BayHunter

McMC transdimensional Bayesian inversion
of receiver functions (RF)
and surface wave dispersion (SWD)

Guide and tutorial

Jennifer Dreiling

Frederik Tilmann

German Research Centre for Geosciences
GFZ-Potsdam, Germany

January 29, 2019

Chapter 1

Development of BayHunter

BayHunter is a tool to perform a Marcov chain Monte Carlo (McMC) transdimensional Bayesian inversion of receiver functions (RF) and surface wave dispersion (SWD), i.e. inverting for the velocity-depth structure, the number of layers and noise parameters (correlation, sigma). The self developed Bayesian inversion tool uses multiple 'Marcov chains' to find the models with the highest likelihood, through the random 'Monte Carlo' sampling.

Each chain contains a current model. In each iteration a new model will be proposed, based on a proposal distribution, by modification of the current model. The acceptance probability is computed. It includes the prior, proposal and posterior ratios from proposal to current model. A proposed model gets accepted with a probability equal to the acceptance probability, i.e. if the likelihood of the proposed model is larger, it gets accepted, but also models that are worst (= less likely) than the current model, can get accepted with a small probability, which prevents the chain to get stuck in a local maximum. If a proposal model got accepted, it will replace the current model; if not, the proposal model gets declined. This process will be repeated until the given number of iterations is worked off. Each accepted model is collected (chain models). The chain models after the burn-in phase define the posterior distribution of the parameters.

1.1 The Optimizer and Chain modules

The *BayHunter.mcmcOptimizer* combines the chains in an overarching module. It starts each single chain's inversion and handles the parallel computing. Each chain and its complete model data can still be accessed after an inversion (in the Python terminal). Before the optimizer initiates the chains, an automatic configfile will be saved, which is necessary for the plotting after the inversion.

Each *BayHunter.SingleChain* explores the parameter space and proposed models get accepted or declined by following Bayes principles. A chain follows multiple steps, which are listed and described below in detail. Equations given are reduced to the number of equations required in BayHunter, but not deduced. The mathematical derivations can be looked up in several fundamental books and papers, I am especially referring here to Bodin et al. (2012) and the corresponding PhD thesis (Bodin, 2010).

Initiate the targets. The first step towards an inversion is to define the target(s). Therefor, the user needs to forward the observed data to the designated *BayHunter* target type. For receiver functions assign observed time-amplitude data to the *PReceiverFunction* (or *SReceiverFunction*) class from *BayHunter.Targets*. You need to update the forward modeling parameters (Gauss filter width, slowness, water level, near surface velocity), if the default values are different from the values used for your RF computation. For surface wave dispersion assign period-

velocity observables to the class that handles your type of surface waves (*RayleighDispersionPhase*, *RayleighDispersionGroup*, *LoveDispersionPhase*, *LoveDispersionGroup*). The forward modeling parameters to update include only the mode.

Each of these target classes comes with a forward modeling code (plugin), which is easily exchangeable. For RF the plugin is based on RFmini from Joachim Saul, GFZ, Potsdam. For surface waves a quick fortran-routine based on CPS-surf96 from Rob Herrmann is pre-installed. Also other completely different targets can be defined.

Parametrization of the model. The model we refer to includes the velocity-depth structure as well as the noise scaling parameters given through the noise covariance matrix.

- **Velocity-depth structure.** The velocity-depth model is parametrized through a variable number of Voronoi nuclei, the position of each is given by a depth and a seismic shear wave velocity (Vs). A model containing only one nucleus represents a half-space model, two nuclei define a model with one layer over a half-space and so on. The layer boundary (depth of an interface) lies equidistant between two nuclei. The advantage to use Voronoi nuclei over a simple thickness-velocity representation is, that one model can be parametrized in many different ways. However, a set of Voronoi nuclei defines only one specific model. The number of layers in a model is undefined and will also be inverted for ("transdimensional").

- **Covariance matrix of data noise.** The noise level is given through two parameters, the correlation r (i.e. the correlation from each to adjacent data points) and the noise amplitude σ. Both r and σ are treated as unknown and can be estimated during the inversion. They are basically the part of the observed data that can not be fitted. Hence, the observed data vector can be described as

$$d_{obs}(i) = d_{True}(i) + \epsilon(i), i = [1, n] \tag{1.1}$$

where n is the size of the vector and $\epsilon(i)$ represents errors that are distributed according to a multivariate normal distribution with zero mean and covariance C_e.

$$C_e = \sigma^2 R \tag{1.2}$$

The covariance matrix C_e is dependent on σ and R, which is the symmetric diagonal-constant or Toeplitz matrix.

$$R = \begin{bmatrix} 1 & c_1 & c_2 & \dots & c_{n-1} \\ c_1 & 1 & c_1 & \dots & c_{n-2} \\ c_2 & c_1 & 1 & \dots & c_{n-3} \\ & & & \vdots & \\ c_{n-1} & c_{n-2} & c_{n-3} & \dots & 1 \end{bmatrix} \tag{1.3}$$

We consider two correlation laws for the correlation matrix R. The exponential law given by

$$c_i = r^i \tag{1.4}$$

and the Gaussian law given through

$$c_i = r^{(i^2)} \tag{1.5}$$

where $r = c_1$ is a constant number between 0 and 1. In *BayHunter* we consider the exponential correlation law for surface waves, and both the exponential and the Gaussian correlation for receiver functions.

Initiate a model. For each chain, the initial model parameters (starting model) are drawn from the uniform prior distributions. These are for the velocity-depth structure the distributions of velocity, depth and the number of layers. The minimum number of layers is thereby the number of layers of the initial model. If set to 0, a half-space model will be drawn, if set to 1 a one layer over a half-space model represents the initial model and so on. The initial number of layers determines how many Voronoi nuclei will be drawn, i.e. how many pairs of velocity and depth. If a shear wave velocity-depth model was drawn, Vp will be computed from the user given Vp/Vs (default: 1.73) and the density will be computed using (Vp * 0.32 + 0.77). Vp/Vs stays constant during the inversion as well as the density computation equation.

If a springboard is needed, it is possible to give a single interface depth estimate (it can be any interface, e.g. the Moho). The estimate includes a mean and a standard deviation of the interface depth. When initiating a model - and only if an estimate was given - an interface depth is drawn from the given normal distribution and two nuclei will be placed equidistant to the interface. If the initial model only consists of a half-space, the interface estimate will be ignored. Giving an estimate helps the chains to quicker converge, but it's also cheating on the prior and on Monte Carlo. It might be practical for testing purposes, but the results will be biased and are not an actual Bayesian posterior distribution.

Each target has two noise scaling parameters (r, σ). The user needs to define the prior distributions for each of the two overarching target types, i.e. for RF and for SWD. Nevertheless, each target (even if from the same target type) will sample an own posterior distribution. You have the option to set noise parameters fix by giving a digit. Then, the initial value drawn is the digit, which will stay constant during the inversion. If you give a range, then this is assumed a uniform prior range and a random value will be drawn from this distribution. The parameter will be inverted for.

When initiating a model it automatically gets assigned as the current model of the chain. The corresponding likelihood is computed. This model is also the first model in the model chain that gets collected for the burn-in phase.

Computation of likelihood. The likelihood is an estimate of the probability of observing the measured data given a particular model m. It is an important measure for accepting or declining proposal models. The likelihood function is described as

$$p(d_{obs}|m) = \frac{1}{\sqrt{(2\pi)^n |C_e|}} \times exp\left\{\frac{-\Phi(m)}{2}\right\} \tag{1.6}$$

where $\Phi(m)$ is the Mahalanobis distance (Mahalanobis, 1936), i.e. the multidimensional distance between observed d_{obs} and estimated $g(m)$ data vectors.

$$\Phi(m) = (g(m) - d_{obs})^T C_e^{-1} (g(m) - d_{obs}) \tag{1.7}$$

As the likelihood often results in very small numbers, the log-likelihood is preferred.

$$\log p(d_{obs}|m) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|C_e|) - \frac{\Phi(m)}{2} \tag{1.8}$$

The computation of the likelihood needs the inverse C_e^{-1} and determinant $|C_e|$ of the covariance matrix. For the exponential correlation law the covariance matrix can be described as

$$C_e = \sigma^2 \begin{bmatrix} 1 & r & r^2 & \dots & r^{n-1} \\ r & 1 & r & \dots & r^{n-2} \\ r^2 & r & 1 & \dots & r^{n-3} \\ & & & \vdots & \\ r^{n-1} & r^{n-2} & r^{n-3} & \dots & 1 \end{bmatrix} \tag{1.9}$$

C_e^{-1} and $|C_e|$ can be solved through linear algebra, and are given by the analytical forms

$$C_e^{-1} = \frac{1}{\sigma^2(1-r^2)} \begin{bmatrix} 1 & -r & 0 & \dots & 0 & 0 \\ -r & 1+r^2 & -r & \dots & 0 & 0 \\ 0 & -r & 1+r^2 & \dots & 0 & 0 \\ & & & \vdots & & \\ 0 & 0 & 0 & \dots & 1+r^2 & -r \\ 0 & 0 & 0 & \dots & -r & 1 \end{bmatrix} \tag{1.10}$$

and

$$|C_e| = \sigma^{2n}(1-r^2)^{n-1} \tag{1.11}$$

These can be easily constructed and quickly computed in the Python language. Obviously, if the correlation of noise r = 0, the R and R^{-1} matrices are simply formed by the diagonal matrix and the determinant is given by σ^{2n}. This is the default case for the surface waves.

For receiver functions, unless they are computed with an exponential filter, the Gaussian correlation law should be considered. In this case C_e^{-1} and $|C_e|$ cannot be solved into an easy analytical form and the numerical computation of these is necessary. Considering a numerical computation each time a noise parameter is perturbed will slow down the Bayesian inversion by a tremendous amount of time. We can use a trick to speed up the computation, however, only if estimating r in advance and setting it fix. The equations

$$C_e^{-1} = (\sigma^2 R)^{-1} = \frac{1}{\sigma^2} R^{-1} \tag{1.12}$$

and

$$|C_e| = |\sigma^2 R| = \sigma^{2n}|R| \tag{1.13}$$

show, that we can pull σ out of the equation. Therefore, the numerical computations of R, R^{-1} and $|R|$ will be executed only once at the beginning of the Bayesian inversion. R^{-1} and $|R|$ will be multiplied by the σ-terms and plugged in equations 1.7 and 1.8 to compute the likelihood. r in R needs to be chosen by the user, but can be estimated from the RF sampling rate and the applied Gaussian filter. If r is set too high R^{-1} gets instable and small eigenvalues need to be suppressed.

To compute the likelihood for multiple data-set inversions, we can simply sum up the log-likelihoods for different targets.

Propose a model. At each iteration a new model is proposed using one of five modification methods. The method is drawn randomly and the current model will be modified according to the method's proposal distribution. Either a parameter is modified (Vs or depth of Voronoi nucleus, r, σ) or the dimension of parameters, i.e. the number of layers in the velocity-depth structure (layer birth, death). The methods are summarized below.

(1) Modification of Vs
(2) Move of Voronoi nucleus
(3) Modification of noise parameter (r, σ)
(4) Modification of dimension (layer birth)
(5) Modification of dimension (layer death)

Each method except for (3) is causing a modification of the velocity-depth structure. For (1) and (2) a random Voronoi nucleus from the current model is selected. For (1) the Vs of the nucleus is modified according to the proposal distribution for Vs. Therefore, a sample from

this normal distribution (centered at zero) is drawn and added to the current Vs value of the nucleus. For (2) a sample from the depth proposal distribution is drawn and added to the depth-coordinate of the nucleus. For (3) one random hyper-parameter from one target is selected (r or σ). The random parameter, of cause, must be one that is wished to be inverted for, i.e. ignoring the fixed noise parameters from each target. The selected value is modified according to the procedure before and according to its own proposal distribution. As mentioned before, the C_e assumed for surface waves is based on the exponential law. For receiver functions the exponential law is only assumed, if the user wants to invert for r. If r is set fix, automatically the Gaussian correlation law is considered.

For (4) and (5) the proposal distributions are equal. For (4) a random depth-value will be drawn from the uniform depth prior distribution, where a new Voronoi nucleus will be born. The new velocity of this nucleus will be computed by the current model velocity at the drawn depth, modified by the proposal distribution. For (5) a random nucleus from the nuclei ensemble of the current model is chosen and simply deleted. Here, the proposal distribution is only relevant for the computation of the acceptance probability, not for the actual modification of the model.

For each of the 5 methods a proposal distribution must be assumed, which is equal for (4) and (5), so only four proposal distribution widths need to be chosen. For the model modification methods (1)-(3) it is obvious, that small standard deviations of the distributions cause a high chance of only small parameter changes. So, the proposal models are very similar to the current model. On the other hand, if the proposal distribution width is large, then the modifications also tend to be larger and the proposal models are likely to be much more different from the current model. For (4) and (5) however, the proposal distribution only plays a subordinate role. If a random nucleus is added or removed, the complete model structure between the adjacent nuclei is modified, which can cause large interface depth shifts – dependent on the closeness of the adjacent nuclei. A nucleus birth with a Vs modification of zero would still result in a shift of a layer interface.

The values chosen for the proposal distributions, however, are not too important, as the width of the normal distributions will be adjusted during the inversion to reach a specific acceptance rate (details later).

Accept a model. After a model is proposed, it needs to be evaluated in comparison to the current model. Therefore, an acceptance probability will be computed. If any parameter of the proposed model does not lie within its prior distribution, the acceptance probability drops to zero and the model will automatically be declined. A model parameter can only lie outside the prior if the current value of it is very close to the prior limits or its proposal distribution width is very large. Additional criteria will set the acceptance probability to zero and force a refusal of the proposal model:

- any value of Vs, depth, r, σ and the number of layers lies outside its prior
- a thickness is not larger than thickmin
- a low / high velocity zone does not fulfill the user defined condition

If a model proposal clears the above criteria, the actual acceptance probability α is computed. The acceptance probability is a combined probability and will be computed from the prior, proposal and likelihood ratios of the proposal model m' and the current model m.

$$\alpha = \text{prior ratio x likelihood ratio x proposal ratio x Jacobian}$$

$$\alpha(m'|m) = \frac{p(m')}{p(m)} \times \frac{p(d_{obs}|m')}{p(d_{obs}|m)} \times \frac{q(m|m')}{q(m'|m)} \times |J| \tag{1.14}$$

The determinant of the Jacobian matrix equals 1 in any case of modification. Furthermore, the acceptance term can be rearranged dependent on the model modification.

• **Voronoi nucleus position and covariance matrix.** The modification of the nucleus position, i.e. Vs and depth, and the modification of the covariance matrix, i.e. r and σ, are proposal types that do not involve a change of dimension. For these model proposals the prior ratio equals 1 and the proposal distributions are symmetrical, i.e. the probability to go from m to m' is equal to the probability to go from m' to m. Hence, the proposal ratio also equals 1. We can shorten the acceptance probability to the likelihood ratio:

$$\alpha(m'|m) = \frac{p(d_{obs}|m')}{p(d_{obs}|m)} \tag{1.15}$$

If the Voronoi nucleus position was modified, the factor $\frac{1}{\sqrt{(2\pi)^n|C_e|}}$ in the likelihood function (Eq. 1.6) is equal for the proposed model m' and the current model m and cancels out. So α is only dependent on the Mahalanobis distance and is defined as:

$$\alpha(m'|m) = exp\left\{-\frac{\Phi(m') - \Phi(m)}{2}\right\} \tag{1.16}$$

If a noise parameter was modified, C_e is different for proposal and current model; therefore the mentioned factor in the likelihood function must be included in the computation of α. Note that the Mahalanobis distance also includes the covariance matrix C_e. The acceptance probability computes as follows:

$$\alpha(m'|m) = \left(\frac{|C_e|}{|C'_e|}\right)^{\frac{1}{2}} \times exp\left\{-\frac{\Phi(m') - \Phi(m)}{2}\right\} \tag{1.17}$$

• **Dimension change of velocity-depth model.** A dimension change of a model implies a birth or death of a Voronoi nucleus, which corresponds to a layer birth or death. In this case, the prior and proposal ratios are no longer unity. For a birth step, the acceptance probability equals:

$$\alpha(m'|m) = \frac{\theta_2\sqrt{2\pi}}{\Delta v} \times exp\left\{\frac{(v'_{k+1} - v_i)^2}{2\theta_2^2} - \frac{\Phi(m') - \Phi(m)}{2}\right\} \tag{1.18}$$

where i indicates the layer in the current tessellation c that contains the depth c'_{k+1} where the birth takes place. The acceptance probability of the birth step is a balance between the proposal probability (which encourages velocities to change) and the difference in data misfit which penalizes velocities if they change so much that they degrade the fit to the data.

The acceptance probability for a death of a Voronoi nucleus is:

$$\alpha(m'|m) = \frac{\Delta v}{\theta_2\sqrt{2\pi}} \times exp\left\{-\frac{(v'_j - v_i)^2}{2\theta_2^2} - \frac{\Phi(m') - \Phi(m)}{2}\right\} \tag{1.19}$$

where i indicates the layer that was removed from the current tessellation c and j indicates the cell in the proposed tessellation c' that contains the deleted point c_i.

The proposal ratio of forward and reverse moves is a probability of birthing or killing a layer. In proposing an increase in the number of layers (a birth step), the likelihood function will tend to encourage acceptance when fit is improved, however, the prior ratio will tend to discourage acceptance due to the increased dimensionality of the space. For more details on this matter see Bodin et al. (2012).

The proposal candidate will be accepted with a probability of α, or rejected with a probability of $1 - \alpha$. The computational implementation is a comparison of α to a number u, which is randomly drawn from a uniform distribution within the range of 0–1. The model gets accepted if $u < \alpha$. This is always the case if $\alpha > 1$. As we consider the log-space for our computations, we use $\log(\alpha)$ and $\log(u)$.

If a proposal model gets accepted, it will be replacing the current model and will also be appended to the chain models.

The posterior distribution. After a chain has finished its iterations, it automatically saves four output files holding velocity-depth models, noise parameters, likelihoods and misfits. Their filenames include the tags *models*, *noise*, *likes* and *misfits*, respectively, as well as the chain identifier and the phase specifier *p1* or *p2* (burn-in and main). The number of models that will be saved orient on the *maxmodels* constraint given by the user, i.e. every i-th model is saved to have a collection of <*maxmodels* models. The files can be found in the subfolder *savepath/data*.

See next section for outlier detection and bundling the *SingleChain* results into one posterior distribution.

1.2 The Saving and Plotting module

As the optimizer module saves a configfile right after initiation, the resaving and plotting can be done any time after the inversion. The *BayHunter.Plotting* module is also responsible for resaving the single chain results into combined output files of one representative posterior distribution for the complete inversion run; these files can be used for specific plotting methods and other statistics.

Outlier detection. There are two phases during the inversion: the burn-in and the second phase. The latter phase is the phase that provides the actual models for the posterior distribution. As the chain starts at a very random position in the parameter space, it is necessary to allow the chain some time to converge to a proper likelihood area within the parameter space. The initial time to reach a somewhat satisfying convergence is called the burn-in phase. In *BayHunter*, the two phases are defined by a user given number of iterations. However, there is no guarantee that each chain will converge. BayHunter provides a method to detect oulier chains after the inversion.

The median likelihood of the posterior sampling phase will be computed for each chain. A deviation *dev*=0.05 will be assumed to compute a threshold by (1-*dev*) times the maximum reached median likelihood of a chain. Any chain that reached only median likelihoods below the threshold will be declared outliers. The user is able to adjust the 5 % deviation. The outlier chain indices will be saved to a file, which will be overwritten when newly evaluating the outlier chains, and considered when resaving the final posterior distribution.

Final posterior distribution. The *BayHunter.PlotFromStorage* class provides a method called *save_final_distribution*. It comes with two options. *dev* is the deviation considered for outlier chains as explained above, *maxmodels* is the number of models that should be selected for the final posterior distribution. The final distribution considers all second phase (*p2*) chain models from all chains that are not outlier chains. An equal number of models per chain will be chosen to assemble *maxmodels* models. Four files will be saved in *savepath/data* representing the velocity-depth models, noise parameters, likelihoods and misfits, with filenames only including the tags *models*, *noise*, *likes* and *misfits*, respectively.

Plotting methods. Here, only a list of plotting methods is presented, the method name is more less self explaining. Some plots are illustrated and explained in the tutorial section below.

Plotting methods for BayHunter		
plot_iiterlikelihood	plot_currentdatafits	plot_posterior_nlayers
plot_iiternlayers	plot_currentmodels	plot_posterior_noise
plot_iiternoise	plot_bestdatafits	plot_posterior_misfits
plot_iitermisfits	plot_bestmodels	plot_posterior_models1d
		plot_posterior_models2d
plot_rfcorr		
merge_pdfs		

1.3 The BayWatch module

As soon as a baywatch configfile is written with *utils.save_baywatch_config*, and an inversion is running, you can use BayWatch for streaming your inversion. No more waiting until the end of the inversion to figure out that your inversion parameters need some more adjustment. No more black-box inversion. See for yourself how the chains are exploring the parameter space, how your fits and models are changing and which direction the inversion goes.

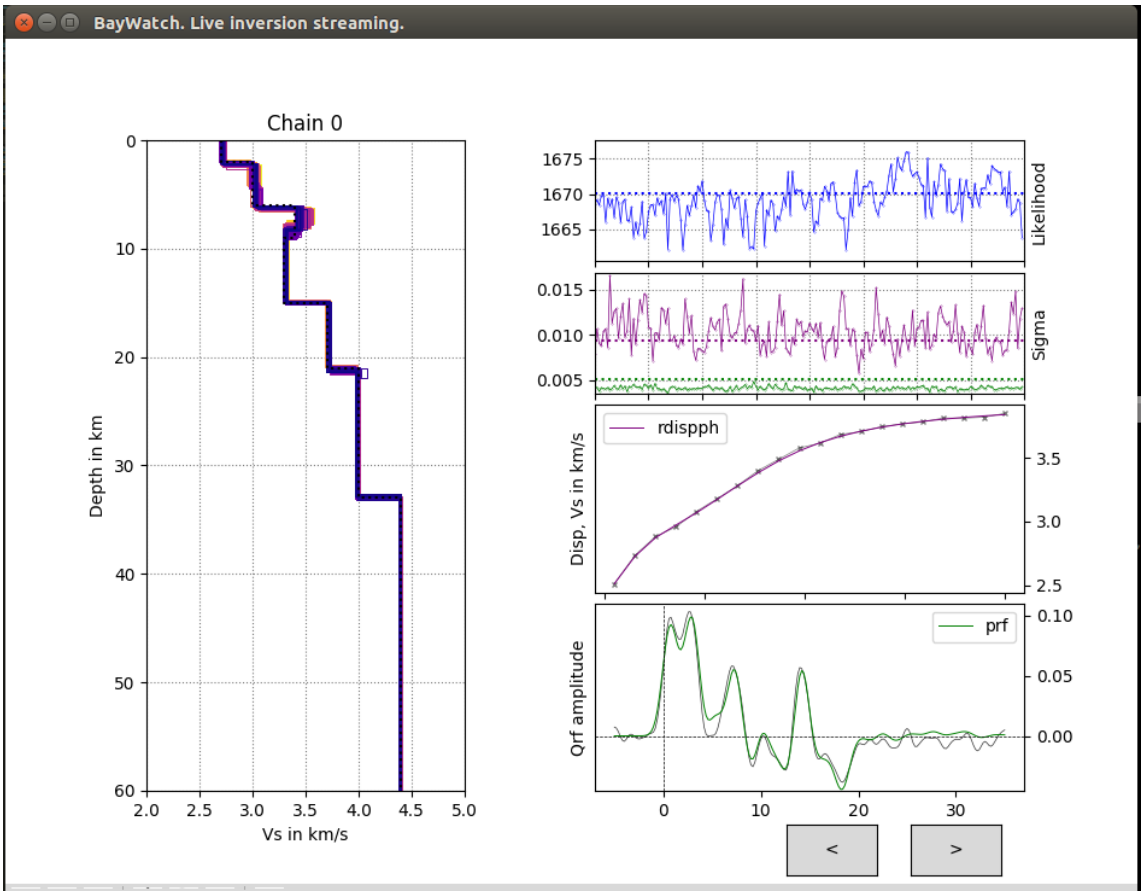

Figure 1.1: Screen shot of BayWatch showing an inversion of synthetic data from a six-layer velocity-depth model. See tutorial for parameter details.

If you set *baywatch=True* within your inversion start command, *BayHunter* opens an additional process only for sending out the latest chain models. When starting *BayWatch* these models get caught and will be illustrated as visible on the screen in Figure 1.1.

Chapter 2

Tutorial

This section explains how to install *BayHunter* on a computer, followed by an example of how to set up an inversion, i.e. the definition of the targets, the parameters and how to start the process of the inversion. An example inversion was done with synthetic data. Please be referred to the actual tutorial in the tutorial folder, providing a script with plenty of comments, and synthetic data of multiple velocity-structures.

2.1 Requirements and installation

BayHunter and *BayWatch* are set up for a python 2 environment. After checking the required python-modules, simply type the following for installation:

```
sudo python setup.py install
```

Python-modules that need to be installed	
numpy	numerical computations
matplotlib	plotting
pyPdf	merging pdfs
configobj	configfile
zmq	BayWatch, inversion live-streaming
rfmini	receiver function forward modeling

The forward modeling code for surface wave dispersion is already included in the *BayHunter* package and will be compiled when installing *BayHunter*. The code for forward modeling receiver functions, however, must be installed, but only if you wish to invert for receiver functions. You will find *rfmini* in the *BayHunter* package (rfmini.tar.gz). Please follow the given instructions for installation. rfmini was developed by Joachim Saul, GFZ, Potsdam. The code is stored at `https://git.gfz-potsdam.de/saul/rfmini`.

2.2 Setting up and running an inversion

Setting up the targets. As mentioned in the *Initiate the targets* section, *BayHunter* provides six types of targets (four SWD and two RF), which use two types of forward modeling plugins, respectively (surf96, rfmini). Both targets have the option to update the default forward modeling parameters with *set_modelparams*. Parameters and default values are given in the table below.

SWD	mode = 1	1=fundamental mode, 2= 1st higher mode, etc.
RF	gauss = 1.	Gauss factor, low pass filter
	water = 0.001	water level stabilization
	p = 6.4	slowness in deg/s
	nsv = None	near surface velocity in km/s for computation of incident angle (trace rotation). If None, nsv is taken from velocity-model.

If you want to use your own forward modeling code, you need to write a python-class for it. After normally initiating your target with *BayHunter*, initiate an instance of your new class and forward it to the *update_plugin* method of the target.

If you wish to include additional data for the inversion, i.e. an additional target with any other type of data than the given, you need to write a new target class, and one that handles the synthetic data computation (the latter as above).

For both the forward modeling class and the new target class, you will find a template in the templates folder. It is important that the classes contain specifically named methods and parameters to ensure the correct call within *BayHunter*. If the class structures are still not clear, please get inspired by the source code of the existing plugins and targets.

Setting up the parameters. Each chain will not only be initiated with the targets, but also with parameter dictionaries. The model priors and inversion parameters that need to be defined are explained below.

• **Model priors**		(defaults)
vpvs	= 1.73	
layers	= (1, 20)	
vs	= (1, 5)	# km/s
z	= (0, 60)	# km
mohoest	= None	# km
r_{RF}	= (0.35, 0.75)	
σ_{RF}	= (1e-5, 0.05)	
r_{SWD}	= 0.	
σ_{SWD}	= (1e-5, 0.1)	

The list of model priors is self explaining and most has already been explained. The priors for the velocity-depth structure include the number of layers, depth and Vs, and Vp/Vs. While the latter is a digit, unaltered during the inversion, the other three parameters must be given as a range and will be interpreted as a uniform distribution. Note, that the parameter *layers* does not include the underlying half space, which is always added to the model. There is also the option to give a single interface depth estimate with the option *mohoest*. It can be any interface, but the initial idea behind was to give a Moho estimate. As explained in the *Initiate a model* section, this should only be used for testing purposes. The estimate should be given as mean and standard deviation. The noise scaling parameters can either be a digit or a range. If you give a digit the value is unaltered during the inversion. If you give a range, then the parameter is inverted for. The range represents the limits of a uniform distribution.

For surface waves the exponential correlation law is a realistic estimate of the correlation between data points and is automatically applied. For receiver functions, the assumed correlation law should be Gaussian, if the RF are computed using a Gaussian filter, and exponential, if the RF are computed applying an exponential filter. While in the latter case the inversion for r_{RF} is possible, it is not for the Gaussian case. The computational time effort to compute the inverse and determinant of the covariance matrix each time r is perturbed would cause a tremendous

impact on the complete inversion time and is therefore not viable (see paragraphs above). Only if r is estimated by the user the Gaussian correlation law is used. Otherwise, if given a range for r, the exponential correlation law is automatically considered. Be aware that the estimation of r_{RF} during the inversion using the exponential law, may not lead to correct results, if the input RF was Gaussian filtered.

However, r_{RF} can be well estimated, as it is dependent on the sampling rate and the applied Gaussian filter width. However, if r_{RF} is too large (i.e. very close to 1) R^{-1} gets instable and small eigenvalues need to be suppressed. To define the cutoff for small singular values you need to chose *rcond*, which is only applied if not set to None. Singular values smaller than *rcond* x largest singular value (both in modulus) are set to zero. *rcond* is not a parameter of the priors dictionary, but the inversion parameters.

• **Initial inversion parameters**		(defaults)
nchains	= 3	
$iter_{burnin}$	= 4096	
$iter_{main}$	= 2048	
propdist	= (0.025, 0.025, 0.015, 0.005)	
acceptance	= (40, 45)	# %
thickmin	= 0.	# km
lvz	= None	
hvz	= None	
rcond	= None	
station	= 'test'	
savepath	= 'results/'	
maxmodels	= 50 000	

The inversion parameters can be subdivided into three categories: actual inversion parameters, model conditions and saving options. Parameters to constrain the inversion are the number of chains, the number of iterations for the burnin and the main phase, the initial proposal distributions and the acceptance rate. A large number of chains is preferable and assures a sampling with good coverage of the parameter space, as each chain starts at a randomly drawn location in the parameter space, i.e. with a completely random initial model only bound by the priors. The number of iterations should also be set high, as it can benefit, but not guarantee, the convergence of the chain towards the global likelihood maximum. The total amount of iterations is the sum of $iter_{burnin}$ and $iter_{main}$. Here you can chose a 1:1 ratio, but also any ratio you prefer. It is wise rather to increase the ratio towards the iterations in the burnin phase (i.e. $iter_{burnin} > iter_{main}$). So the chain is more likely to have converged after the burnin phase. The main phase pools the posterior distribution.

The initial proposal distribution widths for model modifications is given by the user. Here, four values have to be given, each of which represents the standard deviation of a normal distribution centered at zero. The four standard deviations must be in a specific order, representing the proposal distributions for the five methods explained in the *Propose a model* paragraph above. In the following order: Vs, depth, (birth, death), noise. The first three distributions represent velocity-depth model modifications, i.e. modification of Vs, depth and number of Voronoi nuclei in the model ensemble, with the distributions refer to Vs, depth and Vs, again. The fourth distribution represents the modification of noise parameters (r, σ).

If the proposal distributions are kept steady, the percentage of proposal models that get accepted as current models decrease with increasing inversion progress. When a coarse model is found that can describe the major features of the observed data, it gets more and more difficult

to find a proposal model that pleases the acceptance criteria. Therefore, the acceptance rate decreases at the expense of an efficient sampling. Models that are too different from the current model get declined with a much higher probability and more similar ones are more likely to get accepted. To efficiently sample the parameter space an acceptance rate is forced for each proposal method, and reached through narrowing or widening the of proposal distribution by trial. However, there is a minimum width for each proposal distribution, which cannot be undercut. The minimum proposal distribution is given as (0.001, 0.001, 0.001, 0.001), which is a standard deviation of 1 m/s and 1 m for Vs and depth modifications.

Observing the acceptance rate for the different velocity-depth modification methods, the most accepted models are Vs and depth modifications; their acceptance rate gets easily forced to the desired range without even touching the minimum proposal distribution. Birth and death steps, however, get barely accepted after an initial phase of high acceptance. If not limiting the minimum proposal distribution width for birth and death steps, the standard deviations will get as small as 10^{-10} and smaller to try to keep the acceptance rate up. However, as discussed in the *Propose a model* paragraph, the distribution width does not in the first place has influence on the model-modification, but the born or killed Voronoi nuclei, which can cause large interface shifts, even without any Vs alteration. Models modified by birth and death steps will naturally not be accepted very often and less the further the inversion progresses.

The overall acceptance rate is stuck with a specific level below the forced rate. An estimate of the actual overall acceptance rate can be made, assuming a realistic acceptance for the birth and death steps, e.g. 1 %. A user given rate of 40 % for each method would give an actual overall acceptance rate of about 30 %. ($\rightarrow$ 4 methods, 3 reach 40 %, 1 only 1% = 30% over all.) The acceptance rate given by the user is actually a range.

There are three model conditions, which may be used to constrain the velocity-depth model. It can be worthwhile to use them, dependent on the complexity you expect from the resulting models, given your data. However, you should be aware that using any of the following constraints would bias the posterior distribution. You can set a minimum thickness of layers. Furthermore you can suppress low and high velocity zones to exceed a specific velocity percentage relative to the adjacent layer. If you do not want to use the low or high velocity zone condition, just set lvz and hvz to None (default).

If you want to use the lvz criteria, you have to give a percentage of the allowed drop in velocity from each layer relative to the underlying layer. If you chose 0.1, then (for each layer in the model), the velocity of the underlying layer must not be less than 10 % of the current layer's velocity. The same principle is used if extreme high velocity zones want to be suppressed. However, watch out how a possible discontinuity (Moho) could be smoothed out, if chosen too small. The lvz and hvz criteria will be checked after a new velocity-depth model is proposed and the model will be declined if the conditions are not fulfilled.

The saving section includes three key words: station, savepath and maxmodels. The station name is optional and is only used as a reference, and for the automatic configfile saving (filename) of the inversion run. savepath represents the path where all the result files will be stored. The folders of the path will be generated if not existent. A subfolder *data* will contain all the SingleChain output files and the configfile. If using the *BayHunter.Plotting* module for resaving the output files (see above), the *data* folder will also hold the combined posterior distribution files and an outlier information file. The last folder in your savepath will serve as figure path. maxmodels is the number of models that will be saved from each chain, based on second phase.

Running an inversion. The inversion will start through the *optimizer.mp_inversion* command with the option to chose the number of threads, *nthreads*. By default, *nthreads* is equal to the number of CPUs of your PC. One thread is always occupied by a multiprocessing manager

instance. Ideally, one chain is working on one thread. If fully exhausting the capacity of a 8 CPUs PC, give *nthreads*=8 and *nchains*=multiple of (*nthreads*-1). This would cause (*nthreads*-1) chains to run parallel at all times, until nchains are worked off.

The speed of the inversion will not increase by choosing a larger *nthreads*. In fact, the speed is determined by the number of CPUs of your PC. If, for instance, you double *nthreads*, the number of chains running parallel at once is also double, but the chains stand in line for some non-threadable computations blocking one CPU at a time, so its half its speed. Actually, a wise use of *nthreads* is to decrease it, to minimize the workload for your PC that it is still accessible for other (personal) tasks.

Although having access to a cluster, inversions were also performed on a single work station to determine the duration of an inversion with standard PC equipment (e.g. Memory: 15.5 GiB, Processor: 3.60GHz x 8 CPUs). The runtime is not only dependent on the kind of the PC, but also on the number of chains and iterations, and the number of layers of the actual velocity-depth structures, which directly influences the time of the forward modeling. Having 21 chains, 150 000 iterations and an average model layer number of 3–10 layers, the inversion took 20.4 minutes, so each batch of 7 chains took roughly 7 minutes (example below).

When giving the command to start the inversion, you can decide whether you want to watch the inversion progressing. If yes, only set *baywatch*=True and optionally give a sending interval *dtsend* in seconds. However, beforehand, you need to save a baywatch_configfile.

2.3 Testing with synthetic data

For testing BayHunter, multiple synthetic data sets were created. The example test data below are based on a six-layer velocity-depth model, including a low velocity zone. This section will explain the setup of the targets, the input parameters and the result plots that are generated by the *BayHunter.Plotting* module.

Synthetic data are computed with the *BayHunter.SynthObs* module, which provides methods for computing receiver functions (P, S) and surface wave dispersion curves (Love, Rayleigh, phase, group), and also for computing synthetic noise following the exponential or the Gaussian correlation law. We computed the P-receiver function and the fundamental mode SWD of the Rayleigh wave phase velocity from the six-layer velocity-depth model. To generate the observed data for the test, we computed non-correlated noise for the SWD and Gaussian correlated noise for the RF, which was then added to the synthetic SWD and RF. The values r and σ used for noise computation are listed in the table below under *true*.

• **Model priors**		true
vpvs	= 1.73	1.73
layers	= (1, 20)	6
vs	= (2, 5)	see plots
z	= (0, 60)	see plots
r_{RF}	= 0.92	0.92
σ_{RF}	= (1e-5, 0.1)	0.005
r_{SWD}	= 0.	0.
σ_{SWD}	= (1e-5, 0.1)	0.012

• **Initial inversion parameters**	
nchains	= 21
$iter_{burnin}$	= 100 000
$iter_{main}$	= 50 000
propdist	= (0.005, 0.005, 0.005, 0.005)
acceptance	= (50, 55)
rcond	= 1e-6
station	= 'st6'

Two targets (*PReceiverFunction*, *RayleighDispersionPhase*) were initiated with the noise added synthetic data and combined to a *BayHunter.JointTarget* object. The latter and the parameter dictionaries of model priors and initial inversion parameters are forwarded to the optimizer. The parameters that are not defined fall back to the default values. We purposely

run only 150 000 iterations to show the different convergence of different chains and to visualize the outlier detection method.

The inversion finished after 20 minutes, saving and plotting methods were applied afterwards.

Figure 2.1 show the likelihood development over the iterations for all and for a selection of five different chains. A strong increase of likelihood can directly be seen at the first iterations in the burn-in phase, converging towards a stable value with increasing iterations. Some chains reached the final likelihood plateau in the burn-in phase (e.g. $c0$), some within the posterior sampling phase (e.g. $c4$), and some did not converge at all ($c2$). The chain $c2$ (also $c1$ and $c3$) would have reached the maximum likelihood, if the number of iterations would not had stopped the sampling at this early stage. However, the number of iterations can not be eternal, so it is necessary - no matter how many iterations are performed - to check for bad convergence and outlier chains.

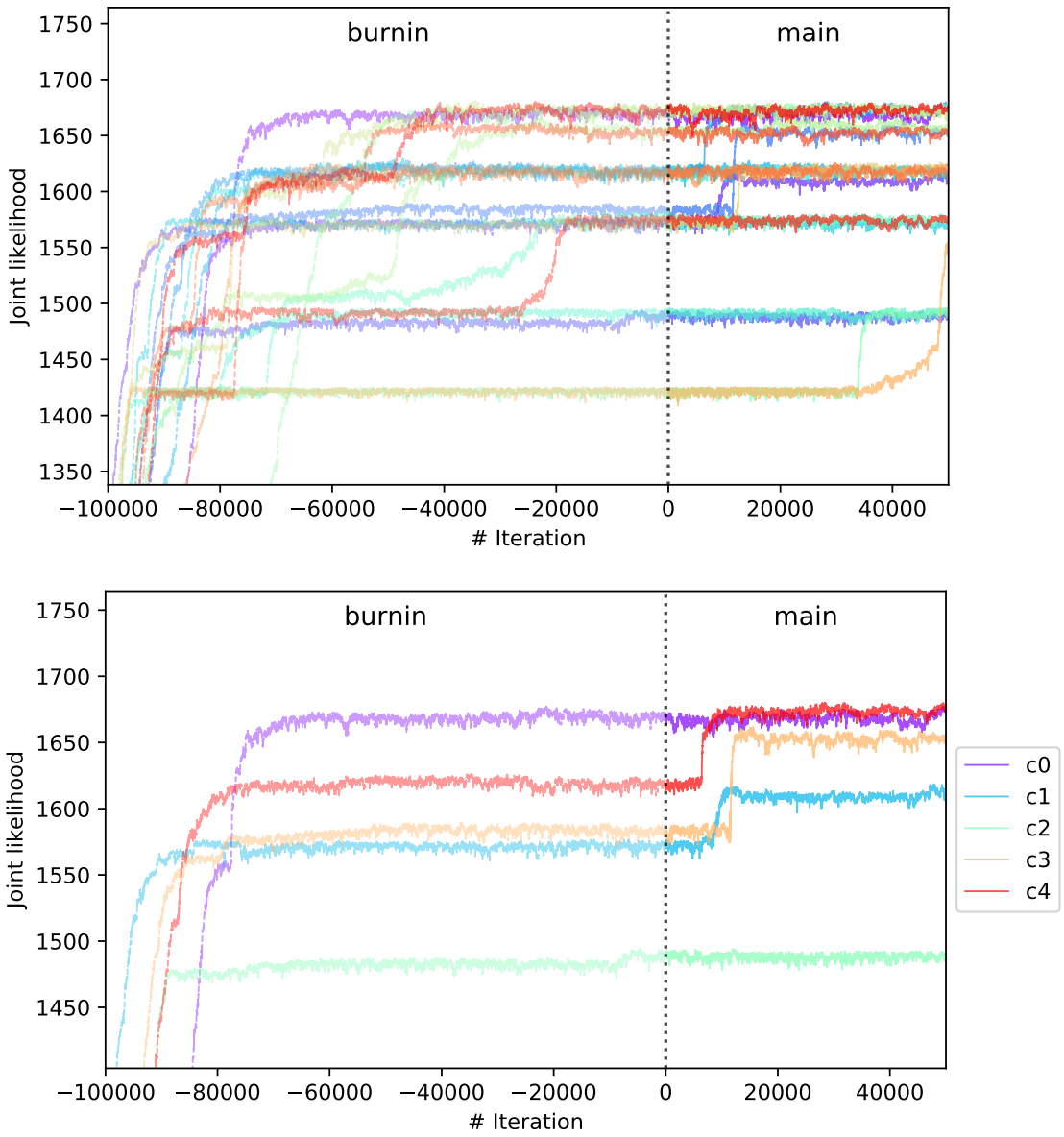

Figure 2.1: Likelihood development over iteration for all and for selected chains.

In this test, we gave a 0.02 deviation condition for outliers. With a maximum median posterior phase likelihood of 1673.6 ($c13$), the likelihood threshold is 1640, which declared 13 outlier chains with actual deviations of 0.032–0.159 (see table below). In a real case only few outlier chains should be detected, as the number of iterations should be much higher. The outlier chains will be discarded for posterior distribution plotting methods. It may be worth to also take a look at the other parameters (i.e. misfits, noise parameters, number of layers) plotted over the iterations, as they show a somewhat correlated behavior.

Outlier chains with >0.02 deviation

c1	0.039	c8	0.109	c15	0.150
c2	0.111	c9	0.059	c16	0.033
c5	0.032	c10	0.150	c17	0.033
c6	0.061	c14	0.033	c19	0.059
c7	0.033				

Figure 2.2 shows the current models from different chains and corresponding data fits (chains correspond to labeled chains in Fig. 2.1). Two chains show an outstandingly bad fit; they correspond to the outlier chains $c1$ and $c2$ as stated above. Additional values in the legend demonstrate the RMS misfit as an orientation. The remaining chains ($c0$, $c3$, $c4$) show a much better fit, the velocity-depth structures are very similar, but parameterized differently. Chains $c0$ and $c4$ already found a six-layer model, $c3$ found a five-layer model averaging the low-velocity zone.

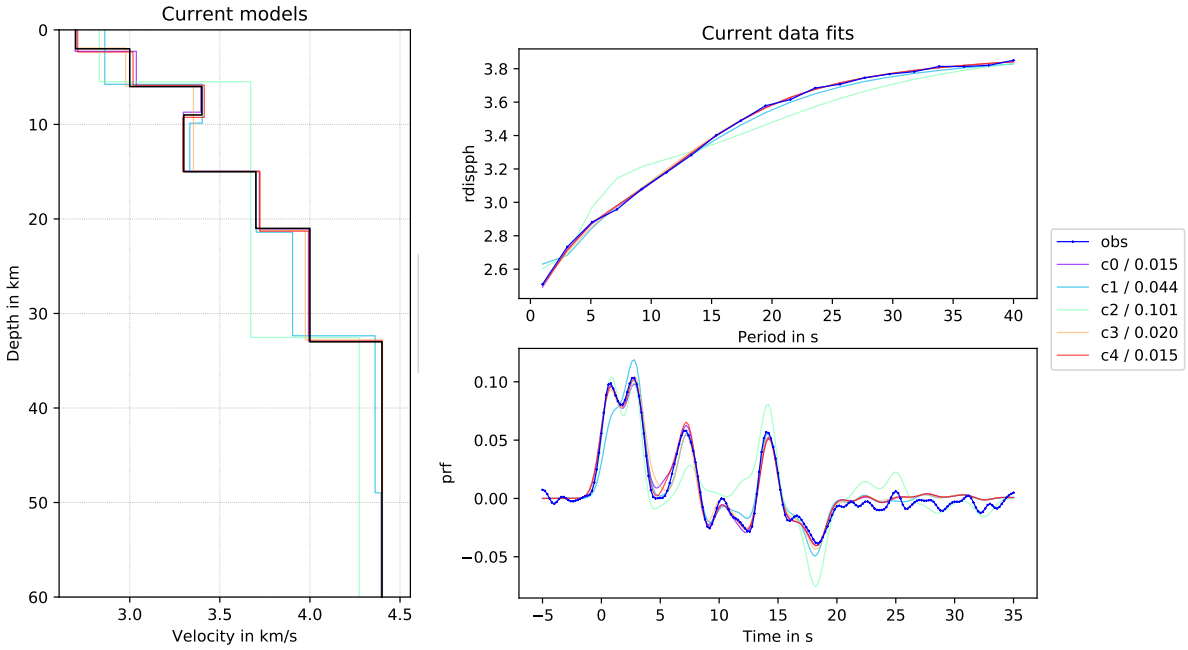

Figure 2.2: Data fits from current models of different chains. Black line (left) shows the synthetic input model. Legend values are RMS misfits of observed and modeled data.

The posterior distributions of the converged 8 chains are illustrated below. The velocity depth-structures and the number of layers are illustrated in Figure 2.3. Figure 2.4 shows the results for the noise scaling parameters for each target, i.e. the noise correlation r (left) and the amplitude σ (right). The red lines indicate the true model as given in the model priors table above.

The velocity-depth 2d histogram also allocates an interface probability plot (gray histogram

bars on the right side). The mean model of 100000 models from the posterior distribution images the true model very well, including the low velocity zone. The number of layers is determined to be most likely six - which is true. The σ distributions of both, RF and SWD show a somewhat Gaussian shape, inhering a long tail of higher values, from which is guessable that there is still an undetected outlier chain included. σ_{SWD} represents a good estimate, still slightly overestimated, because the posterior distribution in this inversion still contains models of unconverged chains, meaning chains that did not fully converge when entering the posterior sampling phase (e.g. $c4$). This falls back to the number of iterations. Tests with more iterations show that the median of σ_{SWD} lies in very good agreement with the true value.

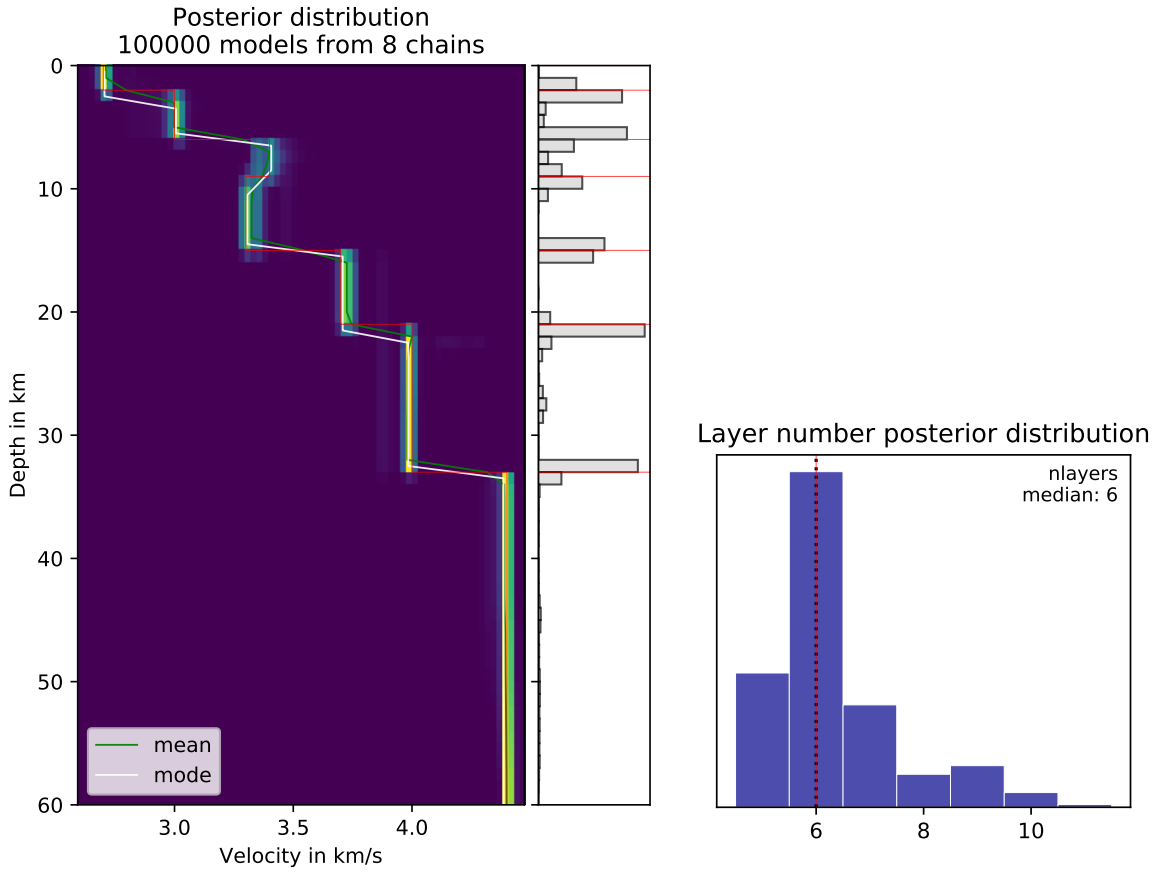

Figure 2.3: Posterior distribution of velocity-depth structures and number of layers.

σ_{RF} is underestimated, which theoretically means that also noise gets fitted. Here being said that it gets complicated if assuming a noise correlation, which is e.g. not assumed by the SWD. The synthetic RF was computed with a Gaussian factor of 1. The noise was generated through a random process, with an r estimated to represent the applied Gauss filter, and added afterwards to the synthetic data. The random process of generating noise does not output a noise vector which exactly matches the given values of r and σ. If only drawing one single realization with a determined amount of samples from the multivariate normal distribution may produce deviations from the targeted r and σ. From the generated noise the real σ can be easily computed (as was done for SWD, the value in the table and the red line in the plot differ slightly). For RF, however, a real σ was computed, but the real r is not reconstructible (?). Assuming a wrong r for the inversion cannot lead to the correct σ. A reason for misestimating the σ_{RF} in this case could also be that signal and noise are not computed and filtered together, but separately. Real data, i.e. signal and noise, are recorded and filtered simultaneously.

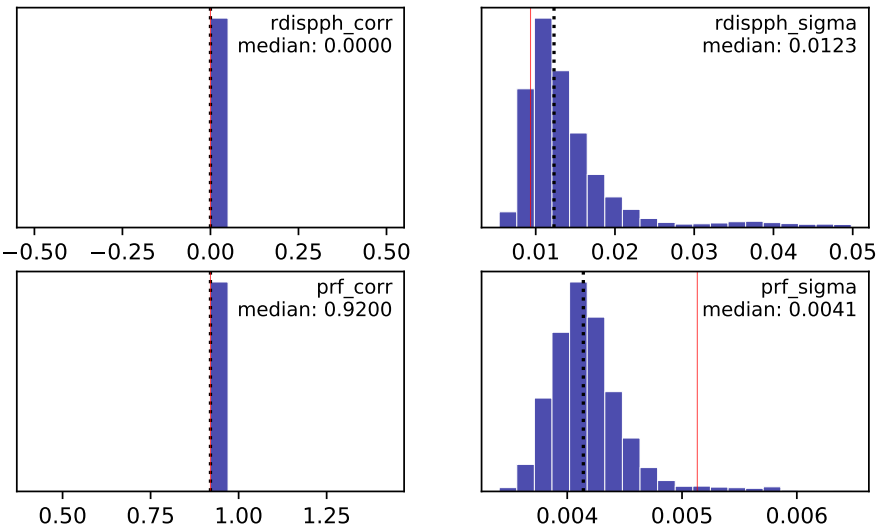

Figure 2.4: Posterior distributions of noise parameters for each of the targets.

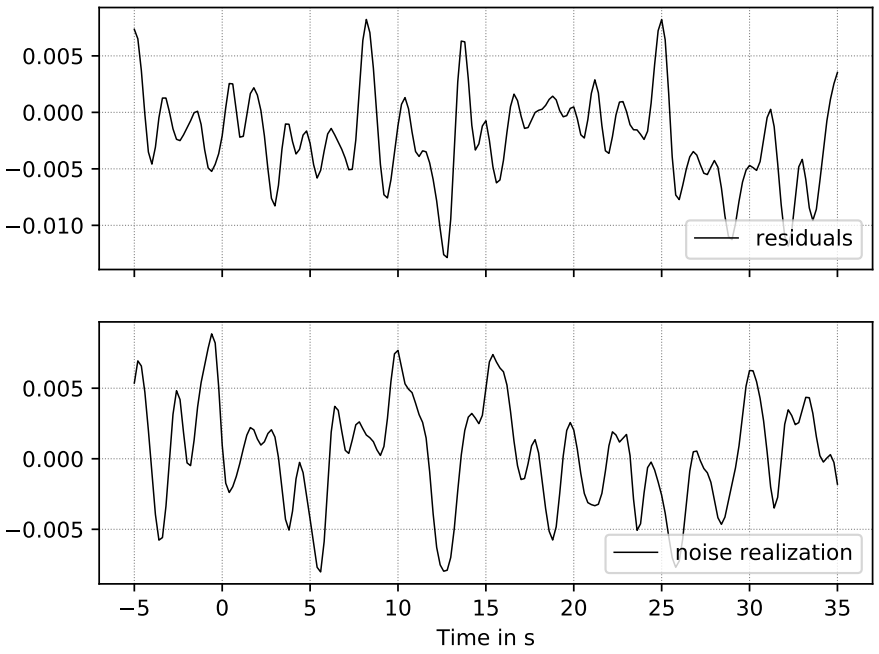

Figure 2.5: Comparison of residuals and one realization of noise through C_e^{RF}.

To estimate a realistic r_{RF} you can use the *plot_rfcorr* method. Figure 2.5 shows an output sample. The top plot shows the residuals of the best fitting model, which basically represents the part of the data that cannot be fitted, i.e. the noise. The lower plot shows one realization of noise with the given correlation r and the estimated σ_{RF}. These two data vectors follow a similar pattern (frequency, amplitude), which shows that r_{RF} was realistically chosen. For more explanations see Bodin et al. (2012).

There are not many parameters to set for the inversion. But please, chose every one of them carefully.

Good luck :)

References

- Bodin, T., Sambridge, M., Tkalčić, H., Arroucau, P., Gallagher, K., and Rawlinson, N. Transdi- mensional inversion of receiver functions and surface wave dispersion. Journal of Geophysical Research: Solid Earth, 117(B2):B02301, Feb. 2012. doi: 10.1029/2011JB008560.

- Mahalanobis, P. On the generalized distance in statistics. 1936.