SparseBayes

An Efficient Matlab Implementation of the
Sparse Bayesian Modelling Algorithm (Version 2.0)

Author: Michael E. Tipping
Date: March 12, 2009
Contact: mail@miketipping.com

www.vectoranomaly.com

Summary

This document is intended as a basic user-guide and implementation overview for the *SparseBayes* Version 2.0 software package for *Matlab*, available from `http://www.relevancevector.com`.

Copyright & Licence

Software Status and Bug Reports

While *SparseBayes* has undergone some considerable testing (under *Matlab* Version 7.4, R2007a, Win32), the status of the current Version 2.0 release would best be considered as "beta". Please report any problems arising with the use of the software to the author at `mail@miketipping.com`.

Acknowledgements

The author would like to thank Mark Hatton, Anita Faul, Ian Nabney, Arnulf Graf and Gavin Cawley for their assistance in the early development of this code.

1. About the *SparseBayes* Software

1.1 The *SparseBayes* Version 2.0 Package

This document briefly summarises "Version 2" of the *SparseBayes* software package, designed to run within the *Matlab* environment (see `http://www.mathworks.com`). The latest version of *SparseBayes*, along with other explanatory materials, should be available from:

`http://www.relevancevector.com`

This documentation refers specifically to the *Version 2.0* release of SparseBayes. Version 2.0 is the first release of an extended implementation of the algorithm described in the paper "Fast Marginal Likelihood Maximisation for Sparse Bayesian Models" [4].

Note that *SparseBayes* "Version 1" is an older, simpler, baseline, implementation, which is still available for reference at the above web-site. The software described in this document is intended to supercede that earlier version.

1.2 A Brief Overview of the "Sparse Bayesian" Model

The concept of the "sparse Bayesian" model was originally introduced in ref. [1], and later described in more comprehensive detail in ref. [2]. The reader who is unfamiliar with the model may wish to initially consult an introductory article which is available as ref. [3]. Download-able versions of these referenced articles are available at `www.relevancevector.com`.

In summary here, the "sparse Bayesian model" is essentially a Bayesian treatment of a generalised linear predictive model with a specific prior over the parameters which favours sparse prediction functions.

1.2.1 The underlying model

For example, consider a predictive modelling task where there is a set of 'input' data samples $\mathbf{X} = \{\mathbf{x}_n\}$ with corresponding desired 'target' values $\mathbf{t} = \{t_n\}$, $n = 1 \ldots N$. The objective is to find an underlying functional model $y(\mathbf{x})$ that "predicts" t well given $\mathbf{x}$, and is not compromised by noisy, real-world, data.

One possible predictor $y(\mathbf{x})$ would be a "generalised linear" one:

$$y(\mathbf{x};\mathbf{w}) = \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}),$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_M)$ is the vector of adjustable parameters. This model is therefore said to be "linear in the parameters", which conveys a number of analytic advantages. At the same time, by choosing the "basis functions" $\phi_m(\mathbf{x})$ to be nonlinear, $y(\mathbf{x};\mathbf{w})$ may be nonlinear too. If M is large, then this type of model is potentially very flexible, and providing the computational and statistical complexity of the model is appropriately managed, it can be very effectively applied to a wide range of problems.

The sparse Bayesian methodology has been designed to be a principled and practically effective mechanism for managing this computational and statistical complexity. The key to the approach is the definition of a hyper-parameterised prior over the model parameters of the form:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{m=1}^{M} p(w_m|\alpha_m) \propto \prod_{m=1}^{M} \alpha_m^{1/2} \exp\left(-\frac{\alpha_m}{2} w_m^2\right).$$

Analysis shows that this type of prior ultimately favours models that both fit the data well and, in particular, are *sparse* [2]. That is, the prior locates most of its probability mass at $w_m = 0$.

Given the model and its prior, the *SparseBayes* software computes the posterior distribution over the hyperparameters α_m given the data, and returns their most-probable values by maximising

the *marginal likelihood* function. This leads to a posterior distribution over the parameters w_m where many are (mathematically) set to zero. As a result, the predictive models derived by the algorithm will, in most cases, be sparse. In practical terms then, the generalised linear model predictor $y(\mathbf{x};\mathbf{w})$ will typically comprise very few non-zero w_m and will thereby incorporate a compact set of basis functions only.

1.2.2 An algorithmic perspective

As a result of the choice of prior, the model described above also exhibits some attractive mathematical properties from an algorithmic perspective, and these are exploited by the accompanying *SparseBayes* Version 2.0 software.

A key element of the *SparseBayes* algorithm is the manner in which it optimises the underlying marginal likelihood function with respect to all the hyperparameters $\{\alpha_m\}$. Although on paper this is a continuous joint optimisation procedure, the algorithm has the facility to perform *discrete* "addition" and "deletion" of *individual* basis functions from the underlying model in a principled way, as discussed in ref. [4]. The optimisation algorithm implemented by *SparseBayes* is essentially a refined version of that outlined in Section 4 of that paper.

1.3 Guide to this Document

The remainder of this document is structured as follows:

Section 2: A listing and summary of the contents of the *SparseBayes* Version 2.0 distribution.

Section 3: A "Quick start" introduction to the use of the *SparseBayes* software via the accompanying `SparseBayesDemo` demonstration program.

Section 4: A guide to the use of `SparseBayes` and other principal functions in the package.

Section 5: Some notes concerning aspects of the implementation that may be of interest.

Section 6: A brief overview of the support functionality in the package.

2. Package Summary: *SparseBayes* 2.0

A listing and brief description of each file in the *SparseBayes* distribution is given in the below table. The relevant functionality in these files is described in more detail later in this document, and reasonably comprehensive information for most functions can also be obtained via the standard MATLAB "help" interface. Note that the files described under "Core User Functionality" below are intended for direct application by the end-user, and are more comprehensively documented herein.

	Example / Demonstration (Section 3)
SparseBayesDemo.m	An example function to illustrate the use of SparseBayes.
	Core User Functionality (Sections 4 and 5)
SparseBayes.m	The main *SparseBayes* functionality: the hyperparameter re-estimation algorithm for a model with arbitrary basis.
SB2_UserOptions.m	Primary user-selectable options.
SB2_ParameterSettings.m	User-adjustable initial parameter setting over-rides.
SB2_ControlSettings.m	Various default parameter settings to control the internal operation of the main algorithm.
	Support Functionality (Section 6)
	Algorithmic
SB2_Initialisation.m	Initialise everything for the SparseBayes algorithm.
SB2_PreProcessBasis.m	Normalise the basis (design) matrix.
SB2_Likelihoods.m	Conveniently encapsulate different likelihood types.
SB2_FullStatistics.m	Computes all relevant statistics in explicit form for the *SparseBayes* algorithm.
SB2_PosteriorMode.m	Posterior-mode finding function for the non-Gaussian likelihood case, called by SB2_FullStatistics.
SB2_Sigmoid.m	Sigmoid link function for Bernoulli likelihood.
	Diagnostics
SB2_Diagnostic.m	Flexible diagnostic output. Allows control over verbosity of output and writing to a log file.
SB2_FormatTime.m	Simple generic support function to pretty-print timings.
	Documentation
Readme.txt	Standard "readme" information file.
licence.txt	Licence file (GPL Version 2).
SB2_Manual.pdf	This document.

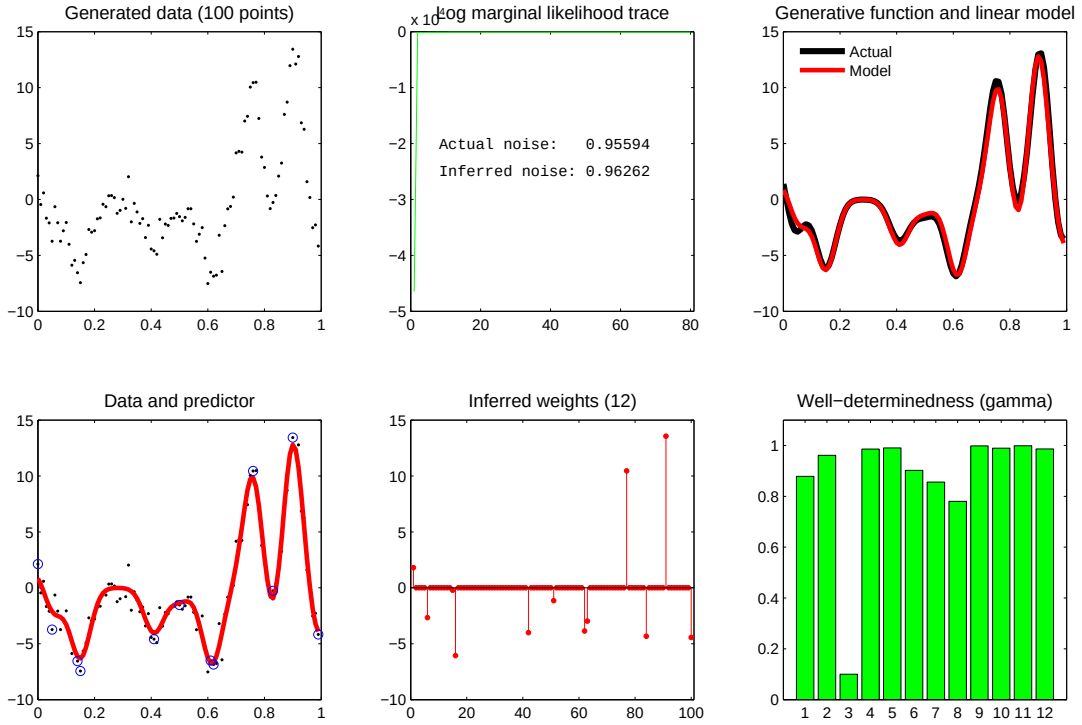
3. Quick Start

The `SparseBayesDemo` program provides a basic illustration of how the *SparseBayes* software may be used. It gives an example of the application of both "regression" and "classification" models (*i.e.* utilising Gaussian and Bernoulli likelihoods respectively) in both one and two dimensions. The functional models employed are nonlinear and based on fixed "Gaussian" basis (kernel) functions. The data used is derived synthetically from the generative model itself using a sparse subset of the possible basis functions, with added noise appropriate to the likelihood function in use.

To see the main *SparseBayes* algorithm at work, type the following at the *Matlab* prompt:

```
>> SparseBayesDemo('Gaussian',1)
```

This should give the results below, illustrating the data, the generative function, the inferred model (posterior-mean) predictor, along with the inferred set of relevant basis functions (which should be sparse).



A similar example in two dimensions can be obtained by running:

```
>> SparseBayesDemo('Gaussian',2) .
```

The data for the above demonstrations was synthesised with additive Gaussian noise of a fixed "noise-to-signal" fraction (relative to the standard deviation of the noise-free data). This defaulted to 0.2. An optional third parameter to `SparseBayesDemo` allows this to be varied. For example, the following two examples may be be of interest to view:

```
>> SparseBayesDemo('Gaussian',1,0.01)
>> SparseBayesDemo('Gaussian',1,1)
```

For examples of "classification", try:

```
>> SparseBayesDemo('Bernoulli',1)
>> SparseBayesDemo('Bernoulli',2)
```

4. Using *SparseBayes*: A Basic Guide

4.1 Overview: User-Level Functionality

The "user-level" functionality in *SparseBayes* is to be found in four principal functions:

- `SparseBayes`: implementation of the core "sparse Bayesian" model inference algorithm
- `SB2_UserOptions`: an interface to allow the user to specify a range of options that modify the behaviour of the algorithm and underlying model
- `SB2_ParameterSettings`: an interface to facilitate initial settings of model parameters and hyperparameters
- `SB2_ControlSettings`: hard-wired specifications of certain algorithmic control parameters which may be modified (in the file) by the interested user

Primarily, only `SparseBayes` and `SB2_UserOptions` are explicitly required, as long as the user is happy with the default initial parameter settings (from `SB2_ParameterSettings`) and algorithmic settings (in `SB2_ControlSettings`). Inspection of the source code of the demonstration function `SparseBayesDemo` introduced in the previous section will hopefully illustrate how the core functions might be called, with more specific detail given in the following subsections.

4.2 Running the Main Inference Algorithm

In its simplest form, `SparseBayes` may be called as:

```
>> SparseBayes('Gaussian', BASIS, Targets)
```

This assumes that argument `BASIS` is an $N \times M$ basis (or design) matrix, where `BASIS(n,m)` represents the value of input variable, or basis function, m (of M) corresponding to example datum n (of N). The argument `Targets` is the corresponding $N \times 1$ vector of desired output values.

The first argument specifies the likelihood model to use. Currently, options are:

- `'Gaussian'`: noise model for real-valued regression (interpolation, function approximation)
- `'Bernoulli'`: standard likelihood for binary classification (pattern recognition)
- `'Poisson'`: likelihood model for positive integer regression ("experimental" at present)

4.3 Configuring *SparseBayes*

4.3.1 General options and settings overview

The default operation of `SparseBayes` is designed to be modified in three ways:

- **User Options:** these are options that the user is likely to want to modify, via a call to `SB2_UserOptions`, on a per-problem basis. Example options are the number of iterations to run for and the level of diagnostic output desired. See Section 4.3.3 shortly.
- **Initial Parameter Settings:** `SparseBayes` automatically obtains sensible default values for $\boldsymbol{\alpha}$, β and the initial basis subset via `SB2_ParameterSettings` (see Section 4.3.4). These defaults may be acceptable in most cases, but the user may easily over-ride them to suit a particular problem if desired. This functionality would also be applicable if the algorithm was to be "restarted" with the previously-learned parameters from an earlier call to `SparseBayes` (perhaps incorporating some alternative basis functions the second time around).
- **Algorithmic Control Settings:** although not explicitly exposed to the user, there are a number of parameters which control how various aspects of the algorithm operate "under the hood". These are are defined in `SB2_ControlSettings.m` which can be thought of as an editable configuration file. See Section 4.3.5 for some further detail.

4.3.2 The configuration interfaces

The functions SB2_UserOptions and SB2_ParameterSettings both behave in a similar manner and offer a similar interface to the user. Each returns a single structure (OPTIONS and SETTINGS respectively) whose fields define appropriate default values for the relevant properties. These defaults can be seen by calling the functions with no arguments, for example:

```
>> OPTIONS = SB2_UserOptions

OPTIONS =

         fixedNoise: 0
          freeBasis: []
         iterations: 10000
               time: 10000
            monitor: 0
    diagnosticLevel: 0
      diagnosticFID: 1
    diagnosticFile_: []
           callback: 0
       callbackFunc: []
       callbackData: {}
```

Some, or all, of these defaults may be over-ridden by the user in their code by supplying the appropriate set of *property-value* argument pairs. For example:

```
>> OPTIONS = SB2_UserOptions('Iterations', 1000,...
                            'DiagnosticLevel', 2,...
                            'Monitor', 10);
```

The range of applicable properties for SB2_UserOptions is detailed in Section 4.3.3 shortly. Note that property names are *not* case-sensitive.

The function SB2_ParameterSettings operates analogously to SB2_UserOptions but incorporates fields designed for the setting of initial values of parameters and hyperparameters. The relevant properties for SB2_ParameterSettings are detailed in Section 4.3.4.

Once both user options and initial parameter settings have been specified, the relevant structures may be passed as optional parameters to SparseBayes. For example, given prior definitions of BASIS and Targets, the following three statements are sufficient to run the full *SparseBayes* algorithm:

```
>> OPTIONS  = SB2_UserOptions('Iterations',1000)
>> SETTINGS = SB2_ParameterSettings('NoiseStd',0.1)
>> SparseBayes('Gaussian', BASIS, Targets, OPTIONS, SETTINGS)
```

Unlike the "user options" (OPTIONS) and "initial parameter settings" (SETTINGS) discussed above, there is no functional user-interface to allow direct adjustment of "algorithmic control settings". Instead, these are "hard-coded" in the file SB2_ControlSettings.m. These settings may be changed via direct editing of the file if desired. See Section 4.3.5 below and the comments in the file itself for more information.

4.3.3 SB2_UserOptions: valid properties

The default values for the various user-determined options (the fields of the `OPTIONS` structure) were shown in the earlier example. Those fields may be modified by passing the appropriate *property-value* pairs to `SB2_UserOptions`, as detailed in the following table.

Property	Type	Description
`'Iterations'`	Integer	Maximum number of iterations to run for.
`'Time'`	String	Maximum time to run for. Specified as a string such as `'45 seconds'`, `'2.5 minutes'` or `'1 hour'`.
`'DiagnosticLevel'`	Integer	Specifies the verbosity of output from `SparseBayes`. Currently understood values range from 0 (no output) to 4 (ultra-verbose). The `SparseBayesDemo` example defaults to a "medium" level of 2.
`'DiagnosticFile'`	String	Name of a log file to output diagnostics to (instead of the *Matlab* console).
`'Monitor'`	Integer	Specifies after how many iterations the evolving model statistics are summarised. Only effective if `'DiagnosticLevel'` is 2 or greater.
`'FixedNoise'`	Boolean	If `true`, the noise estimate is not updated in the Gaussian likelihood case (*i.e.* β is fixed to the value set by `SB2_ParameterSettings`).
`'FreeBasis'`	[Integer]	Indices of basis vectors (columns of `BASIS`) that are considered "free". That is, they are not subject to the sparse Bayesian prior. See Section 5.3.
`'Callback'`	String	Name of function to call at each iteration during the algorithm. See Section 5.5.
`'CallbackData'`	Cell array	List of additional data variables to pass to the callback function.

4.3.4 SB2_ParameterSettings: valid properties

Valid properties for initial parameter specification are given in the below table.

Property	Type	Description
`'Beta'`	Double	Initial setting for inverse noise variance β for Gaussian likelihood models.
`'NoiseStd'`	Double	Initial setting for noise standard deviation σ for Gaussian likelihood models. Note that $\beta = 1/\sigma^2$ and if both `'Beta'` and `'NoiseStd'` are specified, the value of `'Beta'` will take priority.
`'Relevant'`	[Integer]	Vector of indices of initially relevant basis vectors (column indices of `BASIS`).
`'Weights'`	[Double]	Vector of initial weight values corresponding to the `'Relevant'` basis vectors.
`'Alpha'`	[Double]	Vector of initial values for the hyperparameters $\boldsymbol{\alpha}$, again corresponding to `'Relevant'`.

Of the latter three properties, if `'Relevant'` is specified, it is *not* necessary to define corresponding values for either `'Alpha'` or `'Weights'` (these will be "sensibly" initialised if omitted). However, if one or both of `'Alpha'` or `'Weights'` are defined, they must be of matching length to `'Relevant'` (which therefore must also have been specified).

4.3.5 SB2_ControlSettings: configuration summary

This function returns a default structure (CONTROLS) that encapsulates a number of parameters to control the internal workings of the main SparseBayes algorithm. Unlike SB2_UserOptions and SB2_ParameterSettings, there is no explicit functionality in SB2_ControlSettings to allow user over-riding of the defaults. These must be changed by editing the file if desired. The following settings are defined in SB2_ControlSettings:

- Termination tolerances.
- Priority control for "add" and "delete" updates.
- How the Gaussian noise estimates are updated.
- How often the posterior mode approximation is updated in the non-Gaussian case.
- Whether "redundant" basis functions are explicitly handled.

See the comments in the SB2_ControlSettings.m file for more details.

5. Using *SparseBayes*: Implementation Notes

This section describes a number of elements of the implementation of the *SparseBayes* software that may be relevant to the interested user.

5.1 Specifying the Model Basis

Running the `SparseBayes` inference function requires that an appropriate model basis matrix (the argument `BASIS`) for the problem at hand is set up in advance. The earlier "SparseBayes Version 1.1" software included specialised functionality (in `SB1_RVM`) to implement a "kernel" basis model (*i.e.* a "relevance vector machine"). The emphasis of the new "Version 2" *SparseBayes* package described here is that it is "general purpose", and as such, no explicit functionality to define specific model bases is included by default. It is left to the user to pre-compute an appropriate `BASIS` for their problem as desired (although there is no reason not to modify and re-use code from Version 1.1 for that purpose).

Ultimately, the matrix `BASIS` matrix may be derived from arbitrary numbers and types of basis functions $\phi_m(\mathbf{x})$. Noting that `BASIS(n,m)` $= \phi_m(\mathbf{x}_n)$, some examples might be:

- **Bias:** A single constant vector with $\phi(\mathbf{x}_n) = 1$, used to implement a threshold or offset.
- **Linear:** The design matrix is effectively the data matrix, $\phi_m(\mathbf{x}_n) = x_{nm}$.
- **Nonlinear:**
  - *Fixed*. For example, univariate polynomials, $\phi_m(x_n) = x_n^m$.
  - *Data-parameterised*. For example, "Gaussian data-centred" kernel functions, $\phi_m(\mathbf{x}_n) = \exp\{-\eta\|\mathbf{x}_m - \mathbf{x}_n\|^2\}$. See the `SparseBayesDemo` function for an example of these.

Given care, there is no reason not to combine different basis types together: *e.g.* a practical model might define `BASIS` as a concatenation of a bias, a linear basis and a Gaussian kernel basis.

5.2 Initialisation

Unless the user specifies a particular "starting" basis via `SB2_ParameterSettings`, the default initialisation of the model will typically be a single basis vector (but see also Section 5.3 next). This is chosen to be the basis vector (column of `BASIS`) with the largest inner-product with (*i.e.* is best aligned with) the target vector $\mathbf{t}$ ("linearised" in the non-Gaussian case). In respect of computing inner-products of basis vectors, note that `SB2_PreProcessBasis` normalises each basis vector to unit length for the purposes of all internal computations within `SparseBayes`.

5.3 Incorporating a "Free" Basis

One of the configuration options offered by `SB2_UserOptions` is the facility to specify a "free" basis via the `'FreeBasis'` property. This defines a vector of indices m for which the corresponding prior over w_m is to be "inactive", or equivalently, for which α_m is set to zero and remains fixed. Such parameters will therefore be fitted to the data without penalty, and their corresponding basis functions $\phi_m(\mathbf{x})$ will always be incorporated in the model.

A primary motivation for this functionality is to accommodate a "bias", or fixed offset/threshold, parameter (where $\phi_m(\mathbf{x}_n) = 1$ for all n). Although it seems to be conventional in the literature to apply a Bayesian treatment to the bias parameter in most models, this implies the outcome of any inference is *not shift-invariant* with respect to the targets, which ought to be considered undesirable in many applications. (Note that the sparse Bayesian model is implicitly *scale-invariant* with respect to $\mathbf{t}$ already.)

With reference to the discussion on initialisation above, if a `'FreeBasis'` is specified by the user, that particular basis (which may be only the bias vector) will be used to initialise the model. This is in addition to any others that may also be explicitly specified by the user (via the `'Relevant'` property in `SB2_ParameterSettings`).

5.4 Basis Vector Alignment

One of the "under the hood" extensions of the *SparseBayes* algorithm, introduced in response to early user feedback, is the notion of basis function "alignment tests" (which can be turned on and off via the boolean `CONTROLS.BasisAlignmentTest` in `SB2_ControlSettings`). This is a heuristic that has been designed to finesse the issue of multiple identical (or indistinguishable) basis vectors within a model.

One of the features of the sparse Bayesian modelling approach is that if two basis functions, such as $\phi_j(x)$ and $\phi_k(x)$, are similar, then it is unlikely that both will be utilised in the final model. In most cases, one will be deemed irrelevant. This general behaviour can be clearly seen in the `SparseBayesDemo` examples. However, if the basis functions are not just similar but are *identical*, such that $\phi_j(x) = \phi_k(x)$, then if either is ultimately deemed relevant under the model, *both* basis functions will be incorporated to some extent (as there is no way to tell them apart). In such a case, $w_j + w_k$ will be equal to some constant value, although their individual values are essentially arbitrary. This can also occur if $\phi_j(x)$ and $\phi_k(x)$ are not identical but are similar enough to be numerically indistinguishable by the algorithm.

To work around this, there is a parameter `CONTROLS.AlignmentMax`, defined to be slightly less than unity, which specifies a maximum allowable value of the inner-product between any two basis functions in the model (*i.e.*, it is a measure of "maximum allowed similarity"). This imposes a constraint that as long as $\phi_j(x)$ is in the model, then $\phi_k(x)$ can never be "added" if:

$$\sum_{n=1}^{N} \phi_j(x_n).\phi_k(x_n) > \texttt{CONTROLS.AlignmentMax}$$

Since the basis vectors are pre-normalised (transparently to the user), $\sum_{n=1}^{N} \phi_j(x_n).\phi_k(x_n) = 1$ when $\phi_j(x)$ and $\phi_k(x)$ are identical. Note that this does not exclude the possibility of $\phi_j(x)$ being "deleted" at some point and $\phi_k(x)$ then being subsequently "added" (if doing so would increase the marginal likelihood of course).

This alignment-checking functionality is activated by default in the current implementation, with `CONTROLS.AlignmentMax` set to 0.999. Switching off `CONTROLS.BasisAlignmentTest` should not have a noticeable effect on the ultimate predictive model output, but may result in mildly extended optimisation time and a less sparse final predictor.

5.5 The User-Configurable Callback Function

The main `SparseBayes` function includes the provision to call an arbitrary, external, user-specified function via the `'Callback'` property of `SB2_UserOptions`. This callback function is called once at the start of the main optimisation, and once during each iteration thereafter, and each time passes a range of potentially useful data across, such as the current values of the parameters, hyperparameters, active basis set *etc*.

Effectively, this passes control to the user's own function during each cycle of the algorithm, enabling, for example, an ongoing graphical update of the state of the model, perhaps for demonstration purposes. The callback function should be of the form:

```
UserFunction(ITERATION, ACTION, LOGML, USED, WEIGHTS, SIGMA,...
             ALPHA, BETA, GAMMA, PHI, VARARGIN)
```

Here, the variable-length argument list that is passed (as `VARARGIN`) is that associated with the `'CallbackData'` property that can be set by `SB2_UserOptions`.

The next update release of *SparseBayes* will include an additional demonstration program that will illustrate how this "callback" functionality may usefully be exploited.

6. Brief Summary of Other Functions

For the sake of completeness, "support functions" from the *SparseBayes* 2.0 package, not necessarily designed for direct end-use, are briefly described here. Again, there is also some basic "help" text available via the usual *Matlab* interface.

`SB2_Initialisation`

This function is called before the main loop within `SparseBayes` and does the following:

- Calls `SB2_PreProcessBasis` to normalise the basis matrix.
- Validates the choice of likelihood.
- Initialises β in the Gaussian likelihood case if not pre-specified. The heuristic here is to set $\beta = 1/\sigma^2$ where σ is initialised to be 10% of the standard deviation of the targets t_n (*i.e.* there is an assumed initial SNR of 10:1).
- If no initial basis is specified (and there is no "free" basis), an initial model basis $\boldsymbol{\Phi}$ is determined. This is heuristically chosen to be the single basis vector that is best aligned with the (linearised) target vector $\mathbf{t}$.
- Given the initial basis, appropriate "reasonable" initialisations for parameters $\boldsymbol{\mu}$ and hyperparameters $\boldsymbol{\alpha}$ are calculated.

`SB2_PreProcessBasis`

Internally normalises the basis vectors (each column of $\boldsymbol{\Phi}$) to unit length. This should improve numerical robustness, and simplifies a number of internal calculations. The function returns the original basis vector lengths such that the weights (and other dependent parameters) can be appropriately rescaled at the termination of the algorithm, consistent with the scale of the originally supplied basis.

`SB2_Likelihoods`

A convenience function to encapsulate the likelihood specification for the model and to allow for future expandability.

`SB2_FullStatistics`

This function calculates the full posterior statistics given the current settings of the hyperparameters ($\boldsymbol{\alpha}$, β). Variables computed are:

- Parameter (weight) posterior mean and covariance, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.
- "Sparsity" and "quality" factors S_m, Q_m, s_m and q_m, for all basis functions.
- The key "relevance factor" $q_m^2 - s_m$ for all basis functions.
- The log-likelihood and the "well-determinedness" (γ_m) values.

All the above can be calculated by more efficient "update" formulae within `SparseBayes` (see Appendix A of ref. [4]). However, in the Gaussian likelihood case when the noise estimate (β) has been updated, and more generally in the non-Gaussian case, it is necessary to calculate the statistics "in full" via this function.

`SB2_PosteriorMode`

Called by `SB2_FullStatistics` to find the mode of the posterior distribution over parameters, as the basis for the Laplace approximation in the non-Gaussian likelihood case. The optimisation uses a variable-step-length second-order gradient-based (Newton) method.

`SB2_Sigmoid`

Simple generic function to compute the "logistic sigmoid" link function for the Bernoulli likelihood model: $f(x) = 1/(1+e^{-x})$.

`SB2_Diagnostic`

The function facilitates the convenient output of diagnostic information to both the *Matlab* console and/or to a file. Messages have a specific "importance level" and can be filtered out by an appropriate setting of the diagnostic level (see Section 4.3.3). For example, including:

```
>> OPTIONS = SB2_UserOptions('DiagnosticLevel',1,'DiagnosticFile','myfile.log')
```

in the options selections will ensure that only "level-0" and "level-1" messages are output, and these will be written to the specified log file.

`SB2_FormatTime`

Convenient function to output timing information in a user-friendly format.

References

[1] M. E. Tipping. The Relevance Vector Machine. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 652–658. MIT Press, 2000.

[2] M. E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[3] M. E. Tipping. Bayesian inference: An introduction to principles and practice in machine learning. In O. Bousquet, U. von Luxburg, and G. Rätsch, editors, *Advanced Lectures on Machine Learning*, pages 41–62. Springer, 2004.

[4] M. E. Tipping and A. C. Faul. Fast marginal likelihood maximisation for sparse Bayesian models. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, Key West, FL, Jan 3-6*, 2003.