Stratifier1.0

A layer counting algorithm

User guide

Mai Winstrup
mai@gfy.ku.dk
Preliminary version,
December 2014

Introduction

This is a MATLAB-based script developed for counting annual layers in paleoclimate archives. The program is designed for use with ice core profiles, but should be usable for a variety of data series containing a repeated (annual) signal. Note that, in any record, layers of non-annual origin may exist, and thus additional information may be required to establish the annual nature of the obtained layering. The newest version of the script is downloadable from www.bitbucket.org/maiwinstrup/stratifier (upon online request via bitbucket).

The layer counting algorithm is based on the principles of statistical inference of hidden states in semi-Markov processes, with states and their associated confidence intervals being inferred by the Forward-Backward algorithm. The Expectation-Maximization (EM) algorithm is used to find the optimal set of layer parameters for each data batch. Confidence intervals do not account for the uncertainty in estimation of layer parameters. If tiepoints are given, i.e. depths at which the age is assumed known, the algorithm is run between these tiepoints, while assuming constant annual layer signals between each pair. If no tiepoints, the algorithm is run batch-wise down the core, with a slight overlap between consecutive batches. See Winstrup (2011) and Winstrup et al. (2012) for further documentation.

After run of algorithm, manual and automated layer counts can be compared (and, if needed, adjusted) in the environment provided in Matchmaker, a MATLAB tool by Sune Olander Rasmussen (olander@gfy.ku.dk) developed for comparison of paleoclimate timeseries.

The *Stratifier* algorithm was developed for visual stratigraphy data from the NGRIP ice core (Winstrup, 2011, Winstrup et al., 2012). It has later been applied to other cores, and extended to parallel analysis of multi-parameter data sets (e.g. Vallelonga et al., 2014, Sigl et al. (submitted 2014), Sigl et al. (in prep)).

When using this script please provide release date of the algorithm (8/12-2014), and cite: *Winstrup et al., An automated approach for annual layer counting in ice cores, Clim. Past. 8, 1881-1895, 2012*.

Technical issues

MATLAB version

Stratifier has been designed to work with MATLAB R2013a, and updated to work also with R2014b. It requires a license to a range of MATLAB toolboxes.

Platform

Stratifier is known to work on Windows 7 Pro, iOS X, and Linux platforms.

Files and folders

The program consists of a main program, stratifier.m, and a suite of subroutines. Additionally, a settings file is provided, since the model settings must be adjusted according to the data. The package also includes files used in the Matchmaker GUI.

Format of input data

To run *Stratifier*, the required input is

- A preliminary set of manual layer counts
- Data records with an annual signal on common depth scale.

Format of manual layer counts

A file containing an initial set of manual layer counts must be provided. It must be a txt-file having the following format:

Counts(:,1): Depth of layer boundaries

Counts(:,2): Age of layer boundaries

Counts(:,3): Uncertainty of layer (0 = certain, 1: uncertain)

Counts(:,4): Accumulated uncertainty (measured from start of interval).

The file may contain a header, but it is not required.

Since the manual counts are used for constructing templates for the annual layer signature in the various chemical species, it is somewhat important for the manual layer counts to be located relatively precisely at the layer boundaries, whereas the algorithm is much less sensitive to the number of layer boundaries used as input.

Format of ice core data records

The ice core profiles must be saved as MATLAB structure array with the following format:

data.name = {'species 1', 'species 2', …,'species N'}: names of the various data files for core

data.data{1} = data corresponding to species 1,

data.data{2} = data corresponding to species 2,

…,

data.data{N} = data corresponding to species N.

data.depth{1}, …., data.depth{M}: depth corresponding to data profiles.

Data profiles do not need to be on the same depth scale. The depth scale corresponding to each of the data files is given as a Nx1 array with values between 1 and M, e.g.

Data.depth_no = [1, 1, 2, … M]

The data file may contain all data available for the core; the data actively used for layer counting will be a subset of these as specified in the settings-file.

Adjust model settings

Stratifier should be run in a way appropriate to the data and the desired output. The mode in which the algorithm runs can be adjusted by changing various model settings. For each application, a settings file must therefore be constructed and saved in the folder *./Settings/*. An example of a settings-file is provided (*./Settings/sett_example.m*), and a description of each of its fields is given below.

Model.icecore: Name of ice core

Model.species: Name of data series to use for annual layer counting; must be spelled exactly similar to those contained in data.name.

Model.nSpecies: The number of species used. This number is calculated by MATLAB and need not be changed.

Model.wSpecies: Weighting of individual chemical species. In some cases, two data series may record the same signal; for instance if a single process is responsible for delivering the influx of two chemical species, both of which are measured. Despite their similarity, one may want to include both data sets. In order to prevent the two data sets from weighted in too heavily on the resulting layer counts compared to the remaining data sets, they can algorithmically be regarded as a single record by setting the weighting of each of these to 0.5.

There is in general no need to adjust the weighting of the various data series relative to the ambiguity of the annual layer signal in the data. The algorithm internally accounts for the effect of this.

Model.pathData: Provide path to data file (in correct format)

Model.dstart, Model.dend: Depth interval for which to perform the layer counting.

Model.tiepoints: The algorithm may be run using tiepoints, i.e. depths assigned a fixed age. In this case, the algorithm will find the optimal segmentation, while respecting the tiepoint ages as contraints. Tiepoints should generally not be too closely spaced, and preferably have ~50-100 years between them. Shorter sections may destabilize the optimization procedure of the algorithm, and longer sections requires a stable annual signal throughout the entire period, as well as it increases the computation time.

If no tiepoints, the value of Model.tiepoints should be left empty.

If tiepoints are added, the format should be given as follows:

Model.tiepoints(:,1): Depths

Model.tiepoints(:,2): Corresponding ages

Further, an extra field must be added to the Model.array:

Model.ageUnitTiepoints: Age unit of the tiepoint ages. Options are: AD, BP, b2k or layers (this simply being layers without any specific unit).

If given tiepoints, the algorithm will start at the first tiepoints, and run until the deepest tiepoints. Data sections before the first tiepoint and after the last tiepoint are not included.

Model.dx: Resolution of data. The algorithm requires equidistant data, and the provided data files will be downsampled to this resolution. The higher the resolution, the longer the layer counting process will take. Generally, a good value to choose for the resolution parameter is approximately ~15 data points per layer.

Model.dx_center: The data may e.g. be sampled with e.g. 1 cm resolution, but sampled in the midpoint of each interval. In this case, the value of dx_center may be taken as 0.5*dx, in which case unnecessary interpolation is avoided. Otherwise, the value of dx_center should be set equal to 0.

Model.preprocess: Before analysis, the data series is preprocessed to improve performance of the algorithm. The data can be preprocessed in a variety of ways, which can be combined as desired. A very useful method of preprocessing is by log-transforming the data series (optional), and subsequently computing z-scores. A good choice for the normalizing distance is generally ~2-3 layer thicknesses.

Example: The average layer thickness is 0.1 m, and we therefore decide to use a normalization distance of 0.3. For data series 1, we wish to take the log of the data series before normalizing, whereas data series 2 should be normalized directly. The settings-file then reads:

```
Model.preprocess{1} = {'log',[];'zscore',0.3};
```

```
Model.preprocess{2} = {'zscore',0.3};
```

Model.nLayerBatch: The algorithm is run down-core in batches that slightly overlap. The length of each batch is given in terms of the average layer thickness at a given depth, such that each batch contains approximately nLayerBatch layers. Each batch is optimized separately, and the layer signal is assumed to be stable within each batch. The choice of this parameter must therefore balance the requirements of stable annual layering, and sufficient data to provide a basis for the optimization. It is generally found that ~50 is a good value of this parameter.

Model.pathManCounts: Path to manual counts in correct format

Model.ageUnitManual: Age unit of the manual layer counts. Options are: AD, BP, b2k or layers (this simply being layers without any specific unit).

Model.manualtemplates: Depth interval used for constructing layer templates. The algorithm requires templates for the expression of an annual layer in the various chemical species, and these templates are calculated based on the manual layer counts. To obtain the best layer templates, this depth interval should be as large as possible, while still containing stable layer signals. A minimum number of layers to be included is ~50 layers, and the more the better. Upon running the algorithm, an illustration showing the layer templates used as input is saved in the output folder (*templates.jpeg*). If having doubts about the reproducibility of the applied templates, these may be calculated for various sections and compared visually.

Model.initialpar: Depth interval for constructing an initial set of layer parameters, to be used in conjunction with the layer templates. In most cases (depending on the update settings, see below), the algorithm will subsequently improve on the layer parameter values, making the outcome relative insensitive to these initial estimates.

Model.nIter: For each batch, the algorithms iterates to improve the layer parameters according to the signal in the data (the actual iterations depend on the update settings). Model.nIter provides the number of iterations per batch. In most cases, the initial set of parameters is relatively well-determined, and the number of iterations can be kept to a minimum. Experience shows that a good value in this case is generally around 4, with the added precision of using larger values usually being negligible, and not worth the increased computation time.

Model.update: The layer model contains 5 parameters, which describe different aspects of the annual layering in the data. Each of the parameters in the layer model can be iterated in one of two ways: either using the maximum-likelihood solution to update the parameter values, or not to update the parameters. If at all possible, it is recommended to use maximum-likelihood updates for all parameters.

```
Model.update = {'ML', 'ML', 'ML', 'ML', 'ML'};
```

The first entry corresponds to the mean of the log-normal distribution of layer thicknesses.

Second entry corresponds to the standard deviation (sigma) of the log-normal distribution of layer thicknesses.

Third entry is a vector of length nSpecies, which contains the mean layer parameters.

Fourth entry is a matrix of size nSpecies x nSpecies, with diagonal values corresponding to the variance of the layer parameters from year to year for the various species.

Fifth entry is a vector of length nSpecies, which contains the average white noise variance (i.e. variance not explained by annual layer model) for each chemical species.

In case the data only contains a very ambiguous annual signal, the algorithm may need a bit of help, which can be provided by fixing one or more of the five layer parameters. The most appropriate parameter to keep constant is sigma, the spread of the lognormal distribution of the layer thicknesses, i.e.:

```
Model.update = {'ML', 'none', 'ML', 'ML', 'ML'};
```

This sigma parameter generally has a value between 0.15-0.3, and while maximum-likelihood may result in options of layering with much larger values, the much larger variability of layer thicknesses in these scenarios can often be refuted as erroneous.

If setting one or several of the model parameters as constant, the program will subsequently ask whether to use the manual values for this parameter (as based on the layering in the interval Model.initialpar), or whether another value may be preferred. Which one to use depends on the degree to which the manual layer counts are expected to be correct.

Model.dxLambda: Apart from a timescale, the algorithm provides a variety of output to use for evaluating the output of the algorithm. One output is a comparison of the mean layer thickness as computed over equidistant intervals. The interval length(s) over which to compute the mean layer thickness is given in Model.dxLambda. If e.g. desiring to compare the mean layer thicknesses over 1 and 5 m intervals use:

```
Model.dxLambda = [1 5];
```

Model.dMarker: An additional output is the confidence intervals of the number of layers between specific marker horizons. If these exist, the depths of these horizons are added in Model.dMarker, as e.g.:

```
Model.dMarker{1} = [100, 150, 200];
```

In this case, the number of layers (most likely number, and associated confidence interval) will be provided for the two intervals between 100-150 m, and between 150 and 200m, respectively.

Note that the uncertainty associated with the obtained number of annual layers between two depths cannot be calculated directly from the increasing uncertainty band of the timescale with depth. This is because the uncertainties do not combine linearly. If desiring the uncertainty of annual layer counts between two depths, these depths should be added in Model.dMarker.

More than a single set of horizons may be used for calculating number and uncertainties of the annual layers between boundaries, each of these sets will then correspond to a given segmentation of the data record.

Model.ageUnitOut: Age unit in which the output timescale should be given. Options are: AD, BP, b2k or layers (this simply being layers without any specific unit).

Running *Stratifier*

The algorithm is run by typing the following in the MATLAB command line (sett_example being the name of the appropriate settings-file):

>> stratify('sett_example')

The program will now start running. If any updates of the layer parameters are set as "none", the program will further promt for whether or not to use the value from manual counts, and, if not using these, which value to plug in.

The program generally takes a while to run, and the runtime increases with e.g. the number of chemical species, and average number of data points per layer. Please be patient while the code is running.

Upon finalizing the output, the derived layer boundaries are plotted in the MatchMaker GUI, with a comparison to the manual counts used as input. The MatchMaker interface allows easy illustration of their differences, and, if deemed necessary, allows manual changes to the resulting timescale to be made. When using MatchMaker, please be referred to the MatchMaker User's Guide by Sune Olander Rasmussen. If using the option of manually adjusting the automated layers, the adjusted layer counts are located in the folder *./matchfiles/xxx_auto_adj.mat.*

Output

The results of the algorithm are saved in the folder *./Output/icecore/Timescale/*. In this folder, the output is divided into depth intervals, and subsequent runs for the same ice core section are saved in subfolders, such that these can be compared.

Two other output-folders (*./Output/icecore/LayerCharacteristics/* and *./Output/icecore/ProcessedData*) are also created. In these two folders are saved the preprocessed data files and obtained layer templates, respectively, such that they are available for re-use in subsequent runs, thereby speeding up the runtime of the algorithm.

Timescale: timescale.mat, icecore_timescale1yr.txt, timescale.jpeg

Main output of the program is the obtained timescale, which is provided in two different formats. Timescale.mat provides various estimates of the full timescale, in that for each data point the most likely age is given, along with the uncertainty of this age estimate (age of first layer is assumed known). The uncertainty bands are calculated as the 50 and 95 percentiles of the distribution of possible layers at each depth.

A summarized version of the timescale is provided in *icecore_timescale1yr.txt*, which contains the depths of the obtained annual layer boundaries, along with the maximum-likelihood age, and the 95 and 50% confidence intervals (age of first layer is assumed known).

A plot of the timescale is also provided (*timescale.jpeg*).

Mean layer thicknesses: lambda.mat, lambda_xm.jpeg

Plots of the mean layer thicknesses in equidistant intervals are plotted in the figure(s) *lambda_xm.jpeg* (value depending on the length of the equidistant intervals), and compared to that of manual counts.

The corresponding data files containing mean layer thicknesses throughout the core for each of the equidistant intervals are contained in the data file *lambda.mat*.

Marker horizons: markerhorizons.mat, icecore_markerconf.txt (if marker horizons exist)

A MATLAB and a text file containing the most likely layer number, the corresponding probability of this being the actual number, and corresponding 50% and 95% confidence intervals.

The MATLAB file further contains the full probability distributions for each of the intervals.

Additional output:

Further, a plot of the layer templates is provided (*layertemplates.jpeg*), and a couple of MATLAB arrays are saved for future reference. For example, a copy of the array containing all the model settings (*Model.mat*) is saved, and so is a copy of the initial layer parameter input (*Layerpar0.mat*).