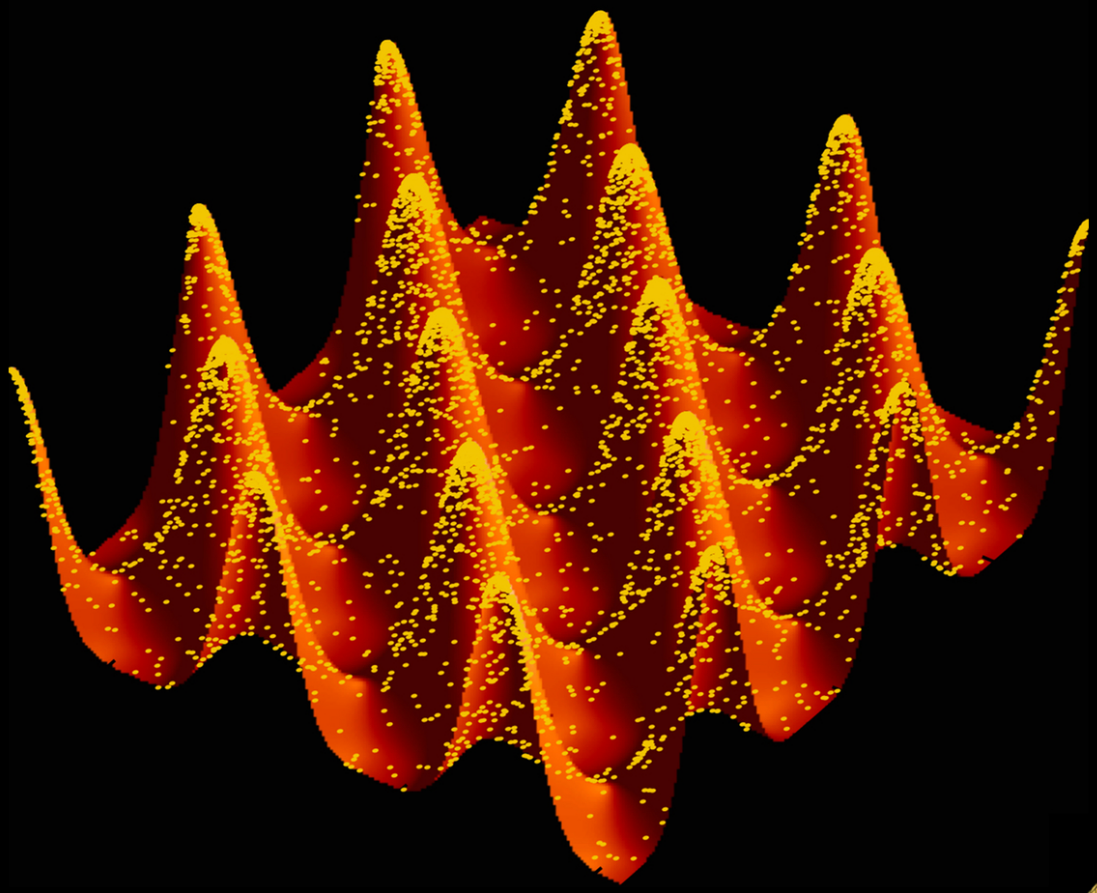
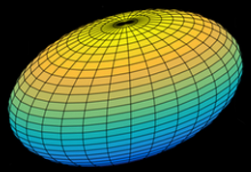
DIAMONDS™
BAYESIAN SOFTWARE

User Guide
Manual

Enrico Corsaro





Based on
Ellipsoidal Sampling

KATHOLIEKE UNIVERSITEIT LEUVEN

2018
Edition

User Guide
Manual

Enrico Corsaro

Marie Sklodowska-Curie Fellow
emncorsaro@gmail.com / enrico.corsaro@oact.inaf.it

INAF - Osservatorio Astrofisico di Catania, Italy

2018
Edition

Contents

Abstract

The DIAMONDS (high-DImensional And multi-MOdal NesteD Sampling) code allows to perform Bayesian parameter estimation and model comparison by means of the nested sampling Monte Carlo (NSMC) algorithm, an efficient and powerful method very suitable for high-dimensional and multi-modal problems. This code can be used for any application involving Bayesian parameter estimation and/or model selection in general, with no given limitations imposed on both the extent of the datasets and the number of free parameters involved in the model. Developed in `C++11`, DIAMONDS is structured in classes for good flexibility and configurability. Any new model, likelihood and prior probability density functions (PDFs) can be defined and implemented upon a basic template known as an abstract class. The first chapter of this user guide manual introduces the different parts of the code and their implementation, providing directly usable code examples. A description of the individual outputs created at the end of the computation is also given. In the second chapter we discuss the individual configuring parameters that characterize the ellipsoidal sampling algorithm, on which the current version of DIAMONDS is based. The third chapter is instead focused on troubleshooting of the main computational failures that may occur during the execution of a process. To conclude, in the fourth chapter we provide a detailed documentation on how to check and understand the reliability of the results, an essential step for using the code with consciousness. The official website with package content description, and installation guide and compilation troubleshooting for both Mac and Linux OS is available at the URL: https://fys.kuleuven.be/ster/Software/Diamonds/. The latest release of the code is free for download from the public GitHub repository: https://github.com/EnricoCorsaro/DIAMONDS.

1 Getting started

The DIAMONDS (high-DImensional And multi-MOdal NesteD Sampling) code presented in this user guide manual is a general Bayesian inference (both parameter estimation and model comparison) tool developed in `C++11` and structured in classes in order to be as much flexible and configurable as possible. The working flow from the main function of the code is as follows (see also Figure 1.1):

1. Read an input dataset
2. Set up model, likelihood, and prior distributions to be used in the Bayesian inference
3. Set up a drawing algorithm
4. Configure and start the nested sampling
5. Compute and print the final results

This represents the general sequence of steps that is necessary to follow in the exact order indicated for guaranteeing the operations. Throughout this manual, we will assume that the user has already read the original paper of the code, provided by [7]. It is essential to have a basic knowledge about all the working method, the algorithm, and the features implemented in DIAMONDS to be able to use in the best way the information provided. All the documentation presented here, especially concerning the meaning and behavior of the configuring parameters of the code, the troubleshooting and the inspection of the results, is based on the experience acquired using DIAMONDS for the scientific works published by [7, 9, 8]. This manual will be updated from time to time whenever more useful testing of the code is available. Users are encouraged to provide their feedback to help us improving the tool and its description. We recommend to use DIAMONDS with consciousness, not as a black box, and to always inspect its results with critical sense before adopting them for any purpose.

We start in this chapter by showing some preliminary information to help the user becoming more familiar with the different parts of the code, thus complementing the documentation with code examples that can be directly implemented in the main function of your application based on DIAMONDS. We consider the drawing algorithm described in [7], namely the Multi Ellipsoidal Sampler based on the simultaneous ellipsoidal sampling (SES) and the X-means clustering algorithms. This drawing system is provided in the core package of the code, available for free download in the official website. A detailed description of the package content is given here, while a comprehensive installation guide for both Mac and Linux OS, including a troubleshooting to compilation errors, is available here.

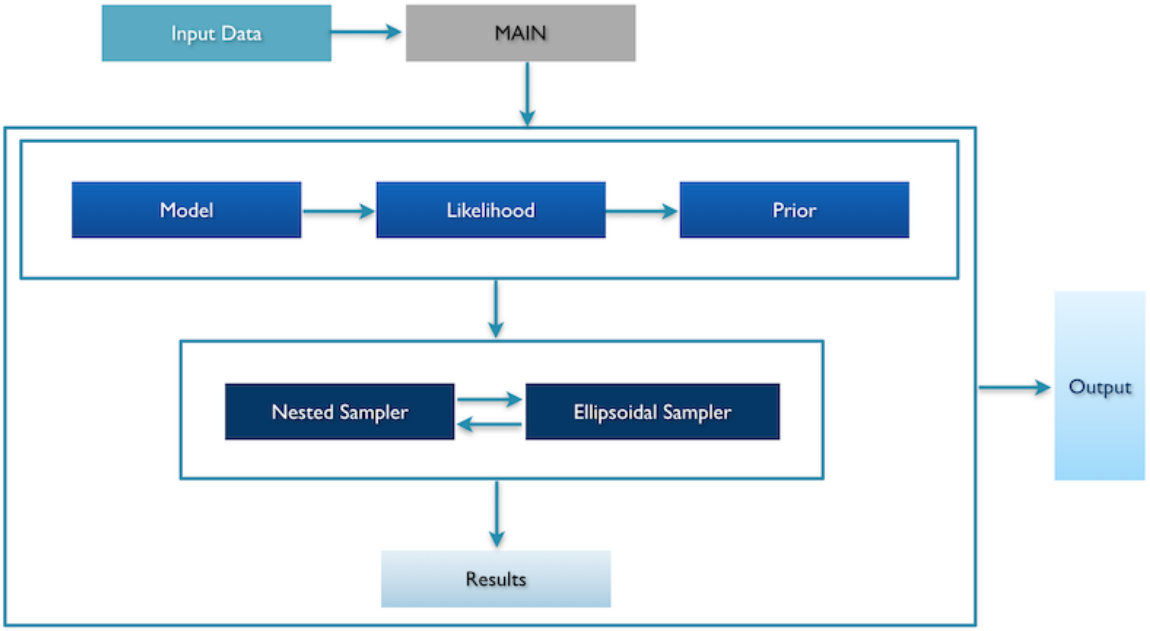


Figure 1.1: General working flow of the DIAMONDS code.

1.1 Setting up the local working path

Setting up the local working path is an essential step before starting any computation. To avoid hard coding the path, being obliged to recompile the code every time it is installed or the local paths change, one can use an input ASCII file containing the local path to be read as a string by the program. This can be done by using the function `vectorStringFromFile()` of the `File` namespace provided in the DIAMONDS code package. An example code to read such a string input file containing the local path is given below

```
// Read the local path for the working session from an input ASCII file
unsigned long Nrows;
ifstream inputFile;
File::openInputFile(inputFile, "localPath.txt");
File::sniffFile(inputFile, Nrows, Ncols);
vector<string> myLocalPath;
myLocalPath = File::vectorStringFromFile(inputFile, Nrows);
inputFile.close();
```

where the integer `Nrows` will contain the number of lines sorted by row, assuming that only one column is present. The file content is open by using the C++ input file stream class `ifstream`. The string variable `inputFileName` will contain the exact name and full path of the input file. The content of the file is stored in the vector of strings `myLocalPath`. The local path, supposed to be provided as a single line file, will then be accessible by using `myLocalPath[0]`.

1.2 The input dataset

The first element in any Bayesian inference or model comparison problem is the input dataset that we want to investigate. The structure of an input dataset is represented by an ASCII file

with a two-column format, namely the covariates (or independent variables, or x-axis values) and the observations (or dependent variables, or y-axis values). In some cases, the input data file could include a third column with uncertainties on the observations. The input data file must be read and stored in Eigen[1] arrays at the beginning of the main function of the specific application using DIAMONDS. Clearly this is an essential step to carry on with the computations. An example of code using Eigen arrays of double type values for reading an input dataset is

```
// Read the input dataset
int Ncols;
ArrayXXd data;
File::openInputFile(inputFile, inputFileName);
File::sniffFile(inputFile, Nrows, Ncols);
data = File::arrayXXdFromFile(inputFile, Nrows, Ncols);
inputFile.close();

// Creating arrays for each data type
ArrayXd covariates = data.col(0);
ArrayXd observations = data.col(1);
ArrayXd uncertainties = data.col(2);
```

where we assumed that also uncertainties were provided as a third column. In order to read the content of the ASCII file, we used again the C++ input file stream class `ifstream`, coupled with the functions `arrayXXdFromFile()`, `openInputFile()` and `sniffFile()` of the `File` namespace provided in the DIAMONDS code package. The integers `Nrows` and `Ncols` will contain the number of data bins sorted by row, and the number of data columns (3 for the given example), respectively. It is important to note that all the data arrays must contain exactly the same number of elements. If this is not not verified, the code will not be able to start the computation.

1.3 The model

A second preliminary step for the Bayesian inference concerns the definition of the model that we want to use for fitting the dataset. This model can be implemented starting from the basic abstract class `Model` included in the code package. The order of the free parameters defined in the implementation of the model is a sensible information through the computation process. The same order of the free parameters will be that of the prior distributions and of the output results. In order to compute the predictions to the observations, the model will require as input the array containing the covariates from the dataset. This is because the array containing the fitting model, usually termed *predictions*, will be of the same length of the covariates array. The object containing the model of the derived class, here labeled `UserSpecifiedModel`, has to be initialized as in the following example:

```
UserSpecifiedModel model(covariates);
```

[1] Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms that is used by DIAMONDS, and it is particularly efficient and optimized for cache-friendliness. You can find the official website with free download and full documentation here. The version of the code that this user guide is referring to implements the release 3.3.4 of the Eigen library.

1.4 The likelihood function

Given the dataset available, the choice of a proper likelihood function is essential. In most of the applications, a Normal likelihood function, as the one described by [6] and used by [9], implemented in the `NormalLikelihood` class of the code package, will be the one that we need. In this case, the input dataset must contain a three-column format, with uncertainties included as a third column. In other applications as those regarding the analysis of Fourier transform datasets (e.g. in asteroseismology), an Exponential likelihood function is instead the most suited (e.g for the problems described in [7, 8]), and you can find it implemented in the `ExponentialLikelihood` class of the package. Nonetheless, the user has the possibility to implement its own likelihood function by starting from the basic abstract class `Likelihood`. The likelihood function will require as input the object containing the model, created in the previous step, and the observations from the input dataset (and additionally the uncertainties if they are used). This is because for each data point corresponding to an observation, a prediction from the model is given so that the likelihood for that measurement can be evaluated. The basic package contains also another interesting likelihood, which is a Normal likelihood averaged over the uncertainties (see [5, 2] for more details), which can be useful for problems where the uncertainties on the observations are not entirely trusted and need to be verified (see the `MeanNormalLikelihood` class). An example code on how to create a `NormalLikelihood` class object reads

```
NormalLikelihood likelihood(observations, uncertainties, model);
```

while one for an `ExponentialLikelihood` class object is

```
ExponentialLikelihood likelihood(observations, model);
```

where this time no uncertainties are given.

1.5 The prior probability distribution

For each of the free parameters of the model, the user has to define a prior probability distribution, which is a key ingredient in the Bayesian approach. The choice of priors should be done consciously as it may have an impact on the final result. The prior distribution can be included either by using the existing distributions provided in the code package (in the classes `UniformPrior`, `NormalPrior` and `SuperGaussianPrior`), or by implementing your own one based on the basic abstract class `Prior`. The priors have to be defined in the main function of the code and they order must follow that in which the free parameters implemented in the model are defined. In addition, they clearly have to be in number equal to the number of free parameters of the model. If this is not happening, the computation will not be able to start.

The optimal prior distribution to guarantee highest computational speed is that of the uniform priors, which only requires the definition of lower and upper parameter bounds allowed within the inference problem, although assuming a fixed range of variation for a free parameter is not reflecting a total status of ignorance about that parameter itself [18], so they have to be chosen with care. However, users can define multiple prior types and combine them in any order based on the set of free parameters available. To use multiple prior types, one has to define a vector of pointers to abstract `Prior` class objects, as shown in the following example for two different types

```
int NpriorTypes = 2;
vector<Prior*> ptrPriors(NpriorTypes);
PriorClassName0 priorClassName0(hyperParameters0);
PriorClassName1 priorClassName1(hyperParameters1);
ptrPriors[0] = &priorClassName0;
ptrPriors[1] = &priorClassName1;
```

where PriorClassName<prior#> is either one of the prior classes provided in the DIAMONDS code package or implemented separately by the user. Each element in the ptrPriors vector will correspond to either a single free parameter or a set of consecutive (stacked contiguously) free parameters for which the same prior type is adopted. Each prior object has to be initialized with a specific set of so-called hyper parameters, which in turn will depend on the corresponding free parameters for which we define the prior PDF[2].

There is also the possibility to store the hyper parameters used in the computation in an ASCII file that one can retrieve afterwards. This can be useful if the user wants to keep track of which prior hyper parameters were used for that specific process, in case multiple tests need to be performed. To do so, we can use the member function writeHyperParametersToFile() of the individual prior class and write the following line of code

```
string fullPathHyperParameters = outputPathPrefix + "hyperParametersClassName.txt";
priorClassName.writeHyperParametersToFile(fullPathHyperParameters);
```

where the string specifies the name of the file completed by the full path where we desire store all the output files (including a filename prefix to identify the name of the application), which we labeled as the string outputPathPrefix. The format of the file will be that of a table of values in which each row corresponds to a free parameter, in the same order given by the model, while each column represents one hyper parameter of the prior in the same order as given as input. We need to keep in mind that if we are using different prior types, this operation must be done separately for each of them. Following the example given before we will have

```
string fullPathHyperParameters0 = outputPathPrefix + "hyperParametersClassName0.txt";
priorClassName.writeHyperParametersToFile(fullPathHyperParameters0);
string fullPathHyperParameters1 = outputPathPrefix + "hyperParametersClassName1.txt";
priorClassName.writeHyperParametersToFile(fullPathHyperParameters1);
```

in which we changed both the prior class name and the file name to avoid overwriting.

1.6 Configuring the cluster algorithm

There are two distinct groups of input parameters that are used to set up the core of the sampling algorithm. The first group is related to the X-means clustering algorithm, with minimum and maximum number of clusters to be specified, and implemented in the KmeansClusterer class. A more detailed discussion about the number of clusters to be used is addressed in Section 2.1. A code example of how to configure the clusterer with input parameters read from an input file is given below:

[2]See [7] for more details on what are the hyper parameters for each prior type of those provided in the package.

```
// Read minimum and maximum number of clusters from an input file
inputFileName = outputDirName + "Xmeans_configuringParameters.txt";
File::openInputFile(inputFile, inputFileName);
File::sniffFile(inputFile, Nparameters, Ncols);
ArrayXd configuringParameters;
configuringParameters = File::arrayXXdFromFile(inputFile, Nparameters, Ncols);
inputFile.close();
int minNclusters = configuringParameters(0);
int maxNclusters = configuringParameters(1);

// Define an Euclidean metric for the X-means clustering
int Ntrials = 10;
double relTolerance = 0.01;
EuclideanMetric myMetric;
KmeansClusterer kmeans(myMetric, minNclusters, maxNclusters, Ntrials, relTolerance);
```

where we need to define an object, of the class type `Metric`, which specifies the metric to be used within the X-means. This is usually an Euclidean metric for all standard applications, hence an `EuclideanMetric` class. Similarly to what done when reading the input dataset, the integers `Nparameters` and `Ncols`, to be declared earlier in the main function, will contain the number of parameters sorted by row (in this case 2), and the number of columns (1), respectively. The parameters defining the number of trails and the relative tolerance are kept fixed to the values indicated because they were calibrated in the testing phase. We considered that the input file containing the configuring parameters is stored in the main folder containing the results. This is a good rule of thumb to avoid messing up the files used for an individual process with those coming from a different run. Reading the configuring parameters from an input files is a very convenient choice to avoid recompiling the code every time we intend to try a different value for one or more of them.

1.7 Configuring the nested sampling algorithm

The second group of configuring parameters concerns the nested sampling algorithm and the effective drawing mechanism done by means of the ellipsoids, implemented in the classes `NestedSampler` and `MultiEllipsoidSampler`, respectively. These parameters were already introduced and described by [7], Section 4.1, but in the context of this user guide we will put more focus on using two of them, namely the initial enlargement fraction f_0 and the shrinking rate α of the ellipsoids. The ellipsoids have the great advantage of allowing us reducing prior volume to draw from as the computation evolves, hence speeding up the computation itself [16, 17]. Nonetheless, ellipsoids have the drawback of requiring careful configuration because a wrong set of input parameters may affect the sampling, hence the reliability of the results, especially of the uncertainties on the free parameters. In the worst case, a wrong configuration will cause the computation to stop prematurely. In the following Chapter 3 and Section 4 we shall investigate the most common problems arising when dealing with the ellipsoids. The recipes provided should ensure a proper configuration of DIAMONDS and therefore producing reliable results for a wide variety of applications. The values suggested for the different parameters and an explanation are provided in Section 2.

The code below shows how to read the configuring parameters of the nested sampler based on the SES from an input file, assuming the parameters are listed in a row format, similarly to what done for the clusterer.

```
inputFileName = outputDirName + "NSMC_configuringParameters.txt";
File::openInputFile(inputFile, inputFileName);
File::sniffFile(inputFile, Nparameters, Ncols);
configuringParameters.setZero();
configuringParameters = File::arrayXXdFromFile(inputFile, Nparameters, Ncols);
inputFile.close();

// Print results on the screen
bool printOnTheScreen = true;

// Initial number of live points
int initialNlivePoints = configuringParameters(0);

// Minimum number of live points
int minNlivePoints = configuringParameters(1);

// Maximum number of attempts when trying to draw a new sampling point
int maxNdrawAttempts = configuringParameters(2);

// The first N iterations, we assume that there is only 1 cluster
int NinitialIterationsWithoutClustering = configuringParameters(3);

// Clustering is only happening every N iterations
int NiterationsWithSameClustering = configuringParameters(4);

// Fraction by which each axis in an ellipsoid has to be enlarged
// It can be a number >= 0, where 0 means no enlargement
double initialEnlargementFraction = configuringParameters(5);

// Exponent for remaining prior mass in ellipsoid enlargement fraction.
// It is a number between 0 and 1.
// The smaller the slower the shrinkage of the ellipsoids.
double shrinkingRate = configuringParameters(6);

// Termination factor for nested sampling process
double terminationFactor = configuringParameters(7);
```

where we enabled the flag `printOnTheScreen` to visualize the status of the computation on the terminal screen every 50 nested iterations (see Section 1.8 for more details). The configuring parameter `initialEnlargementFraction` can alternatively be fixed by using the power law relation provided by Eq. (2.4) in Section 2 (this proves to be very useful for applications involving an asteroseismic analysis, [8]). In the final part, we have to pass this information to the sampler, hence execute the algorithm by using the member function `run()` of the `NestedSampler` class, as shown in the following code

```
MultiEllipsoidSampler nestedSampler(printOnTheScreen, ptrPriors, likelihood, myMetric,
kmeans, initialNlivePoints, minNlivePoints, initialEnlargementFraction, shrinkingRate);

double tolerance = 1.e2;
double exponent = 0.4;
PowerlawReducer livePointsReducer(nestedSampler, tolerance, exponent, terminationFactor);

nestedSampler.run(livePointsReducer, NinitialIterationsWithoutClustering,
NiterationsWithSameClustering, maxNdrawAttempts, terminationFactor, outputPathPrefix);
```

where the PowerLawReducer class object allows for a reduction of the live points according to the power law relation provided by [7] (see their Section 4.4 for more details). Alternatively one could use the law provided by [4], which is implemented in the FerozReducer class. The values of the parameters defining the tolerance and the exponent of the live points reducer are only shown as an example. The reducer has to be defined in any case, even if not used (see also Section 2). The string variable outputPathPrefix is the same as that used for the prior hyper parameters, Section 1.5, and usually contains a subfolder of outputDirName named with the label of the run number that we are using[3] (see also in Section 1.8).

1.8 What output is obtained and how to use it

When the flag printOnTheScreen is turned on, a line of output is printed on the screen every time that 50 consecutive nested iterations are completed. An example of a line of this output reads

```
Nit: 50  Ncl: 1  Nlive: 2000  CPM: 0.0251776  Ratio: 6.45e+102  log(E): -2.375  IG: 3.68
```

Nit is the number of the current nested iteration, in this case corresponding to the second line printed on the screen (the first one is for Nit: 0). Ncl is the actual number of clusters (and not the input ones we provided), and it will be able to vary within the allowed range as the computation proceeds. Nlive is the number of live points used in the given nested iteration, and this information becomes useful especially when the reduction of the live points is taking place, which causes Nlive to vary. CPM represents the cumulated prior mass, which converges to 1 (total prior probability) toward the end of the process. Ratio is the final termination condition, termed $\delta_{\rm final}$ by [7], and it is an important parameter to track for understanding how much is left to sample from the posterior probability distribution. When this number reaches the value of the termination factor given as input, the computation will end and the results can be generated. The last two numbers, log(E) and IG, are the natural logarithm of the cumulated (up to the given nested iteration) Bayesian evidence, and the information gain from the previous iteration, respectively.

When the computation is completed, one displays a message specifying the final natural logarithm of the Bayesian evidence and the error bar, and the total computational time of the process, as in the example below:

```
-----------------------------------------------
 Final log(E): 3.65068e+06 +/- 0.270612
-----------------------------------------------
 Total Computational Time: 1.16 hours
-----------------------------------------------
```

After the computation is completed and this message appears, assuming that there were no errors throughout the process (see Chapter 3 for all the details), the class Results is invoked

[3] The string outputPathPrefix can be defined as in the following: outputPathPrefix = outputDirName + runNumber + "/applicationName_", where runNumber is the string corresponding to the label of the given run (usually you have more than one run and you can label them with progressive integer numbers, e.g. 00, 01, ...), while the last bit of the string represents the label prefix that identifies the given application and that is attached to all the output file names, e.g. for the case of a peak bagging analysis it would be peakbagging_, and we will therefore have for instance the parameter summary file the name peakbagging_parameterSummary.txt. This is a simple way to avoid confusing the results from different runs and applications.

and all the general output files of DIAMONDS are created, except for the one containing the configuring parameters, which is created by the NestedSampler class itself as discussed below. There are several choices that can be included in the printing of the final results, but a standard (and complete) set will include the following ASCII files:

- a 1-column format computationParameters.txt file, containing all the configuring parameters used for the computation and additional key parameters that summarize the process and its status. This file can be very useful especially when running several different processes because it allows us to keep track of which values were used for each configuring parameter of the code, even if we changed the input files, but also if the code has stopped prematurely. This is the only file that is created automatically by the NestedSampler class after the computation starts, and it contains the following default list of parameters:

```
# List of configuring parameters used for the NSMC.
# Row #1: Ndimensions
# Row #2: Initial(Maximum) NlivePoints
# Row #3: Minimum NlivePoints
# Row #4: NinitialIterationsWithoutClustering
# Row #5: NiterationsWithSameClustering
# Row #6: maxNdrawAttempts
# Row #7: terminationFactor
# Row #8: Niterations
# Row #9: Optimal Niterations
# Row #10: Final Nclusters
# Row #11: Final NlivePoints
# Row #12: Computational Time (seconds)
# Row #13: Error: No better likelihood found (1 = yes / 0 = no)
# Row #14: Error: Ellipsoid matrix decomposition failed (1 = yes / 0 = no)
```

  However, not all the parameters are stored automatically, and those belonging specifically to the ellipsoidal sampler must be included in a separate step. This is to allow for more flexibility in the event of a future replacement of the currently adopted ellipsoidal sampler. Since the related output file stream is not closed after the computation has ended, we can append additional parameters to it, by using the following standard sample code made for the ellipsoidal sampler:

```
nestedSampler.outputFile << "# List of configuring parameters used for the
ellipsoidal sampler and X-means" << endl;
nestedSampler.outputFile << "# Row #1: Minimum Nclusters" << endl;
nestedSampler.outputFile << "# Row #2: Maximum Nclusters" << endl;
nestedSampler.outputFile << "# Row #3: Initial Enlargement Fraction" << endl;
nestedSampler.outputFile << "# Row #4: Shrinking Rate" << endl;
nestedSampler.outputFile << minNclusters << endl;
nestedSampler.outputFile << maxNclusters << endl;
nestedSampler.outputFile << initialEnlargementFraction << endl;
nestedSampler.outputFile << shrinkingRate << endl;
nestedSampler.outputFile.close();
```

  If no further parameters are to be printed, one should not forget to close the output file stream in any case.

- a 1-column format parameter<parameter#>.txt file, for as many free parameters as the model and priors have included. These files contain the original sampling of the parameter values following the exact order imposed by the nested sampling, hence sorted in increasing likelihood values, as given in the logLikelihood.txt file (hence the parameter values are not sorted in increasing parameter value conversely to what is given in the marginalDistribution<parameter#>.txt files). These files can be useful to construct the marginal probability distributions if one wants to start from the original sampling obtained by DIAMONDS. In addition they are needed for correlation maps jointly with the logLikelihood.txt file, and they are important to check for the sampling evolution of the free parameters and to inspect problems related to a bad choice of the parameter boundaries in uniform prior distributions that causing premature interruptions of the computations, as discussed in Chapter 3. These files can be created by calling the writeParametersToFile() member function of the Results class, as shown below:

```
Results results(nestedSampler);
results.writeParametersToFile("parameter");
```

  where this time only the initial bit of the string (shown as parameter) is provided, since the number of the parameter is attached automatically and changes for each free parameter used. This numbering follows the exact order in which we defined the free parameters in the model and in the priors. In addition, these files are always produced even if the computation stops prematurely with one of the error types discussed in Chapter 3. The number of parameter values stored will correspond to the total number of nested iterations plus the number of live points used in the final iteration, which are merged to the nested sampled points to complete the final sampling of the parameter space.

- a 1-column format logLikelihood.txt file, containing all the values of the natural logarithm of the likelihood function, sorted in the same order as the computation has evolved, hence by increasing likelihood, and in the same number as in the parameter<parameter#>.txt files. This file is useful to construct correlation maps, jointly with pairs of parameter values provided in the files parameter<parameter#>.txt (see the previous item), as those shown by [7]. An example of correlation map that can be obtained[4] is shown in Figure 1.2. This output file can be created by calling the writeLogLikelihoodToFile() member function of the Results class, as shown below:

```
results.writeLogLikelihoodToFile("logLikelihood.txt");
```

  Likewise the parameter<parameter#>.txt file(s), this file is stored even if the computation stops prematurely with one of the error types discussed in Chapter 3.

- a 3-column format evidenceInformation.txt file, containing the total Skilling's Bayesian evidence in natural logarithm, the corresponding error bar derived through the information theory, and the information gain. This file is essential for performing Bayesian

[4] In the example given, the plot is obtained by means of IDL using the surface routine set plotting in a 3D space, and the plots routine to add sampling points following the gradient in likelihood.

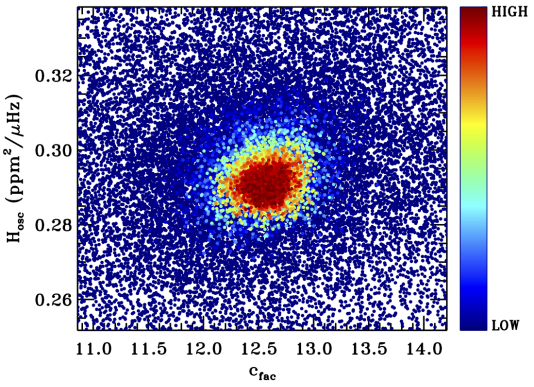


Figure 1.2: Example of correlation map that is obtained by combining the information contained in the file `logLikelihood.txt`, shown in a colored scale, with the sampling of a pair of free parameters stored in the files `parameter<parameter#>.txt`, here reporting the result obtained by [7] for the background fitting of a solar-like oscillator.

model comparison based on the Bayesian evidence information but can be ignored if we are only interested in the parameter estimation problem. It can be created by using the `writeEvidenceInformationToFile()` member function of the `Results` class, as shown below:

```
results.writeEvidenceInformationToFile("evidenceInformation.txt");
```

This file is stored even if the computation stops prematurely with one of the error types discussed in Chapter 3.

- a 1-column format `logEvidence.txt` file, containing the cumulated logarithm of the Bayesian evidence, useful to keep track of how the total evidence has evolved during the computation and to identify the bulk of the information gain, where the evidence saturates. The evolution of the Bayesian evidence as a function of the nested iterations can be essential to understand what stopping criterion is optimal for our analysis, namely it can be used to adjust the termination condition δ_{final} of the code. It can be generated using the `writeLogEvidenceToFile()` member function of the `Results` class, as shown below:

```
results.writeLogEvidenceToFile("logEvidence.txt");
```

- a 1-column format `logMeanLiveEvidence.txt` file, containing the average remaining logarithm of the Bayesian evidence as estimated from the set of live points at each nested iteration. This information can be used, coupled with that contained in the `logEvidence.txt` file, to monitor the quality of the sampling process and improve our choice of the termination condition δ_{final}. It can be generated using the `writeLogMeanLiveEvidenceToFile()` member function of the `Results` class, as shown below:

```
results.writeLogMeanEvidenceEvidenceToFile("logMeanLiveEvidence.txt");
```

- a 1-column format `posteriorDistribution.txt` file, containing the probability values coming from the computation of both likelihood and Bayesian evidence, and sorted according to the evolution of the nested sampling, hence by increasing likelihood as given in the `logLikelihood.txt` file. These values. which are in the same number as in the `logLikelihood.txt` file, can be used to construct the marginal probability distribution for each free parameter, as explained by [7], by using the complementary information contained in the corresponding `parameter<parameter#>.txt` file. The posterior probability values can be created by calling the `writePosteriorProbabilityToFile()` member function of the `Results` class, as shown below:

```
results.writePosteriorProbabilityToFile("posteriorDistribution.txt");
```

  This file is stored even if the computation stops prematurely with one of the error types discussed in Chapter 3.

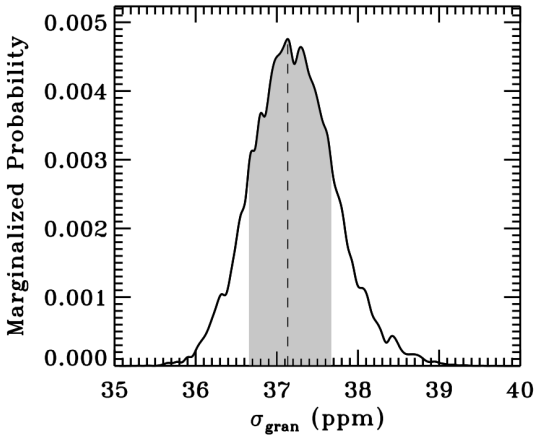


Figure 1.3: Example of a marginal probability distribution obtained by [7] using the information stored in a `marginalDistribution<parameter#>.txt` file for one free parameter of the background model of a solar-like oscillator. The shaded region represents the 68.3% Bayesian credible region computed by the `Results` class.

- a 7-column format `parameterSummary.txt` file, including all the Bayesian estimates computed from the marginal probability distributions (namely mean, median and mode), the Bayesian credible limits, the variance and the skewness of the distribution for each free parameter in the same order as that defined in the model and in the priors. This file is certainly the most important one in the context of Bayesian parameter estimation because it contains the desired estimates of the free parameters of the model. This information can be created by calling the `writeParametersSummaryToFile()` member function of the `Results` class, in the way shown below:

```
double credibleLevel = 68.3;
bool writeMarginalDistributionToFile = true;
results.writeParametersSummaryToFile("parameterSummary.txt",
credibleLevel, writeMarginalDistributionToFile);
```

in which we specified a 68.3 % probability of the Bayesian credible intervals (formally equivalent to the frequentist 1-σ error bar), and we required to also compute the marginal probability distributions through the flag `writeMarginalDistributionToFile` (see next item). If an assertion failure occurs, as explained in Section 3.1, this file will not produced.

- a 2-column format `marginalDistribution<parameter#>.txt` file, for as many free parameters as the model and priors have included, and labeled with the same numbering used by the `parameter<parameter#>.txt` files. These files contain the final marginal probability distribution (MPD), hence the parameter values and the corresponding marginal probabilities, sorted in increasing parameter values. The number of values listed is not corresponding to that given in the previous files, since for their computation the algorithm performs a sorting and a re-binning of the initial sampling. These files are needed when plotting the final results and check their reliability, as it will be discussed in Section 4. An example of MPD from the application done by [7] is shown in Figure 1.3. These output files are created only if specified in the `writeParametersSummaryToFile()` member function of the `Results` class, as described in the previous item. If an assertion failure occurs, as explained in Section 3.1, one or more of these files, depending on which free parameter caused the process to stop, will not be produced.

All the ASCII files will be stored in the output folder of the run, which is specified as an input string in the `run()` member function of the `NestedSampler` class, as shown in Section 1.7.

2 The enlargement fraction f of the sampling ellipsoid

To sample the parameter space in an efficient manner, the nested sampling algorithm implemented in DIAMONDS uses an ellipsoidal sampler, which has been developed based on the existing pioneering works by [16, 17, 3, 4]. The basic concept of the ellipsoidal sampler was already explained by [7], but we briefly recall it in this chapter to help with the reading of the documentation here provided. The procedure to draw a new point from the parameter space is as follows:

1. The set of live points at a given nested iteration is processed by the X-means clustering algorithm. This identifies which and how many clusters the live points can be split into.

2. For each cluster identified, a multidimensional ellipsoid is computed, with a number of dimensions imposed by the number of free parameters of the inference problem [1]. The volume contained within each ellipsoid is the effective volume from which the sampling takes place, therefore approximating the volume of parameter space where all the live points are situated.

3. An ellipsoid among those so constructed is randomly chosen from the set (following a probability criterion based on the volume of the ellipsoid), and a new sampling point is drawn from its volume with an exact algorithm [17].

4. The likelihood of the drawn point is evaluated, in order to verify whether it satisfies the hard likelihood constraint imposed by the nested sampling, namely that the new point must correspond to a likelihood value that is at least better than the lowest among the current set of live points.

5. If the hard constraint in likelihood is satisfied, then the point is accepted and added to the remainder sample of points identified during the nested sampling process, thus moving to the next nested iteration. If not, the drawing process is repeated from step #3 onwards, until a new point satisfying the likelihood constraint is found.

Figure 2.1 shows an example of this process at a fixed nested iteration, where we find a set of live points in three dimensions that is distributed by forming two distinct groups, identified as two different clusters by X-means. The 3D ellipsoids overlaid represent the geometrical construction from the covariance matrices obtained from the individual clusters. As it appears evident, the two three-dimensional shapes well surround each cluster of points, thus allowing us to improve the efficiency in drawing a new point. This is because the volume we are considering for the drawing process is no longer the entire parameter space, in this case a box in three

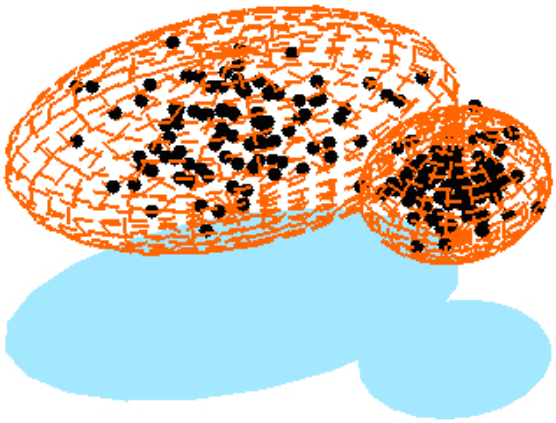


Figure 2.1: An example of two clusters of live points (black dots) in three dimensions for which two distinct ellipsoids are constructed. As visible from the figure, the volume of the ellipsoids (the 3D surfaces shown with red edges) nearly contains all the live points considered, and therefore represents a good approximation of the current distribution of sampling points in the parameter space. This allows drawing a new point very efficiently.

dimensions, but only the small portion contained in each of the ellipsoids that is placed inside this space.

Nonetheless, in real applications a simple ellipsoid directly obtained from the covariance matrix does not provide an adequate approximation of the volume in which the live points lie, as discussed by [16, 17]. The reason is that such ellipsoid would be too small to contain all or at least most of the points in the cluster, hence we need to enlarge its size to avoid losing important regions of parameter space to sample from. For this reason, [16] introduced the so called enlargement of the ellipsoids, an extra parameter that allows the default size of the ellipsoids to be adjusted in order to improve the overall sampling efficiency. Following studies done by [3, 4] showed that this enlargement can also be made dynamic, meaning that it can change through the evolution of the nested sampling in order to additionally improve the efficiency of the method. At this stage we need to understand how the code intervenes on the size and the dynamical evolution of the ellipsoids. We recall the definition of the *dynamical enlargement* f of the ellipsoids (equivalently the total enlargement fraction, or simply the enlargement fraction), provided by [3] and then also adopted in DIAMONDS (see [7]). For a given nested iteration and for a single ellipsoid we have:

$$f\left(f_0, \alpha, N_{\text{live}}, n\right) = f_0 X^{\alpha} \sqrt{\frac{N_{\text{live}}}{n}} \tag{2.1}$$

where we made explicit its dependence upon the parameters f_0, the initial enlargement fraction, the shrinking rate α of the remaining prior volume X, the number of live (or active) points N_{live} used in the nested sampling, and the total number of live points belonging to the ellipsoid considered, n, as extracted by the clustering algorithm. The parameter n in particular depends in turn on N_{live} and N_{clust}, as we will see in detail in Section 2.6. These represent the parameters that need to be configured and that are given as input to the code.

Equation 2.1 is crucial for understanding how f works, and especially how f_0 and α should be tuned by the user, so it is important to have it clear in mind for the remainder of this user guide. The enlargement fraction is applied to the length of each principle axis[1] (or principle

[1] In a bi-dimensional case, the principle (or principal) axes of an ellipse are certain lines that correspond to

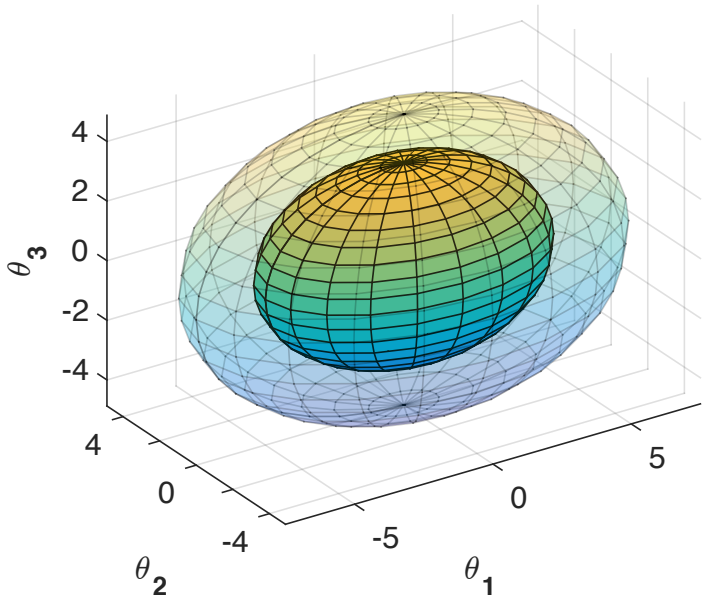


Figure 2.2: An example of a 3D ellipsoid, with initial size shown by the shaded surface in the center of the plot, to which an enlargement fraction of $f = 0.5$ was applied. The enlarged ellipsoid, shown as the transparent shaded surface around the default one, has semi-axes enlarged by 50% with respect to the original length. The three coordinates $\theta_1, \theta_2, \theta_3$ represent three arbitrary free parameters whose prior PDFs define the parameter space in which the sampling takes place.

semi-axis) of the ellipsoid according to $\lambda_{\text{enlarg}} = \lambda_0(1 + f)$, where λ_0 is the default length of a principle axis, and λ_{enlarg} is the length of the principle axis of the enlarged ellipsoid. In Figure 2.2 we show the effect produced by an enlargement fraction $f = 0.5$ on a 3D ellipsoid, meaning that we have extended the original axis length by 50% in each dimension.

To proceed we provide more insights about all the configuring parameters involved in the nested sampling based on the multi ellipsoidal sampler. In reference to the sample code shown in Section 1.7, for most of the applications we can consider that the number of live points, represented by the integer variable `initialNlivePoints`, is not subject to any reduction law of the live points, as those presented by [4] and [7], meaning that we can adopt the condition `minNlivePoints` = `initialNlivePoints`. In addition, from now on we will consider that the variables

- `maxNdrawAttempts` (M_{attempts})
- `NinitialIterationsWithoutClustering` (M_{init})
- `NiterationsWithSameClustering` (M_{same})

are set to the values $M_{\text{attempts}} = 5 \times 10^4$, $M_{\text{init}} = N_{\text{live}}$, $M_{\text{same}} = 50$, following the indications obtained from the calibration performed in Section 2.4.1 and provided by means of the testing done by [7]. These configuring parameters will not be further discussed because either they are known to have a fixed value or their tuning is not representing a crucial point for the success of the computation. Some considerations on the choice of the variable `terminationFactor` (δ_{final}) will be discussed in Section 2.4.1.

the major and minor axes of the ellipse. The principle axes are always orthogonal.

2.1 THE NUMBER OF CLUSTERS $N_{\rm clust}$

This number, given to vary within a range of possible integers, as shown in Section 1.6, is not explicitly appearing in Equation 2.1 but it has a direct impact on the number of live points falling within each ellipsoid, which we indicated as n (see Section 2.6 for more details). As already discussed by [7], and then additionally tested by [8], setting $1 \leq N_{\rm clust} \leq 10$ offers a good compromise between efficiency of the sampling of the parameter space for complex shapes of the likelihood distribution (e.g. pronounced degeneracies, multi-modality, etc.) and the computational time, in a wide variety of applications. This range can of course change depending on the specific problem, although a maximum number of clusters larger than 10 is usually not recommended. When it is possible, e.g. for problems in which no strong degeneracies are expected or the solution is uni-modal, it is useful to reduce the maximum number allowed down to e.g. 6 or even 4. This is because the more clusters are computed the more the nested sampling slows down, causing the cluster algorithm to become the bottleneck of the whole process in cases where the number of dimensions of the inference problem grows up considerably (e.g. > 50). Sometimes allowing for a large number of clusters is not necessarily improving the results and therefore it is advisable to avoid stretching the upper bound of $N_{\rm clust}$ beyond what necessary.

Given the problem in exam is fixed, it is suggested not to change the range of $N_{\rm clust}$ because it may have a strong impact on Equation 2.1, hence on the other configuring parameters of DIAMONDS, as we shall discuss thoroughly in Section 2.6. In the remainder part of the guide we will assume that this range is constant, unless specified otherwise.

2.2 THE NUMBER OF LIVE POINTS $N_{\rm live}$

This number is represented by the integer variable `initialNlivePoints` used in the sample code shown in Section 1.7, and it is subject to change depending on the complexity of the likelihood distribution in exam. When the problem at hand is the same in multiple analyses (e.g. the fitting of the stellar oscillations in a sample of solar-like stars, see also Section 2.4), there is no reason to change $N_{\rm live}$ since we may expect the likelihood distribution to remain approximately of the same type. Some values of $N_{\rm live}$ are provided by the authors in [7, 9, 8, 10, 11]. Numbers of live points ranging from 500 to 2000 should guarantee an optimal sampling for a variety of applications spanning from uni-modal distributions, even with degeneracies and in dimensions from a few up to about one hundred, to highly multi-modal distributions up to at least $\sim$20 different modes in $\sim$10 dimensions, as demonstrated by [7], Section 6.7. The tuning of $N_{\rm live}$ is in general not particularly difficult and critical since even by changing its value by several hundred units the other configuring parameters are not significantly affected. This aspect is further discussed in Section 2.6, which we recommend to read.

As a general advise, the number of live points can be increased if for example:

- A large (and/or high-precision) dataset is used (e.g. NASA's *Kepler* datasets spanning more than one year or solar time-series obtained by GOLF and VIRGO), which will cause the maxima of the likelihood distribution to become highly localized (a condition usually termed as *strong data*, see also [20]), hence requiring a finer sampling to reconstruct them properly. In this case up to 1000 live points can be recommended.

- More complex likelihood distributions (e.g. with multiple modes or pronounced degeneracies or both of them) are expected. To guarantee a better sampling of the posterior distribution at each nested iteration and be able to detect all (or most of) the modes

from the early stages, a number of 2000 live points could be needed (e.g. see [8], Section 6.7).

- The number of dimensions in the problem is subject to vary up to two orders of magnitude (from ∼1 to ∼100 dimensions). In this situation having more live points guarantees the initial sampling of the parameter space to be enough dense to detect the different maxima of the likelihood distribution even if the considered prior volume is increasing. Depending on the complexity of the fitting, values between 500 and 2000 can prove adequate.

A number of live points larger than those mentioned here will likely slow down the computation without a significant gain in accuracy for both the sampling of the local maxima and the evaluation of the final Bayesian evidence, as we shall see more in detail below. Conversely, if N_{live} is too low, we will not be able to retrieve good results because the chances to miss the modes of the likelihood distribution increase (we do not have enough live points to sample adequately all the parameter space and understand which regions should be sampled more than others). In the worst case it may also happen that a computational failure is produced, of the types discussed in Sections 3.2 and 3.3, since the initial sampling provided is too sparse to allow the algorithm to find better likelihood points at later stages of the process.

Finally we remind that using a different N_{live} may have an impact on the estimation of the final Bayesian evidence because the number of nested iterations changes and consequently also the total number of points sampling the posterior distribution (see also [15] for more details). This suggests that when performing a Bayesian model comparison, the different computations should be carried out by adopting exactly the same number of live points in order to be able to compare models for which a consistent sampling method and sampling density has been used. Based on our discussion and given that the type of Bayesian inference problem is fixed, we shall keep N_{live} constant in the different computations, even if the datasets are being changed[2].

2.3 The shrinking rate α

The parameter `shrinkingRate` of the ellipsoidal sampler represents a sensible information for the ellipsoids because it controls the dynamics of the enlargement fraction during the evolution of the nested sampling. It is usually a number so that $0 \leq \alpha \leq 1$, but can in principle be also set to $\alpha < 0$ if for particular reasons we want to compensate the natural shrinking of the ellipsoids by enlarging them as the prior mass decreases. For all the real applications in which DIAMONDS was used, we found α lying in the range $0.00 \leq \alpha \leq 0.04$ only, where steps of 0.01 were adopted to tune its value. In the analyses done up to now, we did not find α correlating with any other parameter such as the number of dimensions or the number of clusters involved, hence also with f_0 itself. However, it is important to note that even a variation of 0.01 could lead to significant changes both in the computational speed, especially for stages approaching the termination condition, and in the resulting marginal distributions of the inferred parameters, as will be discussed in Chapter 4. This parameter has a strong impact on the evolution of the sampling process and therefore it should be chosen with care.

[2]We are assuming that the datasets in use roughly remain of the same type, meaning that for example we could analyze one star by using a time-series spanning one year of observation, and then move to another star with the same or at least similar length of observation as the previous one, although the information contained is clearly different. In situations where we decide to substantially change the features of the dataset in use, e.g. by adopting a new dataset with one order of magnitude more data bins then the first one, then attention has to be paid on what number of live points should be used, and possibly increase it following the suggestions given in this section.

Each kind of application has in principle its optimal value of shrinking rate. A useful guideline is that if this parameter is too large, meaning that the ellipsoids shrink very quickly, there is a high risk to lose sensible information by sampling too small regions of the posterior distribution by the time the nested sampling reaches the termination condition. Conversely, if α is too small, meaning that the ellipsoids shrink slowly or even stop to shrink, the parameter α may cause a significant drop both in speed and sampling efficiency because we are not optimizing anymore for the prior volume of the sampling, especially during the final nested iterations where more focus on the likelihood maxima could be required[3]. In the worst case, this could lead to a premature interruption of the process, as it will be discussed in Chapter 3 in more in detail. For more discussion on the effect of α coupled with that of f_0 we recommended reading Section 4.1. For the peak bagging analysis in particular, we find that a value of $\alpha = 0$ is the one to be preferred to ensure that the resulting marginal distributions do not lead to an underestimation of the errors on the inferred parameters. This implies that the prior mass has no effect on the dynamical enlargement because Equation 2.1 reduces to the simpler *static* version

$$f\left(f_0, N_{\text{live}}, n\right) = f_0 \sqrt{\frac{N_{\text{live}}}{n}}\,, \tag{2.2}$$

which we will consider as a reference for the peak bagging applications with DIAMONDS. Such a condition will nonetheless imply that the ellipsoids shrink as the nested sampling process proceeds toward the termination condition, but that the shrinking is only that naturally produced by the more localized position of the live points as we move to higher values of the likelihood distribution.

2.4 THE INITIAL ENLARGEMENT FRACTION f_0

Specified as the variable `initialEnlargementFraction` in the sample code shown in Section 1.7, this is certainly the most important and most frequently tuned configuring parameter of the ellipsoidal sampler. The reason is that it allows us to have an immediate control on the size of the ellipsoids. According to Equation 2.1 this is done by changing each principle axis length of the ellipsoid by the same fraction, as already shown at the beginning of this chapter. We therefore have that $f_0 \geq 0$, where $f_0 = 0$ means that no enlargement is applied (but never $f_0 < 0$, as this may lead to a negative axis length!). Typical values of this parameter stay on the order of the units and do not change significantly (they will stay within the same order of magnitude as the number of dimensions changes even by more than one order of magnitude). The analyses published by [7, 9, 8, 10, 11, 12] show a variety of applications by providing the related values of f_0.

There is an optimal f_0 for each problem considered. On the one side if f_0 is large, the sampling efficiency decreases and the computation slows down. If f_0 is too large, then the computation typically fails. This is because very big ellipsoids may happen to be larger than the parameter space itself, which is pointless, or because they are still too large as compared to the region containing the likelihood maximum (or maxima), hence disrupting any possible improvement that can be obtained by reducing the useful prior volume to draw from (see [7] and Chapter 3 for a detailed discussion on the computational failures). On the other side, if f_0 is small, the computation is fast but sensible information may be lost during the sampling, similarly to the effect produced by a too large α. This is because the default size of the ellipsoids (without enlargement) is not enough to properly include all or at least most of the live points it is originating from. As a result, the final sampling will likely yield an underestimation of the

[3] See also the explanation presented at the beginning of this chapter, and in Figures 2.1 and 2.2.

Bayesian errors or, in more extreme cases, wrong parameter estimates and Bayesian evidences because too small ellipsoids may fail in sampling the regions of the posterior distribution that contain most or significant portions of the likelihood distribution. A way to obtain an optimal choice of f_0 is thoroughly discussed in Section 2.4.1 using the application of the peak bagging analysis. When setting the optimal value for f_0 we usually deal with three different scenarios:

1. We fix the model, hence the number of free parameters (the dimensions of the problem), and perform a Bayesian inference on different datasets. In this case there is no particular reason to vary f_0 since we expect the problem to behave in a similar way each time. However, Section 2.6 discusses a particular case in which a significant change in the actual number of clusters n may cause f_0 to be readjusted.

2. We have a single dataset but this time we use different models, with a different number of free parameters involved. For this situation we need to tune f_0 as a function of the number of dimensions by using steps of the order of 0.1 or 0.05 each time. This case is important as it represents the condition for performing model comparison by means of the Bayesian evidence.

3. We have a set of different datasets and a different model for each dataset considered. An example in this regard is given by the peak bagging analysis performed on a chunk of power spectrum. Assuming we want to fit a given number of oscillation modes, we thus select a chunk of power spectrum (i.e. a dataset) covering the frequency range of those oscillation modes. If we want to fit more modes (e.g. outside the original frequency range), we can enlarge the frequency range considered, hence fit a more complex model including the new oscillation modes. In this way we increase the number of free parameters and at the same time the number of data bins. This case requires a tuning of f_0 as a function of the number of dimensions in the same way as for scenario # 2.

Figure 2.3 shows the evolution of f_0 as a function of the number of dimensions involved in the inference problem, k, for a set of 37 independent computations of a peak bagging analysis performed on the red giant branch star KIC 12008916. For more details on the calibration process, we refer the reader to Section 2.4.1. By considering all the individual measurements, a sub-linear proportionality between f_0 and k appears evident for both regimes investigated, one using 500 and the other 1000 live points. By fitting a simple power law relation, we obtain

$$f_0^{(500)} = 0.369 \cdot k^{0.574}\,, \tag{2.3}$$

$$f_0^{(1000)} = 0.310 \cdot k^{0.598}\,. \tag{2.4}$$

The exponent to the number of dimensions is roughly equal to 3/5 for both regimes, meaning that at least for this application $f_0 \sim \mathcal{O}(k^{3/5})$, which is similar to the dependency used by [8]. In particular, the relation provided here represents an updated version of that published by [**?**]. This relation, which is ensured to hold in the range of dimensions shown in the plot (more safely $0 \leq k \leq 40$), can be very useful if adopted as a guideline to tune f_0 with a varying number of dimensions[4]. The relation for $N_{\rm live} = 500$ was also tested on another star, the red clump KIC 5112373, up to $k = 54$ and providing successful results.

[4]This relation is not holding if the number of clusters is different than $N_{\rm clust}$ = 3-4. A number of $N_{\rm clust} = 4$, in a range from 1 to 20, was found to be the optimal compromise between sampling speed and efficiency during the testing phase. We therefore encourage the user to perform its own calibration, or at least test the one provided here, for analyses that may require a number of clusters significantly different than 4.

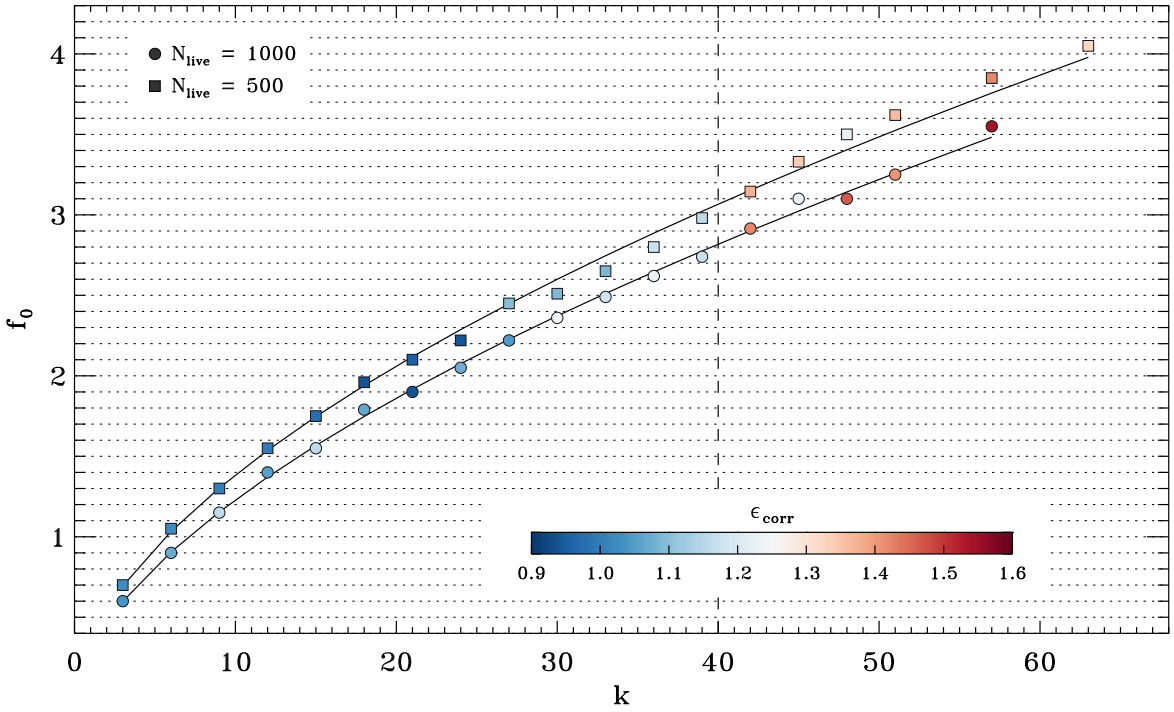


Figure 2.3: The initial enlargement fraction f_0 as a function of the number of dimensions involved in the inference problem, k. The 37 independent computations are performed using the peak bagging analysis [8] on the test star KIC 12008916 and a number of live points of $N_{\rm live} = 500$ (squares) and $N_{\rm live} = 1000$ (circles). The color of the symbols indicates the ratio $\epsilon_{\rm corr}$ between the theoretical, $\sigma_{\rm predicted}$ [13] and measured error, $\sigma_{\rm measured}$ (from the Bayesian inference performed with DIAMONDS) on the $\ell = 0$ mode frequencies involved in the fit, with $\epsilon_{\rm corr} = 1$ representing the optimal condition. The dashed vertical line delimits a transition to a high-dimensional regime in which the calibration of f_0 becomes more critical and the measured errors become underestimated with respect to the prediction by more than 30%.

If the number of clusters varies from one computation to another, one could apply the correction to the proportionality term discussed in Section 2.6 and given by Equation 2.15. However, we cannot guarantee that the correction factor proposed works in all cases because actual computations may differ from the average expectations for a number of reasons, among which the way the sampling itself takes place at each nested iteration[5] and the nature of the inference problem in hand.

From our calibration of f_0 for two different regimes of live points we notice that when $N_{\rm live}$ varies but $N_{\rm clust}$ is kept constant, the dynamical enlargement is not expected to change, at least on average. This follows from Equation 2.2 itself since the number of live points within each ellipsoid is on average expressed as $N_{\rm live}/N_{\rm clust}$ (see the discussion in Section 2.6). We however note that from our calibration f_0 is systematically larger for $N_{\rm live} = 500$ than that used for $N_{\rm live} = 1000$ (see Figure 2.3) by about 10 %. This relative difference is not a function of the number of dimensions and therefore it has to be related to the default size of the ellipsoids (their intrinsic size as deriving from the covariance matrix). As a result, the default size of the ellipsoids appears to increase with decreasing number of live points, although the effect is

[5]The reader should not forget that the nested sampling is a Monte Carlo algorithm and that, as such, it is subject to change at every different realization.

relatively small (only 10 % change with a 200 % change in $N_{\rm live}$). This could be originating because a smaller number of live points yields a sparser sampling in the parameter space, hence corresponding clusters tend to be larger (less localized), and so the associated ellipsoids, than if more live points are being used. Such condition is more likely to occur where the local density of live points (i.e. the number of live points over the volume of parameter space) is lower.

2.4.1 Calibrating the relation f_0-k from the peak bagging analysis

In this section we discuss the procedure used to calibrate the relation f_0-k presented in Figure 2.3. The number of clusters involved in all the fits used for the calibration was allowed to vary in the range $3 \leq N_{\rm clust} \leq 4$, and resulted to be $N_{\rm clust} = 4$ all times. This choice was done to maintain the quantity $\sqrt{N_{\rm live}/n}$ in Equation 2.2 nearly constant throughout the computation, hence stabilize the dynamical enlargement of the ellipsoids for each fit. It is always advisable to keep the number of clusters to a nearly constant value throughout the computation, because if n changes significantly this is likely to have a negative impact on the sampling efficiency and on the reliability of the results (see Section 2.6 for more insights on the effect of a varying n). In addition, the shrinking rate was set to $\alpha = 0.0$ meaning that the dynamical enlargement does not depend on the remaining prior mass (see Equation 2.2), as already anticipated in Section 2.3.

The starting point is to consider a low-dimensional problem for which it is possible to compute the Bayesian evidence numerically (with a direct calculation). In the case of the peak bagging analysis this can be done by the fitting of a single oscillation mode[6], modeled by a Lorentzian profile that requires three parameters for the inference. The Lorentzian profile adopted, as a function of the cyclic frequency ν (expressed in μHz) in the power spectrum of a star, is of the form [7]

$$P(\nu; \nu_0, A_0, \Gamma_0) = \frac{A_0^2/(\pi\Gamma_0)}{1 + 4\left(\frac{\nu-\nu_0}{\Gamma_0}\right)^2} + \bar{B}(\nu) \tag{2.5}$$

with ν_0 the frequency centroid (in μHz), A_0 the oscillation mode amplitude (in ppm), and Γ_0 the linewidth (in μHz). The function $\bar{B}(\nu)$ is a background level that is estimated in a previous phase and that is kept fixed for the fitting of the Lorentzian profile (see [7, 8, 12] for more details on the analysis process). The likelihood to be used is the exponential likelihood that is already implemented in the basic package of DIAMONDS [7], which in logarithmic scale reads as

$$\Lambda(\boldsymbol{\theta}) = -\sum_{i=1}^{N_{\rm bins}} \left[\ln E_i(\boldsymbol{\theta}) + \frac{O_i}{E_i(\boldsymbol{\theta})}\right] \tag{2.6}$$

with E_i the predictions from the Lorentzian profile, O_i the observed power spectrum (in our case a power spectral density in units ppm$^2/\mu$Hz) with a number of $N_{\rm bins}$ data bins, and $\boldsymbol{\theta} = (\nu_0, A_0, \Gamma_0)$ the parameter vector. The numerical calculation is done by means of a grid-based approach, in which each parameter is discretized into a set of values (100 points for each coordinate, regularly spaced) and the likelihood is evaluated for each point in the grid (a total of 10^6 points). The Bayesian evidence is then computed following the approach presented by [6], assuming uniform priors for each free parameter and integrating over the

[6] To avoid confusion, in this section we shall refer to *mode* or *oscillation mode* as the physical oscillation that occurs inside the star. This has not to be confused with the statistical concept of a mode of a probability distribution.

three dimensions considered. In this way we obtain a value for the natural logarithm of the Bayesian evidence from the grid-based approach, that is $\ln Z_{\rm grid} = -1885.0$ and marked in the left panel of Figure 2.4. We checked that this value was not changing even by increasing the number of points in the grid by one order of magnitude along each coordinate (from 100 to 1000 points), so that we can consider it as very accurate. We also computed the 68 % Bayesian credible intervals from the marginal probability density functions obtained with the grid-based approach [6], and we show the resulting density functions in Figure 2.5 with solid black lines.

As an additional comparative factor, we consider the limit error on the frequency centroid ν_0 of a Lorentzian profile, as estimated from theory [13], whose variance can be calculated as

$$\sigma^2_{\nu_0} = \frac{\Gamma_0}{4\pi T}\eta_0(\beta) \tag{2.7}$$

where T is the total observing time of the original time-series, and with

$$\eta_0(\beta) = \sqrt{\beta+1}\left(\sqrt{\beta+1}+\sqrt{\beta}\right)^3 \tag{2.8}$$

a function depending on the inverse of the signal to noise ratio of the fitted oscillation mode, $\beta \equiv \mathrm{SNR}^{-1}$. The function η_0 has this simple form because it is related to an $\ell = 0$ (radial) oscillation mode. Clearly, the oscillation mode we are fitting in this calibration process is a real $\ell = 0$ mode of the star. The parameter β can be easily computed by evaluating the ratio B_0/H_0, between the level of the background in the position of the centroid ν_0, $B_0 \equiv \bar{B}(\nu_0)$, and the oscillation mode height, $H_0 = A_0^2/(\pi\Gamma_0)$. For clarity, from here onwards we shall refer to the theoretical error on the frequency centroid as $\sigma_{\rm predicted}$. As we will see below, this parameter is crucial for assessing the size of the ellipsoids during the nested sampling when the number of dimensions increases and no more numerical solutions are available.

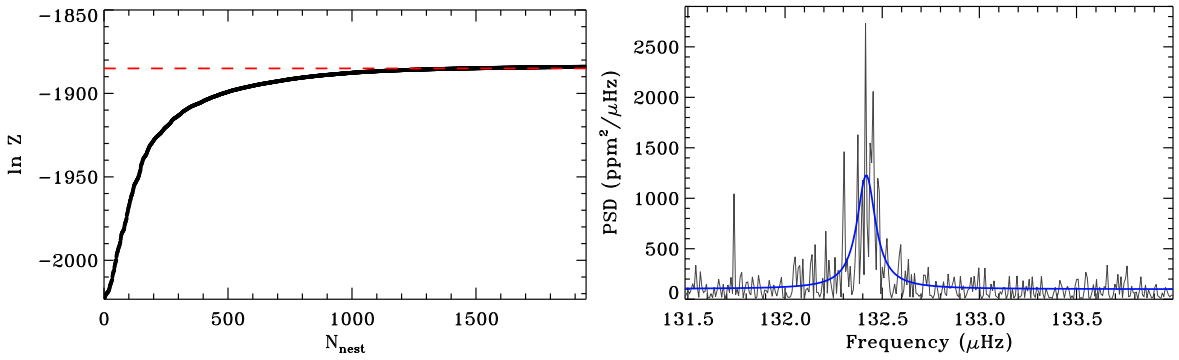


Figure 2.4: The left panel shows the evolution of the natural logarithm of the Bayesian evidence, $\ln Z$ (solid black curve), as a function of the nested iteration during the fitting of a Lorentzian profile to an oscillation mode, as obtained by DIAMONDS. The horizontal dashed red line marks the Bayesian evidence computed with the grid-based approach. The right panel shows a chunk of power spectral density for KIC 12008916 (gray) with overlaid the resulting Lorentzian fit of the oscillation mode done with DIAMONDS (solid blue line) using the median estimators for each free parameter.

Provided that we now have two crucial comparative terms at hand, namely the results from the grid-based approach and the theoretical error on the frequency centroid of the $\ell = 0$ oscillation mode, using DIAMONDS we perform the same Bayesian inference to the same dataset used in the grid-based approach. In this way we are able to tune the configuring parameters f_0, α, $\delta_{\rm final}$, $N_{\rm clust}$ and $N_{\rm live}$ to reproduce at best the already known results. As

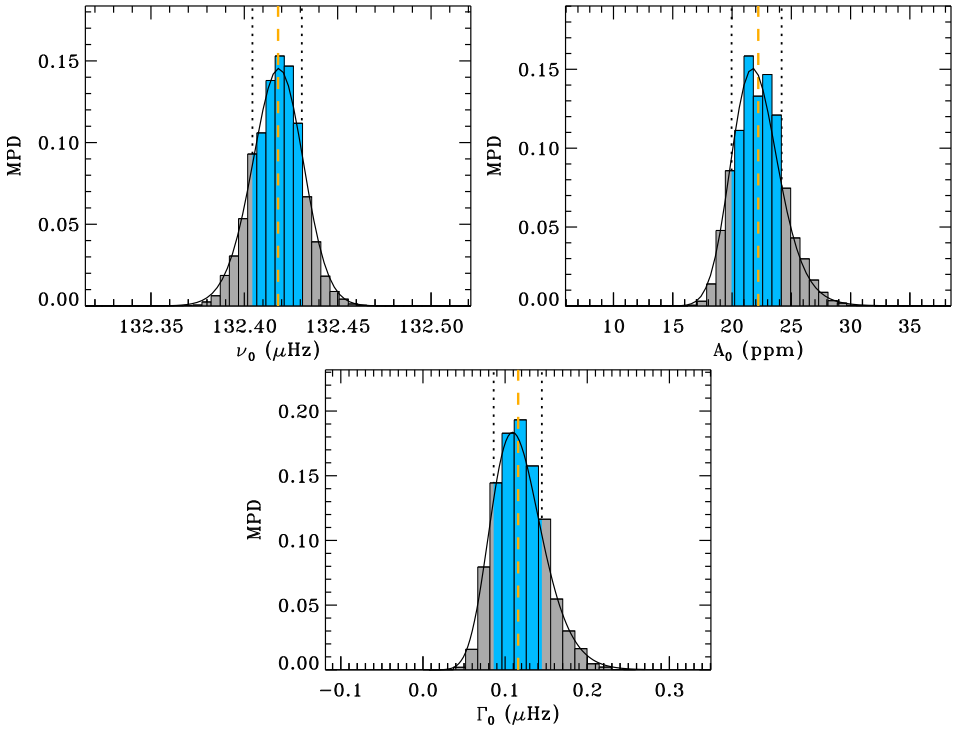


Figure 2.5: Marginal probability distributions for the free parameters defining the Lorentzian profile of an oscillation mode. The histogram shows the MPD obtained by DIAMONDS, while the solid black curve is the marginal probability density function obtained from the grid-based approach and rescaled for comparison to the height of the MPD. The blue regions mark the 68 % Bayesian credible intervals (delimited by the vertical dotted line). The median estimates for each parameter are shown with a vertical dashed orange line.

anticipated at the beginning of this section, in this calibration phase we find that with $3 \leq N_{\rm clust} \leq 4$ and with both $N_{\rm live} = 500$ and 1000 we obtain a successful computation and comparison. In addition, we find that a value of $\delta_{\rm final} = 1.0$ as termination condition represents the best compromise between density of the sampling in the parameter space and the limiting size of the ellipsoids to produce MPDs that match the expected errors from theory[7].

By considering the case of $N_{\rm live} = 500$ for reference, for the three-dimensional problem presented above we find that the an optimal combination of configuring parameters is given by $f_0 = 0.7$, $\alpha = 0.0$. The total natural logarithm of the Bayesian evidence is thus $\ln Z_{\rm DIAMONDS} = -1883.4$. The Bayesian evidence obtained with DIAMONDS therefore agrees on a 0.1 % level in logarithmic scale with that obtained from the grid-based approach. In addition the ratio between the predicted and the measured error on the frequency centroid, which we define for simplicity as the correction factor $\epsilon_{\rm corr} \equiv \sigma_{\rm predicted}/\sigma_{\rm measured}$, is 0.955, so very close to unity[8]. The measured error $\sigma_{\rm measured}$ is given as the geometric average of the left and right

[7]On the one side, a value of $\delta_{\rm final} = 5.0$ has also been tested thoroughly and found to yield MPDs with a poorer sampling, thus possibly affecting the computation of the Bayesian credible intervals. On the other side, a value of $\delta_{\rm final} = 0.01$ as that used by [8, 12], yields to an excessive shrinkage of the ellipsoids and is therefore more likely to produce underestimated Bayesian errors on the inferred parameters. This is because as the nested iterations proceed, the ellipsoids naturally reduce in size due to the localization of the sampling around the maxima of the likelihood distribution.

[8]We note that by repeating the computation multiple times, this value could be subject to a variation within ±10 %, so a factor ranging from 0.9 to 1.1 can be considered an optimal condition. Variations to this term are

68 % Bayesian credible intervals computed with DIAMONDS and corresponds to the classical standard deviation in the case the MPD is a symmetric Gaussian. As shown in Figure 2.5 the overlap between the MPDs obtained with DIAMONDS and the marginal probability density functions computed with the grid-based approach is striking[9]. This is also reflected in the agreement of the median estimates from both sets, which turns out to be 0.0002 % for ν_0, 0.86 % for A_0 and 0.87 % for Γ_0. For a better visualization, when plotting MPDs obtained by DIAMONDS, we advise re-binning the content of the output file by adopting the Scott's rule for normal distributions, giving a bin-size of

$$\Delta\theta = 3.5\sigma \left(N_{\text{points}}\right)^{-\frac{1}{3}} \tag{2.9}$$

where σ is the standard deviation of the marginal probability distribution[10] and N_{points} is the number of points in the output distribution of DIAMONDS, namely that contained in the `marginalDistribution<parameter#>.txt` file. Figure 2.4 also shows the resulting fit obtained with DIAMONDS and the evolution of the Bayesian evidence, which reaches the expected value already after 1300 nested iterations. Interestingly, while DIAMONDS requires just a *one* second computation to obtain the results, the grid-based approach takes about 14 seconds to complete using the same computational resources.

The second stage of the calibration requires that we extend the fitting done with DIAMONDS to a range of dimensions as large as possible. In order to obtain a reliable calibration, we require a comparative term that can be computed for any number of dimensions investigated. Since there is no possibility to compute a Bayesian evidence with a direct numerical approach as done for $k = 3$, we rely on the computation of the theoretical error on the frequency of the $\ell = 0$ modes. We therefore compare this error with that measured by DIAMONDS for the same modes, for as many $\ell = 0$ modes as included in the fit. Increasing the number of dimensions in the peak bagging process is easily done by including more oscillation modes to be fit, hence more Lorentzian profiles, and enlarging the dataset considered by incorporating more data bins. Our calibration is sampled at steps of $k = 3$ dimensions because we can include not less than one Lorentzian profile each time. For each computation we thus tune f_0 such that it gives $\epsilon_{\text{corr}} \simeq 1$. The result is shown in Figure 2.3 for both sets using $N_{\text{live}} = 500$ and 1000. As visible from the figure, the tuning of f_0 becomes more critical for $k > 40$, where we have difficulties in obtaining ellipsoids that are sufficiently large to avoid an underestimation of the Bayesian errors and at the same time be able to finalize the computation without failures. In the worst case we could reach down to $\epsilon_{\text{corr}} \simeq 1.5$ for $k = 57$ and $N_{\text{live}} = 1000$. For the time being, we deem it as a limitation of the ellipsoidal sampler itself, and future releases of DIAMONDS will also aim at improving this aspect by possibly replacing the ellipsoidal sampler

due to the different Monte Carlo realizations that take place each time we run a new nested sampling process.

[9] We note that the MPD obtained with DIAMONDS is not the same object as the marginal probability density function. In the first case, we have direct probabilities, while in the second case we have probability densities, $p = \rho d\theta$, which have to be integrated over the parameter bins. However, since $d\theta$ is constant throughout the range in the grid-based approach, probabilities and probability densities are directly correlated, with $\rho \propto p$ for all the parameter values considered, and therefore the two distributions can be rescaled in height to match one another.

[10] The standard deviation is evaluated as the square root of the second moment. In the case of the marginal probability distribution provided by DIAMONDS, one has the second moment

$$\sigma^2 = \sum_{i=0}^{N_{\text{points}}} p_i \left(\theta_i - \bar{\theta}\right)^2 , \tag{2.10}$$

where p_i is the probability associated to the value θ_i of the parameter, and $\bar{\theta}$ is the mean of the parameter, computed from the same distribution.

with more efficient prior samplers in the high-dimensional regime. From the calibration and testing performed, it also appears that adopting $N_{\rm live} = 500$ yields an easier tuning of f_0, even for a wider range of dimensions (up to $k = 63$), with still a fairly acceptable $\epsilon_{\rm corr}$ value (within 1.3), without losing in sampling density. We therefore suggest the adoption of the power law in Equation 2.4 for $N_{\rm live} = 500$. We note that, if needed, $\epsilon_{\rm corr}$ can also be used *a posteriori* to correct the extent of the Bayesian errors in order to match the theoretical ones. This correction is however meaningful as long as $\epsilon_{\rm corr}$ stays within a value of 2-3, and that we do not need to rely on a very accurate computation of the Bayesian evidence.

2.5 COMPUTATIONAL TIMES $t_{\rm comp}$ AND $t_{\rm norm}$

In this section we show and discuss two important correlations that highlight the performances of the DIAMONDS code and that can be used for estimating expected computational times for different applications of the code, e.g. useful when preparing a proposal for the allocation of computational time or planning the purchase of some computational resources. For this purpose we used the values that were obtained from a set of 37 independent computations made for the peak bagging analysis in the calibration of the f_0-k relation (see Section 2.4.1). These computations used the representative red giant star KIC 12008916 observed by NASA's *Kepler* for more than four years, and a 2.7 GHz CPU. We recall that in the peak bagging analysis the model adopted is a mixture of Lorentzian profiles and that for each number of dimensions considered the dataset is subject to change because generally more data bins are being added when including more oscillation peaks in the model. This study represents an interesting benchmark to investigate and understand the impact of the number of dimensions on the efficiency and computational speed of DIAMONDS.

In the top panel of Figure 2.6 we see how the total computational time for each run with DIAMONDS, termed here $t_{\rm comp}$, is a clear function of the number of free parameters, or dimensions, involved in the inference problem. This dependency can be quite well reproduced by a power law to the number of dimensions with an exponent set to about 3.2, as shown by the relation

$$t_{\rm comp} = 0.0127 \cdot k^{3.1764} \ {\rm s}\,. \tag{2.11}$$

We note that the computational time measured at a given k as shown in the top panel of Figure 2.6 is that obtained from the average of two independent computations, one using $N_{\rm live} = 500$ and the other 1000. This choice was done because we noticed that by optimizing for the initial enlargement fraction f_0 to obtain a correct calibration for each value of live points, the computational time of the run is roughly similar in the two cases, proving that at least in this regime of variation for the number of live points, the total time required by the code to reach the termination condition is not influenced by our choice of $N_{\rm live}$[11]. We note that in the range k = 3-57 the total computational time changes by almost four orders of magnitude, showing that an increase of less than two orders of magnitude in the number of free parameters yields about a factor four increase in magnitude in the total computational time.

Another important aspect is related to the number of data bins, $N_{\rm bins}$, involved in the computation. This information is color-coded in the top panel of Figure 2.6 and shows that by increasing the number of dimensions also the number of data bins increases (about one

[11]This could be explained because by adopting an optimal calibration for each value of live points we are forcing the ellipsoids to maintain a similar size independently of $N_{\rm live}$, thus without changing the overall efficiency of the algorithm. Although a larger $N_{\rm live}$ implies a larger number of nested iterations, it is easier for the nested sampling to identify a new point satisfying the likelihood constraint when the number of live points to compare to is larger.

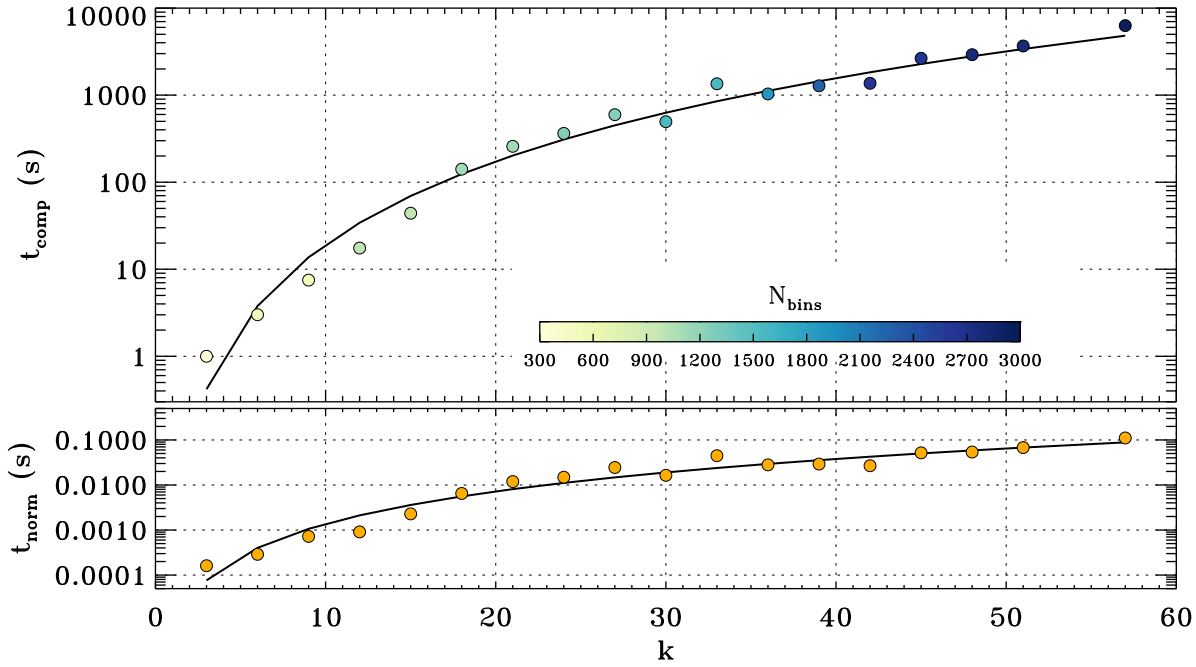


Figure 2.6: *Top panel*: the computational time $t_{\rm comp}$, where each point is an average between the run with $N_{\rm live} = 500$ and 1000, as a function of the number of dimensions involved in the inference problem, k, showing that DIAMONDS has $t_{\rm comp} \sim \mathcal{O}(k^{3.2})$. The different colors represent the number of data bins used in each computation (one value for each measurement point). The solid black line shows the power law fit to the measurements. *Bottom panel*: similar to the top panel but for the computational time normalized by the number of data bins, $N_{\rm bins}$, and by the maximum fractional prior range $\langle \Delta r_{\rm max} \rangle_{f_0\text{-}k} = 19.3978$, yielding $t_{\rm norm} \sim \mathcal{O}(k^{2.4})$. For a better visualization a logarithmic scale in the y-axis is adopted for both panels.

order of magnitude in the range $3 \leq k \leq 57$). The net result is an extra increase in the computational time as a function of k, which is not directly evident from $t_{\rm comp}$ alone. If we want to extrapolate the time it takes to the algorithm to complete a given process for a single data bin we clearly need to get rid of this extra dependency. To do so, we divide the total computational time $t_{\rm comp}$ by the number of data bins, $t_{\rm bin} = t_{\rm comp}/N_{\rm bins}$.

Nonetheless, removing the dependency on the number of data bins is not enough to extrapolate the fundamental time unit required by DIAMONDS to complete a fitting process. For this purpose, we need to consider the dependency of the computational time on the volume of the parameter space, which is still encompassed in $t_{\rm bin}$. The larger the volume of the parameter space set by our prior distributions the longer it will take for the algorithm to converge to a solution. This is because the density of live points, i.e. the number of live points divided by the prior volume, decreases, meaning that with the same number of live points we need to sample a larger volume of parameter space. Clearly, if less points are available to sample a wide region of space, then it becomes more difficult to find a new point that satisfies the likelihood condition imposed by the nested sampling (more drawing attempts from the prior distribution are necessary), hence the efficiency of the whole nested sampling becomes worse. To evaluate the dependency on the prior volume, and assuming uniform priors on all the free parameters for simplicity, one can compute the fractional variation in prior range with respect to each estimated parameter, Δr[12]. This means that, e.g. if we have a one dimensional problem and

[12]One could do the testing for choices of different prior distributions, but dedicated comparisons will depend

our uniform prior range on the free parameter changes from -10 % to 10 % of its estimated median value (or alternatively from -5 % to 15 % or any other combination that yields the same extent in prior range), then the fractional variation in the prior range will be $\Delta r = 0.2$ (or 20 %). For what discussed before, the larger Δr, the bigger the prior volume, the slower will be the computation. Therefore Δr for a given free parameter represents the total range of variation set by the prior boundaries of that parameter and normalized by the parameter value of the median estimator[13]. When we deal with multi-dimensional problems ($k > 1$), as it is often the case, we should instead consider the maximum value among the different Δr (one for each free parameter), which we term $\Delta r_{\rm max} = \max_{i=0,k-1} \Delta r_i$. This is because the efficiency of the sampling algorithm, hence the effective computational time, will be dominated by that parameter (or parameters) producing the lowest density of live points along their coordinate. As a result the normalized computational time of the DIAMONDS code has to be expressed as $t_{\rm norm} = t_{\rm bin}/\Delta r_{\rm max} = t_{\rm comp}/(N_{\rm bins} \cdot \Delta r_{\rm max})$.

For our application on the calibration of the relation f_0-k we have obtained 18 independent computations, one for each value of k that we considered, and each one with a different $\Delta r_{\rm max}$. To calibrate the fundamental $t_{\rm norm}$-k relation, we have thus evaluated an average $\Delta r_{\rm max}$, yielding $\langle \Delta r_{\rm max} \rangle_{f_0\text{-}k} = 19.3978$, which we consider as a more reliable estimate than $\Delta r_{\rm max}$ estimated from a single computation. The resulting $t_{\rm norm}$ as a function of k is shown in the bottom panel of Fig. 2.6, and spans over three orders of magnitude, from 0.1 to 100 ms. By fitting another pawer law as done for $t_{\rm comp}$, as expected we find that $t_{\rm norm}$ has a weaker dependency on k than what $t_{\rm comp}$ has, which corresponds to an exponent of about 2.4 as shown by the relation

$$t_{\rm norm} = 5.4912 \cdot 10^{-6} \cdot k^{2.3958} \ {\rm s} \, . \tag{2.12}$$

Conversely to $t_{\rm comp}$, the time $t_{\rm norm}$ is independent of the length of the dataset in use because it represents the actual time it takes for the algorithm to complete a process when only one bin of data is being used. More importantly $t_{\rm norm}$ is also independent of the way we have set up the priors on each free parameter, because it assumes that the fractional prior ranges are all unitary (or that at least the maximum fractional prior range is unitary). The time $t_{\rm norm}$ for a given k can be considered as the fundamental time unit of DIAMONDS to complete an inference process involving a model with k free parameters (and assuming that the other conditions on the configuring parameters of the code used for the calibration example are also applied). Therefore we can use $t_{\rm norm}$ to predict the effective computational time $t_{\rm comp}$ of a given application. For this purpose, we follow the three steps: (i) predict $t_{\rm norm}$ by adopting the power law given by Equation 2.12 and the number of dimensions of the problem; (ii) multiply the result by the foreseen number of data bins; (iii) multiply the result by the maximum fractional prior range set in our problem. The rescaling by $N_{\rm bins}$ is necessary because the time it takes to evaluate the likelihood function for a fixed number of free parameters is directly proportional to $N_{\rm bins}$. This is true since the likelihood is by definition expressed as a series of $N_{\rm bins}$ identical terms (computationally speaking). Changing $N_{\rm bins}$ by a factor of, e.g. 10,

on the specific problem in use and are out of the scope of this user guide. For example, it is generally possible to convert different prior distributions into a uniform one with some parameter transformation, or by estimating an approximated boundary that can be associated with that of a uniform prior (e.g. a $\pm 3\sigma$ range around the mean estimator of the parameter for a Gaussian prior having σ as a standard deviation). Evaluating Δr from a parameter range will then be trivial, as discussed in the next footnote.

[13] If we consider a free parameter θ, its median estimator $\theta_{\rm median}$, and a uniform prior range $[\theta_{\rm min}, \theta_{\rm max}]$, we have that $\Delta r = (\theta_{\rm max} - \theta_{\rm min}) / \theta_{\rm median}$. When estimators of the free parameters are not available, for example if the need to estimate the computational time for a new application that we have not yet performed, then $\theta_{\rm median}$ can be replaced by the mid point of the prior range, $\theta_{1/2} = (\theta_{\rm max} - \theta_{\rm min}) / 2$.

will increase the time necessary to evaluate the corresponding likelihood (or log-likelihood) value by exactly the same factor. So for example if we want to estimate how long it takes to DIAMONDS to perform the inference of a problem with 20 free parameters and a dataset with $N_{\rm bins} = 2.5 \cdot 10^4$, assuming that we have adopted the same $\Delta r_{\rm max}$ as in our calibration (namely $\langle \Delta r_{\rm max} \rangle_{f_0\text{-}k} = 19.3978$) we will have that $t_{\rm comp} = 0.0072 \times 2.5 \cdot 10^4 \times 19.3978 \simeq 3486\,{\rm s}$, so roughly 1 hour. If, instead we also change $\Delta r_{\rm max}$, as it is usually happening when we modify the number of free parameters, and/or their prior boundaries (especially if the dataset in hand is different), then we need to make sure that the rescaling to the actual $\Delta r_{\rm max}$ is taken into account to obtain reliable estimates of $t_{\rm comp}$.

To show the importance of taking into account $\Delta r_{\rm max}$, and test the reliability of the $t_{\rm norm}$-k relation calibrated in this section, we have performed a simple test by considering a different application of DIAMONDS , namely the fitting of the background signal in a Fourier spectrum (e.g. see [7, 8, 10, 11, 12]). Once again we refer to the star KIC 12008916, the same used for calibrating the f_0-k relation in Sect. 2.4.1. The model adopted for this application is substantially different from that of a peak bagging analysis and it accounts for 10 free parameters in total. In addition the dataset now required contains a much larger number of data bins than those used in the peak bagging analysis, $N_{\rm bins} = 35970$, which comprises the entire power spectrum of the star. By performing the analysis with DIAMONDS, setting $\delta_{\rm final} = 1.0$, $\alpha = 0.0$, $N_{\rm live} = 500$, and $f_0 = 1.384$ as predicted by Equation 2.4 for $k = 10$, we obtain that $t_{\rm comp} = 78\,{\rm s}$. For predicting $t_{\rm comp}$ we again consider Equation 2.12 for $k = 10$, and that $\Delta r_{\rm max} = 1.7476$ according to our choice of the prior boundaries for each free parameter of the background model. Then, the predicted computational time is given as $t_{\rm comp} = 1.3660 \cdot 10^{-3} \cdot 35970 \cdot 1.7476 \simeq 86\,{\rm s}$, which agrees around 10 % with the effective computational time obtained with DIAMONDS. Without taking into account $\Delta r_{\rm max}$ to predict $t_{\rm comp}$, hence relying only on the relation $t_{\rm bin}$-k, we would have overestimated the predicted computational time by a factor of about 11 (which is given by the ratio $\langle \Delta r_{\rm max} \rangle_{f_0\text{-}k} / \Delta r_{\rm max} = 19.3978/1.7476$, so more than one order of magnitude difference!

Finally we caution the reader that the normalized computational time presented here has always to be interpreted as an approximated estimate because the actual $t_{\rm comp}$ can vary for many reasons, among which we mention the actual sampling taking place at each nested iteration, the complexity of the posterior distribution to sample (hence the presence of possible degeneracies and of multi-modality), the adequacy of $\Delta r_{\rm max}$ in representing the dependency from the prior volume (especially if non-uniform priors are being used), and of course the computational resources available[14]. In addition, the relations calibrated in this section assume that we adopt the f_0-k relation presented in Section 2.4 and with the configuring parameters of DIAMONDS specified in the same section. A different f_0-k relation that uses the same values of $N_{\rm live}$ and $N_{\rm clust}$ adopted in this guide (no longer in line with the error calibration described in Section 2.4.1), will clearly yield different computational times. Also, a substantially different choice of the number of clusters, $N_{\rm clust}$, and/or of $N_{\rm live}$, will impact the efficiency of the sampling algorithm, hence the computational time. We however consider that the peak bagging application and related calibration presented here can represent a valid benchmark for a large variety of problems, optimized for uni-modal solutions and preferably not involving high multi-modality as in the examples of the Eggbox or Rastrigin functions discussed by [7], which have to be treated separately.

[14]The relations and time measurements presented in this section refer to a 2.7 GHz CPU (on a single thread/core) and can therefore be easily rescaled to a different clock speed, and/or to a condition of data parallelization on multi-thread or multi-core. In the case of data parallelization, it will be crucial to consider how the dataset is being split for each parallel process, hence consider the computational time of the likelihood for each subset of the total dataset.

2.6 How $N_{\rm clust}$ can change the enlargement fraction f

One last discussion to face is related to the way the number of clusters involved in the sampling impacts the enlargement fraction of the ellipsoids. This can provide some additional and useful insights to help calibrating other f_0-k relations for different values of live points and number of clusters adopted. We recall from Equation 2.1 and Section 2.1 that the number of clusters $N_{\rm clust}$ is not explicitly appearing in the definition of the enlargement fraction but its dependance is hidden in the parameter n, the number of live points falling in an ellipsoid. It is through the quantity $\sqrt{N_{\rm live}/n}$ of Equation 2.1 that $N_{\rm clust}$ has a direct influence on the size of the individual ellipsoids and we shall understand how in the following.

The quantity $\sqrt{N_{\rm live}/n}$ represents the error on the eigenvalues stemming from the covariance matrix of a finite sample of points, which is $\propto 1/\sqrt{n}$, and normalized by the total number of live points used, as discussed by [3]. In practice, the more clusters are used during the computation the lower will be, on average, the number of live points that can be contained within each ellipsoid because the same amount of live points, $N_{\rm live}$, has to be divided among more clusters. By considering the statistical average of n, we have that $\langle n\rangle = N_{\rm live}/N_{\rm clust}$, hence resulting in a net additional enlargement to the ellipsoids for an increasing $N_{\rm clust}$[15], aimed at encompassing a larger degree of uncertainty in the corresponding axis lengths. An immediate conclusion in this regard is that it is not advisable to exceed in the number of $N_{\rm clust}$ being used because this can introduce problems in controlling the size of the ellipsoids, in adequately approximate the volume of the parameter space to follow the likelihood distribution, and thus hamper the overall sampling efficiency and reliability of the code in producing good results.

Changing $N_{\rm live}$ alone (by an amount on the order of several hundreds units) is not changing f significantly (at least on an average basis) since it is the number of identified clusters that matters. This was also shown in Section 2.4 and 2.4.1 to be effective on a level of about 10 %. We can understand that $N_{\rm clust}$ is in principle independent of $N_{\rm live}$ because it mainly relies on the shape of the likelihood distribution and on the way the nested sampling evolves (unless the live points are so sparse that the initial sampling is completely misleading). The total enlargement fraction is on average depending by the number clusters and this can be demonstrated on a statistical basis by simply replacing the parameter n from Equation 2.1 with its averaged value $\langle n\rangle$, already introduced before, yielding the average enlargement fraction

$$\langle f\rangle = f_0 X^{\alpha}\sqrt{N_{\rm clust}}\,, \tag{2.13}$$

in which the explicit dependence on $N_{\rm live}$ has now disappeared.

To show a clarifying example, consider the case in which we want to analyze the granulation signal in the Fourier domain of several stars (see e.g. [11]), meaning that we keep the fitting model fixed, hence the number of dimensions involved, but we change the dataset for each new analysis. Eventually it may happen that the number of clusters involved during the computation changes from one star to another. Now suppose we have tuned the configuring parameters of DIAMONDS for a first computation to force the code using 4 clusters only. If we use the same set of configuring parameters for a new analysis but this time we force the code to use 8 clusters instead of 4, this will imply that the average enlargement fraction should

[15] The average of n is considered over the number of clusters (or equivalently ellipsoids) involved at a certain iteration of the nested sampling. By knowing that $N_{\rm live}$ is constant, if we have, say, p different clusters identified, hence $N_{\rm clust} = p$, each one containing a number of live points n_i, with $i = 1, 2, \ldots, p$, it must be that $N_{\rm live} \equiv n_1 + n_2 + \ldots n_p$ by definition. From the simple arithmetic average we thus have that

$$\langle n\rangle = \frac{n_1 + n_2 + \ldots n_p}{p} = \frac{N_{\rm live}}{N_{\rm clust}}\,.$$

grow by a factor $\sqrt{2}$. The reason is that the parameter n will be on average $\sqrt{2}$ times smaller than in the first case with $N_{\rm clust} = 4$ because the number of clusters is now doubled but the number of live points is still the same. By extrapolating to the general case, we shall consider two different numbers of clusters involved in two separate computations, $N^{(1)}_{\rm clust}$ and $N^{(2)}_{\rm clust}$. The relative difference in the two corresponding average enlargement fractions $\langle f^{(1)} \rangle$ and $\langle f^{(2)} \rangle$, as from Equation 2.13, thus reads

$$\frac{\langle f^{(2)} \rangle}{\langle f^{(1)} \rangle} = \sqrt{\frac{N^{(2)}_{\rm clust}}{N^{(1)}_{\rm clust}}} \,. \tag{2.14}$$

As highlighted in the example above, this aspect becomes relevant if we plan to extend the same analysis to multiple datasets by using a single set of configuring parameters (e.g. those calibrated for a single dataset that is a good representative of all the others, see also Section 4.5). Then, this problem should be taken into account if we notice that by using the same configuring parameters for a new computation either the marginal distributions that are computed appear affected by one of the issues presented in Chapter 4 or the nested sampling itself stops prematurely, as described in Chapter 3. This means that with the new computation (i.e. with the analysis of another dataset) a significant change in the number of clusters being used must have occurred, typically on the order of 4-6 clusters difference. A quick way to track a possible change in the number of clusters is to check the output numbers printed on the screen during the computation, as shown in Section 1.8.

For correcting this undesired behavior of the enlargement fraction it is clear that we have to act either on the range of allowed number of clusters, or on the initial enlargement fraction f_0. In the following we shall describe how to intervene on f_0, since for many applications we may require that the number of clusters can actually vary within some broad range to make sure that curving degeneracies and/or multi-modalities are properly caught. We consider once again that we have fixed the configuring parameters to an optimal set found for our first computation. We now adopt the same f_0 of the first run in a second one, where the number of clusters is changing as shown before from $N^{(1)}_{\rm clust}$ to $N^{(2)}_{\rm clust}$. By considering Equation 2.14, one can rescale the initial enlargement fraction f_0 of the second computation accordingly, yielding the corrected value

$$f_{0,\rm corr} = f_0 \sqrt{\frac{N^{(1)}_{\rm clust}}{N^{(2)}_{\rm clust}}} \,. \tag{2.15}$$

To explain this relation, we assume that the new computation, leading by default to a new enlargement fraction $f^{(2)}$ as from Equation 2.1, has used e.g. a higher number of clusters than the first one for which we calibrated the configuring paramters. Then, the condition $N^{(2)}_{\rm clust} > N^{(1)}_{\rm clust}$ will produce a value of $f^{(2)}$ that is larger than $f^{(1)}$ on average by a factor given by Equation 2.14. As we learned at the beginning of this chapter, we cannot change f in a direct way but only intervene on the corresponding input f_0. To readjust $f^{(2)}$, which is now off from its optimal value, and change it back to approximately the former value $f^{(1)}$, which we know to be working fine for the given problem, we need to evaluate $f_{0,\rm corr}$ by dividing our input f_0 (the same in both computations) by the same factor for which $f^{(2)}$ has increased, easily retrievable by our knowledge of the difference in the number of clusters. According to the example given, we will have that $f_{0,\rm corr} < f_0$ since we want to compensate for a total enlargement fraction that is larger than expected.

However, this problem is not very common and can easily be addressed either by readjusting the enlargement fraction following the recipes presented in this manual (first taking into account the effect described in this section and eventually tuning it according to what discussed

in Chapter 4), or by restricting the allowed range in N_{clust} to avoid two different computations to end up using a significantly different number (e.g. by using a range $4 \leq N_{\mathrm{clust}} \leq 6$, instead of the wider $1 \leq N_{\mathrm{clust}} \leq 10$, if we know that in general the problem requires around 5 clusters only). According to our experience in using the code, we recommend adopting a narrow range of clusters during a computation rather than applying a correction to f_0. This is because the correction is not guaranteed to work all times, especially when the number of clusters involved is high and the term $\sqrt{N_{\mathrm{live}}/n}$ becomes much less accurate (see also the discussion at the beginning of this section).

To conclude this chapter, it is useful to comment about the methodology proposed in this section by asking ourselves the question: "Why do we incorporate the term $\sqrt{N_{\mathrm{clust}}}$ in the average dynamical enlargement of Equation 2.13 (or equivalently the quantity $\sqrt{N_{\mathrm{live}}/n}$ in Equation 2.1) if we eventually need to correct for it when the number of clusters is changing from one computation to another?". To answer this question, we have to consider that this aspect is only useful for the specific case of applications that are in principle identical (or at least of the same type), apart for the different datasets in use. In such kind of applications we have no particular reasons to believe that a difference in the number of clusters being used could be supported by a real change in complexity of the shape of the likelihood distribution. Instead, we expect that the likelihood distribution, though it will never be the same if we change dataset, will still preserve a similar topology, proving that a change in N_{clust} is more likely to be the result of a different initial distribution and subsequent evolution of the live points used by the nested sampling algorithm. For general applications the term $\sqrt{N_{\mathrm{clust}}}$ has to be considered a quantity useful to produce more reliable ellipsoid enlargements. Therefore, the correction to the initial enlargement fraction given by Equation 2.15 is meaningful simply because we have first calibrated f_0 by means of the original definition of the dynamical enlargement, Equation 2.1, which implicitly incorporates the number of clusters being used.

Now that the basic elements to set up a working flow have been provided, and that we have understood what is the role of the enlargement fraction of the ellipsoids and that of the other configuring parameters of the code, in the upcoming chapter we will focus on the incomplete computations that do not allow us to produce the desired final outputs from the Bayesian inference. Lastly we will explain how to inspect the results when the process is completed without errors, thus describing the assessment of their reliability.


DIAMONDS™
BAYESIAN SOFTWARE

3 Tackling incomplete computations

The DIAMONDS code implements a nested sampling algorithm, which as any other Monte Carlo algorithm is subject to the characteristic that a process and its evolution with time will be different every time we execute a new realization[19, 18]. The randomness taking place in both the way the live points are distributed at the first iteration and the way a new live point is drawn from the prior PDF at every nested iteration, can sometimes lead to samplings that cause the computation to fail, just accidentally. A general advise is to try relaunching the code if a failure of any of the types discussed in Sections 3.1, 3.2, 3.3, occurs, but only if this is happening up to a few times in a row, at most.

Based on our experience in using the code with the Multi Ellipsoidal Sampler, we can classify a series of cases that do not allow us to retrieve the desired final results. In this section we will describe each of them in detail by sorting them by increasing severity and complexity, thus explaining what is their possible origin and how to troubleshoot. It is useful to bear in mind that there are two general reasons at the base of the computational failures related to the sampler:

- Wrong choice of the size of the ellipsoids and/or of their dynamical evolution.
- Wrong choice of the prior PDFs for one or more free parameters (especially in the case of uniform priors).

In addition we include two more cases that are not linked to the sampler itself but cause the sampling to fail at the beginning of the computation. We discuss them in Sections 3.4 and 3.5. From now onwards we will often talk about *optimal values* when referring to those values of the configuring parameters that allow the code to successfully complete a computation and produce reliable results. We stress that all the troubleshooting provided here is not guaranteed to work at all times and for any application since there is no precise and absolute recipe for a Monte Carlo method. The descriptions and documentation provided in this chapter are based on the applications where DIAMONDS was used, the detailed knowledge of the working mechanism of the sampling algorithm and its implementation, and the use of some logic reasoning[1].

3.1 Assertion failure

This is the most common error that one can encounter. It is not difficult to handle but it requires some investigation. Although often related with the ellipsoidal sampler, an assertion

[1] Documentation on sampling methods different than the ellipsoidal sampler is not part of this user guide. To help us improving this manual for the future, we encourage you to send your feedback and share the experience gained in testing the code for any other application directly to emncorsaro@gmail.com.

failure is not an internal error of the sampler itself but it originates directly from the Eigen library adopted by DIAMONDS due to some wrong array element indexing. It can display as in the following examples

```
Assertion failed: (index >= 0 && index < size()), function operator(), file
/localPath/Diamonds/include/Eigen/src/Core/DenseCoeffsBase.h, line 394.
```

or

```
Assertion failed: (((SizeAtCompileTime == Dynamic && (MaxSizeAtCompileTime==Dynamic ||
size<=MaxSizeAtCompileTime)) || SizeAtCompileTime == size) && size>=0), function resize,
file /localPath/Diamonds/include/Eigen/src/Core/PlainObjectBase.h, line 262.
```

and only arises during the computation of the Bayesian estimates from the MPDs in the `Results` class. This is the basic type of error and the least severe type among those discussed in this chapter. It could appear either after the nested sampling has reached the imposed termination condition, δ_{final}, meaning that it is the only error outputted by the code, or because the process was terminated prematurely with one of the errors presented in Sections 3.2, 3.3, and 3.4, hence in combination with other errors. This assertion failure implies that despite all the output files of DIAMONDS that contain the original sampling are created, neither parameter estimates nor all the marginal distributions could be computed, meaning that the files `marginalDistribution<parameter#>.txt` for one or more free parameters are not generated, hence the file `parameterSummary.txt` is also not present in the list of outputs (see Section 1.8 for more information on the output files).

We suggest that if this computational error appears for the first time, one relaunches the code at least a few times more to understand whether it is only a problem of a bad sampling/realization or not, unless the assertion failure is subsequent to the the error discussed in Section 3.4. If the problem persists then we should stick to what explained in the following. Since we cannot use the MPDs, tackling this issue requires the inspection of the sampling of the individual free parameters that we are trying to estimate. To do this we have to plot the values of all the output files `parameter<parameter#>.txt`, which we always have available in the directory containing the results, and that store the sampling of the individual free parameters as a function of the nested iterations. Figure 3.1 shows an example of plotting this information for three different free parameters, as derived from the computations performed in the peak bagging analysis described by [8]. The blue dots are the sampling with the standard evolution of the process starting from iteration $M_{\text{init}} = 1000$[2]. The y-axis in all panels shows the allowed range of values for each free parameter as set by their uniform prior PDFs. For the parameter θ_1 (top panel) we have an ideal case where the sampling toward the end of the process is placed in the center of the adopted prior range. The sampling appears also symmetric about the final estimate of the parameter, a case in which no strong correlations with other free parameters appear to be present.

Conversely to the first example, the sampling of the parameter θ_2 (middle panel), approaches to an edge of its prior range. Following this example, a more severe situation in which the sampling directly hits the edge of the allowed parameter range could potentially lead to an assertion failure. To explain this, it can be useful to complement the reading with Section 4.4, where we deal with a MPD falling at the edge of our prior interval. On one hand,

[2]We discarded the first M_{init} points because they do not provide any sensible information on the parameter estimation.

given that the MPD was already computed, when the algorithm implemented in the function `writeParametersSummaryToFile()` of the `Results` class attempts to evaluate the estimators and the Bayesian credible intervals it may happen that the estimators and/or the Bayesian credible limits fall outside the range of allowed parameter values if the MPD is trimmed at one extreme of the prior boundary. The consequence is that the output file containing the MPD (`marginalDistribution<parameter#>.txt`) is created, but not that containing the parameter estimates (`parameterSummary.txt`). This suggests that we are missing the likelihood maximum just by a small extent of parameter range and we therefore need to adjust the prior boundaries accordingly by either shifting the limiting values or extending them[3]. If the prior range cannot be enlarged along the side that is causing the problem[4], one could change the scale of the free parameter into a logarithmic one that would help us obtaining a type of evolution similar to that of the parameter θ_1, which is the most preferred condition. This transforms the uniform prior on θ_2 in a uniform prior in $\ln\theta_2$, now corresponding to a Jeffreys' prior in θ_2, namely $\propto \theta_2^{-1}$ (see also [14, 6]). On the other hand, in a more problematic occurrence it could happen that the MPD itself cannot be computed because the sharp trimming is not even allowing the parameter estimation algorithm to perform the cubic interpolation over the re-binned set of marginal probabilities. As a consequence, both the files containing the MPD and the parameter estimators will not be produced. Similarly to what explained before, we need to intervene on adjusting the prior ranges also this time.

Lastly the bottom panel, representing the sampling evolution of the parameter θ_3, is the typical case showing a wide MPD as compared to the extent of the prior parameter range, hence it is a condition potentially prone to computational assertion failures. This is again because a significant portion of the MPD may fall outside the prior boundaries (and possibly being trimmed at both sides), thus preventing us from computing the MPD itself, therefore also all the Bayesian estimators. To solve the issue it is necessary to enlarge the prior volume (i.e. the interval) considered, at least for the free parameter that is affected by this problem. It is useful to keep in mind that by enlarging the prior range for a given parameter will likely slow down the computational time by an amount proportional to the fractional prior range Δr used on the parameter, as explained in Section 2.5.

We stress that what discussed up to now is mainly related to the adoption of uniform priors. An assertion failure may not appear if a Gaussian or a super Gaussian prior PDF is adopted, where no sharp edges are taken into account. However, Gaussian and super Gaussian priors are computationally less efficient than the uniform priors, as pointed out by [7], and care is required in setting the proper widths of the distributions. Such kind of non-limited-range priors are also more prone to cause a computational failure of the type of Sections 3.2 and 3.3, since they can easily cause the sampling efficiency to drop if they are not adequately set.

Suppose now that we have already adjusted the prior PDFs by following what explained in the above paragraphs, and that we are finally confident about the choice that we have done.

[3] In a *full* Bayesian approach the choice of the prior PDFs is totally independent of the likelihood distribution and it is done before the inference analysis is performed, based solely on our knowledge *a priori* of the problem. In many applications, however, we need to ensure that the likelihood maxima are sampled in their entirety because we want to exploit all the information contained in the data. Since the uniform priors require the limits of the allowed parameter range to be set, we often have to intervene *a posteriori* on their choice, meaning that we apply an *empirical* Bayesian approach. This implies that the posterior distribution is used as a guideline to improve the prior PDFs of a subsequent revised inference analysis of the same problem. We therefore need to pay attention when setting up uniform prior PDFs because they are not reflecting a simple status of ignorance about the inferred parameters, but they are instead imposing a strong limiting condition to the possible outcomes of the parameter estimation (see also [18] for more discussions).

[4] For example, in the case of the free parameter θ_2 presented we have the condition $\theta_2 > 0.00$, meaning that we cannot extend the prior boundary below 0.00.

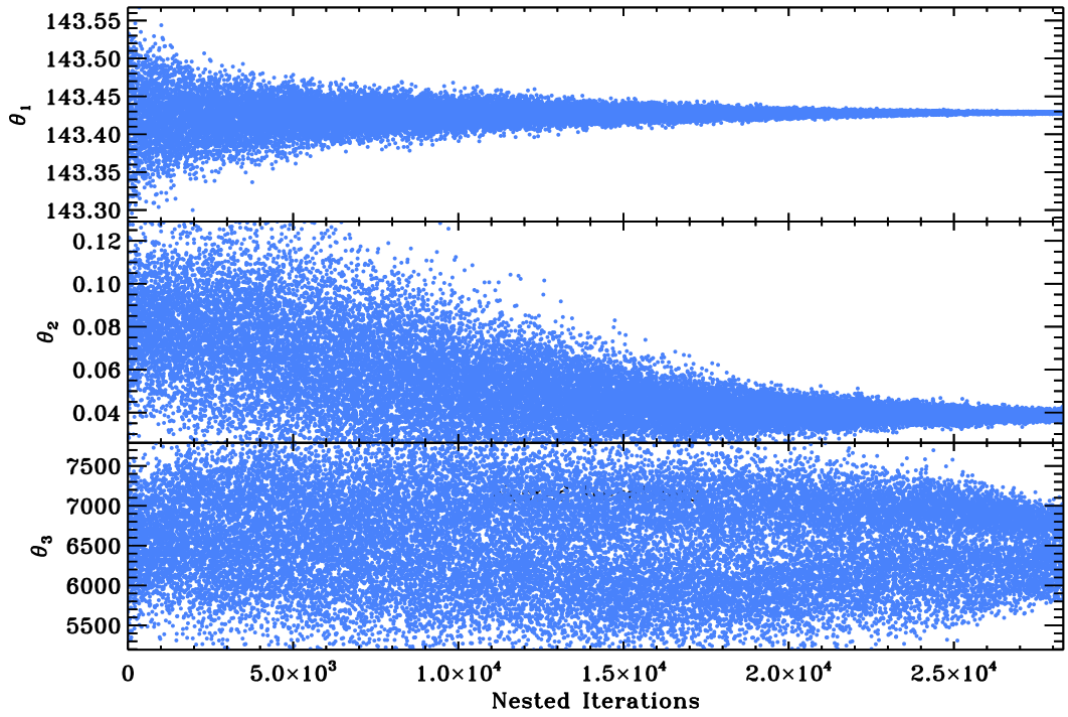


Figure 3.1: The evolution of three different free parameters in the fitting of a Lorentzian profile (see also Section 2.4.1) with the number of nested iterations, and adopting uniform priors. In the top panel, the optimal case of a prior boundary that is well symmetric around the estimator of the free parameter and sufficiently large to ensure that all the likelihood distribution is sampled adequately. In the middle panel, a more critical case of an estimator evaluated close to the edge of the prior boundary, which could lead to an assertion failure as presented in Section 3.1. In the bottom panel, the case of a sparse sampling implying either a small prior range or a large enlargement fraction, also discussed in Section 4.1.

If the assertion failure persists eventually it will be appropriate to intervene on correcting the enlargement fraction of the ellipsoids. The general advise here is to try reducing f_0 and/or increasing α by using steps of 0.1 and 0.01, respectively, for every attempt made until the assertion failure error disappears and the nested sampling is able to generate the output files without any other error. After the results are printed, it will be necessary inspecting them with critical sense. For this purpose we recommend the reading of the subsequent Chapter 4, and in particular of Section 4.1, which explains the effect of f_0 and α on the MPDs. This final aspect is required for understanding whether the final results are reliable or not. If we intend to use our results for other purposes, this represents an inevitable step for any inference process that was successfully completed.

3.2 No better likelihood points found

This is a quite common computational failure that one can encounter and it is typically not difficult to troubleshoot. It can be easily identified when the following error message appears at some point on the terminal screen during the execution of the process:

```
Can't find point with a better Likelihood.
Stopping the nested sampling loop prematurely.
```

The error is produced by default by the `NestedSampler` class but originates directly from the ellipsoidal sampler. It occurs in the process of searching for a new live point that satisfies the hard constraint of having a better likelihood value than the worst of the current set of live points[5]. What happened here is that this searching — involving a drawing of a new live point from the prior PDFs for every attempt made — did not yield any new solution before the maximum number of attempts (M_{attempts}) was reached. As explained by [7], we recall that the parameter M_{attempts} has the purpose of blocking further sampling from occurring because the number of unsuccessful attempts becomes unacceptably large (typically over 10^5), otherwise causing the computational time to increase significantly. The reason why this upper limit is reached depends on a combination of different factors, mainly the shape of the likelihood distribution, the dimensions of the sampling ellipsoids, our choice of the prior boundaries, and sometimes the number of live points adopted with respect to the quality of the dataset, hence the sharpness of the likelihood distribution. In the following we shall try to detail what just mentioned.

If the likelihood region we are sampling is rather flat, hence characterized by a low positive gradient in most of its locations, it will be more difficult than in the case of a stronger positive gradient to quickly find a better likelihood point, implying that on average more attempts need to be made. A situation where a low gradient in likelihood is present could originate from a region of the likelihood distribution that is far from the mode (or any of the multiple modes), which can in turn also be caused by a bad choice of the prior boundaries for one or more free parameters of the fitting model, or just by a poor dataset information as compared to the adopted model. Because of the lack of a clear positive gradient (see the working principle of the nested sampling for more details, [19, 18]), flat or nearly-flat distributions in likelihood can often lead to a sampling that lacks of clear structures because prone to form sparse groups of live points that are scattered over a relatively large region of the parameter space. If clusters including such sparse groups of live points are eventually identified by the X-means or any other clustering algorithm in use, this is likely to produce large axis lengths from the covariance matrix of the associated ellipsoid.

Let assume for simplicity that we take all the principle axes[6] of the given ellipsoid to be symmetric, i.e. to have the same length, and that we term the generic axis in one dimension as λ. The corresponding analytical volume is then given as that of an hyper-sphere, $V \propto \lambda^k$, where k is once again the number of dimensions of the problem. As shown in Figure 3.2 with colored iso-volume levels for $k = 2$, the volume of the ellipsoid increases more rapidly with f_0 when its intrinsic (or default) value[7] is larger, namely for higher values of λ (see the colored shades on the surface). The effect is even more pronounced when the dimensions increase. This means that when by default a new ellipsoid is larger than the others, the enlargement fraction will enhance its volume in comparison to that of other ellipsoids that are instead smaller. We recall that the drawing takes place each time within an ellipsoid that is selected among the others with a probability proportional to its hyper-volume[8], as defined by [3]. This significantly increases the chance of the faulty ellipsoid to be selected for the drawing process

[5] This follows from the working principle of the nested sampling algorithm (e.g. see [7], Section 3).

[6] It is clear that the principle axes are in number equal to the number of dimensions of the inference problem.

[7] With the term intrinsic or default volume we refer to the original volume when no enlargement is applied, that is for $f_0 = 0$.

[8] Suppose that at a given nested iteration we have p different ellipsoids available, each one with a hyper-volume

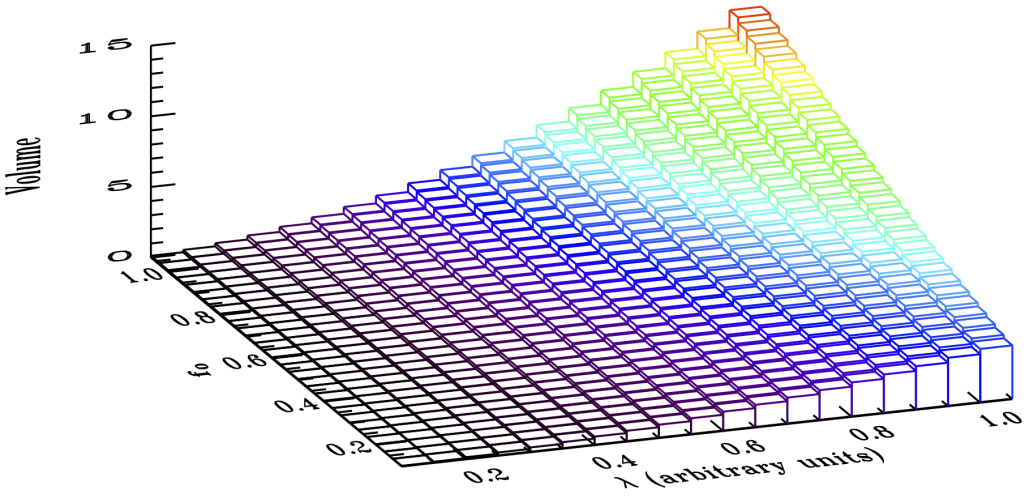


Figure 3.2: The volume of an ellipsoid with symmetric principle axes in $k = 2$ dimensions (namely a circle with $V = \pi\lambda^2$), as a function of the semi-axis length in arbitrary units, λ, and of the initial enlargement fraction f_0. The surface in colored scale shows that when by default λ is larger, the enlargement fraction produces a more enhanced effect on increasing the volume of the ellipsoid than in the case in which λ is smaller.

every time. Drawing repetitively from this ellipsoid, which we assumed to be localized in a region where the likelihood has a low gradient, will quickly push over the threshold the number of attempts needed to find a new point. Since the sampling volume considered is much larger than the optimal condition requires, we have a drastic fall in the sampling efficiency of the algorithm.

Nonetheless, sometimes this computational error is not necessarily depending on the presence of a flat likelihood distribution or a bad set up of the prior boundaries, but it could originate from a wrong choice of the number of live points $N_{\rm live}$. As we mentioned earlier in Section 2.2 it is important to have enough live points to guarantee an initial sampling that is able to detect all the maxima of the likelihood distribution, or at least a good number of them. If we are not using enough live points for the given parameter space (especially when the number of dimensions is high, say around 100), they will result sparsely distributed. For the same reasons explained before, eventually the identified clusters will lead to very large ellipsoids and cause the sampling efficiency to drop very quickly.

We now suppose we are approaching to a completely new problem and we want to overcome this computational failure. Based on what discussed earlier in this section, when attempting to troubleshoot this error we need to take into account three sequential steps:

1. We suggest to check whether $N_{\rm live}$ is reasonably large for the problem at hand (see Section 2.2 for more details), hence possibly repeat the computation with a higher number.

V_i, with $i = 1, 2 \ldots, p$. The drawing is done from one ellipsoid only, which is selected with a probability

$$p_k = \frac{V_k}{V_1 + V_2 + \cdots + V_p} ,$$

meaning that the larger is the hyper-volume V_k the higher is the probability that the k-th ellipsoid is selected. This is done because in general larger volumes correspond to regions of parameter space that require more sampling to be obtained.

2. If the problem persists, we will have to operate on the volume of the ellipsoids, which is likely to be too large as we said in the first part of this section. According to this, the general rule is that we should apply some reduction of the enlargement fraction of the ellipsoids (see also Section 2.4. This will allow the sampling process to become more efficient, hence increase the chance of finding a new live point.

3. If the first two attempts do not solve the error then it will be proper to inspect our choice of the prior PDFs for all the free parameters. This means that if we are unsure about the priors it will be useful to plot the evolution of the free parameters by means of the information contained in the files `parameter<parameter#>.txt`, as already discussed in detail in Section 3.1. In this way we can understand whether the sampling is reaching the limit imposed by the prior boundaries (meaning that the nested sampling wants to explore regions outside the selected parameter space) or it is lying well within the range that we investigated. We thus extend or shift the prior boundaries in order to accommodate for the trend shown by the sampling, if any, and we repeat the computation.

In practice, most often we deal with an application that we already managed to configure by finding the proper number of live points and an adequate choice of the enlargement fraction f_0, for example that given by the calibration presented in Sect. 2.4. This means that we will troubleshoot making use of the attempt #3 of the list, e.g. for applications involving background fitting and peak bagging analysis of oscillations.

For those situations where the tuning of f_0 is instead unsure (for example because we lack a proper calibration of f_0 as a function of the number of free parameters), we have to adhere to step #2 of the list. By following the recipes provided in Sections 2.4 and 2.6 to set up an initial value of f_0, one could try readjusting (hence reducing in this case) it with steps of 0.1 or 0.2 each time until reliable results are produced, and using what discussed in Section 4 as an additional quality check.

We also notice that the *no better likelihood point found* sampling failure could occur at different stages of the computation. We can identify three main ones:

1. The error appears during the first iterations, typically right after the iteration number $N_{\rm init}$.

2. The error appears around half way toward the end of the process (at least several times $N_{\rm init}$).

3. The error appears when approaching the termination condition of the nested sampling where the sampling slows down, namely for iterations close or very close to the end of the process.

The situation will be more severe if it is happening at an earlier phase of the computation. In particular, in the condition #1 it is likely that f_0 is several times off from its optimal value[9]. Another possibility is that the number of live points is wrong (too small) and it is advised to inspect the sampling of each free parameter to discover what is going on (see also the examples shown in Figure 3.1). In the condition #2, a good chance is that f_0 is off from its optimal value by a few to several steps and it is advised to act by directly reducing it, without intervening on any other configuring parameter. The treatment of the condition #3 is quite different. It

[9] In an extreme case, the error discussed in this section could appear even before the first nested iteration is completed. This means that the value adopted for f_0 is really too large. In this particular situation one should reduce f_0 of a factor 10 and check whether the error disappears. Eventually, f_0 will have to be readjusted to tune its value to the optimal condition by using steps of 0.1 or 0.2.

could be that f_0 might still be a good value for most of the computation but that α is wrong, since the ellipsoids tend to be too large only in the ending part of the process. We have to remember that the enlargement of the ellipsoids is dynamic, as shown by Equation 2.1, and it becomes smaller by the time more nested iterations are completed. If this is the case, it is then appropriate to first try changing the shrinking rate only, by increasing its value to allow the ellipsoids shrinking faster and producing smaller enlargements in the final stages of the computation (see also Section 2.3). If by increasing α by 0.02 or 0.03 will not solve the problem, then an additional decrease of f_0 will be needed, with careful steps of 0.1 each time. However, given that estimates of f_0 and α are likely to be close to those obtained in the calibration, the choice of the prior PDFs will also be crucial, especially when the error is produced toward the end of the computation. We therefore recommend to keep this in mind and to always make sure that the sampling of each parameter is not converging in the direction of forbidden regions imposed by the prior PDFs.

To conclude this section, we highlight that the computational failure presented here is often followed by an assertion failure (especially if it is occurring close to the termination condition), as in the example below

```
Can't find point with a better Likelihood.
Stopping the nested sampling loop prematurely.
-------------------------------------------------
 Final log(E): -9689.26 +/- 0.069862
-------------------------------------------------
 Total Computational Time: 4 seconds
-------------------------------------------------
Assertion failed: (index >= 0 && index < size()), function operator(), file
/localPath/Diamonds/include/Eigen/src/Core/DenseCoeffsBase.h, line 394.
```

This is because after the error message is executed, the `NestedSampler` class exits from the nested sampling loop and the code proceeds with the computation of the results using the `Results` class. This is done to avoid losing the sampling information obtained up to the nested iteration where the process has stopped. The `Results` class will generate all the output files that contain the available sampled values (see Section 1.8 for more details), which we can use for inspection according to what described in Section 3.1. When no better likelihood points are found, it is likely that the sampling of the parameter space is wrong and therefore also the parameter estimation and/or the computation of the MPDs done by the member function `writeParametersSummaryToFile()` is subject to fail, but not necessarily all the times.

3.3 Ellipsoid matrix decomposition failed

This is the least common failure error caused by the sampler but at the same time it is the least intuitive and most troublesome. It can be identified by the following error message appearing on the terminal screen:

```
Ellipsoid Matrix decomposition failed.
Quitting program.
```

It is caused by the `Ellipsoid` class used by the `MultiEllipsoidSampler` class, and it is intimately related to the covariance matrices of the clusters of live points. To understand why this error is produced, consider for instance that at a given nested iteration the clustering

algorithm identifies a cluster of live points that is very elongated in one dimension (highly prolate shape), as shown in Figure 3.3. Eventually the associated ellipsoid matrix may become a *ill-conditioned matrix*, meaning that it cannot be decomposed in an eigenvalue problem due to the presence of matrix elements that differ numerically among each other by many orders of magnitude[10]. This stems from the high proximity (or conversely the high dispersion) of the live points along one (or more) principle axis with respect to the others, e.g. because the variance of the live points computed along one direction is very small as compared to that along a direction orthogonal to it. If this happening, the process must be terminated because the entire ellipsoidal sampling method is not reliable anymore.

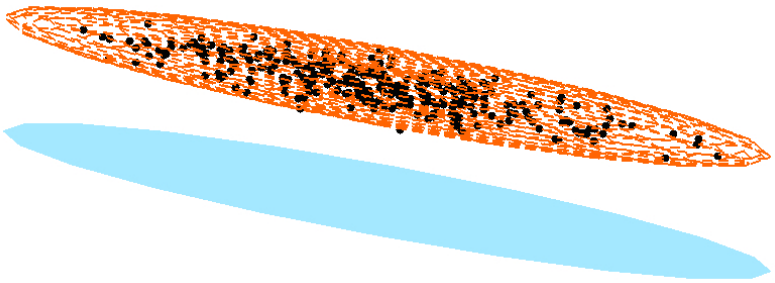


Figure 3.3: An example of a cluster of live points in three dimensions, with an associated ellipsoid very elongated in one dimension, causing a prolate shape that in an extreme case could lead to an ellipsoid matrix decomposition error.

In practice this failure is typically generated by a wrong choice of the prior PDFs (especially if uniform priors are being used) coupled with the presence of a high positive gradient in the likelihood function (steep increase in the likelihood values). Suppose that the maximum of a likelihood distribution that we need to sample is falling off the boundaries of the prior distribution. Such condition may yield a sampling of the parameter space that is highly concentrated along that boundary of the prior range that is close to the position of the maximum of the likelihood. This produces very elongated shapes of sampling points that, as already discussed earlier in this section, can in turn generate an ill-posed eigenvalue problem. In this case one has to readjust the prior boundaries with care, so that they can encompass also the maximum of the likelihood (unless we have a specific reason to avoid that). Most of the times, when using uniform priors and the error *ellipsoid matrix decomposition failed* is outputted by the code, the problem is that we have adopted prior parameter intervals that are either too small or lying off from the values imposed by the likelihood distribution (see also the discussion in Section 3.1 for more details). In addition, this error type is more likely to be produced when highly informative datasets are being used, which cause the maxima of the likelihood distribution to become narrower and steeper in gradient.

More rarely the error could also appear in situations where the enlargement fraction of the ellipsoids is either too small or too large (typically many times), but always in the presence of highly informative datasets that produce very steep gradients in likelihood. This time one has to operate on the initial enlargement fraction by increasing (or decreasing) its value by

[10]This error is occurring in the process of evaluating the overlapping regions between pairs of ellipsoids. The criterion to find that a live point is lying in this region is to check whether it is contained in both the ellipsoids. Numerically this is accomplished by constructing an ellipsoid matrix from the covariance matrices of the two ellipsoids involved in the overlap, according to the algorithm presented by [1]. If the covariance matrices are already ill-conditioned because of e.g. highly prolate shapes, they could cause the eigenvalues decomposition of the ellipsoid matrix to fail.

many steps. Jointly with f_0 one could also attempt reducing (or increasing accordingly) the shrinking rate α since this will prevent the ellipsoids from becoming too small (or big) toward the end of the process.

Finally, similarly to what shown in Section 3.2, we recall that this error could show up followed by an assertion failure, as in the following example:

```
Ellipsoid Matrix decomposition failed.
Quitting program.
-----------------------------------------------
 Final log(E): 3.65051e+06 +/- 0.276146
-----------------------------------------------
 Total Computational Time: 1.23 hours
-----------------------------------------------
Assertion failed: (((SizeAtCompileTime == Dynamic && (MaxSizeAtCompileTime==Dynamic ||
size<=MaxSizeAtCompileTime)) || SizeAtCompileTime == size) && size>=0), function resize,
file /localPath/Diamonds/include/Eigen/src/Core/PlainObjectBase.h, line 262.
```

meaning that once again it was not possible to compute the MPDs and/or the parameter estimators. However, in any case all the output files containing the original sampling done up to the iteration in which the process was terminated, are stored and retrievable for inspection since the `Results` class is executed after the nested sampling has stopped.

3.4 Computation unable to start with NaN values

This computational error only occurs under some specific circumstances, and conversely to the others presented in this chapter, it has nothing to do with the sampling algorithm and the Eigen library. It is produced right after launching the code, when we get the following output message:

```
Nit: 0   Ncl: 1   Nlive: 500   CPM: 0.001998   Ratio: nan   log(E): nan   IG: nan
-----------------------------------------------
 Final log(E): nan +/- nan
-----------------------------------------------
 Total Computational Time: 3 seconds
-----------------------------------------------
Assertion failed: (((SizeAtCompileTime == Dynamic && (MaxSizeAtCompileTime==Dynamic
|| size<=MaxSizeAtCompileTime)) || SizeAtCompileTime == size) && size>=0), function resize,
file /localPath/Diamonds/include/Eigen/src/Core/PlainObjectBase.h, line 262.
```

hence an assertion failure just after the first nested iteration (`Nit: 0`), with `NaN` values for all the parameters related to the Bayesian evidence (namely the Bayesian evidence, its statistical error and the information gain, see also Section 1.8 for more details). This is generated by a very simple, but subtle, mistake present in the dataset. The assertion failure from the Eigen library appears just because the `Results` class is attempting to produce MPDs with only the initial set of live points (from the very first nested iteration), which does not contain any usable information. If the model we selected to fit the observations does not allow a covariate to have a zero value (e.g. because somewhere we are dividing by the covariate itself), this will lead to a `NaN` value, thus causing the program to terminate already after the first nested iteration. For example, when fitting the background model of a solar-like pulsator, or when performing a peak bagging analysis, we have to pay attention to the frequency axis in use,

especially the very first element of the covariates. Since the model is based on a response function (windowing) given as a sinc^2 of the covariates, we cannot include a covariate that corresponds to a zero value of the frequency[11]. A similar mistake could be done when using a particular likelihood function in which for instance we divide by the values of the observations. If for some reasons we have zero values somewhere in the set of observations provided by the dataset, these will produce `NaN` numbers that will cause the code to crash. So the advice is to always pay attention to your input dataset in relation to the model and likelihood you adopt for the inference, eventually removing any zero element that is present in the denominator of an equation.

3.5 Computation unable to start with zero evidence

This error is produced at the beginning of a computation, but after the initial sampling of the parameter space is completed, and can be recognized because it displays the following output:

```
-----------------------------------------------
 Doing initial sampling of parameter space...
-----------------------------------------------
-------------------------------
 Starting nested sampling...
-------------------------------
Can't find point with a better Likelihood.
Stopping the nested sampling loop prematurely.
-----------------------------------------------
 Final log(E): 0 +/- 0
```

After the first set of live points is generated, and their likelihood values computed, the sampling cloud is decomposed into the initial number of clusters set by the users (typically one single cluster if the configuring parameter `NinitialIterationsWithoutClustering > 0`). This implies that a covariance matrix is computed, which then generates a corresponding ellipsoid matrix used to draw the new point from. However, if the free parameters differ significantly in magnitude from one another (e.g. in a bi-dimensional example this could be represented by two parameters taking values that differ by many orders of magnitude from each other), hence also the priors we set on the free parameters, then it is likely that a numerical problem occurs when computing the eigenvalues decomposition of the covariance matrix [17], producing `NaN` eigenvalues that then propagate.

An effective way to overcome this problem is to adopt natural logarithms of your model free parameters. This will ensure that, if a uniform prior is used for the natural logarithm of the free parameter, we flatten the orders of magnitude differences into a simple linear scale[12]. For example, if two free parameters of our model should take values of 10^{30} and 10^{-5}, respectively (so a difference of 35 orders of magnitude), then converting them to the natural logarithm will yield logarithmic parameter values of 69 and -11.5, respectively, which can be easily handled during an eigenvalue decomposition. Another possibility, but less comfortable, is to individually rescale those free parameters that we know could take extreme values, in a way that they exhibit rather homogeneous values[13]. Clearly in this last case one has to apply such

[11] Even a single zero element present in a very large dataset will cause the computation to stop at the beginning.

[12] This corresponds to adopting Jeffreys' priors on the original non-logarithmic free parameters

[13] Typically the difference from one free parameter to another should not exceed 8 orders of magnitude, although this number can change depending on how many free parameters are involved in the model. Dedicated

a rescaling in a proper way every time we change the dataset and/or the model in use, and only if we already have a quite clear idea of the order of magnitude that each free parameter can take.

tests should be done to make sure that the error is not occurring.

4 Checking the results and understanding their reliability

In Section 3 we have shown that when using a Monte Carlo method there is no universal set up to ensure the algorithm successfully converges to the final results for any possible application. At this point we can present the general methodology for inspecting the outcomes and assess their reliability. This process is at the base of a correct use of DIAMONDS and we explicitly recommend a careful reading of all the upcoming sections, especially if you plan to use the results for scientific purposes.

To calibrate the code for a specific application and understand how the individual configuring parameters contribute to the different aspects of its functioning we need to rely on testing and well known examples used as benchmarks. By exploiting the applications done by [7] (including the demos) and later on by [9, 8], we could extrapolate sensible characteristics that can be applied in a more general way and that we explain here to help the user addressing a wider variety of inference problems. Therefore from now on we will consider that the nested sampling could reach the imposed termination condition without any failure error, which would otherwise cause the computation to stop prematurely and not producing the final results that we need. This enables DIAMONDS creating all the desired output files, including the `parameterSummary.txt` and all the files containing the marginal distributions (see Section 1.8). These information constitute the primary material of this last part of the guide that is focused on the results. Since an ellipsoidal sampler requires a proper configuration for each application, it is important and conscientious to understand whether we could have done a good job or not.

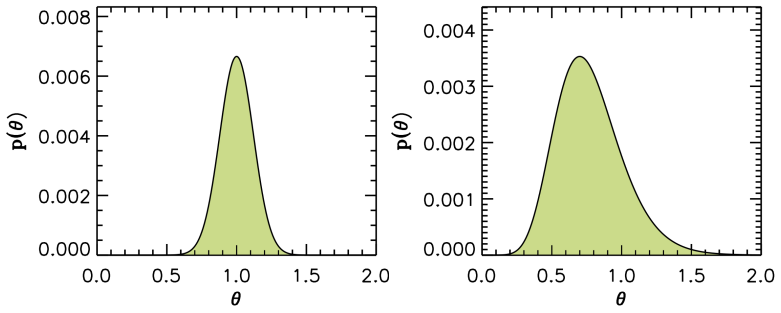


Figure 4.1: Marginal probability distributions with an optimal configuration of the code. In the left panel an example of symmetric Gaussian MPD, while in the right panel an asymmetric Gaussian MPD.

To set up a detailed discussion and present the different problems arising in understanding the reliability of the results, we take by approximation all the expected MPDs to have a

Gaussian-like shape[1]. This type of MPD is very common, and is produced even when adopting likelihood functions of different form, as those provided in the code package[2], thus opening up the possibility to apply this methodology to a large class of applications. Two examples of an optimal MPD for a free parameter θ are shown in Figure 4.1, with a characteristic bell-like shape that is evident even in the asymmetric case. In studies where we are aware that the final likelihood distribution is totally different from that of a Gaussian[3], what follows will not be possible to be accomplished. However, the general sense is to manipulate the configuring parameters of DIAMONDS in such a way that we reproduce (sample) the shape of the likelihood distribution in the best way that is possible.

4.1 How f_0 and α can change the MPD

In Chapter 2 we have seen a detailed overview of the different configuring parameters of the code, especially the initial enlargement fraction f_0 and the shrinking rate α. We have understood that by reducing or enlarging f_0 and/or α we can adjust the dynamical enlargement of the ellipsoids and its evolution within the nested sampling according to Equation 2.1. The parameters f_0 and α are certainly the most important ones to set up in the current version of DIAMONDS based on the multi ellipsoidal sampling. Throughout this manual we have mentioned several times that by directly acting on these two parameters we are able to tune the sampling efficiency and possibly solve some of the main computational failures shown in Chapter 3. In particular in Section 3.1 (see Figure 3.1) we have described that the sampling evolution and its dispersion along the prior range for the given free parameter can be controlled by f_0 and α in order to prevent an assertion failure when the resulting MPDs are larger than the extent of the prior interval[4].

But what if the computation terminates without failures and one or more of the MPDs that we obtain do not show this typical bell-like structure? We shall start by investigating the three main wrong outcomes that can be encountered, plotted in Figure 4.2. All these three MPDs correspond to a set of configuring parameters of DIAMONDS that is however not very far from that of the optimal values, and this is important to remember for improving the result.

The first example on the left panel of Figure 4.2 represents a Lorentzian, characterized by long (or fat) tails and a narrow central part. This distribution shape can appear when the enlargement fraction of the ellipsoids is smaller than the optimal value, yielding a bad sampling in regions of parameter space surrounding the given likelihood maximum, hence a lack of sampling points around it. Despite the MPD may be lying in the middle of the parameter range (as in the example), this is *not* a good result because it is affecting the estimation of the Bayesian credible intervals, which will appear smaller than they are in reality. This is happening because the central region of the distribution is narrow, hence providing larger

[1] This means that the shape originates from a *multivariate normal distribution*, namely the MPD can be either a symmetric Gaussian or an asymmetric one due to the presence of correlations among the different free parameters. In practice, the entire concept of the ellipsoidal sampler arises from assuming that the sampling live points could be distributed by following a mixture of multivariate normal distributions. This is however not limiting the applications of the code for likelihood functions that do not follow such statistics, as clearly demonstrated by the demos shown by [7] and provided in the code package.

[2] The likelihood functions defined in the classes `NormalLikelihood`, `MeanNormalLikelihood` and `ExponentialLikelihood`, are those provided in the basic package of the code, where the latter one is that adopted for all the inferences involving Fourier transform datasets.

[3] This is the case of multimodal distributions such as the eggbox and Rastrigin functions, and the one obtained by [7], or peculiar shapes with pronounced curving degeneracies like the Gaussian shell cylinders or the Rosenbrock function, see the demos presented by [3, 4, 7].

[4] We recall that this would be the case in which we are already confident about our choice of the prior PDFs and we do not need to change them. However, this condition is not very common.

probabilities for smaller intervals of θ as compared to the case of a Gaussian MPD, the latter having a wider central region instead. To solve the problem the first attempt to do is to always try enlarging f_0, using steps of 0.1 each time. In some cases, this may not be enough, meaning that the shrinking rate we are using is still too strong and has to be reduced by steps of 0.01 each time[5]. The optimal values of f_0 and α can thus be found by tuning them in the suggested direction until we recover MPDs similar to those presented in Figure 4.1. This procedure has to be verified for all the free parameters of the inference problem, because if we have even a single parameter whose MPD is still Lorentzian-like, then the global enlargement fraction is not large enough yet and has to be increased. Nonetheless, we have to keep in mind that exceeding with the increase of f_0 and/or the decrease of α may lead to the computational failures discussed in Sections 3.1 and 3.2. If this happens, one has to act the other way around, as already explained in those sections.

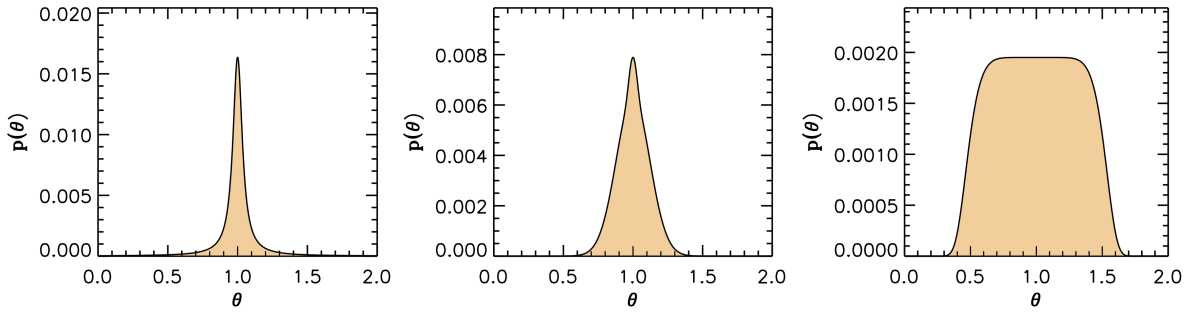


Figure 4.2: Marginal probability distributions resulting from a non-optimal configuration of the code. In the left panel the example of using too small ellipsoids, with a typical Lorentzian MPD. In the middle panel, the case of a quasi-Gaussian distribution with a spiky mode, typical of shrinking rates stronger than the optimal value or of an excess in the termination condition that yields an excess of sampling of the central region of the distribution as compared to its tails. In the right panel the example of using an early termination condition, preventing enough sampling points to be obtained in the central region of the MPD, giving rise to a plateau in the middle of the distribution.

The second distribution (middle panel of Figure 4.2) has the shape similar to a Gaussian (this time not anymore fat tails as for the Lorentzian) but with a spiky central region, where the mode of the distribution is located. This triangular-like MPD is the closest possible to the optimal one and in general it will not affect significantly any of the results. We however suggest to improve it at least to some degree. The triangular shape may arise for two main reasons: (i) because the sampling ellipsoids become very small too rapidly in the late stage of the process, narrowing down the prior volume, hence the parameter range used to sample new points, more than it should actually be done; (ii) because the termination condition δ_{final} is smaller than needed, hence yielding more nested iterations and unbalancing the sampling density of the points between the mode of the MPD and its tails. To correct for the first condition we usually need to act on the shrinking rate α, taking care of reducing its value by 0.01 only once (or twice at most), then repeating the computation. Exceeding with the reduction of α could prevent us from completing the entire computation, yielding the typical error described in Section 3.2, caused by a drastic fall in the sampling efficiency during the final phase of the nested sampling. In the second condition instead, one has to increase the value of δ_{final} (possibly by one order of magnitude per time, or half of it), until the final MPD

[5]In practice it could happen that more than one pair of values for (f_0, α) yields the same optimal result. If this is the case, one can stick to any of the possible pairs being tested.

resembles more closely a Gaussian.

The third case to take into account is represented by a super Gaussian MPD, shown in the right panel of Figure 4.2. This shape is less common than the other two but if present can potentially hamper the final results, especially the Bayesian credible intervals, which will be larger than in the optimal case, and the mode of the distribution because it is not even well defined. This distribution could appear for two main reasons: (i) because we used a total enlargement fraction that is larger than the optimal value but still not enough large to generate an assertion failure (see Section 3.1); (ii) because the termination factor δ_{final} is too large, meaning that we lack sampling in the region that contains the mode of the MPD. The first condition could arise in situations where we either use a value of f_0 that is larger than the optimal value or a shrinking rate that is too small, or a combination of the two. What happens is that we sample a rather large prior volume even toward the end of the process, meaning that the size of the ellipsoids we are using is too big with respect to the initial stage of the sampling to properly approximate the region containing the likelihood maximum. As a consequence, also the runtime of the code will considerably increase, especially in the ending phase of the computation. The best way to proceed is to directly act once again on f_0, this time by reducing its value with steps of 0.1 each time. If the problem persists one ought to change α by increasing it with steps of 0.01 each time. If the reduction of the total enlargement fraction is too strong, we will end up in a Lorentzian shape again, or a pyramidal MPD if we are even closer to the optimal solution. In most cases that we tested, this first condition is not very common and the problem is more likely caused by the second condition, namely a bad choice of the termination factor. In this case, one has to decrease the value of δ_{final}, thus increasing the effective number of nested iterations, until the shape of the MPD resembles that of a Gaussian. If we notice that the computation becomes too slow, possibly producing a *no better likelihood point found* error, then we need to adjust f_0 as well by reducing it by steps of 0.1 per time, until the convergence to the termination factor is accomplished.

We conclude this section with a few useful comments. The computations done by [7, 9, 8] showed that:

- f_0 has the strongest impact on the shape of the MPDs, while α usually remains unchanged for a given application, even if the dimensionality of the inference changes (there is no dependence of the shrinking rate on the number of dimensions, as discussed in Section 2.3).

- Sometimes changing α is not changing the result. This means that the result is quite stable because the likelihood maximum we are sampling is enough well defined (strong data condition, e.g. see [20]) to prevent us from performing a bad sampling even if the dynamical enlargement is not optimal.

- As we have demonstrated in Section 2.4, it is essential to consider the effect of the dimensionality of the problem in exam. For instance, suppose we want to perform a model comparison by using the same dataset and two different models with a different number of free parameters. Clearly, we need to perform a Bayesian inference with both models and retrieve the final total evidence cumulated by the nested sampling for each of the two cases. Suppose that by starting with the first model, we have found the optimal configuration of DIAMONDS to run its related Bayesian inference. To perform the computation for the second model we shall readjust f_0 according to its new dimensionality, following the recipes provided in Section 2.4 and possibly in Section 2.6. It will be crucial to always inspect the final MPDs in order to understand whether they are regular or not, according to what explained before. Eventually we can tune f_0 and α again and repeat the computation until good results are produced (see also Section 4.5 for more

discussion). If this is not done, at least one of the estimated Bayesian evidences will be wrong (or at least partially affected by the bad sampling), implying that the whole model comparison may not be reliable.

- The most general rule is that the balance between size of the ellipsoids and number of nested iterations, hence of sampling points as controlled by δ_{final}, should be such that the MPDs are enough sampled to reasonably resemble a Gaussian function. This should be done by forcing the ellipsoids to be as large as possible to be able to still converge to the termination condition without any failure. In this way one prevents the Bayesian credible limits from being underestimated (see also Section 2.4.1).

4.2 False multimodal MPD

Another problem that we need to present is related to MPDs that show a multi-modality where there is no real multi-modal solution[6]. Two examples of this situation are depicted in Figure 4.3, where the left plot shows a MPD with three modes and the right plot one with two modes. The reason why the multi-modality of these MPDs is not a real multi-modality deriving from the likelihood distribution is because it could be *artificially* produced with an erroneous configuration of the ellipsoidal sampler. What happens in practice is that if the enlargement fraction of the ellipsoids is smaller than the optimal value and by chance more than one cluster (at least two) of live points is identified during the nested sampling process, this could give rise to a sampling evolution similar to that shown in Figure 4.4, an evidence that the sampling is occurring in parallel in multiple regions of the parameter space for a significant portion of the total number of nested iterations. These regions could in principle correspond to either real multiple modes of the likelihood distribution that were correctly identified, or just small portions of a broader likelihood mode that was poorly sampled. Such a multiple sampling, if prolonged enough, could create multiple modes in the corresponding MPD, with these modes occurring at the same positions in θ where the sampling was obtained. To understand whether it is a real effect or not, we need to increase the initial enlargement fraction f_0 by steps of 0.1 each time, and see if the MPD changes, meaning that could recover the optimal case of a Gaussian MPD or one distribution close to it. In alternative, one could impose the number of clusters to be very low (e.g. one or two) and check whether the final outcome still preserves the same multi-modality as the previous test. In most of real applications there is no particular reason to believe that such multi-modality is real, and an investigation of the result is always recommended.

Sometimes, however, we expect a priori a clear multi-modality of the likelihood distribution because of the particular model and dataset adopted. This could be for instance the case of the multi-modal model approach shown by [7] for the peak bagging analysis, or any application in which the datasets produce multi-modal distributions like the eggbox or the Rastrigin functions[7]. Conversely to what just discussed, in all genuine multi-modal likelihood problems we do not intervene on changing the size of the ellipsoids if a multi-modal MPD is obtained, since multi-modality is what we want this time[8].

[6] A multi-modal solution is a solution in which multiple modes of the likelihood distribution are found. In this case it is not possible to conclude the inference process by providing a unique outcome of the corresponding free parameter.

[7] An example of a real application involving a likelihood distribution similar to an eggbox or a Rastrigin function is that of the Scanning Electron Microscopy, in which scanning a material at atom resolution gives rise to many local maxima regularly distributed.

[8] It is important to bear in mind that for multi-modal solutions the Bayesian estimators such as the mode, the mean and the median become meaningless (they can only be defined for a uni-modal MPD). Even if the MPD is

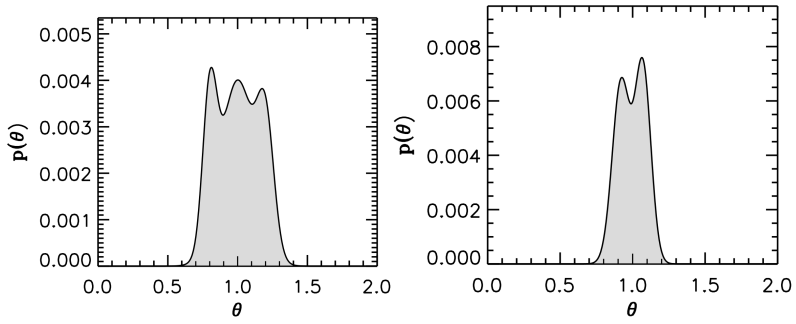


Figure 4.3: Example of marginal probability distributions with a false multi-modality. They could be caused by small ellipsoids coupled with the presence of multiple clusters, which in the plots from left to right are at least 3 and 2, respectively.

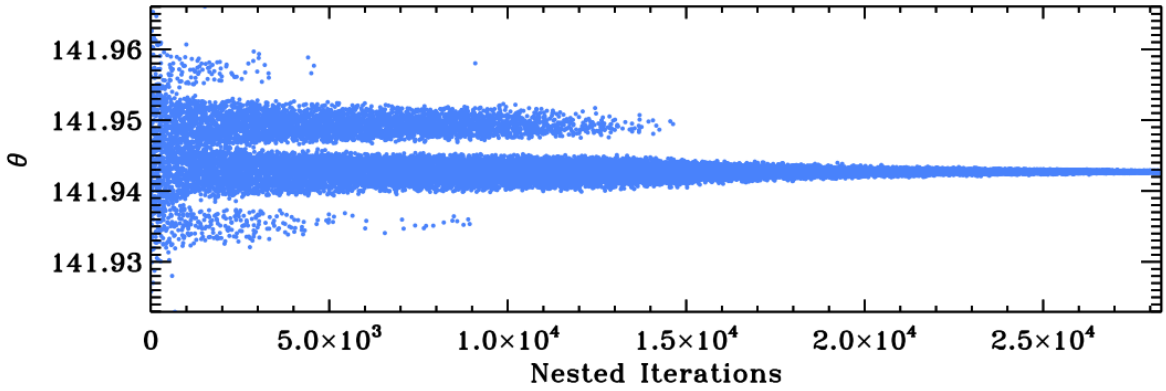


Figure 4.4: The sampling evolution of a free parameter showing the presence of multiple clusters. The sampling in multiple regions of the parameter space stops when about half of the total number of nested iterations is reached, hence converging into a single region containing the global maximum likelihood. This case, if more pronounced, could potentially lead to an MPD with false multimodal features.

4.3 False spike-like MPD

The possible outcomes of marginal distributions described so far are the most common ones that we can encounter and they are the closest cases to the optimal MPDs shown in Figure 4.1. However, there could be another situation that requires some explanation, although it is quite rare. Since DIAMONDS is a code based on a Monte Carlo method, it may happen due to its stochasticity that the initial sampling of the prior volume is just completely wrong and despite the process evolves and it is able to reach the termination condition, the sampling will not be improved. We recall that a similar argument was already provided at the beginning of Chapter 3 in the event of computations that could stop prematurely just by accident. Therefore sometimes a wrong initial sampling of the live points could still yield final outcomes but that are totally wrong. This is the case in which the computation starts with a set of live points unluckily thickened in a small portion of the entire parameter space. What happens is that a single and very small ellipsoid is computed out of the identified cluster of live points, which

computed properly for this case, the evaluation of the estimators could easily yield an assertion failure, discussed in Section 3.1. For these applications we therefore recommend a different approach that is not involving the use of MPDs, as also discussed by [7].

then evolves by becoming even smaller due to Equation 2.1. This causes the nested sampling to reach the termination condition very quickly, and without computational failures, but produces MPDs that have a spike-like structure, as that shown in Figure 4.5.

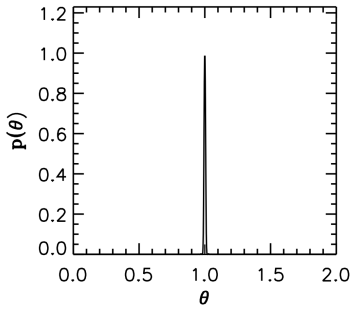


Figure 4.5: Example of a wrong marginal probability distribution with an extremely narrow shape, similar to a spike or a sinc^2 function. This occurs when the initial sampling in the parameter space accidentally leads to a single and very small cluster of live points identified from the very beginning.

This spike-like structure will be similar for all the inferred parameters, and it does not mean that we have extremely precise parameter estimates or that we set too large prior boundaries for all of them, but it is instead a clear sign that we need to repeat the computation at least once more. If the problem persists, then we should proceed as follows: (i) try to increase the initial enlargement fraction by at least a few steps (e.g. 0.3, 0.4) until the MPDs appear fairly similar to one of the cases presented in Section 4.1; (ii) if by enlarging f_0 a failure error of the type shown in Section 3.1 or 3.2 occurs, then it means that our choice of the prior PDFs was wrong from the very beginning and we need to inspect the evolution of the parameter sampling to understand in which way we have to correct the intervals, similarly to what explained in Section 3.1. In this last case, it can also be useful to keep in mind that the density of live points has an important role in producing such a problem. We can estimate this density by dividing the number of live points by the fractional prior range Δr defined in Section 2.5, to make sure that the balance between the extent of the prior range and the number of live points is adequate to allow us identifying the mode (or modes) or the likelihood distribution. We indeed want to avoid conditions in which the initial sampling is too sparse.

4.4 Truncated MPD and the role of uniform priors

The last example that we need to discuss is not caused by a problem in configuring the ellipsoidal sampler but it is directly stemming from a wrong choice of the prior PDFs in the case of uniform distributions. This concerns MPDs that fall at the edge of our uniform prior range, and that could be truncated on one side of the interval (or both in the worst cases). Figure 4.6 depicts a MPD that is truncated on the left side due to a wrong choice of the left prior boundary for the free parameter θ. In the example, the MPD was still computed because luckily the Bayesian estimators could fall within the interval considered. If more pronounced, this situation would cause an assertion failure error, as seen in Section 3.1 and the related Figure 3.1. Such a MPD prevents us from considering the corresponding Bayesian estimates as reliable and, unless we are obliged to maintain our priors for specific reasons, we recommend to relaunch the code with a new set of priors that allows for more room toward the side of the truncated edge. This let us verify that we are able to correctly sample the mode of the likelihood distribution. If the priors cannot be changed and the MPD mode rises too close to

the edge of the parameter interval, by following the same arguments provided in Section 3.1 we suggest to try converting the scale of the free parameter into a logarithmic one.

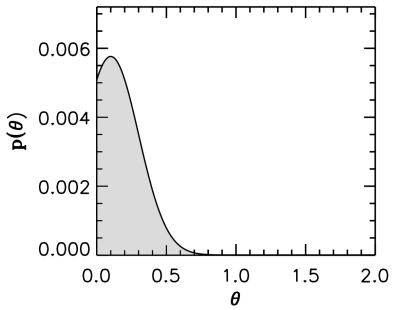


Figure 4.6: Example of a truncated marginal probability distribution, caused by a bad choice of the left prior boundary in a uniform prior PDF. This shows that the sampling algorithm has the tendency to sample new points very close to the edge of the prior range, meaning that the maximum of the likelihood distribution is likely to lie outside the given interval.

4.5 How to extend the methodology to multiple analyses

In this user guide we have often talked about situations where we need to apply a specific analysis to multiple datasets showing similar properties. What shown in Chapter 2 and in Sections 4.1, 4.2, 4.3, 4.4, provided us with a load of information on how to tackle the ellipsoidal sampler and obtain reliable results by using a general approach that can be applied to a wide variety of applications. This paves the way to extend the methodology to two cases of interest:

- The analysis of multiple datasets with a single model and a fixed dimensionality
- The analysis of an individual dataset by means of mixture models with varying dimensionality

We conclude this manual by showing simple recipes on how to proceed in each of the two cases. For the first situation we have already mentioned the typical example of studying the background components in the Fourier domain of an ensemble of stars, where we use a single model for all of them. An example of this analysis was performed by [8, 11]. What is most important in this context is the documentation provided in Section 2.6. Thus to perform this type of analysis on multiple datasets, we advise to adhere to the following steps:

1. Run the analysis on an initial dataset allowing a large range of clusters in the computation (e.g. $1 \leq N_{\rm clust} \leq 10$), hence a higher degree of flexibility in finding the optimal configuring parameters.
2. Once a reliable solution is found, according to what explained in Sections from 4.1 to 4.4, note down the optimal set of configuring parameters and how many clusters are being used on average by this first process. This last information can be obtained either from the Row #10 of the output ASCII file `configuringParameters.txt` or from the printed information on the terminal screen of the parameter `Ncl` at the last iterations of the nested sampling (see Section 1.8).
3. Restrict the range of clusters to the final value from step #2, $\pm$ 1 or 2 clusters at most, or even fix it to the final value itself if you notice that $N_{\rm clust}$ remains very stable throughout the computation (meaning that `minNclusters`=`maxNclusters`).

4. Repeat the analysis for a few datasets and check whether the results remain reliable, according to what discussed in the previous sections.

5. Adopt the new range of clusters and the set of the other configuring parameters found so far for the remainder of the computations to be performed.

The second situation that we mentioned at the beginning of this section is probably less common. As stated different times throughout this manual, this condition is well represented by the peak bagging analysis, namely the extraction of stellar oscillations from the Fourier spectrum of an individual star. This particular analysis usually involves Bayesian inferences with a relatively large number of dimensions (up to a few hundreds) and can alternatively be conducted by considering chunks of the same dataset, hence splitting the problem into separated subsets and performing the computation for a smaller number of oscillations per time (implying fewer dimensions). A clear real example of this approach is shown by [7] and [8]. The model used in this context is a so called *mixture model*, consisting of multiple terms stacked together, in this case a peak profile function for each oscillation peak to be fitted. What we need to apply to this case is essentially related to the effect of the number of dimensions on the initial enlargement fraction, which we discussed in detail in Sections 2.4 and 4.1. For this application we therefore suggest to follow the same steps #1, #2, #3 listed above, and then add the following ones:

4b. Compute the correction factor for the initial enlargement fraction f_0 according to Equation 2.15 by using the number of clusters identified in step #2 for your application (if the number of clusters is 4, no correction is needed).

5b. Check what is the dimensionality of the new analysis (e.g. depending on how many oscillation peaks you consider in a new chunk of the Fourier spectrum of the star), and note down the number.

6b. Evaluate the corresponding initial enlargement fraction f_0 for $N_{\rm clust} = 4$ as given by Equation 2.4, using the given number of dimensions (see Section 2.4 for more details) and subsequently correct this enlargement by rescaling it according to the correction factor computed in the step #4b, if applicable.

This second type of application that we considered could also be that of a Bayesian model comparison, in which we use the same dataset but we want to compare different models, possibly with different numbers of free parameters involved. In particular, we refer to the peak significance test explained by [7], where two different mixture models are compared for every oscillation peak that needs to be tested (see also [8] for more details), using exactly the same dataset for both models.

For applying the methodologies discussed in this section, we assumed that the prior PDFs were already available before running the inference for all the models and datasets used. Although the steps provided in this section are not ensured to work for all or many applications, they certainly represent a good starting point to perform automated analyses on large samples of datasets.


DIAMONDS™
BAYESIAN SOFTWARE

Bibliography

[1] Alfano S., Greer M. L., 2003, *Determining if two solid ellipsoids intersect*, J. Guid. Control Dyn., 26, 106

[2] Bonanno A. & Fröhlich H.-E., 2015, *A Bayesian estimation of the helioseismic solar age*, A&A, 580, 130

[3] Feroz F., Hobson M. P., 2008, *Multimodal nested sampling: an efficient and robust alternative to Markov Chain Monte Carlo methods for astronomical data analyses*, MNRAS, 384, 449

[4] Feroz F., Hobson M. P., Bridges M., 2009, *MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics*, MNRAS, 398, 1601

[5] Fröhlich H.-E., Küker M., Hatzes A. P., & Strassmeier K. G., 2009, *On the differential rotation of CoRoT-2a*, A&A, 506, 263

[6] Corsaro E., Froehlich H.-E., Bonanno A., et al. 2013, *A Bayesian approach to scaling relations for amplitudes of solar-like oscillations in Kepler stars*, MNRAS, 430, 2313

[7] Corsaro E. & De Ridder J., 2014, *DIAMONDS: A new Bayesian nested sampling tool. Application to peak bagging of solar-like oscillations*, A&A, 571, 71

[8] Corsaro E., De Ridder J., García R. A., 2015, *Bayesian peak bagging analysis of 19 low-mass low-luminosity red giants observed with Kepler* , A&A, 579, 83

[9] Corsaro E., De Ridder J., García R. A., 2015, *High-precision acoustic helium signatures in 18 low-mass low-luminosity red giants. Analysis from more than four years of Kepler observations*, A&A, 578, 76

[10] Corsaro, E., Lee, Y.-N., García, R. A., et al. 2017, *Spin alignment of stars in old open clusters*, Nature Astronomy, 1, 0064

[11] Corsaro, E., Mathur, S., García, R. A., et al. 2017, *Metallicity effect on stellar granulation detected from oscillating red giants in open clusters*, A&A, 605, 3

[12] Corsaro E. 2018, *Tutorial: Asteroseismic Data Analysis with DIAMONDS*, ASSP Springer, 49, 137

[13] Libbrecht, K. G. 1992, *On the ultimate accuracy of solar oscillation frequency measurements*, ApJ, 387, 712

[14] Kass R. E. & Wasserman L., 1996, *The selection of prior distributions by formal rules*, J. Am. Stat. Assoc., 91, 1343

[15] Keeton, Charles R., 2011, *On statistical uncertainty in nested sampling*, MNRAS, 414, 1418

[16] Mukherjee P., Parkinson, D., Liddle Andrew R., 2006, *A Nested Sampling Algorithm for Cosmological Model Selection*, ApJ, 638,51

[17] Shaw, J. R., Bridges M., Hobson M. P., 2004, *Efficient Bayesian inference for multimodal problems in cosmology*, MNRAS, 378, 1365

[18] Sivia, D. & Skilling, J. 2006, *Data Analysis: A Bayesian Tutorial*, Oxford OUP

[19] Skilling, J., 2004, *Nested Sampling*, AIP Conf. Proc., 735, 395

[20] Trotta, R., 2008, *Bayes in the sky: Bayesian inference and model selection in cosmology*, Con. Ph., 49, 71