H2M : A set of MATLAB/OCTAVE functions for the EM estimation of mixtures and hidden Markov models

Olivier Cappé
ENST dpt. TSI / LTCI (CNRS-URA 820),
46 rue Barrault, 75634 Paris cedex 13, France.
`cappe` at `tsi.enst.fr`

Jun 13 2006

Keywords: Hidden Markov Model (HMM), Mixture model, Vector Quantization, Expectation-Maximization (EM) algorithm, Multivariate Gaussian distribution, Count data, Poisson distribution, Negative binomial distribution, MATLAB, OCTAVE, GPL

Contents

1 Introduction

1.1 About `H2M`

`H2M` is a set of MATLAB/OCTAVE functions that implement the EM algorithm [1], [2] in the case of mixture models or hidden Markov models with multivariate Gaussian state-conditional distribution. More specifically, three special cases have been considered

1. Gaussian mixture models.
2. Ergodic (or fully connected) Gaussian hidden Markov models.
3. Left-right Gaussian hidden Markov models.

In fact, the case 2 and 3 above do not significantly differ except for the fact that in the case of a left-right HMM, one needs to estimate the parameters from multiple observation sequences. In all three cases, it is possible to use either diagonal or full covariance matrices for the state-conditional distributions.

The `H2M/cnt` extension (added in version 2.0) handles similar models but for scalar count (discrete valued positive) data. Three cases have been considered

1. Mixture of Poisson distributions.
2. Hidden Markov models with Poisson state conditional distribution.
3. Hidden Markov models with Negative binomial state conditional distribution.

Compared to the main `H2M` functions, only the case of ergodic models (ie. models that can be trained from a single long observation sequence rather than from multiple sequences) has been considered.

1.2 Current version and changes

There is a list of the changes that have been made since the first version of `H2M` (Mar. 18, 1997) in the file `h2m/doc/CHANGES`. The current version number is 2.0. Since version 1.3, the code for the multivariate Gaussian case has been pretty stable and changes have mostly concerned the documentation and examples.

The `H2M/cnt` extension appeared only in version 2.0.

1.3 License and warranty

You should read the `h2m/LICENSE` file (especially if you plan selling this code to the MathWorks - or to someone else - or suing me because it doesn't work).

1.4 MATLAB/OCTAVE compatibility

`H2M` functions where originally developed using MATLAB V4 and then MATLAB V5, they will thus run on any currently available version of MATLAB. The only exception is the mex-file `c_dgaus.c` which uses the mex-specific V5 syntax (the obsolete V4 lines are commented out) - if you are using the MATLAB compiler `mcc` however you shouldn't need that anyway (see section 2.5.3).

The `H2M` functions have also been tested using OCTAVE (the GNU alternative to MATLAB), version 2.0.13. It basically works as long as you are using the `--traditional` switch. The sub-directory `octave` contains a minimal set of MATLAB compatibility routines needed to run the three examples `ex_basic`, `ex_bic` and `ex_sprec` (remember to append this sub-directory to your loadpath using the `path` command if you are using OCTAVE). The only significant difference is that OCTAVE does not allow for sparse matrices.

The `H2M/cnt` functions have been tested under version 2.0.16 of OCTAVE (always using the `--traditional` switch). Note that the simulation routines (`pm_gen`, `ph_gen` and `nbh_gen`) do require some functions that have been introduced in version 2.0.14 of OCTAVE (`poisson_rnd` and `gamma_rnd` for simulating random deviates). The `H2M/cnt` functions naturally run without problem using MATLAB V5 but you will need the Statistics toolbox which contains the `poissrnd` and `gammrnd` functions in order for the simulation routines to work (see section 4.2.2 for other possible solutions).

1.5 FAQ

Here are the answers to some questions I have been asked several times (hope this helps):

Q1: Why does the log-likelihood takes positive values? For Gaussian state conditional densities, the likelihood is a value from a probability density function (pdf) and should thus not be interpreted as a probability (it can take any positive values, not only those smaller than 1). For such models, obtaining log-likelihood values greater than 0 is certainly not a bug. It is easy to see that if you scale your observations by, say, 10, and modify the parameters of the model accordingly (divide the means by 10 and the variances by 10 to the square), then the likelihood is multiplied by `10^(T*p)` (the log-likelihood is increased by `T*p*log(10)`) where `T` is the number of observations and `p` their dimension.

Q2: Does `H2M` has some limits on the dimension or on the scaling of the data? The dynamics of the Gaussian pdf increases steadily with the dimension, so that if you are in large dimension and try to compute the value of the Gaussian pdf far from the mean vector (given the covariance matrix) you are likely to obtain 0. But this does not prevent the use of `H2M` tools in large dimension (vectors of dimension 10 to 50), the only thing is that the Gaussian parameters should not be initialized too far from "plausible values". Remember that `exp(-40^2/2)` will already be rounded to zero (in double precision) so that these type of problems indeed occur. The way the problem usually appears is that you get the message "Warning: Divide by zero" caused by the following line of `hmm_fb` (or the equivalent line in `mix_post`):

```
alpha(i,:) = alpha(i,:) / scale(i);
```

because all values of `alpha` for some time index `i` are null. If this occurs check your initialization (are the mean vectors centered on the data?) and try increasing the variances (especially useful if you have outliers).

If you work with very large dimensional mixture parameters (dimension 100 or more), you can use `mix_postl` instead of `mix_post` which is robust with respect to the problems mentioned above (albeit, with an increase computational cost, as the unormalized posterior probabilities are first evaluated on a log-scale).

Q3: Does `hmm_mint` operate well? It is important to understand that `hmm_mint` does not correspond to an "optimal" initialization (if there is one). It is just an heuristic commonly used in speech processing which consist in chopping each parameter sequence into `N` segments of equal length where `N` is the number of states. It will clearly not work if you have many states, many allowed transitions (ie. many entries of the transition matrix not initialized to zero) and few training sequences.

Q4: Is it possible to use discrete (eg., vector quantized) observations? No, `H2M` implements continuous observations as described in section 6.6 of [3]. The so called "discrete HMM" which requires prior quantization of the data using VQ (this is the one considered in section 6.4 of [3] as well as in the first part of the tutorial by Rabiner [4]) can not be implemented using `H2M` (note that these models are not really used anymore nowadays).

Q5: Is it possible to use mixture conditional densities? Not directly. If you think of it though, you will realize that an HMM with `N` states and mixtures of `K` Gaussian densities as state conditional distributions is equivalent to an HMM with `N*K` states with some constraints on the transition matrix. There is however two limitations in using `H2M` for that purpose: (1) you will have to modify the EM re-estimation formulas to take into account the constraints on the transition matrix (should not be too difficult), (2) you will rapidly have to deal with huge transition matrices.

Q6: Why does the covariance matrices becomes ill-conditioned? Problems with covariance matrices nearly always stem from the fact that there are too many HMM states compared to the available training data. In these conditions it is possible that outliers states which are associated to only one (or to a few) data points (a problem also experienced with vector quantization) will appear during the training. Using fewer states and/or more training data usually solve the problem. You may also switch to diagonal covariance matrices if you are using full covariance matrices. In any case you should read section 2.5.2 on heuristics which may prevent this problem from happening.

Q7: In the first example of `ex_basic`, why does the initial distribution converges to a degenerate value? It is easilly checked that the recursion

$$\pi_0^{(k+1)}(i) = P^{(k)}(S_1 = i|X_1) = \frac{\pi_0^{(k)}(i)P(X_1|S_1 = i)}{\sum_j \pi_0^{(k)}(j)P(X_1|S_1 = j)}$$

converges to a value of π_0 such that $\pi_0(i) = 1$ for one of the states indices (this will typically happen when trying to estimate mixture weights from a single observation). By looking at the values of the first line of the arrays `\alpha` and `beta`, it is easily verified that `\alpha` is, by large, the dominant factor so that $P(S_1 = i|X_1, \ldots, X_T) \approx P(S_1 = i|X_1)$. Hence the EM recursion for π_0, $\pi_0^{(k+1)}(i) = P^{(k)}(S_1 = i|X_1, \ldots, X_T)$, behaves as the recursion discussed above. This is of course due to the choice of a random transition matrix `A_sim` together with widely separated Gaussian components. Remember anyway that when observing only one sequence of observations, estimation of the initial probability distribution π_0 is basically hopeless (see discussion on p. 361 of [5]). If you really intend to estimate π_0, then you must have several (independent) observations sequences available.

2 Models with multivariate Gaussian state conditional distribution

2.1 Data structures

No specific data structures have been used, so that a HMM with multivariate Gaussian state conditional distribution consists of:

pi0 Row vector containing the probability distribution for the first (unobserved) state: $\pi_0(i) = P(S_1 = i)$.

A Transition matrix: $a_{ij} = P(S_t + 1 = j|S_t = i)$.

mu Mean vectors (of the state-conditional distributions) stacked as row vectors, such that `mu(i,:)` is the mean (row) vector corresponding to the `i`-th state of the HMM.

Sigma Covariance matrices. These are stored one above the other in two different way depending on whether full or diagonal covariance matrices are used: for full covariance matrices,

```
Sigma((1+(i-1)*p):(i*p),:)
```

(where `p` is the dimension of the observation vectors) is the covariance matrix corresponding to the `i`-th state; for diagonal covariance matrices, `Sigma(i,:)` contains the diagonal of the covariance matrix for the `i`-th state (ie. the diagonal coefficients stored as row vectors).

For a left-right HMM, `pi0` is assumed to be deterministic (ie. `pi0 = [1 0 ... 0]`) and `A` can be made sparse in order to save memory space (`A` should be upper triangular for a left-right model). Using sparse matrices is however not possible if you want to compile your m-files using `mcc` (MATLAB) or if you are using OCTAVE.

A Gaussian mixture model, is rather similar except that as the underlying jump process being i.i.d., `pi0` and `A` are replaced by a single row vector containing the mixture weights `w` defined by $w(i) = P(s_t = i)$.

Most functions (those that have `mu` and `Sigma` among their input arguments) are able to determine the dimensions of the model (size of observation vectors and number of states) and the type of covariance matrices (full or diagonal) from the size of their input arguments. This is achieved by the two functions `hmm_chk` and `mix_chk`.

For more specialized variables such as those that are used during the forward-backward recursions, I have tried to use the notations of L. R. Rabiner in [4] (or [3]) which seem pretty standard:

alpha Forward variables: $\alpha_t(i) = P(X_1, \ldots, X_t, S_t = i)$.

beta Backward variables: $\beta_t(i) = P(X_{t+1}, \dots, X_T | S_t = i)$.

gamma A posteriori distributions of the states:

$$\gamma_t(i) = P(S_t = i | X_1, \dots, X_T)$$

I have also tried to use systematically the convention of multivariate data analysis that the matrices should have "more rows than columns", so that the observation vectors are stacked in `X` as row vectors (the number of observed vectors being usually greater than their dimension). The same is true for `alpha`, `beta` and `gamma` which are `T*N` matrices (where `T` is the number of observation vectors and `N` the number of states).

2.2 Simple types: ex_basic

The script `ex_basic.m` contains some code corresponding to low dimensional examples of the three basic HMM types that are handled by `H2M`.

2.2.1 Ergodic model with full covariance matrices

Let `X` denote a matrix containing `T` observed vectors, the EM estimation of the parameters take the following form:

```
for i = 1:n_iter
  [alpha, beta, logscale, dens] = hmm_fb(X, A, pi0, mu, Sigma);
  logl(i) = log(sum(alpha(T,:))) + logscale;
  [A, pi0] = hmm_tran(alpha, beta, dens, A, pi0);
  [mu, Sigma] = hmm_dens(X, alpha, beta, COV_TYPE);
end;
```

`COV_TYPE` is just a flag that should be set to 0 when using full covariance matrices and to 1 for diagonal covariance matrices.

Notice that at each step, the log-likelihood is computed from the forward variables using a correction term `logscale` returned by `hmm_fb` (for forward-backward) which contains the sum of the logarithmic scaling factors used during the computation of `alpha` and `beta`. In the present version scaling of the forward variable is performed at each time index `t = 2:T` (which means that each row of the `alpha` matrix sums to one, except the first one). This systematic scaling appears to be much safer when using input data with largely varying range. The backward variable is scaled using the same normalization factors as indicated in [4] (using exactly the same normalization factors is important for the re-estimation of the coefficients of the transition matrix). Note that the mere suppression of the scaling procedure would lead to numerical problems in almost every cases of interest (when the length of the observation sequences if greater than 50 for instance) despite the double precision representation used in MATLAB/OCTAVE.

If you don't want to see what's going on you can simply use

```
[A, pi0, mu, Sigma, logl] = hmm(X, A, pi0, mu, Sigma, n_iter);
```

which calls the very same piece of code except for the fact that the messages concerning the execution time are suppressed: all the computational functions print those messages by default but this can be suppressed by supplying and optional argument (named `QUIET`) different from zero.

2.2.2 Left-right HMM

This case is not really different from the last one except that the case of multiple observation sequences has been considered:

```
for i = 1:n_iter
  [A, logl(i), gamma] = hmm_mest(X, st, A, mu, Sigma);
  [mu, Sigma] = mix_par(X, gamma, COV_TYPE);
end;
```

In this case, the matrix `X` contains all the observation sequences and the vector `st` yields the index corresponding to the beginning of each sequence so that `X(1:st(2)-1,:)` contains the vectors that correspond to the first observation sequence, and so on until `X(st(length(st)),length(X(1,:),:)` which corresponds to the last observation sequence.

The transition parameters are re-estimated inside `hmm_mest` and the a posteriori distribution of the states are returned in `gamma`. Once again, if you don't want to see what's happening you could more simply use

```
[A, mu, Sigma, logl] = hmm(X, st, A, mu, Sigma, n_iter);
```

In fact, if you need to estimate the parameter of an ergodic model using multiple (independent) observation sequences, you may use `hmm_mest` as well, but `hmm_mest` assumes that `pi0 = [1 0 ... 0]` (ie. that the Markov chain starts from the first state).

2.2.3 Mixture model

The EM estimation in the case of mixture model is achieved through

```
for i = 1:n_iter
  [gamma, logl(i)] = mix_post(X, w, mu, Sigma);
  [mu, Sigma, w] = mix_par(X, gamma, COV_TYPE);
end;
```

or, more simply

```
[w, mu, Sigma, logl] = mix(X, w, mu, Sigma, n_iter);
```

2.3 Another mixture example: ex_bic

`ex_bix` is a richer example with mixtures where you analyze some simulated data consisting of mixture observations plus a small portion of outliers (simulated from another density). To assess the fit provided by mixture models with different numbers of components, the BIC (Bayesian Information Criterion) is used [6]. The aim of this example is not to promote the use of BIC for selecting the number of component in a mixture model (see the comments in `ex_bix` concerning the sensitivity of the procedure to various parameters of the analysis) but rather to illustrate the use of `H2M` functions in a more "realistic" setting.

2.4 A more advance example (sequence recognition with HMM): ex_sprec

The script `ex_sprec.m` contains a more elaborate example of the use of `H2M` functions for an isolated words speaker-dependent recognition task (with no recording variability). I included this example because it provided a simple way of answering most of the question I was asked concerning `H2M` (please, don't ask me about speech recognition anymore!). **I do however insist on the following: (1) `ex_sprec.m` is not meant to be used for real-world speech recognition (see section 2.5.3 on computation time and memory usage), (2) I am not a speech recognition expert and thus cannot answer questions concerning this particular use of `H2M` functions**.

This is a very simplistic example where basic models (left-right models with five states and a single shared covariance matrix) are sufficient to get rid of all errors on a small size training set (which is again not at all typical of real-world speech recognition applications). The code is commented so that it should not be too difficult to figure out what's going on.

If you run into trouble, it will probably happens when reading the signal file `data/digits.sig` which is a binary file contains shorts (16 bits signed integers) recorded on a *little-endian* system (PCs running Windows or Linux are typically little-endian systems, most Unix workstations are big-endian). I have successfully run `ex_sprec` on various Unix systems (little *and* big-endian) using either MATLAB V5 or OCTAVE 2.0, so I hope to have cleared all the input/output problems. The script tries to determine which of MATLAB or OCTAVE is used so as to add the `octave` subdirectory to the search path if needed.

Running `ex_sprec` requires about 6 Mb of memory, or 15 Mb if you have to overwrite the signal file (which is my own stupid way of coping with old versions of MATLAB or OCTAVE running on big-endian systems). On my own Linux box (Pentium III - 1 GHz hardarwe), the computation time for `ex_sprec` (parameterization, training and recognition) ranges from 48.5 s under OCTAVE to 9.1 s using MATLAB with all computational routines compiled.

2.5 Implementation issues

2.5.1 Initialization

Initialization plays an important part in iterative algorithms such as the EM. Usually the choice of the initialization point will strongly depend on the application considered. I use only two very basic methods of initializing the parameters:

For left-right models `hmm_mint` initializes all the model parameters using a uniform "hard" segmentation of each data sequence (each sequence is splitted in `N` consecutive sections, where `N` is he number of states in the HMM, the vectors thus associated with each state are used to obtain initial parameters of the state-conditional distributions).

For mixture models `svq` implements a binary splitting vector quantization algorithm. This usually provides efficient initial estimates of the parameters of the Gaussian densities. Note that `svq` uses the unweighted Euclidean distance as performance criterion. If the components of the input vectors are strongly correlated and/or of very different magnitude, it would be preferable to use `svq` on the vectors $\mathbf{\Phi x}_t$ where $\mathbf{\Phi}$ is the Cholevski factor associated with $\mathrm{Cov}^{-1}(\mathbf{x})$, ie. $\mathbf{\Phi}'\mathbf{\Phi} = \mathrm{Cov}^{-1}(\mathbf{x})$.

2.5.2 Modifications of the EM recursions

It is common practice to introduce some modifications of the EM algorithm in order to avoid known pitfalls. The fact that the likelihood becomes infinite for singular covariance matrices is maybe the problem most often encountered in practice. Solutions include thresholding the individual variances coefficients in the diagonal case or adding a constant diagonal matrix at each iteration. This is certainly useful, particularly in the case where there is "not enough" training data (compared to the complexity of the model). There is however a risk of modifying the properties of the EM algorithm with such modifications, in particular *the likelihood may decrease at some iteration.* A perhaps more elegant way of handling such problems consists in using priors for the HMM parameters in a Bayesian framework [7].

Most HMM packages such as HTK (popular development tool for speech processing applications) use such modifications in order to avoid these problems (the same is true for vector quantization where heuristics can be introduced to avoid the appearance of singleton clusters). No such modification has been used here but these are easy to code using something like:

```
for i = 1:n_iter
  [A, logl(i), gamma] = hmm_mest(X, st, A, mu, Sigma);
  [mu, Sigma] = mix_par(X, gamma, DIAG_COV);
  Sigma = Sigma + SMALL_VALUE*ones(size(Sigma)); % EM Modif. (assuming
                                                 % diagonal covariances)
end;
```

Another frequently used solution is to "share" some model parameters (ie. to constrain them to be equal) such as the variances of different states. This usually necessitates only minor modifications of the EM re-estimation equations [8] - see the function `ex_sprec` (section 2.4) for an example of variance sharing.

Finally the aforementioned HTK toolkit uses a modification of the HMM model which force the forward-backward recursions to give null probability to sequences that do not end in the final state (these modified equations are obtained simply by assuming that each observed sequence is terminated by an END_OF_SEQUENCE symbol associated with a terminal state located after the actual final state of the HMM). This doesn't modify much the EM estimates, except in case where very few training sequences are available (moreover this is clearly limited to left-right HMMs).

2.5.3 Computation time and memory usage

Each function is implemented in a rather straightforward way and should be easy to read. In some case (such as `hmm_tran` for instance) however, the code may be less easy to read because of aggressive "matlabization" (vectorization) which helps save computing time. Note that one of the most time-consuming operation is the computation of Gaussian density values (for all input vectors and all states of the model). In the case I use more frequently (Gaussian densities with diagonal covariance), I have included a mex-file (`c_dgaus`) which is used in priority if it is found on MATLAB's search path (expect a gain of a factor 5 to 10 on `hmm_fb` and `mix_post`). Some routines could easily be made more efficient (`hmm_vit` for instance) if someone has some time to do so.

Especially if you are using full covariance matrices or if you can't compile the mex-file `c_dgaus`, these routines can be made much faster by using the MATLAB compiler `mcc` (if you paid for it): I apologize if it looks like a plain commercial, but execution time gets approximately divided by 10 on functions such as `hmm_fb` when using `mcc`. In the four components 2-D mixture model (last example in script `ex_basic.m`) with 50 000 (fifty thousand) observation vectors, the execution time (on an old fashioned 1998 SUN SPARC workstation) for each EM iteration was: 3 seconds when using diagonal covariance with the mex-file `c_dgaus`; 3 minutes when using full covariance; 10 seconds for full covariance matrices when the files `mix_post` and `mix_par` had been compiled using MATLAB's `mcc`. Only the ratios between these figures are actually of interest since these should run faster on modern computers (with a Pentium III - 1 GHz PC running Linux, the full covariance case, without compilation, boils down to 25 seconds).

Users of the MATLAB compiler should compile in priority the file `gauseval` (computation of the Gaussian densities) which represents the main computational load in many of the routines. Compiling the high-level functions like `mix`, `hmm` (and `vq`) fails because I used variable names ending with a trailing underscore to denote the updated parameters (sorry for that!) It wouldn't be very useful anyway since only the compilation of the low-level functions significantly speeds up the computation. Note that functions compiled with `mcc` can't handle sparse matrices, which is a problem for left-right HMMs (for this reason, I don't recommend compiling a function like `hmm_fb`). Finally, in version 5.2 (the first MATLAB V5 version on which the compiler actually runs) it is possible to use compilation pragmas (or to specify them as arguments of `mcc`). Using the pragma `#inbounds` is an absolute requirement if you want to obtain any gain in execution time and to avoid memory overflows (`#realonly` also helps but to a much lesser extent - moreover `mcc` does not seem to handle this properly in the version I use which is 5.2.0.3084). I have included these pragmas when possible in `gauseval.m`, `gauslogv.m` `mix_par.m` and `mix_post.m`.

Memory space is also a factor to take into account: typically, using more than 50 000 training vectors of dimension 20 with HMMs of size 30 is likely to cause problems on most computers. Usually, the largest matrices are `alpha` and `beta` (forward and backward probabilities), `gamma` (a posteriori distribution for the states) and `dens` (values of Gaussian densities). The solution would consists in reading the training data from disk-files by blocks... but this is another story!

3 The `H2M/cnt` extension: models for scalar count data

The `H2M/cnt` extension contains a set of functions for estimation of mixture and hidden Markov models appropriate for count data (positive, discrete valued time series). Currently implemented are functions for the EM estimation of Poisson mixtures and HMMs as well as HMMs with Negative Binomial state conditional distributions.

> **If you are only interested in speech processing, then you can safely skip this section (and delete the `cnt` subdirectory if you want).**

3.1 Nomenclature

The names of the functions in the `H2M/cnt` extension follow these conventions:

p* **P**oisson something...

pm_* function for **P**oisson **M**ixture modeling

ph_* function for **P**oisson **H**idden Markov modeling

nb* **N**egative **B**inomial something..

nbm_* function for **N**egative **B**inomial **M**ixture modeling

nbh_* function for **N**egative **B**inomial **H**idden Markov modeling

The meaning of the suffix is similar to what has been used for the main `H2M` functions:

***_ch** **CHeck**s that the data structure is correct (and return the model dimension)

***_gen** **GEN**erates simulated data from the model

***_em** **EM** estimation of the parameters of the model

***_vit** Maximum a posteriori estimation of the sequence of states using dynamic programming (also known as **VIT**erbi algorithm)

Note that only the high-level function *_em is provided (in the main `H2M` functions, EM estimation is split into several routines which can used separately such as `gauseval`, `hmm_fb`, `hmm_tran` and `hmm_dens`).

3.2 Data structures

3.2.1 Poisson mixture model

A Poisson mixture model is defined by the two (1-D) arrays:

wght Mixture weights (positive and sum to 1)

rate Component rates (parameters of the Poisson distributions)

3.2.2 Poisson HMM

A Poisson HMM is defined by

TRANS The transition matrix of the hidden chain

rate Component rates (parameters of the Poisson distributions)

3.2.3 Negative binomial mixture model

A negative binomial mixture model is defined by three (1-D) arrays:

wght Mixture weights (positive and sum to 1)

alpha Component (positive) shape parameters

beta Component (positive) inverse scales

3.2.4 Negative binomial HMM

A negative binomial HMM is defined by

TRANS The transition matrix of the hidden chain

alpha Component (positive) shape parameters

beta Component (positive) inverse scales

The Negative Binomial distribution is such that

$$\mathrm{P}(N = n) = \begin{pmatrix} n+\alpha-1 \\ \alpha-1 \end{pmatrix} \left(\frac{\beta}{1+\beta}\right)^{\alpha} \left(\frac{1}{1+\beta}\right)^{n} \qquad \text{for} \quad n \in \mathbb{N}$$

which has mean α/β and variance $\alpha(1+\beta)/\beta^2$. The negative binomial distribution can be viewed as a Poisson (continuous) mixture for which the rate follows a Gamma(α, β) distribution (this is the method used for simulating from the model in `nbh_gen`). If you don't know what the negative binomial distribution is, you should refer, for instance, to [9] or perhaps to [10].

3.2.5 Note on the initial distribution of HMMs in `H2M/cnt`

Contrary to what was the case for the `H2M` main functions, the `H2M/cnt` HMM functions are mostly intended to deal with ergodic models which are estimated from a single long observation sequence (whereas left-right HMMs such as those used in speech processing need to be trained using multiple observation sequences). With a single (long) training sequence, the initial distribution is a parameter that has little influence and that cannot be estimated consistently. Taking this into account, it is assumed that the initial distribution (usually called `pi0` in the `H2M` functions is uniform (equal probabilities for all states of the model).

3.3 Examples

The file `ex_cnt` contains examples of use of the `H2M/cnt` functions on simulated data for the three models:

1. Poisson mixture
2. Poisson hidden Markov model
3. Negative-binomial hidden Markov model

All the models considered in `ex_cnt` have two states. The script tries to figure out which of OCTAVE or MATLAB is used so as to make sure that the plots do not look too bad.

4 Reference

4.1 Functions in the main directory

4.1.1 Alphabetical list of functions

c_dgaus Computes a set of multivariate normal density values in the case of diagonal covariance matrices (mex-file).

ex_1d A simple 1 Dimensional Gaussian HMM example (3 states)

ex_basic How to use the h2m functions on the three basic model types:

ex_bic Example of cluster analysis with varying number of mixture

ex_fb_bms Script to check that the results of Forward-Backward and and Backward Markovian Smoothing are the same.

ex_sprec (Unrealistic) example of isolated word recognition

gauselps Plots 2D projections of Gaussian ellipsoids.

gauseval Computes a set of multivariate normal density values.

gauslogv Computes a set of multivariate normal log-density values.

hmm Performs multiple iterations of the EM algorithm.

hmm_chk Checks the parameters of an HMM and returns its dimensions.

hmm_dens Reestimates the Gaussian parameters for an HMM.

hmm_fb Implements the forward-backward recursion (with scaling).

hmm_gen Generates a sequence of observation given an HMM.

hmm_mest Reestimates the transition parameters for multiple observation sequences.

hmm_mint Initializes the distribution parameters using multiple observations (left-right model).

hmm_post Implements backward Markovian smoothing (forward-backward alternative).

hmm_psim Generates a random sequence of conditional HMM states.

hmm_tran Reestimates the transition part of an HMM.

hmm_vit Computes the most likely sequence of states (Viterbi DP algorithm).

lrhmm Performs multiple iterations of the EM algorithm for a left-right model.

mix Performs multiple iterations of the EM algorithm for a mixture model.

mix_chk Checks the parameters of a mixture model and return its dimensions.

mix_gen Generates a sequence of observation for a Gaussian mixture model.

mix_par Reestimates mixture parameters.

mix_post Computes a posteriori probabilities for a Gaussian mixture model.

mix_postl Alternative to mix_post which uses logarithmic computation to avoid underflows (useful in very large dimensional models).

randindx Generates random indexes with a specified probability distribution.

statdis Returns the stationary distribution of a Markov chain.

svq Vector quantization using successive binary splitting steps.

vq Vector quantization using the K-means (or LBG) algorithm.

4.1.2 Notes

The main functions are described in section 2.2 (or in the example scripts), other functions include:

`hmm_gen` and `mix_gen` generate data vectors according to a given model. This is useful for testing algorithms on "prototype data". `hmm_psim` generates a random sequence of HMM state conditional to an observation sequence. This can be used for doing Monte Carlo simulations (the way it works is described, for instance, in [11] as "sampling the indicator variables").

`gauseval` and `gauslogv` compute values of the Gaussian probability density (or the logarithm of it for `gauslogv`) for several Gaussian distributions and several observed vectors at the same time. Computing as many values as possible at the same time is much faster than calling the function several times (especially when the number of Gaussian distributions is large).

`gauselps` plots the 2-D projections of the Gaussian ellipsoids corresponding to the Gaussian distribution (this is certainly one of the most useful things in order to see what's going on, at least for low dimensional models).

`statdis` computes the stationary distribution of a finite state-space Markov chain from its transition matrix.

4.2 Functions in the `H2M/cnt` extension

4.2.1 Alphabetical list of functions

digamma Computes the digamma (also called psi) function (`d log gamma`).

ex_cnt Script to illustrate the three basic models handled by `H2M/cnt`.

nb_ml Maximum likelihood estimation for negative binomial data.

nbh_chk Checks the parameters of a negative binomial HMM and returns its dimension.

nbh_em Estimates the parameters of a negative binomial HMM using EM.

nbh_gen Simulates data from a negative binomial HMM.

nbh_vit A posteriori sequence estimation for negative binomial HMM.

nbm_chk Checks the parameters of a negative binomial mixture

nbm_em Estimates the parameters of a negative binomial miture using EM.

nbm_gen Simulates data from a negative binomial mixture.

ph_chk Checks the parameters of a Poisson HMM and returns its dimension.

ph_em Estimates the parameters of a Poisson HMM using the EM algorithm.

ph_gen Simulates data from a Poisson HMM.

ph_vit A posteriori sequence estimation for Poisson HMM.

pm_chk Checks the parameters of a Poisson mixture and returns its dimension.

pm_em Estimates the parameters of a Poisson mixture using the EM algorithm.

pm_gen Simulates data from a Poisson mixture.

trigamma Computes the trigamma function (`d^2 log gamma`).

4.2.2 Notes

The simulation routines `pm_gen`, `ph_gen` and `nbh_gen` need functions to generate random numbers from the Poisson and Gamma distributions. For this reason, they must be run either under OCTAVE version 2.014 (or above) or with MATLAB equipped with the Statistics Toolbox.
If you do not fall into one of the two cases above (which probably means that you are using MATLAB but don't want to pay for the Statistics toolbox), you can still get around using the free Statbox toolbox by Gordon K Smyth but you will have to modify the names of the random number generators. Another option, would be to use GSL - The GNU Scientific Library whose recent versions contain a compiled binary named `gsl_randist` which can be used directly for simulating random numbers from the command line (if you are courageous and have a decent C compiler, you can also use the GSL library modules for writing a mex file). Linux users should find this packaged in any recent version of their favorite distribution under the name `gsl` or `gsl-bin` (Debian).

The function `nbh_em` uses a modified version of the EM algorithm in which after the E step, the EM intermediate quantity is maximized explicitly with respect to the `beta` parameters while the `alpha` parameters are updated using a single Newton step. The first and second derivative of the part of the EM intermediate quantity which depends on the `alpha` parameters are computed using the special functions `digamma` and `trigamma`. See [12] for details concerning the convergence of such modified versions of EM.

If you are using OCTAVE, you will also need the `gammaln` m-file (which computes the log of the gamma function) to run `ph_em`, `ph_vit`, `nbh_em` or `nbh_vit`. This function is in the subdirectory `h2m/octave` which you should thus append to your loadpath using the `path` command.

4.2.3 Known problems

To avoid the use of a line search routine, the Newton steps are used in `nb_ml` and `nbh_em` without checking that the objective function indeed increases and that alpha does not become negative. This is of course something that could break down convergence (and at least make the likelihood non strictly increasing from one iteration to the other). Note that it is easily checked though that in both cases, the part of the likelihood or of the EM intermediate quantity which depends only on `alpha` is concave, and thus the situation is rather easy compared to a general optimization task.

In practice problems never seem to happen as long that you have at least a few non null observations and that your starting values are not too crazy when using `nbh_em` (for real data, you can for instance use `nb_ml` to obtain credible initialization values). If you see the message

```
Warning: could not update alpha
```

too often when using `nbh_em` then you are probably in one of those bad cases and should check your starting values (the other option is that you have very few data points - say less than 10 - and/or that these are almost all zero, in which case there is not much to be expected from estimating the parameters anyway).

5 To do

What follows is a remark that could be useful for people needing to adapt the code to a new type of model rather than an actual wishlist

since I don't plan to to this myself at the moment.

If you take a look at the code in `nbh_em`, `ph_em` or `pm_em` and compare these with the corresponding functions for multivariate Gaussian state conditional distributions (`mix`, `hmm`, etc) you will se that they are all very similar and that one could indeed develop generic routines for EM estimation of mixtures and HMMs. The only model specific elements that are needed are

1. A routine that compute the density values given the data and the parameters characterizing the different model states (this is done by the function `gauseval` for multivariate Gaussian distributions).

2. A routine that updates the parameters of the state conditional distributions given the a posteriori state probabilities (this is done by the `hmm_dens` for multivariate Gaussian distributions). This second function can be somewhat more difficult to implement since one needs to consider the precise form of the EM intermediate quantity, but usually this just boils down to a straightforward modification of the maximum likelihood computation in the corresponding model (see, for instance, the end of `pm_em` for the case of Poisson distributions).

Everything else (including the forward-backward recursions and the reestimation of the transition matrix and, if needed, of the initial states - that is functions such as `hmm_fb`, `hmm_vit`, `hmm_tran` or `mix_post`) is absolutely generic.

For doing this with MATLAB/OCTAVE, one would have to use `eval` statements (so as to allow passing the names of the above two routines as parameters) as well as variable number of input arguments (since all distributions do not have the same number of parameters). The latter could be done easilly in MATLAB (starting from V 5) using the `varargin` construct.

6 Downloading `H2M`

(Updated on 28 Aug 2012)

`H2M` is available as a Unix gz-compressed tape archive at
http://perso.telecom-paristech.fr/~cappe/Code/H2m/h2m.tgz

A PC zip file `h2m.zip` is also available in the same directory. The size of both archives is about 1.2 Mb (two third of which corresponding to the data file used to run the example `ex_sprec` of section 2.4). The `.tgz` and `.zip` archives also contain the documentation in HTML and PDF formats.

The permanent adress for `h2m` is
http://perso.telecom-paristech.fr/~cappe/Code/H2m.html

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. B*, 39(1):1–38 (with discussion), 1977.

[2] C. F. J. Wu. On the convergence properties of the EM algorithm. *Ann. Statist.*, 11(1):T95–103, 1983.

[3] L. R. Rabiner and B-H. Juang. *Fundamentals of speech recognition.* Prentice-Hall, 1993.

[4] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–285, February 1989.

[5] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models.* Springer, 2005.

[6] C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. Technical Report 329, University of Washington, Department of statistics, 1998.

[7] J-L. Gauvain and C-H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of Markov chains. *IEEE Trans. Speech Audio Process.*, 2(2):291–298, April 1994.

[8] O. Cappé, C. Mokbel, D. Jouvet, and E. Moulines. An algorithm for maximum likelihood estimation of hidden Markov models with unknown state-tying. *IEEE Trans. Speech Audio Process.*, 6(1):61–70, January 1998.

[9] N. L. Johnson and S. Kotz. *Discrete Distributions*, volume 2. Wiley, 1969.

[10] J. Grandell. *Mixed Poisson Processes.* Chapman & Hall, 1997.

[11] C. K. Carter and R. Kohn. On Gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994.

[12] X-L. Meng and D. B. Rubin. Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika*, 80(2):267–278, 1993.