The GTM Toolbox - User's Guide

Markus Svensén
Neural Computing Research Group
Aston University, Birmingham B4 7ET, UK
svensen@cns.mpg.de

October 4, 1999
Version 1.01

Abstract

This is the User's Guide to the GTM Toolbox — a set of MATLAB functions and scripts that implements and demonstrates the *generative topographic mapping*, a method for density modelling, dimensionality reduction and data visualisation. This document gives a brief description of the GTM, the content of the toolbox and what is required to use it. It describes how to use the toolbox and provides a section of practical advice and tips. Finally, it contains a reference section for the functions and scripts in the toolbox.

Contents

Preface

Acknowledgement

Although I have produced almost all of the code and documentation of this package myself, the development work has been carried out in close co-operation with my supervisors, Prof. Christopher M Bishop and Dr Chris Williams. Many other members of the Neural Computing Research Group have given helpful comments and suggestions. In particular, I would like to thank Dr Mike Tipping and Iain Strachan who have both taken a lot of interest in the GTM and this toolbox, and contributed with valuable ideas and code. Ric Lister patiently helped me to sort out all my (self-invented) LaTeXproblems.

Legal Issues

Information sources

The GTM has a homepage on the Internet at `http://www.ncrg.aston.ac.uk/GTM`, from which you can obtain this toolbox with associated documentation, various data sets, papers related to the GTM, etc.

Chapter 1

Introduction

This section gives a brief introduction to the GTM, the content of this toolbox, and what you need to use it. It is assumed that you have read at least one of the papers on the GTM [Bishop, Svensén, and Williams 1997b; Bishop, Svensén, and Williams 1997a; Bishop, Svensén, and Williams 1996][1].

1.1 What is GTM

GTM, which stands for *generative topographic mapping*, is a mathematical model for density modelling and visualisation [Bishop, Svensén, and Williams 1997b]. It generates a constrained mixture of Gaussians in the data space, which is fitted to the data using a modified form of the EM (expectation-maximisation) algorithm [Dempster, Laird, and Rubin 1977; Bishop 1995]. More specifically, we constrain the model by confining the centres of the mixture to a low-dimensional manifold embedded in the data space. This is achieved by a latent variable model approach, where we map a low-dimensional latent variable space into the data space using a parametric non-linear mapping.

Figure 1.1 shows an example where a discrete sample from a two-dimensional latent variable space, $\mathbf{x}$, is mapped into a three-dimensional data space by a parameterised non-linear mapping, $\mathbf{y}(\mathbf{x}; \mathbf{W})$. Each point in the latent space maps to a corresponding point in the data space and treating each of these points as a centre of a Gaussian, we get a mixture of Gaussians. As far as this release of the toolbox is concerned, we only consider mixtures with equal mixing coefficients and a single variance parameter, common to all mixture components. However, this could easily be extended to more general models.

Note that, although we work with a discrete latent variable sample, the mapping is defined continuously over the latent space, sweeping out a corresponding manifold in the data space, on which the centres of the Gaussians lie.

We alter the shape of the mixture by modifying the parameters of the mapping, $\mathbf{W}$, using an EM algorithm where the M-step is modified to suit our partic-

[1] These papers are available from the GTM homepage, `http://www.ncrg.aston.ac.uk/GTM`.

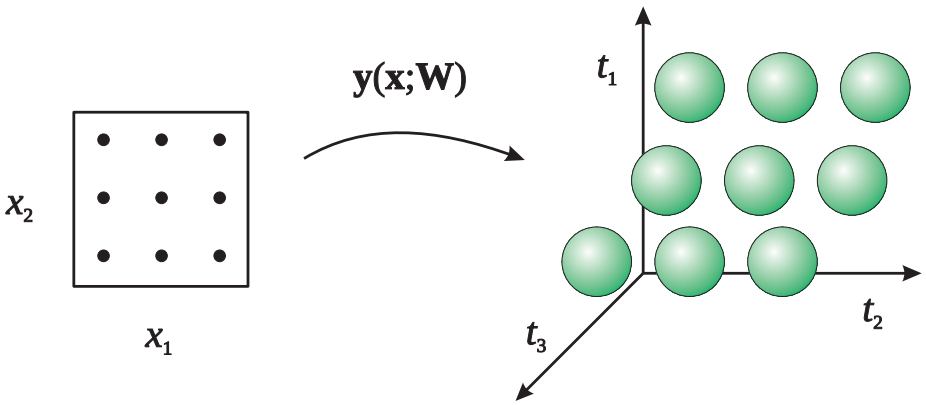

Figure 1.1: The basic idea of the GTM — points in the low-dimensional latent space is mapped to corresponding centres of a Gaussians in the (potentially) high-dimensional data space.

ular choice of mapping. This toolbox uses a generalised linear regression model, which uses a linear combination of fixed non-linear basis functions. This gives significant computational advantages, since the corresponding M-step reduces to a matrix inversion, avoiding the iterative search procedures often associated with the fitting of non-linear models.

An important feature of the GTM is that it can be used for visualisation of data, provided the latent space has no more than two or possibly three dimensions. Since the GTM defines a probability distribution in the data space, $p(\mathbf{t}|\mathbf{x}; \mathbf{W}^*)$ (where $\mathbf{t}$ denotes a point in the data space and $\mathbf{W}^*$ denotes parameters of the model fitted to training data), we can use Bayes' theorem to compute $p(\mathbf{x}|\mathbf{t}_n; \mathbf{W}^*)$, for a data point $\mathbf{t}_n$, which then can be plotted against the latent variable. You can think of it as reversing the mapping to go from data to latent space. Remember, however, that a single data point in the data space will map to a complete distribution over the latent space, not just a single point.

1.2 What does this toolbox contain?

This package provides a set of MATLAB functions which together have all the necessary machinery to generate GTMs and use them for visualisation of data. It has been developed as a part of ongoing research, with the intention of being modular to allow for easy extensions and changes. There are also functions included which are designed to facilitate efficient use and to lower the 'threshold of practicalities'.

At the moment, it comes as a MATLAB implementation accompanied by two short C programs; it does not require any extra toolboxes etc. to run. If you have a C compiler supported by MATLAB, the C files can be compiled into mex-files which are called directly from MATLAB [The MathWorks, Inc. 1993]. This

will give a speed up of the GTM training process, but the package can also be used as a pure MATLAB implementation. There is a demo provided (`gtm_demo`), which shows the GTM in action on a toy problem. The corresponding code forms an example of how some of the other functions can be used.

In terms of documentation, there is this document, which provides a description of the package and how to use it. It does not describe any of the underlying theory. It is important that the users of this package understand its theoretical foundations, which can be achieved by reading the papers published on the GTM model [Bishop, Svensén, and Williams 1997b; Bishop, Svensén, and Williams 1997a; Bishop, Svensén, and Williams 1996]. This document is written with the assumption that the readers are already familiar with the GTM and its associated terminology; it is also assumed that the readers are familiar with MATLAB.

The second part of this document is the reference manual, describing all functions and scripts in the toolbox. This reference documentation is also delivered as a set of html-files, which can be displayed in html-browsers like Netscape and Mosaic.

1.2.1 What's required to use the toolbox?

As has already been said, this package can be used as it is under MATLAB v4.2 (or later). There are, however, requirements in terms of hardware which may impose restrictions on the size of problems you can tackle with the package. In particular, the training algorithm requires significant amounts of memory to be allocated. This is largely due to the fact that the GTM training algorithm provided with this version of the toolbox operates in batch-mode only[2]. Some of the matrices manipulated during training scale like the product of the size of the set of training data and the number of the latent variable points; e.g. with 1000 training data points and 400 latent variable points, the distance and responsibility matrices[3] will have 400 000 elements, requiring over 3 megabytes of storage each, with 64 bit floating point representation. The algorithm is also demanding in terms of CPU usage.

Consequently, you may experience problems if you try to tackle larger scale problems on a machine with limited resources. MATLAB is itself quite demanding in terms of memory and its memory management is geared towards speed of execution rather than limited memory usage; the MATLAB documentation [The MathWorks, Inc. 1992] gives some tips on how to tackle memory problems.

[2] This is an inherent problem with batch-algorithms, arising from the need of storing large amounts of intermediate results. The same problem occurs also with other batch algorithms, e.g. the batch version of the self-organizing map [Kohonen 1995].

[3] These matrices are generated during the training of a GTM, as described in section 2.3.

1.3 The rest of this document

In the next chapter, the first two sections (2.1 and 2.2) describe how to generate and train a GTM, and how to use it for visualisation. The last section (2.3) describes how the toolbox works at a more detailed level, and can be omitted until the methods described in sections 2.1 and 2.2 no longer suffice for your needs.

The last chapter gives some hints on the selection of parameters; thereafter follows the reference section.

Chapter 2

How to use this toolbox

Every session using this package 'from scratch' (i.e. starting with just a data set) will involve two main steps: set-up and training. Set-up refers to the process of generating an initial GTM model, made up by a set of components (MATLAB matrices). Training refers to adapting the initial model to a data set, in order to improve the fit to that data.

Both steps can be more or less automated, depending on which level of detail you want to work. This package contains functions that automate what is believed to be the most common usage procedures. All that these functions do is to call other functions of the package, so by instead doing this by hand you can exercise more control over the exact details of your model.

The first sections of this chapter describe how to set up and train a GTM, and how to use it for visualisation of data. The following section take a look under the bonnet, explaining the steps of set-up and training in greater detail; for convenience we first introduce some notation used in text, figures, examples and also in the MATLAB code; most of it also conforms (more or less) to the notation used in the papers on the GTM. A few self-explanatory variable names are omitted.

Notation

`T` denotes a matrix containing the data we want to model and visualise (stored row-wise).

`X` denotes a matrix containing the latent variable sample points (stored row-wise).

`MU` denotes a matrix containing the positions of the basis functions in the latent space (stored row-wise).

`sigma` denotes a scalar giving the relative width of the basis functions; the absolute width is calculated as `sigma` times the distance between two neighbouring basis function centres.

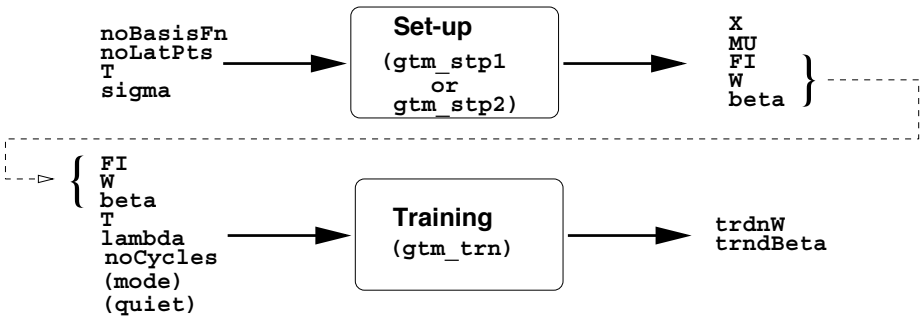

Figure 2.1: The automated set-up and training procedures for generating a GTM. The components to the left and right of the boxes denotes input and output arguments respectively; arguments within parentheses are optional.

`FI` denotes a matrix containing the output the basis functions corresponding to the latent variable sample `X`.

`W` denotes the weight matrix mapping the output of the basis functions to the data space.

`Y` denotes a matrix containing the centres of Gaussian mixture generated in the data space. (`Y = FI*W`)

`beta` denotes a scalar giving the inverse variance of the components of the Gaussian mixture that is generated in the data space.

`lambda` denotes a scalar giving the weight regularisation coefficient used when training the model.

`mode` denotes an integer that selects the mode of calculation (this is discussed further in section 2.3).

`gtm_*` denotes functions of the GTM toolbox, where `*` is replaced by 2–4 letters or digits, to form the name of the function.

2.1 Set-up and training

Figure 2.1 illustrates the procedure for automated set-up and training. The set-up functions (`gtm_stp1` or `gtm_stp2`) take the data set we want to model, along with parameters of the GTM, and generate the initial GTM components, which are subsequently fed into the training function, `gtm_trn`.

These components represent a default initial GTM, utilising a uniformly gridded latent sample (`X`), Gaussian basis functions (`FI`) uniformly gridded over the space of the latent sample, weights (`W`) mapping the outputs of the basis functions corresponding to the latent sample to the first principal component(s)

of the data, and an accordingly chosen value for `beta`. The user chooses the size of the latent sample and the number and relative width of the basis functions.

`gtm_trn` iteratively adapts the parameters `W` and `beta`, each iteration improving the fit to the data, and returns the new values after a given number of cycles. If desired, these new values can be used as input arguments in second call to `gtm_trn`, continuing the training from where it ended.

`gtm_trn` also takes arguments that control the degree of weight regularisation, mode of calculation and echoing of diagnostic information — see the reference section for details.

2.2 Visualisation

The GTM can be used for visualisation of either individual data points or whole sets of data. In the former case, the result is a probability distribution over the latent space; in the latter, we summarise the distributions by their corresponding means or modes. This toolbox contains three corresponding functions, which all return vectors or matrices suited for visualisation using MATLAB's graphics routines.

`gtm_ppd` computes and returns the posterior probability distribution induced over the latent space given a single points in the data space. Depending on the dimensionality of the latent space, it returns either two vectors that can be used with `plot` or three mesh matrices that can be used with MATLAB's routines for 3D graphics, e.g. `pcolor`, `mesh` and `surf`. In the latter case, it assumed that the latent sample was generated using `gtm_stp2`, in order for the 3D graphic routines to work.

`gtm_pmn` takes a whole set of data points, together with the components of a (trained) GTM and returns, for each data point, the mean of the corresponding distribution in the latent space. Using this form of visualisation, one should always bear in mind that the mean might be a poor descriptor of the distribution, e.g. in the case it is multi-modal.

For a multi-modal distribution, the mean is often significantly different from the mode, so we may be able to detect such cases by comparing means and modes. `gtm_pmd` works just like `gtm_pmn`, with the difference that it computes the mode rather than the mean of the posterior distribution.

Example

The example in box 2.1 illustrates the use of the functions described in the previous sections. A file named `Data.mat` is present in the directory where MATLAB is started, containing a data set living in a high-dimensional space.

Figure 2.2 shows the result from the calls to `surfl` and `plot`. Note that for most mean-mode pairs in the right plot, although there may be some discrepancies, these merely indicate that a few nearby mixture components share the responsibility for the corresponding data point.

```
                          < M A T L A B (R) >
               (c) Copyright 1984-94 The MathWorks, Inc.
                          All Rights Reserved
                            Version 4.2c.1
                              Dec 31 1994

Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info, subscribe

>> load Data
>> who

Your variables are:

Data

>> noLatPts = 400; % 20-by-20 latent sample grid
>> noBasisFn = 81; % 9-by-9 basis function grid
>> sigma = 1.5;
>> [X, MU, FI, W, beta] = gtm_stp2(Data, noLatPts, ...
noBasisFn, sigma);
>> lambda = 0.001;
>> cycles = 40;
>> [trndW, trndBeta, llhLog] =  gtm_trn(Data, FI, W, ...
lambda, cycles, beta, 'quiet');
>> [xl, yl, p]  = gtm_ppd(Data(58,:), FI*trndW, ...
trndBeta, X, 20, 20); % 20*20 = 400 = noLatPts
>> surfl(xl, yl, p)
>> means = gtm_pmn(Data, X, FI, trndW, trndBeta);
>> modes = gtm_pmd(Data, X, FI, trndW);
>> plot(means(1:250,1), means(1:250,2), 'o', ...
modes(1:250,1), modes(1:250,2), 'x' ...
) ; % plot only subset for clarity
>> for i = 1:250
plot([means(i,1);modes(i,1)], [means(i,2), modes(i,2)], '-')
end
>>
```

Box 2.1: Transcript of the MATLAB session that produced the plots in figure 2.2. A MATLAB workspace file named `Data.mat` is present in the directory where MATLAB was started.

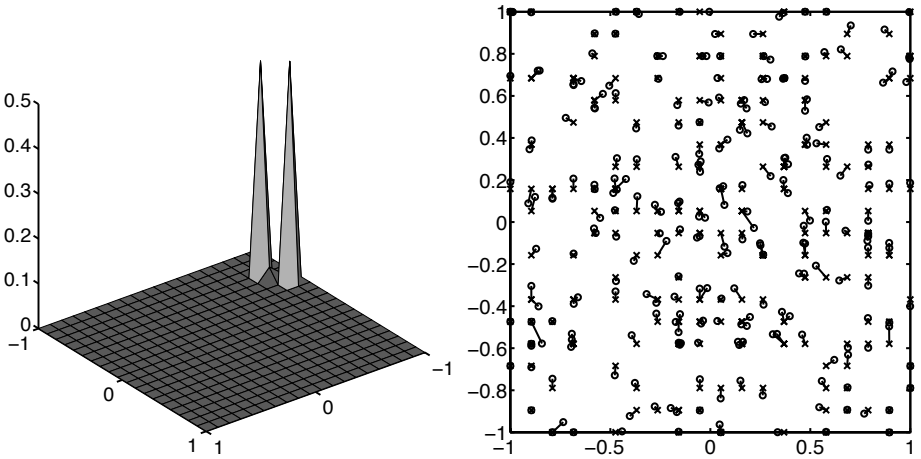

Figure 2.2: Examples of visualisation with the GTM. The left plot shows the posterior distribution over the latent space induced by the 58th data point (which in this case proved to be pretty interesting). The right plot shows the posterior mean ($\circ$) and mode ($\times$) projection of a subset of the data set in the latent space, with corresponding mean and mode points joined by a line.

2.3 Under the bonnet

Rather than using the automated set-up functions, you can generate the necessary initial components 'manually'. This gives you more detailed control over the model, and lets you try out configurations not directly supported by the automated set-up.

Manual set-up essentially means carrying out the steps of the automated set-up functions by hand, so an easy way to get an understanding of what to do is to look at the inner workings of these functions; figure 2.3 gives a pictorial description of what is happening inside `gtm_stp2`. (`gtm_stp1` is very similar — just slightly simpler.)

The procedure consists of four steps:

1. Generate a latent variable sample, `X`.
2. Generate the centres of the basis functions, `MU`.
3. Compute the activations in the basis functions, `FI`, given the latent variable sample.
4. Compute an initial weight matrix, `W`, mapping from the output of the basis functions to the data space, and an initial value for `beta`, the inverse variance of the Gaussian mixture.

Steps 1 and 2 both generate sets of points in the latent space, and hence can both be carried out in the same way. `gtm_stp1` and `gtm_stp2` both generate

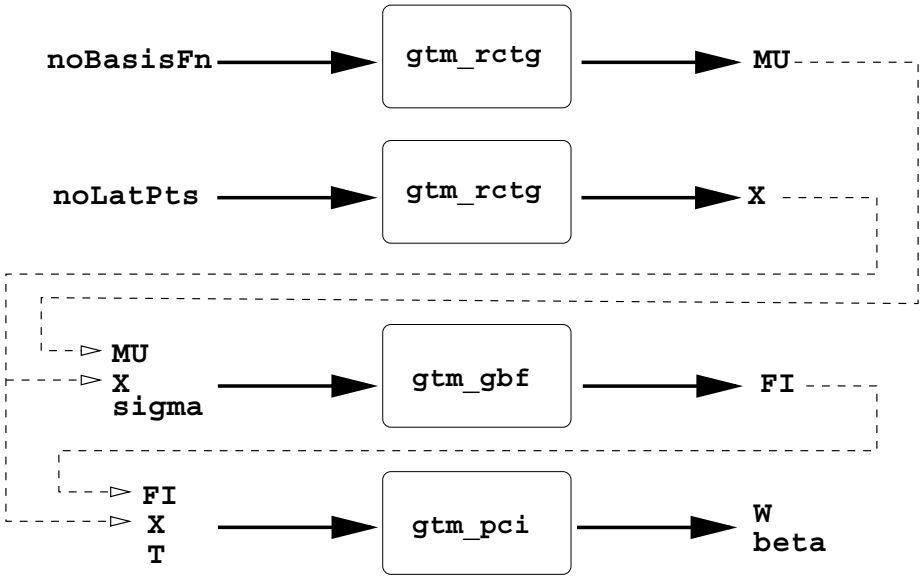

Figure 2.3: The inner workings of `gtm_stp2`, illustrating the steps of creating an initial GTM.

uniformly gridded samples over the latent space; this is trivial in the case of a one-dimensional space. For a two-dimensional latent space, `gtm_rctg` generates a set of points taken from a regular, rectangular lattice over the latent space; `gtm_hxg` does the same thing, but instead uses a hexagonal lattice[1].

Alternatively, these steps could be accomplished using MATLAB's `rand` or `randn` functions, which generate random samples from a uniform and normal distribution respectively. This is generally not recommended, since GTMs generated in exactly the same way could still give slightly different results[2]; moreover, you would not be able to use a (2D) latent variable sample generated this way with MATLAB's functions for 3D visualisation (`mesh, surf, pcolor`, etc.). However, it does provide a fast way of generating GTMs with latent spaces of higher dimension then two; be aware, though, that the number of points required to provide an adequate sample grows exponentially with the number of dimensions of the latent space — a phenomenon known as the *curse of dimensionality* [Bellman 1961; Bishop 1995] — and therefore higher dimensional models are computationally expensive.

Step 3 is carried out using `gtm_gbf`. It returns the activations of a set of spherical Gaussian basis functions whose centres were generated in step 2 and

[1] `gtm_hxg` was developed as part of the study of the relationship between GTM and the self-organizing map [Kohonen 1995]. For the case of the GTM, using a rectangular or hexagonal lattice appears to have small impact on the resulting density model, although in visualisation, the underlying lattice may 'shine through' to some extent.

[2] This could be avoided by initialising the random generator each time before generating an initial GTM.

the width chosen by the user, given the latent sample generated in 1. There is also a function, `gtm_lbf`, which returns the activations from linear basis functions; with a Gaussian latent distribution and linear basis functions, the GTM will implement a constrained variant of factor analysis. However, the use of linear basis functions is not considered further in this document.

For step 4, finally, `gtm_pci` calculates a weight matrix which maps the L-dimensional latent variable to the plane spanned by the L first principal components of the target data, so as to match the mean and variance of the data projected onto this plane. The corresponding value for `beta` is chosen so its *inverse* equals larger of

- half the average distance from one component of the mixture to its nearest neighbour, or
- the $(L+1)$th eigenvalue of the covariance matrix of the data, i.e. the largest variance orthogonal to the L-dimensional hyperplane to which the latent sample is mapped.

This is the initialisation used by `gtm_stp1` and `gtm_stp2`.

Alternatively, `gtm_ri` returns a weight matrix generated at random from an axis-aligned Gaussian distribution (in the weight space), with parameters chosen so that the mean and variance of the set of points generated in target space (taken to be the centres in the Gaussian mixture) roughly match the mean and variance of the data on each dimension; `beta` is set to the average distance between each mixture component and its nearest data point.

2.3.1 The steps of training

Whereas it is reasonably likely that you might want to try set up GTM models by hand, in order to try configurations not catered for by the automated set-up procedures, it is rather unlikely that you ever will want to do 'manual' training. This section is rather intended as additional documentation on the implementation of training algorithm in this toolbox (`gtm_trn`); it should be particularly useful when you want to experiment with, or change this code. The whole training procedure is illustrated in figure 2.4; details of the figure are explained in the following paragraphs.

The training procedure can, initially, be divided into the two steps of the EM-algorithm. In the E-step we calculate a matrix, `R`, containing the responsibilities assumed by each Gaussian mixture component for each of the training data points. These responsibilities are subsequently used in the M-step, for calculating new parameters of the Gaussian mixture, by means of new values for `W` and `beta`.

The E-step can, in turn, be broken down into two sub tasks. `gtm_dist` takes care of the first one: calculating the distances between all mixture components and all training data points. It takes the corresponding two matrices, `T` and `Y`, as input arguments and returns a matrix with all the distances, `DIST`. This is a computationally demanding step; the overall speed of the algorithm

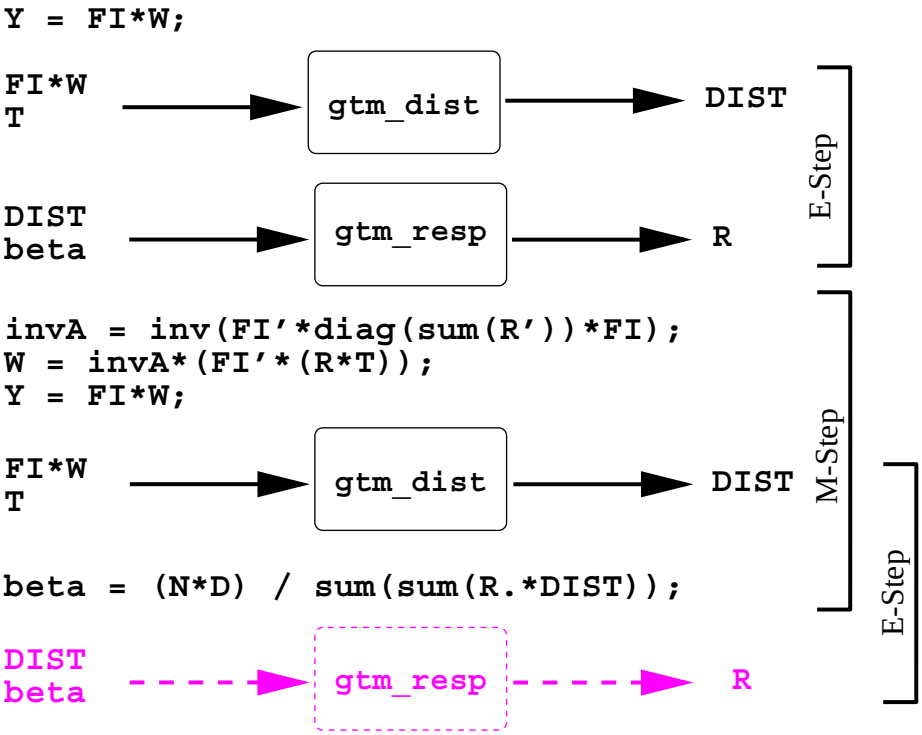

Figure 2.4: A simplified illustration of the steps of training. Note a) how information is flowing down the figure; b) how all E-steps but the first overlaps with the preceding M-step by re-using `DIST`.

can be increased if this calculation is done using an external C or FORTRAN implementation[3].

The `DIST` matrix is used as an argument to `gtm_resp` which calculate responsibilities from the distances, which are returned in a matrix, `R`.

There are optional argument for both `gtm_dist` and `gtm_resp` controlling the mode of calculation — greater accuracy can be obtained at the expense of more calculations. At the moment there are three modes of calculations: 0, 1 and 2, in increasing order of accuracy and required number of operations. No serious attempt has been made to quantify the differences in accuracy, but experience so far point towards very marginal improvements in accuracy, and that the relative largest gain is made going from mode 0 to mode 1. (See code for details.)

As has already been said, the GTM training algorithm is quite demanding in terms of memory, a problem further worsened by the fact that output arguments from functions are passed by value in MATLAB. This leads to problems with repeated calls to `gtm_dist` and `gtm_resp`; each call will allocate return matrices from the memory heap, and if these matrices take sizes in order of megabytes, MATLAB will run out of memory before long[4]. Therefore, `gtm_dist` and `gtm_resp` have sibling functions, `gtm_dstg` and `gtm_rspg` which utilise global variables to pass on their results, rather than normal output arguments. For these functions to work efficiently, the necessary global variables should have been pre-allocated. It is highly recommended that you use `gtm_dstg` and `gtm_rspg` rather than `gtm_dist` and `gtm_resp` inside any kind of loop structures (`gtm_trn` indeed does so). There is a C implementation of `gtm_dstg`; note that this has pre-allocated global variables of correct size as an absolute prerequisite.

The M-step can also be divided into two steps, first maximising with respect to `W`, then with respect to `beta`. The first step corresponds to solving a system of linear equations, i.e. a matrix inversion followed by a matrix multiplication. As the matrix that is inverted is symmetric and often also positive definite, it is worthwhile trying Cholesky decomposition, with the option of resorting to the pseudo-inverse if the matrix would prove to be singular; this normally gives better performance both in terms of speed and accuracy. In figure 2.4 this is just represented by the line `invA = inv(...);`, for the sake of simplicity. Admittedly, the savings in CPU-time are normally marginal, as the matrix inversion only accounts for a small part of total amount of calculations necessary the determine `W`. Most of the time is spent calculating the factor `(FI'*(R*T))`; note that the multiplication should be forced to go right-to-left, which normally requires significantly fewer operations[5].

[3] The difference in speed between the C and MATLAB implementations delivered in the toolbox varies between platforms, but it appears as if the C implementation is faster (if only slightly) in most cases.

[4] One would have hoped that MATLAB would re-use allocated memory upon repeated functions calls, but experience shows that this is not the case.

[5] This is due to the fact that these are not square matrices; 'absorbing' the largest dimension first, which is normally the number of training data points, will minimise the total number

For the maximisation of `beta`, we simply apply the update formula from the papers. Note, however, that the `DIST` matrix should be re-calculated with the new set of weights. This may seem to be a costly update formula, but the new `DIST` matrix can be retained and used for calculating a new matrix of responsibilities (using the updated `beta`) for the next iteration of training.

of operations required. MATLAB does not seem to recognise this fact, but do the calculations left-to-right, regardless of the shape of the matrices involved.

Chapter 3

Practical advice and tips

So far we have covered the procedures involved in building and experimenting with GTMs. This section gives additional guidelines for the usage of GTMs and can hopefully help you to some extent in tackling the problem of parameter setting. The setting of parameters corresponds to choosing a prior distribution over possible models, which generally is a difficult task. However, reasonable assumptions about the distributions we are trying to model give directions for this choice. The GTM already prescribes a fairly restrictive prior, as the density models that can be generated in the data space will necessarily take the shape of low-dimensional manifolds. It also seems reasonable to assume that this manifold is relatively smooth, which in turn means that we should choose a smooth mapping from latent to data space.

The smoothness of the mapping is controlled both by the parameters we choose at set-up time and the ones we set during training, as will be explained in the following two sections. As the choice of priors is inevitable in statistical density modelling, it is an important advantage of the GTM that the relationship between the model parameters and the corresponding prior distributions over density models is relatively straightforward.

3.1 Parameters chosen at set-up time

During set-up you are required to specify the number of latent points and basis functions, together with a width parameter of the basis functions. The smoothness of the mapping is largely determined by the properties of the basis functions. Clearly, a limited number of basis functions will necessarily restrict the possible forms that the mapping can take. Moreover, as basis functions overlap, their response will be correlated — it is this correlation that causes the smoothness of the mapping — which decreases the 'effective' number of basis functions. Hence, more or narrower basis functions will allow a more flexible mapping, while fewer or broader basis functions will prescribe a smoother mapping. Beware that very broad basis functions will cause the matrix of basis

function activations, `FI`, to be ill-conditioned (`gtm_pci` will warn of this), while on the other hand, very narrow basis functions will be close to uncorrelated so the smoothness of the mapping is lost. The choice of parameters must be done individually for each problem you try; experience so far has shown that `sigma` = 1.0 is a good starting point, from were you can search by halving or doubling subsequent values of `sigma` (i.e. searching on a log_2 scale).

As we are working with a finite latent variable sample, the size of this will also affect the final mapping; the effective measure of overlap between basis functions is how many latent points they 'share'. With too few points per basis function — and even fewer shared between neighbouring basis functions — the smoothness of the mapping is lost. Ideally, we would choose the latent variable sample to be very large, but this is computationally prohibitive. A good rule of thumb is to have $O(10^L)$ number of latent points in the support of each basis function, where L is the number of dimensions of the latent variable.

3.2 Parameters chosen for training

There is just one parameter to set for training: the weight regularisation factor, `lambda`. It governs the degree of weight decay applied during training. While the basis function parameters controls the smoothness of the manifold in the data space, the weight regularisation parameter controls the scaling, by restricting the magnitude of the weights.

In practice, because we work with finite number of latent and data points, a small degree of weight regularisation, say 10^{-3}, is generally advisable as this prevents the weights from growing very large, which otherwise could cause the smoothness imposed by the basis function parameter to break down. (In particular, this can be the case when using many broad basis functions.)

In general, it is recommended that prior belief about the smoothness of the mapping is primarily expressed in the choice of basis function parameters, while the weight regularisation parameter is chosen to ensure that this smoothness is maintained, while respecting the overall scale of the training data.

3.2.1 Local maxima

Like the general EM-algorithm, the training algorithm for the GTM suffers from the problem of local maxima. Empirical evidence shows that the PCA initialisation performed by `gtm_pci` (which is used by `gtm_stp1` and `gtm_stp2`) often yields good results, although not optimal. You will be well advised to also try a few random initialisations, once a suitable model architecture has been determined.

Bibliography

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. New Jersey: Princeton University Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bishop, C. M., M. Svensén, and C. K. I. Williams (1996). EM Optimization of Latent-Variable Density Models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems*, Volume 8, pp. 465–471. MIT Press.

Bishop, C. M., M. Svensén, and C. K. I. Williams (1997a). GTM: A Principled Alternative to the Self-Organizing Map. In M. C. Mozer, M. I. Jordan, and T. P. T (Eds.), *Advances in Neural Information Processing Systems*, Volume 9. MIT Press.

Bishop, C. M., M. Svensén, and C. K. I. Williams (1997b). GTM: the generative topographic mapping. Accepted for publication in Neural Computation. Available as NCRG/96/015 from `http://www.ncrg.aston.ac.uk/`.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B* ***39***(1), 1–38.

Kohonen, T. (1995). *Self-Organizing Maps*. Berlin: Springer-Verlag.

The MathWorks, Inc. (1992). *MATLAB User's Guide*. The MathWorks, Inc.

The MathWorks, Inc. (1993). *MATLAB External Interface Guide*. The MathWorks, Inc.

Reference Manual

This part of the document contains reference information about all the functions included with the GTM toolbox. This information is also available via MATLAB's online help and as html-documents.

As these reference pages have been generated automatically from the MATLAB help comments, please excuse any deviations and 'glitches' in the typesetting, as compared to the first part of this document.

gtm_bi

Calculate an initial value for beta.

The value is calculated from the average distance between the nearest neighbours in Y, the centres of the constrained Gaussian mixture generated in the target space from latent sample.

Synopsis

```
beta = gtm_bi(Y)
```

Arguments

`Y` — a matrix containing the positions of the centres of the Gaussian mixture induced in target space from the latent variable samples.

Return

`beta` — an initial value for the inverse variance of the Gaussian mixture

gtm_demo

Demonstrates the GTM with a 2D target space and a 1D latent space.

This script generates a simple data set in 2 dimensions, with an intrinsic dimensionality of 1, and trains a GTM with a 1-dimensional latent variable to model this data set, visually illustrating the training process

Synopsis

```
gtm_demo
```

Notes

The script generates a number of variables which may overwrite variables already existing in the workspace. The generated variables remain in the work space after the script has finished executing.

gtm_dist

Calculate the squared distances between two sets of data points.

This function calculates distances between all data points in the two data sets T and Y and returns them in a matrix.

Synopsis

```
[DIST, minDist, maxDist] = gtm_dist(T, Y, m)
[DIST] = gtm_dist(T, Y)
```

Arguments

`T, Y` — data set matrices in which each row is a data point; dimensions N-by-D and K-by-D respectively

`m` — mode of calculation; iff m > 0, min- and maxDist (below) are calculated; the default mode is 0

Return

`DIST` — matrix containing the calculated distances; dimension K-by-N; DIST(k,n) contains the squared distance between T(n,:) and Y(k,:).

`minDist, maxDist` — vectors containing the minimum and maximum of each column in DIST, respectively; 1-by-N; required iff m > 0.

Notes

This m-file provides this help comment and a MATLAB implementation of the distance calculation. If, however, a mex-file with the same name is present in the MATLABPATH, this will be called for doing the calculation. As this is a computationally demanding step of the algorithm, an efficient mex-file implementation will improve the performance of the GTM training algorithm.

See also

`gtm_dstg`

gtm_dstg

Calculate the squared distances between two sets of data points.

This function calculates distances between all data points in the two data sets T and Y and returns them via the global variable matrix gtmGlobalDIST.

In addition, the minimum and maximum value of each column in gtmGlobalDIST may be calculated and returned via the global variables gtmGlobalMinDist and gtmGlobalMaxDist.

Synopsis

```
gtm_dstg(T, Y, m)
gtm_dstg(T, Y)
```

Arguments

`T, Y` — data set matrices in which each row is a data point; dimensions N-by-D and K-by-D respectively

`m` — mode of calculation; the default mode is m = 0 (see gtmGlobalMinDist/MaxDist below)

Global variables

`gtmGlobalDIST` — Matrix containing the calculated distances; dimension K-by-N; DIST(k,n) contains the squared distance between T(n,:) and Y(k,:). This matrix is assumed to be pre-allocated; if this is not the case, performance deteriorates dramatically

`gtmGlobalMinDist, gtmGlobalMaxDist` — vectors containing the minimum and maximum of each column in DIST, respectively; 1-by-N; calculated iff m > 0.

Notes

This m-file provides this help comment and a MATLAB implementation of the distance calculation. If, however, a mex-file with the same name is present in the MATLABPATH, this will be called for doing the calculation. As this is a computationally demanding step of the algorithm, an efficient mex-file implementation will improve the performance of the GTM training algorithm. A mex-file implementation will have pre-allocated global matrices as an absolute requirement.

See also

`gtm_dist`

gtm_gbf

Calculates the output of Gaussian basis functions for a given set of input

Synopsis

```
FI = gtm_gbf(MU, sigma, X)
```

Arguments

`MU` — a M-by-L matrix containing the centers of the basis functions

`sigma` — a scalar giving the standard deviation of the radii-symmetric Gaussian basis functions,

`X` — the latent variable sample forming the set of inputs; K-by-L

Return

`FI` — the matrix of basis functions output values; K-by-(M+1), "+1" for a bias basis function with a fixed value of 1.0

See also

`gtm_lbf`

gtm_hxg

Produces a 2D grid with points arranged in a hexagonal lattice.

The grid is centered on the origin and scaled so the dimension (X or Y) with largest number of points ranges from -1 to 1.

Synopsis

```
grid = gtm_hxg(xDim, yDim)
```

Arguments

`xDim, yDim` — number of points along the X and Y dimensions, respectively; must be >=2.

Return

`grid` — a (xDim*yDim)-by-2 matrix of grid points with the first point being the top-left corner and subsequent points following columnwise.

See also

`gtm_rctg`

gtm_lbf

Calculates the output of linear basis functions for a given set of inputs

This simply amounts to returning the set of inputs with an extra bias column of ones after the last column in the input set matrix.

Synopsis

```
FI = gtm_lbf(X)
```

Arguments

`X` — the latent variable sample forming the set of inputs; K-by-L

Return

`FI` — the matrix of basis functions output values; K-by-(L+1), "+1" for the bias basis function

See also

`gtm_gbf`

gtm_m2r

Converts from a mesh-matrix to vector representation

Returns a matrix in which each row corresponds to a point on the grid defined by the mesh-matrices X and Y. The enumeration of points goes from the top-left corner of the mesh to the bottom-right, columnwise.

Synopsis

```
cXcYcZ = gtm_m2r(X, Y, Z)
cXcYcZ = gtm_m2r(X, Y)
cXcYcZ = gtm_m2r(X)
```

Arguments

`X, Y, Z` — mesh-matrices for x-, y- and z- coordinates respectively; M-by-N.

Return

`cXcYcZ` — matrix of rows of tuples; (M*N)-by-k, where k is 1, 2 or 3

See also

`gtm_r2m`

gtm_pca

Calculates the principal components of a data set.

The principal components equals the eigenvectors of the covariance matrix of the data..

Synopsis

```
[eVts, eVls] = gtm_pca(T)
```

Arguments

`T` — the data set for which the principal components are to be calculated. Every row is assumed to be a data point; N-by-D

Return

`eVts` — an D-by-D matrix in which each column is a unit length eigenvector of the covariance matrix of the data, sorted in descending order w.r.t. the corresponding eigenvalues

`eVls` — a D-dimensional vector holding the eigen- values of the covariance matrix of the data, sorted in descending order

gtm_pci

Returns a weight matrix initialised using principal components.

The returned weight matrix maps the mean of the latent variable to the mean of the target variable, and the L-dimensional latent variable variance to the variance of the target data along its L first principal components.

An initial value for beta can also be calculated, based on the noise of the data (the "L+1"th eigenvalue) and the interdistances between Gaussian mixture centres in the data space.

Synopsis

```
[W, beta] = gtm_pci(T, X, FI)
W = gtm_pci(T, X, FI)
```

Arguments

`T` — target distribution sample; one data point per row; N-by-D

`X` — the latent distribution sample, K-by-L

`FI` — basis functions' activation when fed the latent data, X, plus a bias, K-by-(M+1)

Return

`W` — the initialised weight matrix, (K+1)-by-D

`beta` — the initial beta value, scalar. This is an optional output argument; if ommitted, the corresponding (rather time consuming) calculations are ommitted too.

Notes

The first dimension (column) of X will map to the first principal component, the second dimension (column) of X will map to the second principal component, and so on. This may be of importance for the choice of sampling density along the different dimensions of X, if it differs between different dimensions

See also

`gtm_ri`

gtm_pmd

Calculates the posterior mode projection of data into the latent space.

The posterior mode projection of a point from the target space, t, is the mode of the correspondig posterior distribution induced in the latent space.

Synopsis

```
modes = gtm_pmn(T, X, FI, W)
```

Arguments

`T` — data points representing the distribution in the target space. N-by-D

`X` — data points forming a latent variable sample of the distribution in the latent space. K-by-L

`FI` — activations of the basis functions when fed X; K-by-(M+1)

`W` — a matrix of trained weights

Return

`modes` — the posterior modes in latent space. N-by-L

See also

`gtm_ppd`, `gtm_pmn`

gtm_pmn

Calculates the posterior mean projection of data into the latent space.

The posterior mean projection of a point from the target space, t, is the mean of the correspondig posterior distribution induced in the latent space.

Synopsis

```
means = gtm_pmn(T, X, FI, W, b)
```

Arguments

`T` — data points representing the distribution in the target space. N-by-D

`X` — data points forming a latent variable sample of the distribution in the latent space. K-by-L

`FI` — activations of the basis functions when fed X; K-by-(M+1)

`W` — a matrix of trained weights

`b` — the trained value for beta

Return

`means` — the posterior means in latent space. N-by-L

See also

`gtm_ppd`, `gtm_pmd`

gtm_ppd

Latent space posterior probability distribution for a given data point.

This function calculates the posterior probability distribution induced in the latent space of a trained GTM model for a given data point, and returns it in a format suitable for MATLAB's 2D or 3D graphic plotting routines, depending on the latent space dimensionality.

Synopsis

```
[xl, yl, p]  = gtm_ppd(t, Y, beta, X, xDim, yDim)
[xl, p]  = gtm_ppd(t, Y, beta, X)
```

Arguments

`t` — a point in the data space; 1-by-D

`Y` — centres of the Gaussian mixture generated by the GTM in the data space, Y = FI*W; K-by-D

`beta` — variance of Gaussian mixture; scalar

`X` — latent sample

`xDim, yDim` — number of points along the 2 dimensions of the latent space meshgrid sample

Return

`xl, yl` — latent sample; if the latent space is 2D, xl and yl are mesh matrices; if it is 1D, xl is identical to X

`p` — posterior distribution over latent space given data point t; if the latent space is 2D, p is a mesh matrix, otherwise it is a vector of same length as xl

Notes

If the latent sample X is 2 dimensional, it is assumed to have been constructed from a mesh-grid, e.g. as if generated by gtm_stp2

See also

`gtm_pmn`, `gtm_pmd`, `gtm_stp2`

gtm_r2m

Convert data from column vector to a mesh-matrix representation

The mesh-matrices are filled columnwise, starting from the top left corner, with the elements from the corresponding column vectors. The exact (cX, cY, cZ) - (X, Y, Z) relationship being:

X(i,j) = cX(meshRows*(i-1)+j)
Y(i,j) = cY(meshRows*(i-1)+j)
Z(i,j) = cZ(meshRows*(i-1)+j)

Synopsis

```
[X, Y, Z] = gtm_r2m(cX, cY, cZ, meshRows, meshCols)
[X, Y] = gtm_r2m(cX, cY, meshRows, meshCols)
X = gtm_r2m(cX, meshRows, meshCols)
```

Arguments

`cX, cY, cZ` — column vectors with x-, y-, and x-data respectively; N-by-1

`meshRows, meshCols` — number of rows and colmuns of the mesh matrices; meshRows*meshCols = N

Return

`X, Y, Z` — mesh matrices; meshRows-by-meshCols

See also

`gtm_m2r`

gtm_rctg

Produces a 2D grid with points arranged in a rectangular lattice.

The grid is centered on the origin and scaled so the dimension (X or Y) with largest number of points ranges from -1 to 1.

Synopsis

```
grid = gtm_rctg(xDim, yDim)
```

Arguments

`xDim, yDim` — number of points along the X and Y dimensions, respectively; must be >=2.

Return

`grid` — a (xDim*yDim)-by-2 matrix of grid points with the first point being the top-left corner and subsequent points following column-wise.

See also

`gtm_hxg`

gtm_resp

Log-likelihood and component responsibilities under a Gaussian mixture

The responsibility R(k,n) is the probability of a particular component in the Gaussian mixture, k, having generated a particular data point, n. It is calculated from the distances between the data point n and the centres of the mixture components, 1..K, and the inverse variance, beta, common to all components.

Synopsis

```
[llh, R] = gtm_resp(DIST, minDist, maxDist, beta, D, mode)
[llh, R] = gtm_resp(DIST, beta, D)
```

Arguments

`DIST` — a K-by-N matrix in which element (k,n) is the squared distance between the centre of component k and the data point n.

`minDist, maxDist` — vectors containing the minimum and maximum of each column in DIST, respectively; 1-by-N; required iff m > 0.

`beta` — a scalar value of the inverse variance common to all components of the mixture.

`D` — dimensionality of space where the data and the Gaussian mixture lives; necessary to calculate the correct log-likelihood.

`mode` — optional argument used to control the mode of calculation; it can be set to 0, 1 or 2 corresponding to increasingly elaborate measure taken to reduce the amount of numerical errors; mode = 0 will be fast but less accurate, mode = 2 will be slow but more accurate; the default mode is 0

Return

`llh` — the log-likelihood of data under the Gaussian mixture

`R` — an K-by-N responsibility matrix; R(k,n) is the responsibility takened by mixture component k for data point n.

Notes

'llh' is put as the first output argument, as 'R' is not of interest in the fairly common task of calculating the log-likelihood of a data set under a given model. This allows for calls like: llh = gtm_resp(...);

See also

`gtm_dist`, `gtm_rspg`, `gtm_dstg`

gtm_ri

Returns an initial random weight matrix.

Generates a weight matrix with the bias weights set to map to the mean of the target distribution and remaining weights drawn at random from a Gaussian distribution with zero mean and variances choosen so that the variances of the generated distribution roughly match the variances of the target distribution.

In addition, an initial value for beta may be calculated as the inverse of the average distance between each Gaussian centre, calculated with the random mapping, and its nearest neighbours in the set of data points.

Synopsis

```
[W, beta] = gtm_ri(T, FI)
W = gtm_ri(T, FI)
```

Arguments

`T` — sample of target distribution, used for calculating the mean; one data point per row; N-by-D

`FI` — basis functions' activations when fed the latent data, X, plus a bias, K-by-(M+1)

Return

`W` — the initialised weight matrix; (M+1)-by-D

`beta` — the initial beta value, scalar. This is an optional output argument; if omitted, the corresponding calculations are omitted too.

See also

`gtm_pci`

gtm_rspg

Log-likelihood and component responsibilities over a Gaussian mixture

The responsibilities are returned via the global variable matrix gtmGlobalR. The responsibility gtmGlobalR(k,n) is the probability of a particular component in the Gaussian mixture, k, having generated a particular data point, n. It is calculated from the distances between the data point n and the centres of the mixture components, 1..K, and the inverse variance, beta, common to all components.

Synopsis

```
llh = gtm_rspg(beta, D, mode)
llh = gtm_rspg(beta, D)
```

Arguments

`beta` — a scalar value of the inverse variance common to all components of the mixture.

`D` — dimensionality of space where the data and the Gaussian mixture lives; necessary to calculate the correct log-likelihood.

`mode` — optional argument used to control the mode of calculation; it can be set to 0, 1 or 2 corresponding to increasingly elaborate measure taken to reduce the amount of numerical errors; mode = 0 will be fast but less accurate, mode = 2 will be slow but more accurate; the default mode is 0

Return

`llh` — the log-likelihood of data under a the Gaussian mixture.

Global variables

`gtmGlobalR` — an K-by-N responsibility matrix; gtmGlobalR(k,n) is the responsability takened by mixture component k for data point n.

`gtmGlobalDIST` — an K-by-N matrix in which element (k,n) is the Euclidean distance between the centre of component m and the data point n.

`gtmGlobalMinDist, gtmGlobalMaxDist` — vectors containing the minimum and maximum of each column in DIST, respectively; 1-by-N; required iff m > 0.

See also

`gtm_resp`, `gtm_dstg`, `gtm_dist`

gtm_sort

Sorts the columns of argument matrix R in increasing order.

Synopsis

```
srtdR = gtm_sort(R)
```

Arguments

`R` — an (unsorted) matrix

Return

`srtdR` — the corresponding sorted matrix

Notes

The m-file implementation is simply an alias for MATLAB's built-in sort function. However, if a corresponding mex-file exists, this will be used instead; experience has shown that a C-implementation of (e.g.) quicksort works much faster.

gtm_stp1

Generates the components of a GTM with a 1D latent space.

Synopsis

```
[X, MU, FI, W, beta] = gtm_stp1(T, noLatVarSmpl, noBasFn, s)
```

Arguments

`T` — target data, to be modelled by the GTM.

`noLatVarSmpl` — number of samples in the latent variable space

`noBasFn` — number of basis functions

`s` — the width of basis functions relative to the distance between two neighbouring basis function centres, i.e. if s = 1, the basis functions will have widths (std.dev) equal to (i.e. 1 times) the distance between two neighbouring basis function centres.

Return

`X` — the grid of data points making up the latent variable sample; a vector of length noLatVarSmpl, in which each row is a data point

`MU` — a noBasFn-element vector holding the coordinates of the centres of the basis functions

`FI` — the activations of the basis functions when fed the latent variable sample X, and a bias unit fixed to 1.0; a matrix with the same number of rows as X and noBasFn+1 columns (+1 for the bias).

`W` — the initial matrix of weights, mapping the latent variable sample X linearly onto the first principal component of the target data (T)

`beta` — the intial value for the inverse variance of the data space noise model

Notes

The latent variable sample is constructed as a uniform grid on the interval [-1, 1]. Similarly the centres of the basis function are gridded uniformly over the latent variable sample, with equal standard deviation, set relative to the distance between two centres. The initial linear mapping maps mean and std.devs. 1:1 from the latent to the target sample along the principal component.

See also

`gtm_stp2`

gtm_stp2

Generates the components of a GTM with a 2D latent space.

Synopsis

```
[X, MU, FI, W, beta] = gtm_stp2(T, noLatVarSmpl, noBasFn, s)
```

Arguments

`T` — target data, to be modelled by the GTM.

`noLatVarSmpl` — number of samples in the latent variable space; must be an integer^2, e.g. 1, 4, 9, 16, 25, 36, 49, ...

`noBasFn` — number of basis functions in the; must be an integer^2

`s` — the width of basis functions relative to the distance between two neighbouring basis function centres, i.e. if s = 1, the basis functions will have widths (std.dev) equal to (1 times) the distance between two neighbouring basis function centres.

Return

`X` — the grid of data points making up the latent variable sample; a matrix of size noLatVarSmpl-by-2, in which each row is a data point

`MU` — a noBasFn-by-2 matrix holding the coordinates of the centres of the basis functions

`FI` — the activations of the basis functions when fed the latent variable sample X, and a bias unit fixed to 1.0; a matrix with the same number of rows as X and noBasFn+1 columns (+1 for the bias).

`W` — the initial matrix of weights, mapping the latent variable sample X linearly onto the 2 first principal components of the target data (T)

Notes

The latent variable sample is constructed as a uniform grid in the square [-1 -1; -1 1; 1 1; 1 -1]. Similarly the centres of the basis function are gridded uniformly over the latent variable sample, with equal standard deviation, set relative to the distance between neigh- bouring centres.The initial linear mapping maps the std.devs. 1:1 from the latent to the target sample

See also

`gtm_stp1`

gtm_trn

Optimize (train) the parameters of a GTM model, using an EM algorithm.

Synopsis

```
[W, beta, llhLog] = gtm_trn(T, FI, W, l, cycles, beta, m, q)
[W, beta] = gtm_trn(T, FI, W, l, cycles, beta)
```

Arguments

`T` — matrix containing a sample of the distribution to be modeled; N-by-D

`FI` — matrix containing the output values from the basis functions, when fed the latent variable sample; K-by-(M+1)

`W` — an initial weight matrix; (M+1)-by-D

`l` — weight regularisation factor

`cycles` — no of training cycles

`beta` — an initial value for beta, the inverse variance of the Gaussian mixture generated in the data space

`m` — mode of calculation; it can be set to 0, 1 or 2 corresponding to increasingly elaborate measure taken to reduce the amount of numerical errors; mode = 0 will be fast but less accurate, mode = 2 will be slow but more accurate; the default mode is 1

`q` — quiet execution; if q equals the string 'quiet', the plottingand echoing of the values of log- likelihood and beta during traaining is supressed. This argument is optional; if omitted the training is run non-quiet.

Return

`W, beta` — the corresponding weight matrix and inverse variance after training

`llhLog` — the log-likelihood after each cycle of training; optional output argument