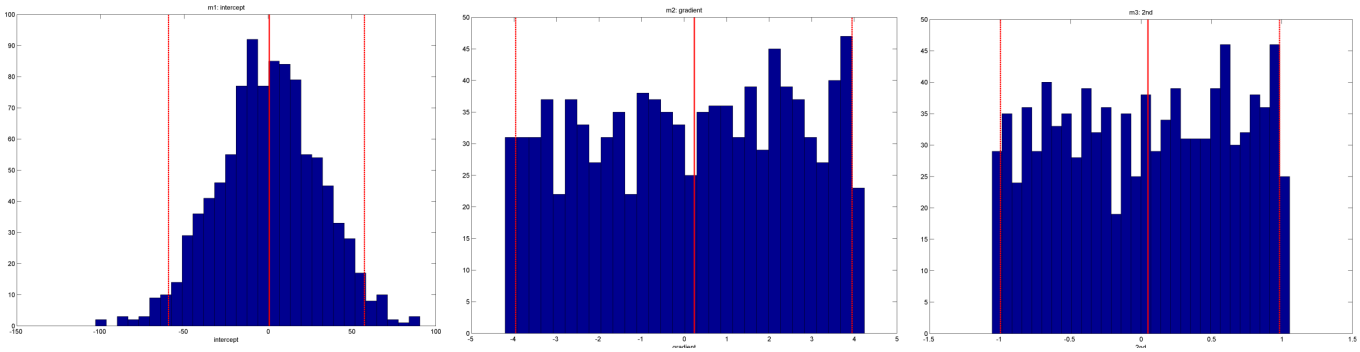
```
%% Setting up the prior model

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

% 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;
prior{im}.m_true=m_ref{2};

% 2nd order
im=3;
prior{im}.type='gaussian';
prior{im}.name='2nd';
prior{im}.m0=0;
prior{im}.std=1;
prior{im}.norm=80;
prior{im}.m_true=m_ref{3};

sippi_plot_prior_sample(prior);
```



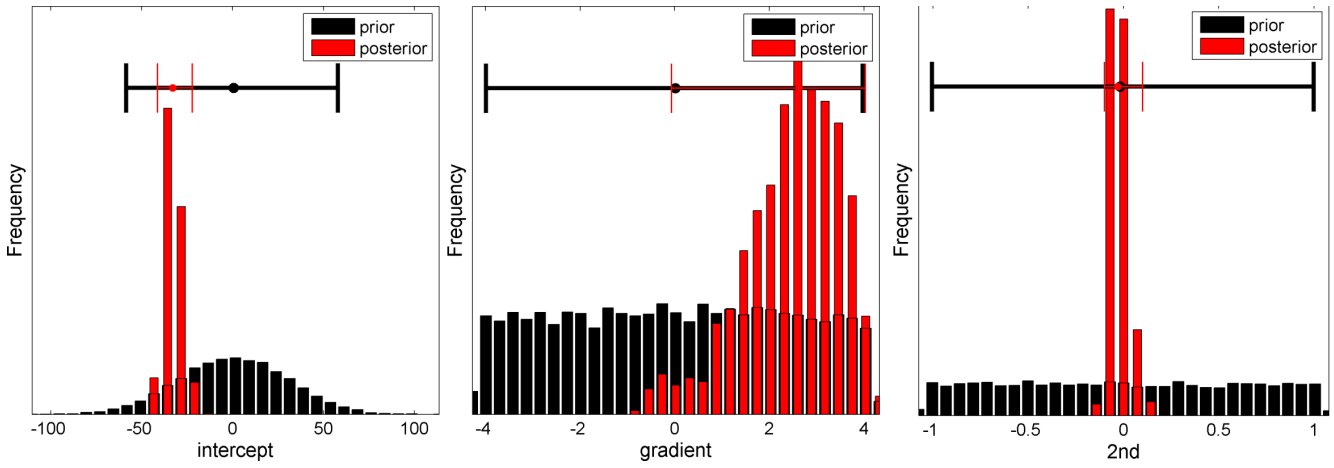
4.2.4 Setup and run the Metropolis sampler

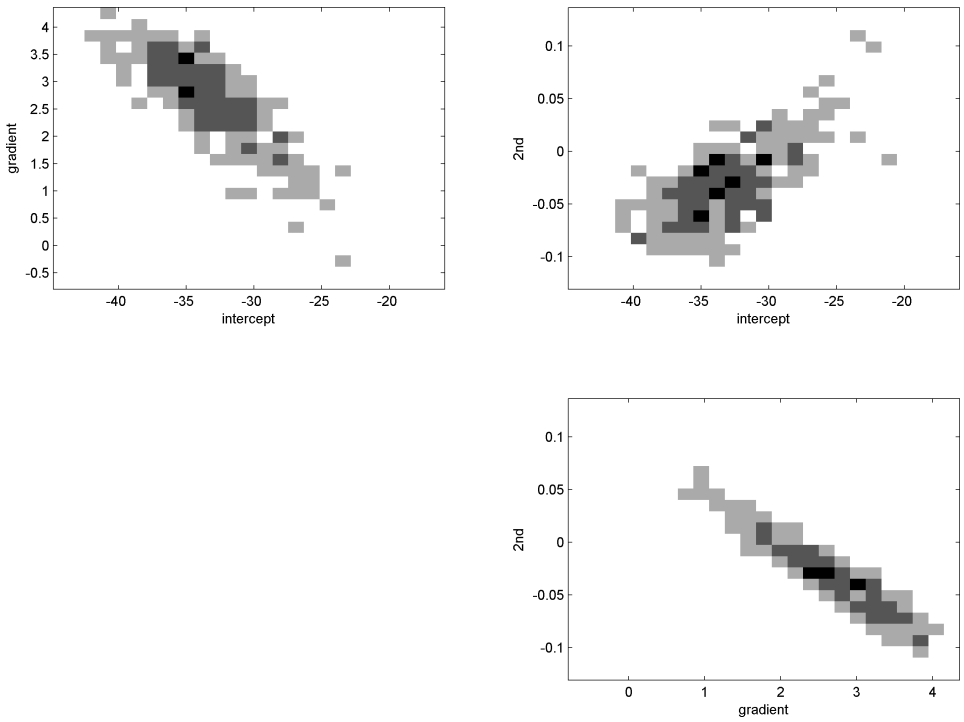
Now, information about the model parameters can be inferred by running the extended Metropolis sampler using

```
options.mcmc.nite=40000;  % Run for 40000 iterations
options.mcmc.i_sample=50; % Save every 50th visited model to disc
options.mcmc.i_plot=2500; % 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);

% 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
```





4.2.5 Setup and run the rejection sampler

In a similar manner the rejection sampler 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);
```

4.3 Cross hole tomography

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

In addition, a reference cross hole GPR data set is also avaialble, and will be used here to demonstrate the use of SIPPI to solve cross hole tomographic inversion in a probabilistic framework.

Please see [HCLM13b] for more details on the example of using SIPPI to sample the posterior for cross hole tomograohic inverse problems. See [LHC10] for more details on the data from Arrenæs.

4.3.1 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/cross-hole' folder.

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

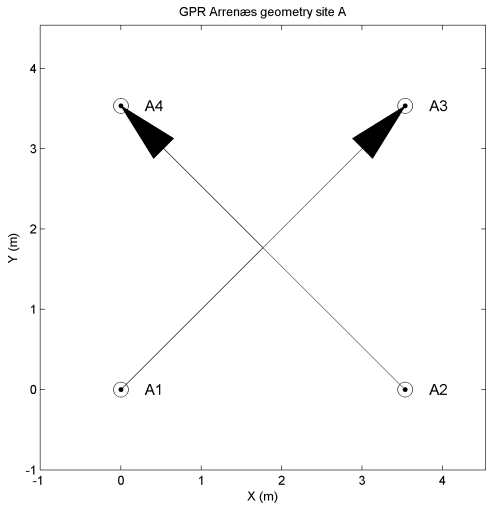


Figure 4.1: Location of boreholes AM1, AM2, AM3, and AM4 at Arrenæs.

Traveltime data were collected between boreholes AM1 and AM3, and AM2 and AM4 respecitvely, in a depth interval betwee 1m and 12m. The traveltimes 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 traveltime in milliseconds
d_std --> [ndata,1] each row contains the standard deaviation of the uncertainty of the ↩
    observed traveltime 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.

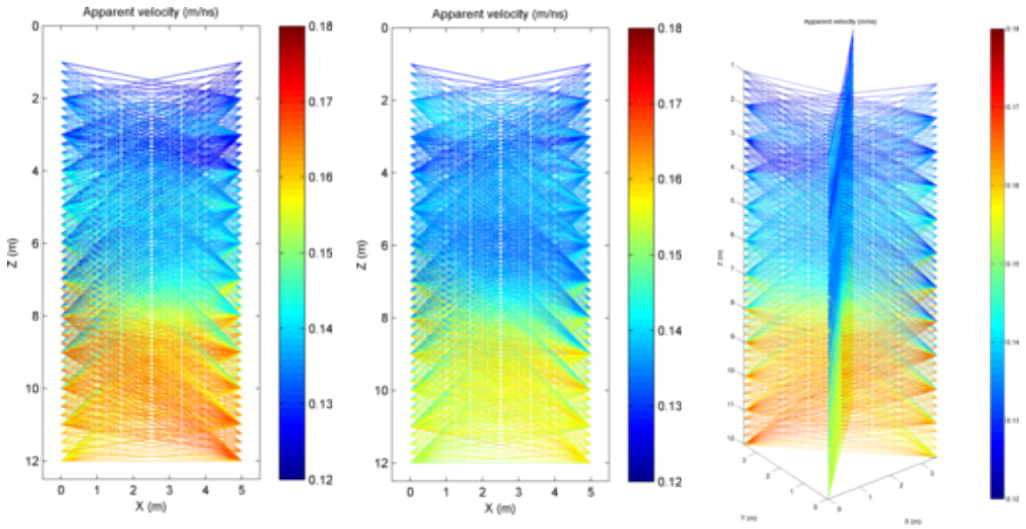


Figure 4.2: Ray coverage beetween wells left) AM1-AM3, middle) AM2-AM4, right) AM1-4.

4.3.2 Travel 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 descrie the forward problem sepcify 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';
```

4.3.2.1 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 travelling 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}'

4.3.2.2 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

4.3.2.3 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.

4.3.2.4 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 eficcient to solve the eikonal eqation directly, that to used the 'forward.type='ray';' type forward model.

To coose 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.

4.3.3 Inversion of cross hole GPR data from Arrenaes data

4.3.3.1 2D, Gaussian prior

Chapter 5

Bibliography

[CHM12] K. S. Cordua, T. M. Hansen, and K. Mosegaard, Monte Carlo full waveform inversion of crosshole GPR data using multiple-point geostatistical a priori information, H19--H31.

Geophysics, 77, 2012.

[HCLM13a] T.M. Hansen, K.S. Cordua, M.C. Looms, and K. Mosegaard, SIPPI: a Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 1, methodology, 470--480.

Computers & Geosciences, 52, 03 2013.

[HCLM13b] T.M. Hansen, K.S. Cordua, M.C. Looms, and K. Mosegaard, SIPPI: a Matlab toolbox for sampling the solution to inverse problems with complex prior information: Part 2, Application to cross hole GPR tomography, 481--492.

Computers & Geosciences, 52, 03 2013.

[HCM12] T. M. Hansen, K. C. Cordua, and K. Mosegaard, Inverse problems with non-trivial priors - efficient solution through sequential Gibbs sampling, 593--611.

Computational Geosciences, 16, 2012.

[HCM14] T. M. Hansen, K. S. Cordua, B. H. Jacobsen, and K. Mosegaard, Accounting for imperfect forward modeling in geophysical inverse problems - exemplified for cross hole tomography, xx.

Accepted for publication in Geophysics, xx, 2014.

[LHC10] M. C. Looms, T. M. Hansen, K. S. Cordua, L. Nielsen, K. H. Jensen, and A. Binley, Geostatistical inference using crosshole ground-penetrating radar : Geostatistical inference using GPR, J29--J41.

Geophysics, 75, 2010.

Chapter 6

Reference

6.1 SIPPI

6.1.1 getinunits

```
GETINUNITS   Get object properties in specified units
   V = GETINUNITS(H, PROP, UNITS) returns the object property
   in the specified UNITS. It will leave the 'Units' and 'FontUnits'
   property unchanged afterwards.

   H is the handle of the object. If it is an M-element array of handles,
   the function will return an M-by-1 cell array. PROP can be a string or
   a cell array of strings. If it is a 1-by-N or N-by-1 cell array, the
   function will return an M-by-N cell array of values. UNITS can be a
   string or a cell array. If it is a cell array, then PROP must also be a
   cell array with the same size as UNITS, and each cell element of UNITS
   corresponds to a cell element of PROP.

   V = GETINUNITS(H, PROP) is the same as GET(H, PROP)

   Examples:
     V = GETINUNITS(H, 'Position', 'Pixels')
     V = GETINUNITS(H, {'FontSize', 'Position'}, 'Normalized')
     V = GETINUNITS(H, {'FontSize', 'Position'}, {'Points', 'Pixels'})

   See also GET, SET
```

6.1.2 logdet

```
LOGDET Computation of logarithm of determinant of a matrix

   v = logdet(A);
       computes the logarithm of determinant of A.

       Here, A should be a square matrix of double or single class.
       If A is singular, it will returns -inf.

       Theoretically, this function should be functionally
       equivalent to log(det(A)). However, it avoids the
       overflow/underflow problems that are likely to
```

```
        happen when applying det to large matrices.

        The key idea is based on the mathematical fact that
        the determinant of a triangular matrix equals the
        product of its diagonal elements. Hence, the matrix's
        log-determinant is equal to the sum of their logarithm
        values. By keeping all computations in log-scale, the
        problem of underflow/overflow caused by product of
        many numbers can be effectively circumvented.

        The implementation is based on LU factorization.

    v = logdet(A, 'chol');
        If A is positive definite, you can tell the function
        to use Cholesky factorization to accomplish the task
        using this syntax, which is substantially more efficient
        for positive definite matrix.

    Remarks
    -------
        logarithm of determinant of a matrix widely occurs in the
        context of multivariate statistics. The log-pdf, entropy,
        and divergence of Gaussian distribution typically comprises
        a term in form of log-determinant. This function might be
        useful there, especially in a high-dimensional space.

        Theoretially, LU, QR can both do the job. However, LU
        factorization is substantially faster. So, for generic
        matrix, LU factorization is adopted.

        For positive definite matrices, such as covariance matrices,
        Cholesky factorization is typically more efficient. And it
        is STRONGLY RECOMMENDED that you use the chol (2nd syntax above)
        when you are sure that you are dealing with a positive definite
        matrix.

    Examples
    --------
        % compute the log-determinant of a generic matrix
        A = rand(1000);
        v = logdet(A);

        % compute the log-determinant of a positive-definite matrix
        A = rand(1000);
        C = A * A';     % this makes C positive definite
        v = logdet(C, 'chol');
```

6.1.3 pathdef

```
PATHDEF Search path defaults.
   PATHDEF returns a string that can be used as input to MATLABPATH
   in order to set the path.
```

6.1.4 plotboxpos

```
PLOTBOXPOS Returns the position of the plotted axis region

 pos = plotboxpos(h)

 This function returns the position of the plotted region of an axis,
 which may differ from the actual axis position, depending on the axis
 limits, data aspect ratio, and plot box aspect ratio.  The position is
 returned in the same units as the those used to define the axis itself.
 This function can only be used for a 2D plot.

 Input variables:

   h:      axis handle of a 2D axis (if ommitted, current axis is used).

 Output variables:

   pos:    four-element position vector, in same units as h
```

6.1.5 sippi_adjust_step_size

```
 sippi_adjust_step_size Adjust step length length for Metropolis sampler in SIPPI

 Call :
   step=sippi_adjust_step_size(step,P_average,P_target);

 step : current step
 P_current : Current acceptance ratio
 P_target  : preferred acceptance ratio (def=0.3);

 See also sippi_compute_acceptance_rate, sippi_prior_set_steplength
```

6.1.6 sippi_anneal_adjust_noise

```
 sippi_anneal_adjust_noise : Adjust noise level in annealing schedul

 Call:
    [data_adjust,mcmc]=sippi_anneal_adjust_noise(data,i,mcmc,prior);

 See also: sippi_metropolis, sippi_anneal_factor
```

6.1.7 sippi_anneal_factor

```
 sippi_anneal_factor : compute simple noise multiplication factor for
 annealing type sampling

 See also sippi_metropolis, sippi_anneal_adjust_noise
```

6.1.8 sippi_colormap

```
sippi_colormap Default colormap for sippi

Call :
  sippi_colormap; % the same as sippi_colormap(3);

or :
  sippi_colormap(1) - Red Green Black
  sippi_colormap(2) - Red Green Blue Black
  sippi_colormap(3) - Jet
```

6.1.9 sippi_compute_acceptance_rate

```
sippi_compute_acceptance_rate Computes acceptance rate for the Metropolis sampler in ↩
    SIPPI

Call:
  P_acc=sippi_compute_acceptance_rate(acc,n_update_history);
```

6.1.10 sippi_forward

```
sippi_forward Simple forward wrapper for SIPPI

Assumes that the actual forward solver has been defined by
forward.forward_function

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

6.1.11 sippi_forward_traveltime

```
sippi_forward_traveltime Traveltime computation in SIPPI

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

  forward.type determines the method used to compute travel times
  forward.type='ray';
  forward.type='fat';
  forward.type='eikonal';
  forward.type='born';
```

6.1.12 sippi_get_sample

```
sippi_get_sample Get a posterior sample

Call :
 [reals,etype_mean,etype_var]=sippi_get_sample(data,prior,id,im,n_reals,options);
```

6.1.13 sippi_least_squares

```
sippi_least_squares Least squares type inversion for SIPPI

Call :
   [m_reals,m_est,Cm_est]=sippi_least_squares(data,prior,forward,n_reals,lsq_type,id,im);



  lsq_type : 'lsq' (def), classical least squares
             'error_sim', simulation through error simulation
             'visim', simulation through SGSIM of DSSIM
```

6.1.14 sippi_likelihood

```
sippi_likelihood Compute likelihood given an observed dataset

Call
  [logL,L,data]=sippi_likelihood(d,data);


 data{1}.d_obs [N_data,1] N_data data observations
 data{1}.d_std [N_data,1] N_data uncorrelated Gaussian STD

 data{1}.d_var [N_data,1] N_data uncorrelated Gaussian variances


Gaussian modelization error, N(dt,Ct), is specified as
 data{1}.dt [N_data,1] : Bias/mean of modelization error
 data{1}.Ct [N_data,N_data] : Covariance of modelization error

 data{1}.Ct [1,1] : Constant Covariance of modelization error
                    imples data{1}.Ct=ones(N_data.N_data)*data{1}.Ct;



data{id}.recomputeCD [default=0], if '1' then data{1}.iCD is recomputed
each time sippi_likelihood is called. This should be used if the noise model
changes between each call to sippi_likelihood.

 data{id}.full_likelihood [default=]0; if '1' the the full likelihood
 (including the determinant) is computed. This not needed if the data
 civariance is constant, but if it changes, then use
 data{id}.full_likelihood=1;
```

6.1.15 sippi_mcmc_init

```
sippi_mcmc_init Initialize McMC options for Metropolis and rejection sampling in SIPPI

Call:
   options=sippi_mcmc_init(options,prior);
```

6.1.16 sippi_metropolis

```
sippi_metropolis Extended Metropolis sampling in SIPPI

Metropolis sampling.
  See e.g. Hansen, T. M., Cordua, K. S., and Mosegaard, K., 2012.
    Inverse problems with non-trivial priors - Efficient solution through Sequential ↩
        Gibbs Sampling.
    Computational Geosciences. doi:10.1007/s10596-011-9271-1.

Call :
   [options,data,prior,forward,m_current]=sippi_metropolis(data,prior,forward,options)
Input :
   data : sippi data structure
   prior : sippi prior structure
   forward : sippi forward structure

options :
   options.txt [string] : string to be used as part of all output files

   options.mcmc.nite [1]  : Number if iterations
   options.mcmc.i_plot [1]: Number of iterations between updating plots
   options.mcmc.i_sample=: Number of iterations between saving model to disk

   options.mcmc.m_init : Manually chosen starting model
   options.mcmc.m_ref  : Reference known target model

   options_mcmc.accept_only_improvements [0] : Optimization

  %% PERTUBATION STRATEGY
  options.mcmc.pert_strategy.perturb_all=1; % Perturb all priors in each
                                            % iteration. def =[0]
   %% SIMULATED ANNEALING
   options.mcmc.anneal.i_begin=1; % default, iteration number when annealing begins
   options.mcmc.anneal.i_end=100000; %  iteration number when annealing stops
   options.mcmc.anneal.fac_begin=20; % default, noise is scaled by fac_begin at iteration ↩
        i_begin
   options.mcmc.anneal.fac_end=1; % default, noise is scaled by fac_end at iteration  ↩
       i_end


See also sippi_rejection
```

6.1.17 sippi_plot_current_model

```
sippi_plot_current_model Plots the current model during Metropolis sampling

Call :
```

```
   sippi_plot_current_model(mcmc,data,d,m_current,prior);
```

6.1.18 sippi_plot_data

```
sippi_plot_data plot data in SIPPI

Call.
   sippi_plot_data(d,data);
```

6.1.19 sippi_plot_loglikelihood

```
sippi_plot_loglikelihood Plot loglikelihood time series

Call :
   acc=sippi_plot_loglikelihood(logL,i_acc,N,itext)
```

6.1.20 sippi_plot_model

```
sippi_plot_model Plot a 'model', i.e. a realization of the prior model


Call :
  sippi_plot_model(prior,m,im_array);

  prior : Matlab structure for SIPPI prior model
  m : Matlab structure for SIPPI realization
  im_array : integer array of type of models to plot (typically 1)


Example
  m=sippi_prior(prior);
  sippi_plot_model(prior,m);

  m=sippi_prior(prior);
  sippi_plot_model(prior,m,2);

See also sippi_plot_prior
```

6.1.21 sippi_plot_movie

```
sippi_plot_movie plot movie of prior and posterior realizations

Call :
  sippi_plot_movie(fname);
  sippi_plot_movie(fname,im_array,n_frames,skip_burnin);
     fname : name of folder with results (e.g. options.txt)
     im_array : array of indexes of model parameters to make into movies
     n_frames [200] : number of frames in movie
     skip_burnin [200] : start movie after burn_in;
```

```
Ex:
sippi_plot_movie('20130812_Metropolis');
sippi_plot_movie(options.txt);

%% 1000 realization including burn-in, for prior number 1
sippi_plot_movie('20130812_Metropolis',1,1000,0);
```

6.1.22 sippi_plot_posterior

```
sippi_plot_posterior Plot statistics from posterior sample

Call :
   sippi_plot_posterior(fname,im_arr,prior,options,n_reals);

See also sippi_plot_prior
```

6.1.23 sippi_plot_prior

```
sippi_plot_prior Plot a sample of the prior in SIPPI

Call :
   sippi_plot_prior(prior,ip,n_reals,cax,supt);

 See also sippi_plot_posterior, sippi_plot_model
```

6.1.24 sippi_prior

```
sippi_prior A priori models for SIPPI

To generate a realization of the prior model defined by the prior structure use:
  [m_propose,prior]=sippi_prior(prior);

To generate a realization of the prior model defined by the prior structure,
in the vicinity of a current model (using sequential Gibbs sampling) use:
  [m_propose,prior]=sippi_prior(prior,m_current);

The following types of a priori models can be used
  SNESIM  [1D-3D] : based on a multiple point statistical model inferref from a training ↩
      images. Relies in the SNESIM algorithm
  SISIM   [1D-3D] : based on Sequential indicator SIMULATION
  VISIM   [1D-3D] : based on Sequential Gaussian and Direct Sequential simulation
  FFTMA   [1D-3D] : based on the FFT-MA method (Multivariate Gaussian)
  GAUSSIAN   [1D] : 1D generalized gaussian model


%%% SIMPLE EXAMPLE %%%

% A simple 2D multivariate Gaissian based prior model based on the
% FFT-MA method, can be defined using
   im=1;
   prior{im}.type='FFTMA';
```

```
   prior{im}.name='A SIMPLE PRIOR';
   prior{im}.x=[0:1:100];
   prior{im}.y=[0:1:100];
   prior{im}.m0=10;
   prior{im}.Va='1 Sph(10)';
   prior=sippi_prior_init(prior);
% A realization from this prior model can be generated using
   m=sippi_prior(prior);
% This realization can now be plotted using
   sippi_plot_prior(m,prior);
% or
   imagesc(prior{1}.x,prior{1}.y,m{1})

%%% A PRIOR MODEL WITH SEVERAL 'TYPES OF A PRIORI MODEL'

   im=1;
   prior{im}.type='GAUSSIAN';
   prior{im}.m0=100;
   prior{im}.std=50;
   prior{im}.norm=100;
   im=2;
   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)';
   im=3;
   prior{im}.type='SISIM';
   prior{im}.x=[0:1:100];
   prior{im}.y=[0:1:100];
   prior{im}.m0=10;
   prior{im}.Cm='1 Sph(10)';
   im=4;
   prior{im}.type='SNESIM';
   prior{im}.x=[0:1:100];
   prior{im}.y=[0:1:100];

   sippi_plot_model(prior);

%% Sequential Gibbs sampling

   All a priori model types can be perturbed, such that a new realization
   is generated in the vicinity of a current model.
   To do this Sequential Gibbs Sampling is used.
   For more information, see <a href="matlab:web('http://dx.doi.org/10.1007/s10596 ↩
       -011-9271-1')">Hansen, T. M., Cordua, K. S., and Mosegaard, K., 2012. Inverse  ↩
       problems with non-trivial priors - Efficient solution through Sequential Gibbs  ↩
       Sampling. Computational Geosciences</a>.
   The type of sequential Gibbs sampling can be controlled in the
   'seq_gibbs' structures, e.g. prior{1}.seq_gibbs

   im=1;
   prior{im}.type='SNESIM';
   prior{im}.x=[0:1:100];
   prior{im}.y=[0:1:100];

   [m,prior]=sippi_prior(prior);
   prior{1}.seq_gibbs.step=1; % Large step--> independant realizations
   prior{1}.seq_gibbs.step=.1; % Smaller step--> Dependant realizations
   for i=1:30;
      [m,prior]=sippi_prior(prior,m); % One iteration of Sequential Gibbs
      sippi_plot_model(prior,m);
```

```
    end

  See also: sippi_prior_init, sippi_plot_prior, sippi_prior_set_steplength.m

  TMH/2012
```

6.1.25 sippi_prior_fftma

```
  sippi_prior A priori models for SIPPI

  To generate a realization of the prior model defined by the prior structure use:
    [m_propose,prior]=sippi_prior(prior);

  To generate a realization of the prior model defined by the prior structure,
  in the vicinity of a current model (using sequential Gibbs sampling) use:
    [m_propose,prior]=sippi_prior(prior,m_current);

  The following types of a priori models can be used
    SNESIM  [1D-3D] : based on a multiple point statistical model inferref from a training ↩
        images. Relies in the SNESIM algorithm
    SISIM   [1D-3D] : based on Sequential indicator SIMULATION
    VISIM   [1D-3D] : based on Sequential Gaussian and Direct Sequential simulation
    FFTMA   [1D-3D] : based on the FFT-MA method (Multivariate Gaussian)
    GAUSSIAN   [1D] : 1D generalized gaussian model


%%% SIMPLE EXAMPLE %%%

% A simple 2D multivariate Gaissian based prior model based on the
% FFT-MA method, can be defined using
    id=1;
    prior{id}.type='FFTMA';
    prior{id}.name='A SIMPLE PRIOR';
    prior{id}.x=[0:1:100];
    prior{id}.y=[0:1:100];
    prior{id}.m0=10;
    prior{id}.Va='1 Sph(10)';
    prior=sippi_prior_init(prior);
% A realization from this prior model can be generated using
    m=sippi_prior(prior);
% This realization can now be plotted using
    sippi_plot_prior(m,prior);
% or
    imagesc(prior{1}.x,prior{1}.y,m{1})

%%% A PRIOR MODEL WITH SEVERAL 'TYPES OF A PRIORI MODEL'

    id=1;
    prior{id}.type='FFTMA';
    prior{id}.x=[0:1:100];
    prior{id}.y=[0:1:100];
    prior{id}.m0=10;
    prior{id}.Cm='1 Sph(10)';
    id=2;
    prior{id}.type='SISIM';
    prior{id}.x=[0:1:100];
    prior{id}.y=[0:1:100];
    prior{id}.m0=10;
    prior{id}.Cm='1 Sph(10)';
```

```
   id=3;
   prior{id}.type='GAUSSIAN';
   prior{id}.m0=100;
   prior{id}.std=50;
   prior{id}.norm=100;
   prior=sippi_prior_init(prior);

   sippi_plot_model(prior);

%% Sequential Gibbs sampling
% For more information, see <a href="matlab:web('http://dx.doi.org/10.1007/s10596 ↩
    -011-9271-1')">Hansen, T. M., Cordua, K. S., and Mosegaard, K., 2012. Inverse problems ↩
    with non-trivial priors - Efficient solution through Sequential Gibbs Sampling. ↩
    Computational Geosciences</a>.


 See also: sippi_prior_init, sippi_plot_prior, sippi_prior_set_steplength.m

 TMH/2012
```

6.1.26 sippi_prior_init

```
 sippi_prior_init Initialize PRIOR structure for SIPPI

 Call
   prior=sippi_prior_init(prior);

 See also sippi_prior
```

6.1.27 sippi_prior_new

```
 sippi_prior A priori models for SIPPI

 To generate a realization of the prior model defined by the prior structure use:
   [m_propose,prior]=sippi_prior(prior);

 To generate a realization of the prior model defined by the prior structure,
 in the vicinity of a current model (using sequential Gibbs sampling) use:
   [m_propose,prior]=sippi_prior(prior,m_current);

 The following types of a priori models can be used
   SNESIM  [1D-3D] : based on a multiple point statistical model inferref from a training ↩
       images. Relies in the SNESIM algorithm
   SISIM   [1D-3D] : based on Sequential indicator SIMULATION
   VISIM   [1D-3D] : based on Sequential Gaussian and Direct Sequential simulation
   FFTMA   [1D-3D] : based on the FFT-MA method (Multivariate Gaussian)
   GAUSSIAN   [1D] : 1D generalized gaussian model


%%% SIMPLE EXAMPLE %%%

% A simple 2D multivariate Gaissian based prior model based on the
% FFT-MA method, can be defined using
   id=1;
   prior{id}.type='FFTMA';
```

```
   prior{id}.name='A SIMPLE PRIOR';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Va='1 Sph(10)';
   prior=sippi_prior_init(prior);
% A realization from this prior model can be generated using
   m=sippi_prior(prior);
% This realization can now be plotted using
   sippi_plot_prior(m,prior);
% or
   imagesc(prior{1}.x,prior{1}.y,m{1})

%%% A PRIOR MODEL WITH SEVERAL 'TYPES OF A PRIORI MODEL'

   id=1;
   prior{id}.type='FFTMA';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Cm='1 Sph(10)';
   id=2;
   prior{id}.type='SISIM';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Cm='1 Sph(10)';
   id=3;
   prior{id}.type='GAUSSIAN';
   prior{id}.m0=100;
   prior{id}.std=50;
   prior{id}.norm=100;
   prior=sippi_prior_init(prior);

   sippi_plot_model(prior);

%% Sequential Gibbs sampling
% For more information, see <a href="matlab:web('http://dx.doi.org/10.1007/s10596 ↩
    -011-9271-1')">Hansen, T. M., Cordua, K. S., and Mosegaard, K., 2012. Inverse problems ↩
    with non-trivial priors - Efficient solution through Sequential Gibbs Sampling. ↩
    Computational Geosciences</a>.


 See also: sippi_prior_init, sippi_plot_prior, sippi_prior_set_steplength.m

 TMH/2012
```

6.1.28 sippi_prior_old

```
 sippi_prior A priori models for SIPPI

 To generate a realization of the prior model defined by the prior structure use:
   [m_propose,prior]=sippi_prior(prior);

 To generate a realization of the prior model defined by the prior structure,
 in the vicinity of a current model (using sequential Gibbs sampling) use:
   [m_propose,prior]=sippi_prior(prior,m_current);

 The following types of a priori models can be used
```

```
   SNESIM  [1D-3D] : based on a multiple point statistical model inferref from a training ↩
       images. Relies in the SNESIM algorithm
   SISIM   [1D-3D] : based on Sequential indicator SIMULATION
   VISIM   [1D-3D] : based on Sequential Gaussian and Direct Sequential simulation
   FFTMA   [1D-3D] : based on the FFT-MA method (Multivariate Gaussian)
   GAUSSIAN   [1D] : 1D generalized gaussian model


%%% SIMPLE EXAMPLE %%%

% A simple 2D multivariate Gaissian based prior model based on the
% FFT-MA method, can be defined using
   id=1;
   prior{id}.type='FFTMA';
   prior{id}.name='A SIMPLE PRIOR';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Va='1 Sph(10)';
   prior=sippi_prior_init(prior);
% A realization from this prior model can be generated using
   m=sippi_prior(prior);
% This realization can now be plotted using
   sippi_plot_prior(m,prior);
% or
   imagesc(prior{1}.x,prior{1}.y,m{1})

%%% A PRIOR MODEL WITH SEVERAL 'TYPES OF A PRIORI MODEL'

   id=1;
   prior{id}.type='FFTMA';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Cm='1 Sph(10)';
   id=2;
   prior{id}.type='SISIM';
   prior{id}.x=[0:1:100];
   prior{id}.y=[0:1:100];
   prior{id}.m0=10;
   prior{id}.Cm='1 Sph(10)';
   id=3;
   prior{id}.type='GAUSSIAN';
   prior{id}.m0=100;
   prior{id}.std=50;
   prior{id}.norm=100;
   prior=sippi_prior_init(prior);

   sippi_plot_model(prior);

%% Sequential Gibbs sampling
% For more information, see <a href="matlab:web('http://dx.doi.org/10.1007/s10596 ↩
    -011-9271-1')">Hansen, T. M., Cordua, K. S., and Mosegaard, K., 2012. Inverse problems ↩
    with non-trivial priors - Efficient solution through Sequential Gibbs Sampling. ↩
    Computational Geosciences</a>.


 See also: sippi_prior_init, sippi_plot_prior, sippi_prior_set_steplength.m

 TMH/2012
```

6.1.29 sippi_prior_set_steplength

```
sippi_prior_set_steplength Set step length for Metropolis sampler in SIPPI

Call
  prior=sippi_prior_set_steplength(prior,mcmc,im);
```

6.1.30 sippi_rejection

```
sippi_rejection Rejection sampling

Call :
    options=sippi_rejection(data,prior,forward,options)

input arguments

  options.mcmc.i_plot
  options.mcmc.nite     % maximum number of iterations
  options.mcmc.logLmax

  options.mcmc.rejection_normalize_log = log(options.mcmc.Lmax)

  options.mcmc.adaptive_rejection=1, adaptive setting of maxiumum likelihood
                (def=[0])
                At each iteration Lmax will be set if log(L(m_cur)=>options.mcmc.logLmax


  options.mcmc.max_run_time_hours = 1; % maximum runtime in hours
                                       % (overrides options.mcmc.nite if needed)

See also sippi_metropolis
```

6.1.31 sippi_set_path

```
sippi_set_path Set paths for running sippi
```

6.2 SIPPI toolbox: Traveltime tomography

6.2.1 calc_Cd

```
Calc_cd Setup a covariance model to account for borehole imperfections

Call: Cd=calc_Cd(ant_pos,var_uncor,var_cor1,var_cor2,L)
This function sets up a data covariance matrix that accounts for static
(i.e. correlated) data errors.

Inputs:
* ant_pos: A N x 4 array that contains N combinations of transmitter/source
and receiver positions. The first two columns are the x- and y-coordinates
of the transmitter/source position. The last two columns are the x- and
```

```
  y-coordiantes of the receiver position.
  * var_uncor: The variance of the uncorrelated data errors.
  * var_cor1: The variance of the correlated data errors
  related to the transmitter/source positions.
  * var_cor2: The variance of the correlated data errors
  related to the receiver positions.
  * L: The correlation length for the correlation between the individual
  transmitter/source or receiver positions using an exponential covariance
  function. For typical static errors the correlation length is set to a
  small number (e.g. 10^-6).

  For more details and practical examples see:
  Cordua et al., 2008 in Vadose zone journal.
  Cordua et al., 2009 in Journal of applied geophysics.

  Knud S. Cordua (2012)
```

6.2.2 eikonal

```
  eikonal Traveltime computation by solving the eikonal equation

  tmap=eikonal(x,y,z,V,Sources,type);

   x,y,z : arrays defining the x, y, and z axis
   V: velocity field, with size (length(y),length(x),length(z));
   Sources [ndata,ndim] : Source positions
   type (optional): type of eikonal solver: [1]:Fast Marching(default), [2]:FD

   tmap [size(V)]: travel times computed everywhere in the velocity grid

 %Example (2D):
    x=[1:1:100];
    y=1:1:100;
    z=1;
    V=ones(100,100);V(:,1:50)=2;
    Sources = [10 50;75 50];
    t=eikonal(x,y,z,V,Sources);
    subplot(1,2,1);imagesc(x,y,t(:,:,1,1));axis image;colorbar
    subplot(1,2,2);imagesc(x,y,t(:,:,1,2));axis image;colorbar

  See also eikonal_traveltime
```

6.2.3 eikonal_raylength

```
  eikonal_raylength : Computes the raylength from S to R using the eikonal equaiton

  Call:
    raylength=eikonal_raylength(x,y,v,S,R,tS,doPlot)
```

6.2.4 eikonal_traveltime

```
 eikonal_traveltime Computes traveltime between sources and receivers by solving the ↩
     eikonal equation

 t=eikonal_traveltime(x,y,z,V,Sources,Receivers,iuse,type);

  x,y,z : arrays defining the x, y, and z axis
  V: velocity field, with size (length(y),length(x),length(z));
  Sources [ndata,ndim] : Source positions
  Receivers [ndata,ndim] : Receiver positions
  iuse (optional): optionally only use subset of data. eg.g i_use=[1 2 4];
  type (optional): type of eikonal solver: [1]:Fast Marching(default), [2]:FD

  tmap [size(V)]: travel times computed everywhere in the velocity grid

%Example (2%

 Example 2d traveltime compuation

 Example (2D):
   x=[1:1:100];
   y=1:1:100;
   z=1;
   V=ones(100,100);V(:,1:50)=2;
   S=[50 50 1;50 50 1];
   R=[90 90 1; 90 80 1];
   t=eikonal_traveltime(x,y,z,V,S,R)

 Example (3D):
   nx=50;ny=50;nz=50;
   x=1:1:nx;
   y=1:1:ny;
   z=1:1:nz;
   V=ones(ny,nx,nz);V(:,1:50,:)=2;
   S=[10 10 1;10 10 1;10 9 1];
   R=[40 40 40; 40 39 40; 40 40 40];
   t=eikonal_traveltime(x,y,z,V,S,R)


 See also eikonal
```

6.2.5 kernel_buursink_2d

```
 kernel_buursink_2k Computes 2D Sensitivity kernel based on 1st order EM scattering theory

 See
   Buursink et al. 2008. Crosshole radar velocity tomography
                         with finite-frequency Fresnel. Geophys J. Int.
                         (172) 117;

  CALL :
     % specify a source trace (dt, wf_trace):
     [kernel,L,L1_all,L2_all]=kernel_buursink_2d(model,x,z,S,R,dt,wf_trace);
     % Use a ricker wavelet with center frequency 'f0'
     [kernel,L,L1_all,L2_all]=kernel_buursink_2d(model,x,z,S,R,f0));


 Knud Cordua, 2009,
 Thomas Mejer Hansen (small edits, 2009)
```

6.2.6 kernel_finite_2d

```
kernel_finite_2d 2D sensitivity kernels

 Call:
   [Knorm,K,dt,options]=kernel_finite_2d(v_ref,x,y,S,R,freq,options);
```

6.2.7 kernel_fresnel_2d

```
kernel_fresnel_2d Sensitivity kernel for amplitude and first arrival

Call:
  [kernel_t,kernel_a,P_omega,omega]=kernel_fresnel_2d(v,x,y,S,R,omega,P_omega);


Based on Liu, Dong, Wang, Zhu and Ma, 2009, Sensitivity kernels for
seismic Fresenl volume Tomography, Geophysics, 75(5), U35-U46

See also kernel_fresnel_monochrome_2d

Run with no argument for an example.
```

6.2.8 kernel_fresnel_monochrome_2d

```
kernel_fresnel_monochrome_2d 2D monchrome kernel for amplitude and first arrival

Call:
  [kernel_t,kernel_a]=kernel_fresnel_monochrome_2d(v,x,y,S,R,omega);
or
  [kernel_t,kernel_a]=kernel_fresnel_monochrome_2d(v,x,y,S,R,omega,L,L1,L2);

Based on Liu, Dong, Wang, Zhu and Ma, 2009, Sensitivity kernels for
seismic Fresenl volume Tomography, Geophysics, 75(5), U35-U46

See also, kernel_fresnel_2d
```

6.2.9 kernel_multiple

```
kernel_multiple Computes the sensitivity kernel for a wave traveling
from S to R.

CALL :
   [K,RAY,Gk,Gray,timeS,timeR,raypath]=kernel_multiple(Vel,x,y,z,S,R,T,alpha,Knorm);

IN :
   Vel [ny,nx] : Velocity field
   x [1:nx] :
   y [1:ny] :
   z [1:nz] :
   S [1,3] : Location of Source
   R [1,3] : Location of Receiver
```

```
   T : Donminant period
   alpha: controls exponential decay away ray path
   Knorm [1] : normaliztion of K [0]:none, K:[1]:vertical

OUT :
   K : Sensitivity kernel
   R : Ray sensitivity kernel (High Frequency approx)
   timeS : travel computed form Source
   timeR : travel computed form Receiver
   raypath [nraydata,ndim] : the center of the raypath

The sensitivity is the length travelled in each cell.


See also : fast_fd_2d

TMH/2006
```

6.2.10 kernel_slowness_to_velocity

```
kernel_slowness_to_velocity Converts from slowness to velocity parameterizations

G : kernel [1,nkernels]
V : Velocity field (


CALL:
  G_vel=kernel_slowness_to_velocity(G,V);
or
  [G_vel,v_obs]=kernel_slowness_to_velocity(G,V,t);
or
  [G_vel,v_obs,Cd_v]=kernel_slowness_to_velocity(G,V,t,Cd);
```

6.2.11 mspectrum

```
mspectrum : Amplitude and Power spectrum
Call  :
    function [A,P,smoothP,kx]=mspectrum(x,dx)

1D (A)mplitude and (P)owerspectrum of x-series with spacing dx
```

6.2.12 munk_fresnel_2d

```
2D frechet kernel, First Fresnel Zone

See Jensen, Jacobsen, Christensen-Dalsgaard (2000) Solar Physics 192.

Call :
S=munk_fresnel_2d(T,dt,alpha,As,Ar,K);

T : dominant period
```

```
dt :
alpha : degree of cancellation
As : Amplitude fo the wavefield propagating from the source
Ar : Amplitude fo the wavefield propagating from the receiver
K : normalization factor
```

6.2.13 munk_fresnel_3d

```
3D frechet kernel, First Fresnel Zone

See Jensen, Jacobsen, Christensen-Dalsgaard (2000) Solar Physics 192.

Call :
```

6.2.14 tomography_kernel

```
tomography_kernel Computes the sensitivity kernel for a wave traveling from S to R.

CALL :
   [K,RAY,Gk,Gray,timeS,timeR,raypath]=tomography_kernel(Vel,x,y,z,S,R,T,alpha,Knorm);

IN :
   Vel [ny,nx] : Velocity field
   x [1:nx] :
   y [1:ny] :
   z [1:nz] :
   S [1,3] : Location of Source
   R [1,3] : Location of Receiver
   T : Donminant period
   alpha: controls exponential decay away ray path
   Knorm [1] : normaliztion of K [0]:none, K:[1]:vertical

OUT :
   K : Sensitivity kernel
   R : Ray sensitivity kernel (High Frequency approx)
   timeS : travel computed form Source
   timeR : travel computed form Receiver
   raypath [nraydata,ndim] : the center of the raypath

The sensitivity is the length travelled in each cell.
```