StratiCounter

Version 1.2

A layer counting algorithm

User guide

Mai Winstrup
mai@gfy.ku.dk
July 2015

Introduction

StratiCounter 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 additional information may thus be required to establish the annual nature of the layering. The newest version of the script is downloadable from www.github.com/maiwinstrup/StratiCounter.

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 and synchronization of paleoclimate timeseries.

The *StratiCounter* 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. (accepted 2015), Sigl et al. (submitted 2015)).

When using this script please provide release date of the algorithm (7/7-2015), 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

StratiCounter has been designed to work with MATLAB R2013a, and is known to work also with MATLAB R2015a. *StratiCounter* requires a license to a range of MATLAB toolboxes.

Platform

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

Files and folders

The program consists of a main program, *straticounter.m*, and a suite of subroutines. Additionally, an adjustable settings file is provided, containing the suite of model settings that can be adjusted according to the data. Further, the package contains an example of data from the NEEM-2011-S1 ice core. The package also includes files used in the Matchmaker GUI.

Format of input data

To run *StratiCounter*, the required input is

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

A further requirement is a settings-file adjusted according to the data (see next section).

Format of manual layer counts

A file containing an initial set of manual layer counts must be provided, and located in the subfolder *./Manualcounts*. 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.

An empty example of such file is provided (*counts_empty.txt*), as well as a set of manual layer counts for the NEEM-2011-S1 core (*counts_example.txt*).

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

An example data file is provided for a section from the NEEM-2011-S1 shallow ice core from Greenland (*data_example.mat*). The original version of this data file is available online as

supporting data to Sun et al. (2014), *Ash from Changbaishan Millennium eruption recorded in Greenland ice: Implications for determining the eruption's timing and impact*, GRL 41,2, pp. 694–701, DOI: 10.1002/2013GL058642. An empty data file (*data_empty.mat*) is also provided.

Adjust model settings

StratiCounter 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 empty settings-file is provided (*sett_empty.m*), as well as an example of a settings-file for the NEEM-2011-S1 core data (*./Settings/sett_ example.m*). A description of each of the fields in the settings-file 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 weighting 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 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.path2data: Provide path to data file (make sure that the data file has 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, leave the value of Model.tiepoints 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 should be added to the model structure array, namely:

Model.ageUnitTiepoints: Age unit of the tiepoint ages. Options are: *'AD'*, *'BP'*, *'b2k'* or *'layers'* (the latter 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_offset: 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_offset may be taken as 0.5, in which case unnecessary interpolation is avoided. Otherwise, the value of dx_offset should be set equal to 0.

Model.preprocsteps: 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. It should not be smaller than two layer thicknesses. If desired not to do any preprocessing for a given data series, set the corresponding entry in Model.preprocsteps equal to [].

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 should then read:

```
Model.preprocess{1,1} = {'log',[];'zscore',0.3};
Model.preprocess{2,1} = {'zscore',0.3};
```

The preprocessing distances can also be specified as function of the mean layer thickness at a given depth. For each batch, the preprocessing distance will then be determined based on the mean layer thickness of the previous batch. Preprocessing method and distance on batch level is given in a second column in Model.preprocsteps, with the distance given in terms of mean layer thicknesses.

Example: Instead of specifying a fixed distance as in the example above, we wish to constantly maintain a preprocessing distance corresponding to three layer thicknesses. In all other respects, the preprocessing procedure should be similar. In this case, use the following preprocessing specification:

```
Model.preprocsteps{1,1} = {'log'};
Model.preprocsteps{2,1} = [];
Model.preprocsteps{1,2} = {'zscore',3};
Model.preprocsteps{2,2} = {'zscore',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.nameManualCounts: Name of data file containing the manual counts. This file must be located in the subfolder *./Manualcounts* and have correct format as described earlier.

Model.ageUnitManual: Age unit of the manual layer counts. Options are: *'AD'*, *'BP'*, *'b2k'* or *'layers'* (the latter 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 initial set of 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 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. In this case, the update parameter should read:

```
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 amount of data that this value is based on, and the degree of confidence in the original manual layer counts.

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 included 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'* (the latter simply being layers without any specific unit).

Running *StratiCounter*

The algorithm is run by typing the following in the MATLAB command line (replace 'sett_ example' with the name of the appropriate settings-file):

>> straticounter('sett_ example')

The program will now start running. If any updates of the layer parameters are set as "none", the program will further ask 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 length of the data set, the number of chemical species, and average number of data points per layer. Please be patient while the code is running.

Display of results in *Matchmaker*

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. A short description of the Matchmaker interface is provided here. For further questions regarding Matchmaker please be referred to the Matchmaker User's Guide by Sune Olander Rasmussen (olander@nbi.ku.dk).

The top panel of Matchmaker displays the data series actively used for layer counting by the algorithm, and the corresponding automatically generated layer counts. The bottom panel displays the available data series for the core, and the manual layer counts used for initiating the algorithm. If using a large number of data series, not all of these may be displayed at once. In that case, the data records to view can be selected interactively from the drop-down menus marked with a downward-facing triangle.

Each data series can be displayed as the logarithm to the data (if selecting the "Log" button), and with axis limits as defined in the boxes below. The default is to use automatic scaling of the axes ("Aut" button).

The data is shown on a depth scale. The depth interval can be set in the boxes to the left of the window. Alternatively, forward- and backward arrow buttons (or keyboard arrows) allow the user to step forward or backward in depth, with a step size as given in the lowermost box. When comparing the two sets of counts, it is necessary always to ensure that the same depth interval is displayed in the upper and lower panels.

Layer counts are shown as thick grey bars with a number, and in case of uncertain manually-derived layer counts, these are shown as thin grey bars without numbers. Note that as the numbering scheme of the grey bars does not account for the existence of uncertain layers (none of which are included), these numbers do not necessarily reflect the corresponding timescale. If tie points have been used for creating the timescale, the location of these are shown as thin blue bars.

It is possible, although not recommended, to use the comparison of the two counts to manually adjust the *StratiCounter* results. To do this, the button "Mark?" in the bottom of the window should be selected. This allows old layers to be removed and new ones to be added by clicking in the panel at the appropriate locations. If using the option of manually adjusting the derived layers, the resulting adjusted sets of layer boundaries will be saved in the folder *./Output/icecore/Matchfiles* under the names *layers_auto.mat* (i.e. based on the automated counts) and *layers_manual.mat* (i.e. based on the manual counts). When referring to layer counts derived in this way, it should be specified that these have been manually adjusted on basis of automatically derived layer counts.

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: icecore_timescale.txt, timescale.mat, 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_timescale.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*).

Example of a timescale.txt output:

Depth [m]	Maximum Likelihood age [layers]	95% conf int (younger bound)	50% conf int (younger bound)	50% conf int (older bound)	95% conf int (older bound)
206.42	25	25	25	25	25
206.62	26	25	26	26	26
206.84	27	26	27	27	28
207.03	28	27	28	29	29

This result can be interpreted as follows:

At 206.42 m, the 95% confidence interval is [25; 25], i.e. we are with 95% confidence in layer number 25.

At 206.62 m, the most likely layer number is 26, and the 95% confidence interval is [25; 26]. I.e. with 95% confidence, we cannot say whether we are here still in layer 25 or has progressed to layer 26. The layer boundary at 206.62 is thus not certain with 95% confidence. However, since the 50% confidence interval is [26; 26], the designation of this layer boundary is certain with 50% confidence.

At 206.84m, the most likely layer number is 27, and the 95% confidence interval is [26; 28]. The lower limit of this confidence interval is caused by the previous layer being uncertain on

the 95% level. The higher limit is caused by the algorithm finding a possible layer boundary between 206.62 and 206.84 m, which, however, can be rejected as a layer boundary on the 50% confidence level (given that the 50% confidence interval is [27; 27]).

With increased number of layer boundaries, keeping track of the uncertainties of the individual layer boundaries becomes more difficult, since the individual probabilities are combined in often non-trivial ways. At 207.03 m depth, for instance, the most likely layer number is 28, the 50% confidence interval is now [28; 29], and the 95% confidence interval is [27; 29]. Parts of these are inherited from the uncertain layer boundaries previously encountered in the data series at 206.62 m and between 206.62 and 206.84 m. However, the numbers indicate that there is some added ambiguity to the layering from the data section between 206.84 m and 207.03 m, which may suggest an extra layer in the interval. To account for these uncertainties, and when combining with the uncertainties of the possible layer number distribution prior to 206.84 m, the upper limit of the 50% confidence level corresponding to the likely layer boundary at 207.03 m has to be increased with an extra layer.

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 of *x* 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*.

The mean layer thicknesses are calculated in two ways: One way (solid red line and red areas) takes advantage of the full probabilities calculated by the algorithm, and hence include uncertainties associated with layer boundaries located close to the edges. A second estimate of the layer thickness (dark red dashed line) uses the most likely layer boundaries to obtain a mean layer thickness without any associated uncertainty estimate. Given the different approach for accounting for layer boundaries located close to the interval edges, the two estimates may not be exactly similar.

Marker horizons: icecore_markerconf.txt, markerhorizons.mat (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.

Layer templates: layertemplates.jpeg, layertemplates_new.jpg

Two plots of the layer templates are provided. One plot (*layertemplates.jpeg*) displays the employed layer templates for the various species used for the layer counting. The upper panels display the mean layer signal, with the first principal component of the various species shown in the lower panels.

A second plot of the layer templates (*layertemplates_new.jpeg*) include the layer templates used by the algorithm as input for the layer counting (i.e. the same as in *layertemplates.jpeg*), as well

as the layer templates obtained based on the layer counting output of the algorithm. Depending on their similarity, it might be an idea to run the algorithm using the new set of layer templates as input to the algorithm.

Additional output:

Further, 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*).