Dynamic Display of Changing Posterior in Bayesian Survival Analysis: The Software

Hani J. Doss
Department of Statistics
The Ohio State University
Columbus, OH 43210

B. Narasimhan
Department of Statistics
Stanford University
Stanford, CA 94305

$Revision : 1.34$ of $Date : 1998/07/0217 : 16 : 33$

Contents

Abstract

We consider the problem of estimating an unknown distribution function F in the presence of censoring under the conditions that a parametric model is believed to hold approximately. We use a Bayesian approach, in which the prior on F is a mixture of Dirichlet distributions. A hyperparameter of the prior determines the extent to which this prior concentrates its mass around the parametric family. A Gibbs sampling algorithm to estimate the posterior distributions of the parameters of interest is reviewed. An importance sampling scheme enables us to use the output of the Gibbs sampler to very quickly recalculate the posterior when we change the hyperparameters of the prior. The calculations can be done sufficiently fast to enable the dynamic display of the changing posterior as the prior hyperparameters are varied.

This paper provides a literate program completely documenting the code for performing the dynamic graphics.

1 Copyright

We begin with our usual copyright.

4 ⟨*Copyright* 4⟩≡ (5)

Defines:
`copyright`, never used.

2 Introduction

This document is a literate program implementing the theory described in our paper [2]. A literate program is a program written in a style that makes it easy for humans to read, understand and modify. For more information on Literate Programming, see [4]. A quicker introduction is available on the World Wide Web; see [6]. This document uses the `Noweb` [7], [8], [9] literate programming tools. Although it is not required, we recommend that any serious user of this software have the `Noweb` tools installed. `Noweb` tools, besides being free, are extremely easy to install and require no special expertise other than basic knowledge about how TEX or LATEX work. Having `Noweb` allows one to take full advantage of our software—syntax errors will be minimized and the code we have written can be reused with the user's modifications spliced into place automatically by `Noweb`.

We expect any serious user of our software to read the original paper [2], a copy of which is included in the software distribution.

This document is available in three forms: Postscript, PDF and HTML. All versions are accessible from the web pages of the authors.

We wish to remark that the software only does sensitivity analysis. No general facility is provided for generating observations from Markov chains. Indeed, since the range of models for which MCMC methods are applicable is large and such methods most likely involve problem-specific issues, it is our opinion that building such a supertool, if it is at all possible, is a non-trivial task. However, the Fortran program used in generating the output for our example is included along with this software and can be used for models similar to ours. Of course, any appropriate method may be used to generate the samples as long as the output is available in a form usable by our software. The requirements on the data that can be used with our software are spelt out below.

Corresponding to each Markov chain output, there must be two files with the extensions `.in` (input file) and `.out` (output file). For example, `mc1.in` and `mc1.out`.

The input file must have the following structure. The first four items in the file can be anything, string or number, either on a single line or any conceivable combination of lines. The next three items *must* be the shape of the Gamma distribution on θ, the scale of the Gamma distribution on θ—the parametrization for shape a and scale b is proportional to $\theta^{a-1}\exp(-b\theta)$—and $\alpha(\mathbf{R})$. The next three values values following these quantities can be anything, but the one following it should be the number of data points, or sets. Nothing else is read from the input file.

The output file must have the following structure for each data point generated by the Markov chain. The value of θ must be followed by the number of distinct values of the data points, which must be followed by a frequency table of the actual data value and the corresponding frequency. The layout of the values on lines does not matter as long as at least a single white space delimits values. If this structure is violated, errors will result. A peek at the data files included with this software will help the reader.

A note on performance. The calculations involved in reweighting are non-trivial and require a reasonably powerful computer for smooth performance. The efficiency can be improved by dynamically loading C programs that compute various quantities. The version of software described here does so by default. An older version that does not use dynamic loading which is available upon request from the authors. Without dynamic loading, the performance is very bad indeed.

Dynamic libraries for Windows and Macintosh are provided. Suggestions for various Unix platforms are also provided. Section 7 has more details.

It is assumed that a proper installation of `Lisp-Stat` described in [10] is available. The version number on `Lisp-Stat` should be 3.52.0 or higher since dynamic loading uses the new shared library mechanism.

Some additions planned for the future are listed in Section 8.

3 The Software

The software consists of the following components.

5 ⟨* 5⟩≡
⟨*Copyright* 4⟩
```
(require "utility")
```

```
(require "call-by-reference")
(defpackage "BSA" (:use "XLISP" "USER" "UTILITY" "CALL-BY-REFERENCE"))
(in-package "BSA")
(import
        '(user::rseq
          user::standard-deviation))
⟨The Master Prototype 7⟩
⟨The Slave Prototype 43b⟩
(export
        '(master-proto))
```

Uses `master-proto` 7.

4 The Master Object Prototype

The master prototype `master-proto` inherits from `dialog-proto` of `Lisp-Stat` and contains a number of slots. A rich set of methods facilitate interaction with the master.

7 ⟨*The Master Prototype* 7⟩≡ (5)
```
(defproto master-proto
  '(identifier number-of-markov-chains
               number-of-points
               number-of-data-values
               number-of-months
               data-file-names
               summary-data
               indicator-counts
               log-constants-of-proportionality
               slaves
               hyperparameters-used-in-markov-chains
               initially-specified-hyperparameter-values
               current-hyperparameter-values
               hyperparameter-names
               log-mixture-density
               importance-weights
               hyperparameter-ranges
               hyperparameter-sliders
               work-space
               shared-library
               density-abscissae
               density-ordinates
               expectation-abscissae
               expectation-ordinates
               standard-deviation-ordinates
               statistics
               statistics-print-formats
               statistics-labels
               number-of-slider-stops
               superimpose
               timing
               timing-button
               lazy)
  () dialog-proto
  "The Master prototype. Creates and manipulates a harem of slaves.")
⟨Methods for Master Prototype 10⟩
⟨Defaults for Master 43a⟩
```
Defines:
- `current-hyperparameter-values`, used in chunks 15a, 25, 38–40, 46a, and 61.
- `data-file-names`, never used.
- `density-abscissae`, never used.
- `density-ordinates`, never used.
- `expectation-abscissae`, never used.
- `expectation-ordinates`, never used.
- `hyperparameter-names`, used in chunks 14c, 24b, and 40a.
- `hyperparameter-ranges`, used in chunks 16b, 20, 24b, 25a, 27d, and 36b.
- `hyperparameter-sliders`, used in chunk 16c.
- `hyperparameters-used-in-markov-chains`, used in chunk 16a.

`identifier`, used in chunks 12a, 18–20, 22c, 23b, 25c, 31, 37b, and 40a.
`importance-weights`, used in chunks 17b and 61.
`indicator-counts`, used in chunks 13c, 26a, 28–30, 35b, and 37a.
`initially-specified-hyperparameter-values`, used in chunks 14b, 19b, 22b, and 39b.
`lazy`, never used.
`log-constants-of-proportionality`, used in chunk 13d.
`log-mixture-density`, used in chunk 17a.
`master-proto`, used in chunks 5, 12–18, 26a, 29a, 31, 35b, and 37–42.
`number-of-data-values`, used in chunk 13a.
`number-of-markov-chains`, used in chunks 12b, 16–19, 31, 32b, and 43a.
`number-of-months`, used in chunks 12d, 13c, 22a, 29–31, and 43a.
`number-of-points`, used in chunks 12c, 17–19, 22c, 26a, 27a, 31, 34a, and 43a.
`number-of-slider-stops`, never used.
`shared-library`, never used.
`slaves`, used in chunks 14a and 38.
`standard-deviation-ordinates`, never used.
`statistics`, used in chunks 19a, 24b, 39–41, 44, 45, 47a, 51, 56, 60, and 61.
`statistics-labels`, used in chunks 19a, 41c, 45, and 47a.
`statistics-print-formats`, used in chunks 19a, 41b, 44a, 45, and 47a.
`summary-data`, used in chunks 13b, 26a, 28b, 31, 34, 35, and 37a.
`superimpose`, used in chunk 42b.
`timing`, used in chunks 24b and 42a.
`work-space`, never used.

Throughout, we shall use m for the number of Markov chains, and n for the number of data points in the output of each Markov chain. Note that arrays are indexed from 0 so that index i refers to the $(i + 1)$-th position.

Here is a description of all the slots in `master-proto`.

identifier holds a string that is used to identify instances of the object. This string also helps in repeating any interesting run—all inputs needed for the run are saved in a file whose name is generated by adding an extension `.run` to the string. In addition, a file with the extension `.lsp` is also created so that invoking `xlispstat` on that file with automatically recreate everything without annoying questions.

number-of-markov-chains holds the number of Markov chains (m)to be used in the exploration.

number-of-points holds the number of data points from each Markov chain (n) to be used in the exploration.

number-of-data-values holds the number of data values, that is the number of sets.

number-of-months holds the number of months for which the law of $F(t)$ is computed. Default is given by the variable `*default-number-of-months*`.

data-file-names holds a list of the data files corresponding to the Markov chains, without the extension. Dimension is m.

summary-data This slot holds the summary data from all the Markov chains. The data is stored as a 2-d array with each row holding (d_x, θ), where d_x is the the number of distinct values of the data points. The first set of n pairs correspond to the first Markov chain, the next set to the second and so on, which means that the indices have to be suitably translated to access values. There are mn pairs in all. Note that we only handle balanced data, i.e., same n for all Markov chains.

indicator-counts This slot is a 2-d array of size $mn \times 2$ containing the a count of the number of data-values less than t, the number of months. The number of columns in this array is indicated by the slot `number-of-months`. If this value is 61, say, then one would be able to calculate the law of $F(t)$ for t ranging from 0 to 60 months.

log-constants-of-proportionality holds a vector of estimates of the logs of the constants of proportionality, $(\tilde{C}(\kappa_i))^{-1}$, in the paper. The first constant is implicitly -1 and so the dimension of this slot is $m - 1$.

slaves stores a list of slaves who need to be informed of changes in the hyperparameter values. Each slave is an instance of `slave-proto`.

hyperparameters-used-in-markov-chains stores an m by 3 array holding the values of hyperparameters at which the Markov chains were run. If a is the array by a, then $a[j, i]$ is the value of the i-th hyperparameter for the j-th Markov chain, $0 \leq j < m$ and $0 \leq i < 3$. The three hyperparameters in every row are $\alpha(\mathbf{R})$, the shape and scale of the Gamma prior on θ respectively.

initially-specified-hyperparameter-values is a 4×1 array of initial values for the hyper-parameters at which the exploration should begin. These values serve are used in starting and restarting the exploration. The one extra hyperparameter is the value of time t in $F(t)$.

current-hyperparameter-values is a 4×1 array of the current values of the three hyperparameters. These are the quantities that are changed by the user via sliders.

hyperparameter-names holds a list of names for the hyper-parameters. Default is `Alpha`, `Theta Shape`, `Theta Scale`, and `Time (Months)`.

log-mixture-density is a an array holding $\log h_{mix}(x)$ at the data points. Size is mn. See [3]. For speed in dynamic graphics, this quantity is calculated once and saved.

importance-weights holds the importance weights $w[i]$, $0 \leq i < nm$, used for the Reweighting Mixtures (RM) scheme. See [1] and [3].

hyperparameter-ranges stores a list of hyperparameter ranges to be used in exploration. For internal use only.

hyperparameter-sliders is a list of slider objects for internal use only.

work-space is a slot used for storing results from dynamically loaded C routines.

shared-library holds a library for shared handle when dynamic loading is used. For internal use only.

density-abscissae is a vector of abscissa values between 0 and 1 at which the density of $F(t)$ will be plotted. This is an array of length `*default-number-of-plot-stops*`.

density-ordinates is a vector of values of the density of $F(t)$. This is an array of length `*default-number-of-plot-stops*`.

expectation-abscissae is a vector of values of $E(\overline{F}(t))$ for t ranging from 0 to `*default-number-of-months*` (minus 1, as we start at 0).

expectation-ordinates is a vector of values of $E(\overline{F}(t))$ for t ranging from 0 to `*default-number-of-months*` (minus 1, as we start at 0).

standard-deviation-ordinates is a vector of values of $\sigma_{\overline{F}(t)}$.

statistics is a vector of 3 values that will hold $E(\overline{F}(t))$, $\sigma_{\overline{F}(t)}$ and the effective sample size. The effective sample size is calculated using the formula

$$\text{Effective Sample Size} = m * n/(1 + cv(W)^2),$$

where $cv(W)$ is the coefficient of variation of the importance weights W. See, for example, [5].

statistics-print-formats is a list of lists indicating the format to be used in printing the statistics values.

statistics-labels is a list of strings (labels).

number-of-slider-stops stores the number of slider stops for hyperparameters.

superimpose toggles superimposition on and off. Default is off.

timing signifies if timing is needed.

timing-button is a button for toggling timing on and off.

lazy is a slot used for efficient synchronization. It is for internal use by programs and the user shouldn't mess with it.

The methods for `master-proto` can be broken down as follows.

10 ⟨*Methods for Master Prototype* 10⟩≡ (7)
⟨*The Master :identifier Method* 12a⟩
⟨*The Master :number-of-markov-chains Method* 12b⟩
⟨*The Master :number-of-points Method* 12c⟩
⟨*The Master :number-of-months Method* 12d⟩
⟨*The Master :number-of-data-values Method* 13a⟩
⟨*The Master :summary-data Method* 13b⟩
⟨*The Master :indicator-counts Method* 13c⟩
⟨*The Master :log-constants-of-proportionality Method* 13d⟩
⟨*The Master :slaves Method* 14a⟩
⟨*The Master :initially-specified-hyperparameter-values Method* 14b⟩
⟨*The Master :hyperparameter-names Method* 14c⟩
⟨*The Master :current-hyperparameter-values Method* 15a⟩
⟨*The Master :number-of-hyperparameters Method* 15b⟩
⟨*The Master :hyperparameters-used-in-markov-chains Method* 16a⟩
⟨*The Master :hyperparameter-ranges Method* 16b⟩
⟨*The Master :hyperparameter-sliders Method* 16c⟩
⟨*The Master :log-mixture-density Method* 17a⟩
⟨*The Master :importance-weights Method* 17b⟩
⟨*The Master :loglik Method* 17c⟩
⟨*The Master :compute-log-hmix Method* 17d⟩
⟨*The Master :calc-weights Method* 18a⟩
⟨*The Master :isnew Method* 18b⟩
⟨*The Master :graphical-interface Method* 31⟩
⟨*The Master :process-run-file Method* 26a⟩
⟨*The Master :process-frequency-table Method* 29a⟩
⟨*The Master :create-run-file Method* 35b⟩
⟨*The Master :synchronize Method* 38⟩
⟨*The Master :consolidate-computation Method* 39a⟩
⟨*The Master :reset Method* 39b⟩
⟨*The Master :effective-sample-size Method* 39c⟩
⟨*The Master :print-all-statistics Method* 40a⟩
⟨*The Master :labelled-hyperparameter-values Method* 40b⟩
⟨*The Master :statistics Method* 41a⟩
⟨*The Master :statistics-print-formats Method* 41b⟩
⟨*The Master :statistics-labels Method* 41c⟩
⟨*The Master :superimpose Method* 42b⟩
⟨*The Master :toggle-timing Method* 42a⟩
⟨*The Master :close Method* 42c⟩

Some of these methods are mere accessor and modifier methods for the slots and we can get them easily out of the way.

4.1 The `:identifier` Method

12a ⟨*The Master :identifier Method* 12a⟩≡ (10)

```
(defmeth master-proto :identifier (&optional name)
  "Method args: (&optional name)
Sets or retrieves the identifier slot."
  (if name
      (setf (slot-value 'identifier) name)
    (slot-value 'identifier)))
```

Defines:
`:identifier`, used in chunks 19a, 25c, 37b, and 40a.
Uses `identifier` 7 and `master-proto` 7.

4.2 The `:number-of-markov-chains` Method

12b ⟨*The Master :number-of-markov-chains Method* 12b⟩≡ (10)

```
(defmeth master-proto :number-of-markov-chains ()
  "Method args: ()
Returns m, the number of Markov chains used."
  (slot-value 'number-of-markov-chains))
```

Defines:
`:number-of-markov-chains`, used in chunk 19a.
Uses `master-proto` 7 and `number-of-markov-chains` 7.

4.3 The `:number-of-points` Method

12c ⟨*The Master :number-of-points Method* 12c⟩≡ (10)

```
(defmeth master-proto :number-of-points ()
  "Method args: ()
Returns n, the number of data points."
  (slot-value 'number-of-points))
```

Defines:
`:number-of-points`, used in chunk 19a.
Uses `master-proto` 7 and `number-of-points` 7.

4.4 The `:number-of-months` Method

12d ⟨*The Master :number-of-months Method* 12d⟩≡ (10)

```
(defmeth master-proto :number-of-months ()
  "Method args: ()
Returns n, the number of months for which F(t) is computed."
  (slot-value 'number-of-months))
```

Defines:
`:number-of-months`, used in chunk 22a.
Uses `master-proto` 7 and `number-of-months` 7.

4.5 The `:number-of-data-values` Method

13a ⟨*The Master :number-of-data-values Method* 13a⟩≡ (10)
```
  (defmeth master-proto :number-of-data-values ()
    "Method args: ()
  Returns n, the number of data values, or sets."
    (slot-value 'number-of-data-values))
```
Defines:
`:number-of-data-values`, never used.
Uses `master-proto` 7 and `number-of-data-values` 7.

4.6 The `:summary-data` Method

13b ⟨*The Master :summary-data Method* 13b⟩≡ (10)
```
  (defmeth master-proto :summary-data ()
    "Method args: ()
  Returns the summary data containing pairs theta and number of distinct
  data points. A 2d array of size mn by 2."
    (slot-value 'summary-data))
```
Defines:
`:summary-data`, never used.
Uses `master-proto` 7 and `summary-data` 7.

4.7 The `:indicator-counts` Method

13c ⟨*The Master :indicator-counts Method* 13c⟩≡ (10)
```
  (defmeth master-proto :indicator-counts ()
    "Method args: ()
  Returns the data values a 2d array of size mn by number-of-months."
    (slot-value 'indicator-counts))
```
Defines:
`:indicator-counts`, never used.
Uses `indicator-counts` 7, `master-proto` 7, and `number-of-months` 7.

4.8 The `:log-constants-of-proportionality` Method

13d ⟨*The Master :log-constants-of-proportionality Method* 13d⟩≡ (10)
```
  (defmeth master-proto :log-constants-of-proportionality ()
    "Method args: ()
  Returns a list of values of the log constants of
  proportionality for the posteriors for each Markov chain."
    (slot-value 'log-constants-of-proportionality))
```
Defines:
`:log-constants-of-proportionality`, never used.
Uses `log-constants-of-proportionality` 7 and `master-proto` 7.

4.9 The `:slaves` Method

14a ⟨*The Master :slaves Method* 14a⟩≡ (10)
```
(defmeth master-proto :slaves ()
  "Method args: () Retrieves the slaves."
  (slot-value 'slaves))
```
Defines:
`:slaves`, never used.
Uses `master-proto` 7 and `slaves` 7.

4.10 The `:initially-specified-hyperparameter-values` Method

14b ⟨*The Master :initially-specified-hyperparameter-values Method* 14b⟩≡ (10)
```
(defmeth master-proto :initially-specified-hyperparameter-values ()
  "Method args: ()
Returns a list of initially specified values of the hyper
parameters. Used mainly for resetting the hyper parameter values
before dynamic exploration."
  (slot-value 'initially-specified-hyperparameter-values))
```
Defines:
`:initially-specified-hyperparameter-values`, used in chunks 19b, 22b, and 39b.
Uses `initially-specified-hyperparameter-values` 7 and `master-proto` 7.

4.11 The `:hyperparameter-names` Method

14c ⟨*The Master :hyperparameter-names Method* 14c⟩≡ (10)
```
(defmeth master-proto :hyperparameter-names (&optional index)
  "Method args: (index)
Returns a string identifying the hyperparameter index or the whole slot."
  (if index
       (select (slot-value 'hyperparameter-names) index)
     (slot-value 'hyperparameter-names)))
```
Defines:
`:hyperparameter-names`, used in chunk 24b.
Uses `hyperparameter-names` 7 and `master-proto` 7.

4.12 The `:current-hyperparameter-values` Method

15a ⟨*The Master :current-hyperparameter-values Method* 15a⟩≡ (10)

```
(defmeth master-proto :current-hyperparameter-values (&optional i x)
  "Method args: (&optional i x)
Returns a list of hyper-parameter values. If i is given and a sequence,
then all values in the list are set. The dimensions must match. If i is
not a sequence, returns the i-th value. If x is specified, sets the
i-th hyperparameter value to x. Note that x is ignored if i is a sequence."
  (if x
      (if (slot-value 'timing)
        (time
           (progn
             (setf (elt (slot-value 'current-hyperparameter-values) i) x)
             (send self :synchronize)))
        (progn
           (setf (elt (slot-value 'current-hyperparameter-values) i) x)
           (send self :synchronize)))
    (if i
        (if (sequencep i)
            (progn
                (setf (slot-value 'lazy) t)
                (dotimes (j (send self :number-of-hyperparameters))
                    (send (select (slot-value 'hyperparameter-sliders) j)
                          :value (select i j)))
                (setf (slot-value 'lazy) nil)
                (send self :synchronize))
           (elt (slot-value 'current-hyperparameter-values) i))
      (slot-value 'current-hyperparameter-values))))
```

Defines:
`:current-hyperparameter-values`, used in chunks 25, 38–40, 46a, and 61.
Uses `current-hyperparameter-values` 7, `master-proto` 7, `:number-of-hyperparameters` 15b, and `:synchronize` 38.

4.13 The `:number-of-hyperparameters` Method

15b ⟨*The Master :number-of-hyperparameters Method* 15b⟩≡ (10)

```
(defmeth master-proto :number-of-hyperparameters ()
  "Method args: ()
Returns the number of hyperparameters."
  (length (slot-value 'current-hyperparameter-values)))
```

Defines:
`:number-of-hyperparameters`, used in chunks 15a, 25, and 40b.
Uses `master-proto` 7.

4.14 The **:hyperparameters-used-in-markov-chains** Method

16a ⟨*The Master :hyperparameters-used-in-markov-chains Method* 16a⟩≡ (10)

```
(defmeth master-proto :hyperparameters-used-in-markov-chains ()
  "Method args: ()
Returns the hyperparameter-values used in running the Markov
chains. 2D array of size number-of-markov-chains by 3."
  (slot-value 'hyperparameters-used-in-markov-chains))
```

Defines:
`:hyperparameters-used-in-markov-chains`, never used.
Uses `hyperparameters-used-in-markov-chains` 7, `master-proto` 7, and `number-of-markov-chains` 7.

4.15 The **:hyperparameter-ranges** Method

16b ⟨*The Master :hyperparameter-ranges Method* 16b⟩≡ (10)

```
(defmeth master-proto :hyperparameter-ranges (&optional ranges)
  "Method args: (&optional ranges)
Returns the ranges within which the hyperparameters will be
varied. List of lists. This is calculated as the maximum and minimum
of all the values used in the Markov chains if not given."
  (if ranges
      (setf (slot-value 'hyperparameter-ranges) ranges)
    (if (slot-value 'hyperparameter-ranges)
        (slot-value 'hyperparameter-ranges)
      (let* ((values (column-list
                      (slot-value 'hyperparameters-used-in-markov-chains)))
             (max (mapcar #'max values))
             (min (mapcar #'min values)))
       (append (mapcar #'(lambda (x y) (list x y)) min max)
               (list (list 0 (1- (slot-value 'number-of-months)))))))))
```

Defines:
`:hyperparameter-ranges`, used in chunks 20, 24b, 27d, and 36b.
Uses `hyperparameter-ranges` 7 and `master-proto` 7.

4.16 The **:hyperparameter-sliders** Method

16c ⟨*The Master :hyperparameter-sliders Method* 16c⟩≡ (10)

```
(defmeth master-proto :hyperparameter-sliders ()
  "Method args: ()
Returns the sliders associated with the hyperparameters."
  (slot-value 'hyperparameter-sliders))
```

Defines:
`:hyperparameter-sliders`, never used.
Uses `hyperparameter-sliders` 7 and `master-proto` 7.

4.17 The `:log-mixture-density` Method

17a ⟨*The Master :log-mixture-density Method* 17a⟩≡ (10)

```
(defmeth master-proto :log-mixture-density ()
  "Method args: ()
Returns the log of the mixture density at each of the data points.
An array of nm values."
  (slot-value 'log-mixture-density))
```

Defines:
`:log-mixture-density`, never used.
Uses `log-mixture-density` 7 and `master-proto` 7.

4.18 The `:importance-weights` Method

17b ⟨*The Master :importance-weights Method* 17b⟩≡ (10)

```
(defmeth master-proto :importance-weights ()
  "Method args: ()
Retrieves the importance sampling weights. An array
of size number-of-markov-chains by number-of-points."
  (slot-value 'importance-weights))
```

Defines:
`:importance-weights`, never used.
Uses `importance-weights` 7, `master-proto` 7, `number-of-markov-chains` 7, and `number-of-points` 7.

4.19 The `:loglik` Method

17c ⟨*The Master :loglik Method* 17c⟩≡ (10)

```
(defmeth master-proto :loglik (n)
  "Returns the log-quasi-likelihood as a function of n. The dimension
of n should be one less than the number of Markov chains."
   (let ((result (slot-value 'work-space)))
      (call-by-reference-oldcfun "logLikelihood"
                        (slot-value 'shared-library)
                        (coerce n '(vector c-double)) result)
         (aref result 0)))
```

Defines:
`:loglik`, used in chunk 23c.
Uses `loglik` 54a and `master-proto` 7.

4.20 The `:compute-log-hmix` Method

17d ⟨*The Master :compute-log-hmix Method* 17d⟩≡ (10)

```
(defmeth master-proto :compute-log-hmix ()
"Method args: ()
Computes the log of the mixture density at the various data points."
   (call-by-reference-oldcfun "computeLogHmix"
                        (slot-value 'shared-library)))
```

Defines:
`compute-log-hmix`, used in chunk 23b.
Uses `computeLogHmix` 55 and `master-proto` 7.

4.21 The `:calc-weights` Method

18a ⟨*The Master :calc-weights Method* 18a⟩≡ (10)

```
(defmeth master-proto :calc-weights ()
"Method args: ()
Calculates the importance weights."
  (call-by-reference-oldcfun "calcWeights" (slot-value 'shared-library)))
```

Defines:
`:calc-weights`, used in chunk 18b.
Uses `calcWeights` 56 and `master-proto` 7.

4.22 The `:isnew` Method

The `:isnew` method for `master-proto` is a bit involved.

18b ⟨*The Master :isnew Method* 18b⟩≡ (10)

```
(defmeth master-proto :isnew (&key identifier?
                                   number-of-markov-chains?
                                   number-of-points?
                                   file-names?
                                   log-constants?)
  "Method args: (&key identifier?
                     number-of-markov-chains?
                     number-of-points
                     file-names?
                     log-constants?)
Please refer to the literate program and the paper for a thorough
discussion."
  ⟨Set up slot values for master object 18c⟩
  (send self :calc-weights)
  ⟨Create slaves of master object 24a⟩
  ⟨Set up dialog and wait for user input 24b⟩
  ⟨Some final touches 25c⟩)
```

Defines:
`:isnew`, used in chunk 51.
Uses `:calc-weights` 18a, `identifier` 7, `master-proto` 7, `number-of-markov-chains` 7, and `number-of-points` 7.

The `:synchronize` message at the end of this method forces the computations anyway, so we avoid the computations now. In other words, initially, we want to be *lazy* in performing intensive computations.

18c ⟨*Set up slot values for master object* 18c⟩≡ (18b) 19a ▷

```
(setf (slot-value 'work-space)
      (make-array 1 :initial-element 0.0 :element-type 'c-double))
(setf (slot-value 'lazy) t)
```

We need to determine m, the number of chains and n, the number of data points to be used. If the argument `identifier?` is given, then all the necessary inputs can be read from the "run" file. Otherwise, we need to prompt the user for everything.

19a ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◃18c 19b▹

```
(setf (slot-value 'hyperparameter-names)
      (list "M(R)" "Theta-Shape" "Theta-Scale" "Time (months)"))
(setf (slot-value 'shared-library)
          (shlib::shlib-open *default-shared-lib-name*))
(setf (slot-value 'statistics) (make-array 3 :initial-element 0.0
                                    :element-type 'c-double))
(setf (slot-value 'statistics-print-formats)
       *default-statistics-print-formats*)
(setf (slot-value 'statistics-labels)
       *default-statistics-labels*)
(if identifier?
    (progn
        (setf (slot-value 'identifier) identifier?)
        (send self :process-run-file :number-of-points? number-of-points?))
  (send self :graphical-interface :identifier? identifier?
             :number-of-markov-chains? number-of-markov-chains?
             :number-of-points? number-of-points?
             :file-names? file-names?
             :log-constants? log-constants?))
```

Uses `*default-shared-lib-name*` 43a, `*default-statistics-labels*` 47a, `*default-statistics-print-formats*` 47a, `:identifier` 12a, `identifier` 7, `:number-of-markov-chains` 12b, `number-of-markov-chains` 7, `:number-of-points` 12c, `number-of-points` 7, `:process-run-file` 28b, `statistics` 7 50, `statistics-labels` 7, and `statistics-print-formats` 7.

Time to get information on how the exploration is to proceed.

19b ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◃19a 22a▹

```
⟨Set up Hyperparameter-ranges and stops 20⟩
(setf (slot-value 'current-hyperparameter-values)
      (copy-vector (send self :initially-specified-hyperparameter-values)))
```

Uses `:initially-specified-hyperparameter-values` 14b and `initially-specified-hyperparameter-values` 7.

This code chunk sets up a large dialog box for all the various quantities. In most cases, the user will just continue without change.

20 ⟨*Set up Hyperparameter-ranges and stops* 20⟩≡ (19b)

```
(let* ((dialog-items ())
       (result nil)
       (ranges (select (send self :hyperparameter-ranges) (iseq 3)))
       (init-vals (if identifier?
                      (slot-value 'initially-specified-hyperparameter-values)
                     (mapcar #'(lambda(x) (* 0.5 (sum x))) ranges)))
       (stop-vals (if identifier?
                      (slot-value 'number-of-slider-stops)
                     (mapcar #'(lambda(x) (1+ (- (second x) (first x))))
                                 ranges)))
       (hypernames (select (slot-value 'hyperparameter-names) (iseq 3)))
       (prompt-item
        (send text-item-proto :new
              (format nil
                "You can change the interval between which the~%~
                hyperparameters may be varied and the number~%~
                of stops in the interval (end points included).~%~
               Please note that NO ERROR CHECKING is done!~%~%~
               The default settings computed from the data files~%~
               are shown below. Please click OK when done.")))
       (col-1 (send text-item-proto :new "Hyperparameter"))
       (col-2 (send text-item-proto :new "Minimum"))
       (col-3 (send text-item-proto :new "Maximum"))
       (col-4 (send text-item-proto :new "Stops"))
       (col-5 (send text-item-proto :new "Initial"))
       (headings (list col-1 col-2 col-3 col-4 col-5))
       (widths (map-elements #'send headings :width))
       (hnames (mapcar #'(lambda(x)
                           (send text-item-proto :new x)) hypernames))
       (mins (mapcar #'(lambda(x)
                         (send edit-text-item-proto :new
                               (format nil "~g" (select x 0))
                               :text-length 10))
                     ranges))
       (maxs (mapcar #'(lambda(x)
                         (send edit-text-item-proto :new
                               (format nil "~g" (select x 1))
                               :text-length 10))
                     ranges))
       (stops (mapcar #'(lambda(x)
                          (send edit-text-item-proto :new
                            (format nil "~g" x)
                            :text-length 10))
                      stop-vals))
       (inits (mapcar #'(lambda(x)
                          (send edit-text-item-proto :new
                            (format nil "~g" x)
                            :text-length 10))
```

```
                      init-vals))
          (abort (send modal-button-proto :new "Abort" :action #'top-level))
          (dialog nil)
          (ok (send modal-button-proto :new "OK"
                    :action
                    #'(lambda()
                        (list
                         (map-elements #'send mins :text)
                         (map-elements #'send maxs :text)
                         (map-elements #'send stops :text)
                         (map-elements #'send inits :text))))))
     (dotimes (i (length hnames))
              (send (select hnames i) :width (select widths 0))
              (send (select mins i) :width (select widths 1))
              (send (select maxs i) :width (select widths 2))
              (send (select stops i) :width (select widths 3))
              (send (select inits i) :width (select widths 4)))
     (setf dialog-items
           (list prompt-item
                 headings
                 (list (mapcar #'(lambda (x y z w u) (list x y z w u))
                               hnames mins maxs stops inits))
                 (list ok abort)))
     (setf dialog (send modal-dialog-proto :new  dialog-items))
     (send dialog :title "Hyperparameter Ranges and Stops")
     (setf result (mapcar #'convert-to-numbers (send dialog :modal-dialog)))
     (send self :hyperparameter-ranges
         (append
             (mapcar #'(lambda(x y) (list x y))
                     (select result 0) (select result 1))
           (list (list 0 (1- (slot-value 'number-of-months))))))
     (setf (slot-value 'initially-specified-hyperparameter-values)
       (make-array 4 :element-type 'c-double
           :initial-contents (append (select result 3)
                                     (list
                                      (truncate
                                        (* 0.5
                                        (slot-value 'number-of-months)))))))
     (setf (slot-value 'number-of-slider-stops)
          (append (select result 2) (list (slot-value 'number-of-months)))))
```
Uses `:hyperparameter-ranges` 16b, `hyperparameter-ranges` 7, and `identifier` 7.

Now, make the initial arrays for calling the C functions.

22a ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◁19b 22b▷
```
(setf (slot-value 'density-abscissae)
       (make-array *default-number-of-plot-stops*
             :initial-contents (rseq 0 1 *default-number-of-plot-stops*)
              :element-type 'c-double))
(setf (slot-value 'density-ordinates)
       (make-array *default-number-of-plot-stops*
             :initial-element 0 :element-type 'c-double))
(let ((number-of-stops (send self :number-of-months)))
    (setf (slot-value 'expectation-abscissae)
            (make-array number-of-stops
               :initial-contents (iseq 0 (1- number-of-stops))
               :element-type 'c-double))
    (setf (slot-value 'expectation-ordinates)
            (make-array number-of-stops
               :initial-element 0 :element-type 'c-double))
    (setf (slot-value 'standard-deviation-ordinates)
            (make-array number-of-stops
               :initial-element 0 :element-type 'c-double)))
```
Uses `:number-of-months` 12d and `number-of-months` 7.

The current values of the hyperparameters must be those specified initially.

22b ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◁22a 22c▷
```
(let ((tmp-array (send self :initially-specified-hyperparameter-values)))
  (setf (slot-value 'current-hyperparameter-values)
      (make-array (length tmp-array) :initial-contents tmp-array
              :element-type 'c-double)))
```
Uses `:initially-specified-hyperparameter-values` 14b and `initially-specified-hyperparameter-values` 7.

We set up the number of points to be used in Reverse Logistic Regression if necessary. After that, all the data areas will have been set up and so we pass addresses of the data-arrays to the C routines.

22c ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◁22b 23b▷
```
(let* ((n (slot-value 'number-of-points))
       (nc (min *default-number-of-points* n)))
    (unless (or log-constants? identifier?
            (= (slot-value 'number-of-markov-chains) 1))
      (setf nc
        (select (get-tested-value-dialog
           (format nil "How many points should I use from each chain for~%~
                estimating the constants of proportionality?~%~
                Unless you are prepared to wait for a long time, you~%~
                should go with the default or less!~%")
         :initial nc
         :test #'(lambda(x) (and (numberp x) (> x 0) (<= x n)))
         :error-message "Invalid entry. Please try again!")
        0)))
  ⟨Pass data array addresses to C Routines 23a⟩)
```
Uses `*default-number-of-points*` 43a, `identifier` 7, and `number-of-points` 7.

23a ⟨*Pass data array addresses to C Routines* 23a⟩≡ (22c)
```
(call-by-reference-oldcfun "initializeAddress"
          (slot-value 'shared-library) (slot-value 'summary-data)
          (slot-value 'hyperparameters-used-in-markov-chains)
          (slot-value 'current-hyperparameter-values)
          (slot-value 'importance-Weights)
          (slot-value 'log-mixture-density)
          (slot-value 'log-constants-of-proportionality)
          (slot-value 'number-of-markov-chains)
          (slot-value 'number-of-points)
          (slot-value 'number-of-months)
          (slot-value 'number-of-data-values)
          (slot-value 'indicator-counts)
          (slot-value 'statistics)
          nc)
```
Uses `initializeAddress` 51.

If the constants of proportionality have to be estimates, this is the time to do it after duly notifying the user.

23b ⟨*Set up slot values for master object* 18c⟩+≡ (18b) ◁22c
```
(unless (or log-constants? identifier?)
    (unless (= (slot-value 'number-of-markov-chains) 1)
        (message-dialog "The Constants of proportionality will now be~%~
                         be estimated by Reverse Logistic Regression.~%~
                         This will take a while and only happens once.~%~%~
                         You will see the results of iterations on the~%~
                         Console as the maximization is done.~%~%
                         Click OK to continue.")
        ⟨Perform Reverse Logistic Regression 23c⟩))
(unless identifier?
        (send self :compute-log-hmix)
        (send self :create-run-file))
```
Uses `compute-log-hmix` 17d and `identifier` 7.

23c ⟨*Perform Reverse Logistic Regression* 23c⟩≡ (23b)
```
(flet ((loglik (n) (send self :loglik n)))
   (setf initial-guess
         (nelmeadmax #'loglik (slot-value 'log-constants-of-proportionality)
                  :epsilon *default-maximization-tolerance*
                  :count-limit 50000))
   (dotimes (i (length initial-guess))
        (setf (aref (slot-value 'log-constants-of-proportionality) i)
              (select initial-guess i))))
```
Uses `*default-maximization-tolerance*` 43a, `:loglik` 17c, and `loglik` 54a.

We are ready to create the slave objects, a plot of the density of $F(t)$ as well as a plot of $E(S(t))$ for various values of t.

24a ⟨*Create slaves of master object* 24a⟩≡ (18b)

```
(setf (slot-value 'slaves)
        (list
            (send slave-proto :new self)
            (send slave-proto :new self :survival-plot t)))
(send (second (slot-value 'slaves)) :title
                     (strcat "Plot of E(S(t))-" (slot-value 'identifier)))
```

Uses `slave-proto` 43b.

The sliders for controlling the hyperparameters need to be set up. There are three dialog items that correspond to every hyperparameter: a `text-item` where the hyperparameter name will be displayed, a `value-text-item` where the value of the hyperparameter will be displayed and underneath the first two, a slider showing the slider stop. Figure 1 shows a typical slider. For nice looks, the width of the hyperparameter name string and the value string should add up to the width of the slider.

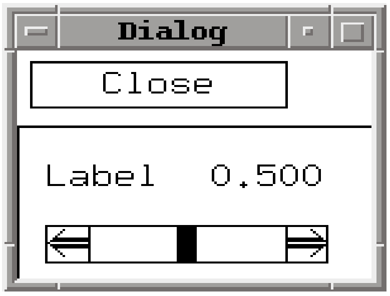

Figure 1: A typical slider in a dialog

24b ⟨*Set up dialog and wait for user input* 24b⟩≡ (18b)

```
(let* ((hyperparameter-labels (send self :hyperparameter-names))
       (hyperparameter-ranges (send self :hyperparameter-ranges))
       (triples ⟨Make triples for sliders 25a⟩)
       (slider-and-dialog-items ⟨Make sliders with triples 25b⟩)
       (dialog-items (select slider-and-dialog-items 0))
       (sliders-alone (select slider-and-dialog-items 1))
       (reset-button (send button-item-proto :new "Reset"
                           :action #'(lambda() (send self :reset))))
       (timing-button (send button-item-proto :new "Timing:OFF"
                           :action #'(lambda() (send self :toggle-timing))))
       (stats-button (send button-item-proto :new "Statistics"
                           :action #'(lambda()
                                       (send self :print-all-statistics)))))
  (setf (slot-value 'hyperparameter-sliders) sliders-alone)
  (setf (slot-value 'timing-button) timing-button)
  (call-next-method (list (list reset-button stats-button timing-button)
                          (list dialog-items))))
```

Uses `:hyperparameter-names` 14c, `hyperparameter-names` 7, `:hyperparameter-ranges` 16b, `hyperparameter-ranges` 7, `:print-all-statistics` 40a, `:reset` 39b, `statistics` 7 50, `timing` 7, and `:toggle-timing` 42a.

Let us first create the triples we need for the hyperparameter sliders for use with the function `make-sliders`. The triples are (a) the label for the hyperparameter, (b) the range within which the hyperparameter will be varied, and (c) the action that is to be taken when the slider is pressed as a function.

25a ⟨*Make triples for sliders* 25a⟩≡ (24b)

```
(mapcar
   #'(lambda(x y z)
        (list x y #'(lambda(w)
                       (send self :current-hyperparameter-values z w))))
   hyperparameter-labels hyperparameter-ranges
        (iseq (send self :number-of-hyperparameters)))
```

Uses `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `hyperparameter-ranges` 7, and `:number-of-hyperparameters` 15b.

We are now ready to create the sliders and associated text and value items. The function `make-sliders` returns a `multiple-values` list of `slider-items` using `triples` with a specified layout along with a list of the appropriate `scroll-items`. A `slider-item` is a list of two elements, the first element being a list containing a `text-item` and a `value-item` and the second is a `scroll-item`.

25b ⟨*Make sliders with triples* 25b⟩≡ (24b)

```
(multiple-value-list
    (make-sliders triples
          :layout (list (list t) (list t t) (list t))
          :formats *default-hyperparameter-print-format*
          :no-of-slider-stops (slot-value 'number-of-slider-stops)))
```

Uses `*default-hyperparameter-print-format*` 43a.

Finally, we create a decent title, become alert once again as opposed to being lazy and then synchronize everything and show those windows.

25c ⟨*Some final touches* 25c⟩≡ (18b)

```
(send self :title (strcat (send self :identifier) "-Master"))
(let ((hd (send self :number-of-hyperparameters))
      (hyper-sliders (slot-value 'hyperparameter-sliders))
      (hyper-vals (send self :current-hyperparameter-values)))
   (setf hyper-vals (coerce hyper-vals 'list))
   (send (select hyper-sliders 1) :display-value)
   (dotimes (l hd)
      (unless (= l 1)
        (send (select hyper-sliders l) :value (select hyper-vals l)))))

(setf (slot-value 'lazy) nil)
(send self :synchronize)
```

Uses `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `:identifier` 12a, `identifier` 7, `:number-of-hyperparameters` 15b, and `:synchronize` 38.

4.23 The `:process-run-file` Method

In handling the "run" file, note the assumptions made about the file structure and how some irrelevant headings are expected and skipped.

26a ⟨*The Master :process-run-file Method* 26a⟩≡ (10)

```
(defmeth master-proto :process-run-file (&key number-of-points?)
"Method args: (&key number-of-points?)
 Processes a run file to get all inputs."
(format t "~%Processing Run file ... ")
(let ((fh (open (strcat (slot-value 'identifier) ".run") :direction :input))
      (file-names nil)
      (alpha-shape-scale nil)
      (constants nil)
      (m nil)
      (n nil)
      (mn nil)
      (index 0)
      (indicator-counts nil)
      (summary-data nil)
      (log-hmix nil))
  ⟨Read number of Markov chains 26b⟩
  ⟨Read number of points 27a⟩
  (setf mn (* m n))
  ⟨Read number of sets 27b⟩
  ⟨Read number of months 27c⟩
  (setf (slot-value 'importance-weights)
        (make-array mn :initial-element 0.0 :element-type 'c-double))
  ⟨Process the table containing info on Markov chains 28a⟩
  ⟨Process Hyperparameter Ranges etc. 27d⟩
  ⟨Read summary data, log mixture density, indicator counts 28b⟩
  (setf (slot-value 'indicator-counts) indicator-counts)
  (setf (slot-value 'summary-data) summary-data)
  (setf (slot-value 'log-mixture-density) log-hmix)
  (close fh))
  (format t "done~%"))
```

Uses `indicator-counts` 7, `master-proto` 7, `number-of-points` 7, `:process-run-file` 28b, and `summary-data` 7.

26b ⟨*Read number of Markov chains* 26b⟩≡ (26a)

```
;; Ignore the information in first line---just some info.
(read fh nil)
;; Ignore the heading "Number of Markov chains"
(read fh nil)
(setf m (read fh nil))
(setf (slot-value 'number-of-markov-chains) m)
```

27a ⟨*Read number of points* 27a⟩≡ (26a)

```
;; Ignore the heading "Number of Points"
(read fh nil)
(setf n (read fh nil))
(if number-of-points?
    (setf n number-of-points?))
(setf (slot-value 'number-of-points) n)
```

Uses `number-of-points` 7.

27b ⟨*Read number of sets* 27b⟩≡ (26a)

```
;; Ignore the heading "Number of Data Values"
(read fh nil)
(setf (slot-value 'number-of-data-values) (read fh nil))
```

27c ⟨*Read number of months* 27c⟩≡ (26a)

```
;; Ignore the heading "Number of Months"
(read fh nil)
(setf (slot-value 'number-of-months) (read fh nil))
```

27d ⟨*Process Hyperparameter Ranges etc.* 27d⟩≡ (26a)

```
;; Ignore the leading line
(read fh nil)
(let ((range nil)
      (r (iseq 3))
      (s (iseq 3))
      (i (iseq 3)))
  (dotimes (j 3)
    (read fh nil) (read fh nil)
    (setf range (cons (read fh nil) nil))
    (read fh nil)
    (setf range (cons (read fh nil) range))
    (setf (select r j) (reverse range))
    (read fh nil)
    (setf (select i j) (read fh nil))
    (read fh nil)
    (setf (select s j) (read fh nil)))
  (send self :hyperparameter-ranges
        (append r
         (list (list 0 (1- (slot-value 'number-of-months))))))
  (setf (slot-value 'number-of-slider-stops)
        (append s (list (slot-value 'number-of-months))))
  (setf (slot-value 'initially-specified-hyperparameter-values)
        (append i (list (* 0.5 (slot-value 'number-of-months))))))
```

Uses `:hyperparameter-ranges` 16b and `hyperparameter-ranges` 7.

28a ⟨*Process the table containing info on Markov chains* 28a⟩≡ (26a)

```
;; Now read the table of file name, alpha,
;;    shape, scale and log constant after ignoring the heading.
(read fh nil)
(dotimes (i m)
     (setf file-names (cons (read fh nil) file-names))
     (setf alpha-shape-scale (cons (read fh nil) alpha-shape-scale))
     (setf alpha-shape-scale (cons (read fh nil) alpha-shape-scale))
     (setf alpha-shape-scale (cons (read fh nil) alpha-shape-scale))
     (setf constants (cons (read fh nil) constants)))
  (setf (slot-value 'data-file-names) (reverse file-names))
  (setf (slot-value 'log-constants-of-proportionality)
     (if (= m 1)
         (make-array 1 :initial-element 0
                    :element-type 'c-double)
       (make-array (1- m) :initial-contents (rest (reverse constants))
                    :element-type 'c-double)))
  (setf (slot-value 'hyperparameters-used-in-markov-chains)
        (make-array (list m 3)
                 :initial-contents (reverse alpha-shape-scale)
                 :element-type 'c-double))
```

28b ⟨*Read summary data, log mixture density, indicator counts* 28b⟩≡ (26a)

```
;; Ignore heading.
(read fh nil)
(setf indicator-counts
      (make-array (list mn (slot-value 'number-of-months))
                 :element-type 'c-long))
(setf summary-data
        (make-array (list mn 2) :initial-element 0.0
                                :element-type 'c-double))
(setf log-hmix
      (make-array mn :initial-element 0.0 :element-type 'c-double))
(dotimes (j n)
   (dotimes (k m)
       (setf index (+ (* n k) j))
       (setf (aref summary-data index 0) (read fh nil))
       (setf (aref summary-data index 1) (read fh nil))
       (setf (aref log-hmix index) (read fh nil))
       (dotimes (l (slot-value 'number-of-months))
         (setf (aref indicator-counts index l) (truncate (read fh nil))))))
```

Defines:
  `:process-run-file`, used in chunks 19a and 26a.
Uses `indicator-counts` 7 and `summary-data` 7.

4.24 The `:process-frequency-table` Method

29a ⟨*The Master :process-frequency-table Method* 29a⟩≡ (10)
```
(defmeth master-proto :process-frequency-table (fh nd ind)
  "Args: (fh nd ind)
This is a convenience method used in reading the data files. It reads
the frequency table of size nd from fh, the file stream, and fills
the indicator-counts slot at index ind."
   (let ((x-vals (make-array nd))
         (freq (make-array nd))
         (sort-index nil)
         (indicator-counts (slot-value 'indicator-counts))
         (number-of-months (slot-value 'number-of-months))
         (j 0)
         (k 0)
         (freq-sum 0)
         (smallest-month 0))
     ⟨Read frequency table and sort values 29b⟩
     (dotimes (i nd)
         (setf smallest-month
              ⟨Find smallest month not less than i-th ordered x-val 29c⟩))
         (when smallest-month
              ⟨Fill table entries with current frequency sum 29d⟩
              ⟨Bump starting index for next stretch 29e⟩
              ⟨Update frequency sum 30a⟩))
     ⟨Handle situation when x-values run out before month values 30b⟩))
```
Defines:
`:process-frequency-table`, used in chunk 34c.
Uses `indicator-counts` 7, `master-proto` 7, and `number-of-months` 7.

29b ⟨*Read frequency table and sort values* 29b⟩≡ (29a)
```
(dotimes (l nd)
         (setf (aref x-vals l) (read fh nil))
         (setf (aref freq l) (read fh nil)))
(setf sort-index (make-array nd :initial-contents (order x-vals)))
```
In the code snippet below, `nil` is returned if no such month is found.

29c ⟨*Find smallest month not less than i-th ordered x-val* 29c⟩≡ (29a)
```
(loop
    (if (>= j number-of-months)
           (return nil))
    (if (>= j (elt x-vals (elt sort-index i)))
          (return j))
    (setf j (1+ j)))
```
Uses `number-of-months` 7.

29d ⟨*Fill table entries with current frequency sum* 29d⟩≡ (29a)
```
(dotimes (x (- smallest-month k))
   (setf (aref indicator-counts ind (+ x k)) freq-sum))
```
Uses `indicator-counts` 7.

29e ⟨*Bump starting index for next stretch* 29e⟩≡ (29a)
```
(setf k smallest-month)
```

30a ⟨*Update frequency sum* 30a⟩≡ (29a)

```
(setf freq-sum (+ freq-sum (elt freq (elt sort-index i))))
```

30b ⟨*Handle situation when x-values run out before month values* 30b⟩≡ (29a)

```
(dotimes (x (- number-of-months k))
     (setf (aref indicator-counts ind (+ x k)) freq-sum))
```

Uses `indicator-counts` 7 and `number-of-months` 7.

4.25 The :graphical-interface Method

31 ⟨*The Master :graphical-interface Method* 31⟩≡ (10)

```
  (defmeth master-proto :graphical-interface
                              (&key identifier?
                                number-of-markov-chains?
                                number-of-points?
                                file-names?
                                log-constants?)
  "Method args: (&key identifier?
                                number-of-chains?
                                number-of-points
                                file-names?
                                log-constants?)
  Sets up a graphical interface for some inputs if not specified."
    (let* ((id (if identifier?
                   identifier?
                  ⟨Get an identifier 32a⟩))
           (m (if number-of-markov-chains?
                  number-of-markov-chains?
                 ⟨Get the number of Markov chains 32b⟩))
           (file-names (if file-names?
                           file-names?
                          ⟨Get data file names 33⟩))
           (n (if number-of-points?
                  number-of-points?
                 ⟨Get the number of points 34a⟩))
           (initial-guess (if log-constants?
                              log-constants?
                            (if (> m 1)
                                ⟨Get initial guess 34b⟩))))
      (setf (slot-value 'identifier) id)
      (setf (slot-value 'data-file-names) file-names)
      (setf (slot-value 'number-of-markov-chains) m)
      (setf (slot-value 'number-of-points) n)
      (setf (slot-value 'number-of-months) *default-number-of-months*)
      (setf (slot-value 'indicator-counts)
            (make-array (list (* m n) (slot-value 'number-of-months))
                     :element-type 'c-long))
      (setf (slot-value 'importance-weights)
             (make-array (* m n) :initial-element 0 :element-type 'c-double))
      (setf (slot-value 'log-mixture-density)
              (make-array (* m n) :initial-element 0
                  :element-type 'c-double))
      (setf (slot-value 'log-constants-of-proportionality)
         (if (= m 1)
             (make-array 1
                  :initial-element 0
                  :element-type 'c-double)
           (make-array (1- m)
                 :initial-contents initial-guess
```

```
                 :element-type 'c-double)))
      (let* ((alphas nil)
             (shapes nil)
             (scales nil)
             (summary-data (make-array (list (* m n) 2) :initial-element 0
                                       :element-type 'c-double )))
          ⟨Read in data files and set up data 34c⟩))))
```

Uses `*default-number-of-months*` 43a, `identifier` 7, `master-proto` 7, `number-of-markov-chains` 7, `number-of-months` 7, `number-of-points` 7, and `summary-data` 7.

32a ⟨*Get an identifier* 32a⟩≡ (31)

```
  (get-nonempty-string-dialog
   (format nil "Please enter a unique short descriptive name~%~
                for this exploration.~%~
                Ex: CancerData") :initial "BreastCancer")
```

32b ⟨*Get the number of Markov chains* 32b⟩≡ (31)

```
  (select (get-tested-value-dialog
           (format nil "How many Markov chains for ~a?" id)
           :initial *default-number-of-markov-chains*
           :test #'(lambda(x) (and (numberp x) (> x 0)))
           :error-message "No. of Markov chains must be >= 1!")
          0)
```

Uses `*default-number-of-markov-chains*` 43a and `number-of-markov-chains` 7.

33 ⟨*Get data file names* 33⟩≡ (31)

```
(let* ((dialog-items ())
       (filenames nil)
       (prompt-item
           (send text-item-proto :new
               (format nil "Please enter names of all data files without~%~
                 the extensions for run ~a and click OK.~%" id)))
       (col-1 (send text-item-proto :new "MC number"))
       (col-2 (send text-item-proto :new "   Data Filename   "))
       (headings (list col-1 col-2))
       (width-a (send col-1 :width))
       (width-b (send col-2 :width))
       (widths (map-elements #'send headings :width))
       (file-name-items (map-elements
                         #'send edit-text-item-proto :new
                         (repeat "" m)
                         :text-length 30))
       (mc-numbers (mapcar #'(lambda(x)
                               (send text-item-proto :new
                                    (format nil "~5d" x)))
                                    (1+ (iseq m))))
       (abort (send modal-button-proto :new "Abort" :action #'top-level))
       (dialog nil)
       (ok (send modal-button-proto :new "OK"
                 :action
                 #'(lambda()
                      (map-elements #'send file-name-items :text)))))
  (dolist (x mc-numbers)
        (send x :width width-a))
  (dolist (x file-name-items)
        (send x :width width-b))
  (setf dialog-items
        (list prompt-item
              headings
              (list (mapcar #'(lambda (x y) (list x y))
                            mc-numbers file-name-items))
              (list ok abort)))
  (loop
    (setf dialog (send modal-dialog-proto :new  dialog-items))
    (setf file-names (send dialog :modal-dialog))
    (if (or (some-files-dont-exist (map-elements #'strcat file-names ".in"))
            (some-files-dont-exist (map-elements #'strcat file-names ".out")))
        (message-dialog "Some files don't exist. Please try again!")
     (return file-names))))
```

34a ⟨*Get the number of points* 34a⟩≡ (31)

```
(select (get-tested-value-dialog
          (format nil "How many points to use for ~a in reweighting?" id)
          :initial *default-number-of-points*
          :test #'(lambda(x) (and (numberp x) (> x 0)))
          :error-message "No. of points must be > 0!")
         0)
```

Uses `*default-number-of-points*` 43a and `number-of-points` 7.

34b ⟨*Get initial guess* 34b⟩≡ (31)

```
(select (get-nonnil-value-dialog
          (format nil "Enter an initial guess for estimating the ~% ~
                       constants in the format shown below.~% ~
                       Dimension should be ~d!~%" (1- m))
          :initial (1+ (iseq (1- m)))
          :test #'(lambda(x)
                    (let ((val (select x 0)))
                         (and val (listp val) (= (length val (1- m))))))
          :error-message "Improper guess!")
         0)
```

To process the data, we use both the input and output file used in running the Markov chains.

34c ⟨*Read in data files and set up data* 34c⟩≡ (31) 35a ▷

```
(dotimes (j m)
;;; First process the input file parameters
    (let ((fh (open (strcat (select file-names j) ".in") :direction :input)))
;; discard the first four values
      (dotimes (i 4)
          (read fh nil))
      (setf shapes (cons (read fh nil) shapes))
      (setf scales (cons (read fh nil) scales))
      (setf alphas (cons (read fh nil) alphas))
;; discard next three values (warmup, iterations and gap)
;; Assumes all input files are consistent.
      (read fh nil) (read fh nil) (read fh nil)
      (setf (slot-value 'number-of-data-values) (read fh nil))
      (close fh))
;;; Now process the corresponding output file.
    (let ((fh (open (strcat (select file-names j) ".out") :direction :input))
          (ind (* j n))
          (nd nil))
      (dotimes (i n)
          (setf (aref summary-data ind 1) (read fh nil))
          (setf nd (read fh nil))
          (setf (aref summary-data ind 0) nd)
          (send self :process-frequency-table fh nd ind)
          (setf ind (1+ ind)))
          (close fh)))
```

Uses `:process-frequency-table` 29a and `summary-data` 7.

Now that all the Markov chain files have been processed, the slot-values can be set up.

35a ⟨*Read in data files and set up data* 34c⟩+≡ (31) ◁34c

```
(setf (slot-value 'hyperparameters-used-in-markov-chains)
      (make-array (list m 3)
           :initial-contents (bind-columns (reverse alphas)
                                           (reverse shapes)
                                           (reverse scales))
           :element-type 'c-double))
(setf (slot-value 'summary-data) summary-data)
```

Uses summary-data 7.

4.26 The **:create-run-file** Method

35b ⟨*The Master :create-run-file Method* 35b⟩≡ (10)

```
(defmeth master-proto :create-run-file ()
"Method args: ()
Creates a data and a lisp file for subsequent runs."
(format t "~%Creating Run file for subsequent runs ... ")
(let ((mn nil)
      (index 0)
      (m (slot-value 'number-of-markov-chains))
      (n (slot-value 'number-of-points))
      (p (slot-value 'number-of-data-values))
      (file-names (slot-value 'data-file-names))
      (summary-data (slot-value 'summary-data))
      (indicator-counts (slot-value 'indicator-counts))
      (hypers-used (slot-value 'hyperparameters-used-in-markov-chains))
      (log-constants (slot-value 'log-constants-of-proportionality))
      (log-hmix (slot-value 'log-mixture-density))
      (fh (open (strcat (slot-value 'identifier) ".run")
                :direction :output)))
  (setf mn (* m n))
  ⟨Write number of Markov chains, etc. 36a⟩
  ⟨Write table of info on Markov chains 36c⟩
  ⟨Write info on hyperparameters 36b⟩
  ⟨Write summary data, log mixture density, indicator counts 37a⟩
  (close fh))
⟨Create lisp file for subsequent runs 37b⟩
(format t "done~%")
⟨Show informative message 37c⟩)
```

Uses indicator-counts 7, master-proto 7, and summary-data 7.

36a ⟨*Write number of Markov chains, etc.* 36a⟩≡ (35b)
```
(format fh "~s~%~%" "Automatically generated file. Do not edit unless~
                     you know what you are doing!")
(format fh "~s ~g~%" "Number of Markov Chains" m)
(format fh "~s ~g~%" "Number of Points" n)
(format fh "~s ~g~%" "Number of Data Values"
        (slot-value 'number-of-data-values))
(format fh "~s ~g~%" "Number of Months used"
        (slot-value 'number-of-months))
```

36b ⟨*Write info on hyperparameters* 36b⟩≡ (35b)
```
(let ((r (send self :hyperparameter-ranges))
      (s (slot-value 'number-of-slider-stops))
      (i (slot-value 'initially-specified-hyperparameter-values)))
  (format fh "~%~s~%"
     "Hyperparameter Exploration Range, Initial Value, stops, etc.")
  (format fh "~%~s ~s ~g ~s ~g ~s ~g ~s ~g~%"
      "M(R)" "Min" (select (select r 0) 0)
             "Max" (select (select r 0) 1)
             "Initial" (select i 0)
             "Stops" (select s 0))
  (format fh "~%~s ~s ~g ~s ~g ~s ~g ~s ~g~%"
      "Theta-Shape" "Min" (select (select r 1) 0)
             "Max" (select (select r 1) 1)
             "Initial" (select i 1)
             "Stops" (select s 1))
  (format fh "~%~s ~s ~g ~s ~g ~s ~g ~s ~g~%"
      "Theta-Scale" "Min" (select (select r 2) 0)
             "Max" (select (select r 2) 1)
             "Initial" (select i 2)
             "Stops" (select s 2)))
```
Uses `:hyperparameter-ranges` 16b and `hyperparameter-ranges` 7.

36c ⟨*Write table of info on Markov chains* 36c⟩≡ (35b)
```
(format fh "~%~%~s~%~%"
     "Table of file name, alpha, shape, scale, log constants")
(dotimes (j m)
     (format fh "~s ~g ~g ~g ~g~%"
            (select file-names j)
            (aref hypers-used j 0)
            (aref hypers-used j 1)
            (aref hypers-used j 2)
            (if (= j 0) -1 (elt log-constants (1- j)))))
```

37a ⟨*Write summary data, log mixture density, indicator counts* 37a⟩≡ (35b)

```
(format fh "~%~%~s~%~%"
    "Table of no of distinct values, theta, log-hmix, Delta-X-t.")
(dotimes (j n)
   (dotimes (k m)
      (setf index (+ (* k n) j))
      (format fh "~g ~g " (aref summary-data index 0)
                          (aref summary-data index 1))
      (format fh "~g " (aref log-hmix index))
      (dotimes (l (slot-value 'number-of-months))
         (format fh "~d " (aref indicator-counts index l)))
      (format fh "~%")))
```

Uses `indicator-counts` 7 and `summary-data` 7.

37b ⟨*Create lisp file for subsequent runs* 37b⟩≡ (35b)

```
(let* ((id (slot-value 'identifier))
       (fh (open (strcat id ".lsp") :direction :output)))
    (format fh ";;;Automatically generated file. Do not edit~
                unless you know what you are doing.~%")
    (format fh "(require ~s)~%" "bsa")
    (format fh "(use-package ~s)~%" "BSA")
    (format fh "(defvar ~a-master (send master-proto :new
                              :identifier? ~s))~%" id id)
  (close fh))
```

Uses `:identifier` 12a, `identifier` 7, and `master-proto` 7.

37c ⟨*Show informative message* 37c⟩≡ (35b)

```
(let ((id (slot-value 'identifier)))
  (message-dialog
    (format nil "For your information: Two files~%~
             ~a.run  and  ~a.lsp~%were created.~%~
             To repeat this run quickly the next time around~%~
             you only need to load the file ~a.lsp into~%~
             xlispstat." id id id)))
```

4.27 The `:synchronize` Method

The `:synchronize` method is responsible for synchronizing the slaves so that the density estimates they display is for the current values of the hyperparameters. Thus, if any hyperparameter value is changed, the `:synchronize` method should be invoked.

38 ⟨*The Master :synchronize Method* 38⟩≡ (10)
```
(defmeth master-proto :synchronize ()
  "Method args: ()
Synchronizes all slaves."
   (when (not (slot-value 'lazy))
         (send self :consolidate-computation)
         (let* ((t-format (select *default-hyperparameter-print-format* 3))
                (slave1 (first (slot-value 'slaves)))
                (slave2 (second (slot-value 'slaves)))
                (t-value (send self :current-hyperparameter-values 3)))
              (send slave1
                    :title (format nil "Law of F(~v,vf)-~a"
                                   (first t-format)
                                   (second t-format) t-value
                                   (slot-value 'identifier)))
              (setf (select (slot-value 'statistics-labels) 0)
                  (format nil "E(S(~g))" t-value))
              (setf (select (slot-value 'statistics-labels) 1)
                  (format nil "SD(S(~g))" t-value))
     (send slave1 :start-buffering)
     (send slave1 :clear-lines :draw nil)
     (send slave1 :add-lines
                (slot-value 'density-abscissae)
                (slot-value 'density-ordinates))
     (send slave1 :adjust-to-data)
     (send slave1 :buffer-to-screen)
     (send slave2 :start-buffering)
     (send slave2 :clear-lines :draw nil)
     (send slave2 :add-lines
                  (slot-value 'expectation-abscissae)
                  (slot-value 'expectation-ordinates))
     (send slave2 :adjust-to-data)
     (send slave2 :buffer-to-screen))))
```
Defines:
`:synchronize`, used in chunks 15a and 25c.

Uses `:consolidate-computation` 39a, `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `*default-hyperparameter-print-format*` 43a, `master-proto` 7, and `slaves` 7.

4.28 The `:consolidate-computation` Method

In this method we compute all the relevant statistics such as $E(S(t))$ and $\sigma_{S(t)}$ for various values of t, as well as the effective sample size. This method was added to consolidate all computations for acceptable performance and makes several other methods above superfluous.

39a ⟨*The Master :consolidate-computation Method* 39a⟩≡ (10)
```
(defmeth master-proto :consolidate-computation ()
  "Method args: ()
Consolidates all computations for sake of speed. This method
recalculates the importance weights, computes relevant statistics and
saves them in the slots."
  (call-by-reference-oldcfun "consolidateComputation"
          (slot-value 'shared-library) *default-number-of-plot-stops*
          (slot-value 'statistics)
          (slot-value 'density-abscissae) (slot-value 'density-ordinates)
          (slot-value 'expectation-abscissae)
          (slot-value 'expectation-ordinates)
          (slot-value 'standard-deviation-ordinates)))
```
Defines:
`:consolidate-computation`, used in chunk 38.
Uses `consolidateComputation` 60, `master-proto` 7, and `statistics` 7 50.

4.29 The `:reset` Method

39b ⟨*The Master :reset Method* 39b⟩≡ (10)
```
(defmeth master-proto :reset ()
  "Method args: ()
Resets the state of all objects."
  (send self :current-hyperparameter-values
        (send self :initially-specified-hyperparameter-values)))
```
Defines:
`:reset`, used in chunk 24b.
Uses `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `:initially-specified-hyperparameter-value`
`initially-specified-hyperparameter-values` 7, and `master-proto` 7.

4.30 The `:effective-sample-size` Method

The actual computation of the effective sample size is now done in the C code that computes the importance weights (see section 6.9). We note that that the formula for computing effective sample size was originally used in estimating standard deviations in stratified sampling. It can be used to give ballpark figures, but should not be taken too literally.

39c ⟨*The Master :effective-sample-size Method* 39c⟩≡ (10)
```
(defmeth master-proto :effective-sample-size ()
  (select (slot-value 'statistics) 2))
```
Defines:
`:effective-sample-size`, never used.
Uses `master-proto` 7.

4.31 The :print-all-statistics Method

40a ⟨*The Master :print-all-statistics Method* 40a⟩≡ (10)

```
(defmeth master-proto :print-all-statistics ()
  (let* ((hyperparameter-names (slot-value 'hyperparameter-names))
         (hyper-strings (send self :labelled-hyperparameter-values))
         (max-name-len (max (mapcar #'length hyperparameter-names))))
     (format t "~%~%*** Statistics for ~a ***~%~%" (send self :identifier))
     (format t "Hyperparameter Settings:~%" )
     (dolist (x hyper-strings)
             (format t " ~a~%" x))
     (format t "Total Sample Size     = ~5d.~%"
              (* (slot-value 'number-of-markov-chains)
                 (slot-value 'number-of-points)))
     (format t "Effective Sample Size = ~5d.~%"
              (select (slot-value 'statistics) 2)))
  (dolist (x (slot-value 'slaves))
        (send x :print-summary))
  (format t "~%*** End of Statistics ***~%~%"))
```

Defines:
  `:print-all-statistics`, used in chunks 24b and 61.
Uses `hyperparameter-names` 7, `:identifier` 12a, `identifier` 7, `:labelled-hyperparameter-values` 40b, `master-proto` 7, `:print-summary` 46a, and `statistics` 7 50.

4.32 The :labelled-hyperparameter-values Method

40b ⟨*The Master :labelled-hyperparameter-values Method* 40b⟩≡ (10)

```
(defmeth master-proto :labelled-hyperparameter-values ()
  (let* ((hyper-names (slot-value 'hyperparameter-names))
         (max-name-len (max (mapcar #'length hyper-names))))
     (mapcar
        #'(lambda(a b)
            (let ((y (select *default-hyperparameter-print-format* a)))
               (if (listp y)
                     (format nil "~va = ~v,vf" max-name-len
                                 b (first y) (second y)
                           (send self :current-hyperparameter-values a))
                  (format nil (strcat "~va = " y) max-name-len b
                           (send self :current-hyperparameter-values a)))))
    (iseq (send self :number-of-hyperparameters)) hyper-names)))
```

Defines:
  `:labelled-hyperparameter-values`, used in chunk 40a.
Uses `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `*default-hyperparameter-print-format*` 43a, `master-proto` 7, and `:number-of-hyperparameters` 15b.

4.33 The `:statistics` Method

The `:statistics` method for the master returns the value of the slot `statistics`.

41a ⟨*The Master :statistics Method* 41a⟩≡ (10)

```
(defmeth master-proto :statistics ()
  "Method args: ()
Returns the slot value statistics."
  (slot-value 'statistics))
```

Defines:
`:statistics`, used in chunks 41, 44a, and 45.
Uses `master-proto` 7 and `statistics` 7 50.

4.34 The `:statistics-print-formats` Method

The `:statistics-print-formats` method for the master returns the value of the slot `statistics-print-formats`.

41b ⟨*The Master :statistics-print-formats Method* 41b⟩≡ (10)

```
(defmeth master-proto :statistics-print-formats ()
  "Method args: ()
Returns the slot value statistics-print-formats."
  (slot-value 'statistics-print-formats))
```

Defines:
`:statistics-print-formats`, used in chunks 44a and 45.
Uses `master-proto` 7, `:statistics` 41a, `statistics` 7 50, and `statistics-print-formats` 7.

4.35 The `:statistics-labels` Method

The `:statistics-labels` method for the master returns the value of the slot `statistics-labels`.

41c ⟨*The Master :statistics-labels Method* 41c⟩≡ (10)

```
(defmeth master-proto :statistics-labels ()
  "Method args: ()
Returns the slot value statistics-labels."
  (slot-value 'statistics-labels))
```

Defines:
`:statistics-labels`, used in chunk 45.
Uses `master-proto` 7, `:statistics` 41a, `statistics` 7 50, and `statistics-labels` 7.

4.36 The `:toggle-timing` Method

This method toggles timing on and off.

42a ⟨*The Master :toggle-timing Method* 42a⟩≡ (10)
```
(defmeth master-proto :toggle-timing ()
  "Method args: ()
Toggles timing on and off."
  (send self :hide-window)
  (setf (slot-value 'timing) (not (slot-value 'timing)))
  (if (slot-value 'timing)
      (send (slot-value 'timing-button)
            :slot-value 'text "Timing:ON")
     (send (slot-value 'timing-button) :slot-value 'text
              "Timing:OFF"))
  (send self :show-window))
```
Defines:
`:toggle-timing`, used in chunk 24b.
Uses `master-proto` 7 and `timing` 7.

4.37 The `:superimpose` Method

This method returns the value of the `superimpose` slot.

42b ⟨*The Master :superimpose Method* 42b⟩≡ (10)
```
(defmeth master-proto :superimpose ()
  "Method args: ()
Returns the value of slot superimpose."
  (slot-value 'superimpose))
```
Defines:
`:superimpose`, never used.
Uses `master-proto` 7 and `superimpose` 7.

4.38 The `:close` Method

The `:close` method for the master must close the slaves window that are active. Finally, it must commit hara-kiri.

42c ⟨*The Master :close Method* 42c⟩≡ (10)
```
(defmeth master-proto :close ()
  "Method args: ()
Kills all subordinate slave and commits suicide."
  (dolist (x (slot-value 'slaves))
          (send x :remove))
  (call-next-method))
```
Defines:
`:close`, used in chunk 46b.
Uses `master-proto` 7.

4.39 Defaults for Master

We shall use the prefix “BSA” for Bayesian Sensitivity Analysis.

43a ⟨*Defaults for Master* 43a⟩≡ (7)
```
(defvar *default-master-object-prefix* "BSA-")
(defvar *default-number-of-markov-chains* 8)
(defvar *default-number-of-points* 50)
(defvar *default-number-of-plot-stops* 51)
(defvar *default-no-of-slider-stops* '(64 51 121 61))
(defvar *default-hyperparameter-print-format*
         '((7 2) (7 2) (7 2) (7 2)))
(defvar *default-number-of-months* 61)
(defvar *default-maximization-tolerance* 1e-5)
(defvar *default-shared-lib-name*
        (if (member ':msdos xlisp::*features*)
            "win/bsa.dll"
          "./libbsa@SHLIB_SUFFIX@"))
```
Defines:
- `*default-hyperparameter-print-format*`, used in chunks 25b, 38, and 40b.
- `*default-master-object-prefix*`, never used.
- `*default-maximization-tolerance*`, used in chunk 23c.
- `*default-no-of-slider-stops*`, never used.
- `*default-number-of-markov-chains*`, used in chunk 32b.
- `*default-number-of-months*`, used in chunk 31.
- `*default-number-of-points*`, used in chunks 22c and 34a.
- `*default-shared-lib-name`, used in chunk 19a.

Uses `number-of-markov-chains` 7, `number-of-months` 7, and `number-of-points` 7.

5 The Slave Object Prototype

The Slave prototype inherits from `scatterplot-proto`.

43b ⟨*The Slave Prototype* 43b⟩≡ (5)
```
(defproto slave-proto '(master survival-plot color-index)
  () scatterplot-proto
  "The Slave prototype. Master is its master upon whom the slave
  relies for all data. Survival-plot is nil or t indicating what is
  to be plotted. Color-index is used in superimposition.")
```
⟨*Methods for Slave Prototype* 43c⟩
⟨*Defaults for Slave* 47a⟩

Defines:
- `slave-proto`, used in chunks 24a and 44–46.

The methods for `slave-proto` follow.

43c ⟨*Methods for Slave Prototype* 43c⟩≡ (43b)
⟨*The Slave :isnew Method* 44a⟩
⟨*The Slave :redraw-background Method* 44b⟩
⟨*The Slave :redraw-statistics Method* 45⟩
⟨*The Slave :print-summary Method* 46a⟩
⟨*The Slave :close Method* 46b⟩

5.1 The `:isnew` Method

44a ⟨*The Slave :isnew Method* 44a⟩≡ (43c)

```
(defmeth slave-proto :isnew (master &key (go-away t) (survival-plot nil))
  "Method args: master &rest args
Creates a new instance of the slave-proto object."
  (setf (slot-value 'master) master)
  (setf (slot-value 'survival-plot) survival-plot)
  (setf (slot-value 'color-index) 0)
  (call-next-method 2 :go-away go-away :draw nil)
  (if (not (slot-value 'survival-plot))
    (let ((skip (+ (send self :text-ascent) (send self :text-descent)))
          (len (length (send master :statistics-print-formats))))
         (send self :margin
                    0 (+ (* len skip) (send self :text-descent)) 0 0
                    :draw nil)))
  (send self :redraw))
```

Defines:
`:isnew`, used in chunk 51.
Uses `slave-proto` 43b, `:statistics` 41a, `statistics` 7 50, `:statistics-print-formats` 41b, and `statistics-print-formats` 7.

5.2 The `:redraw-background` Method

44b ⟨*The Slave :redraw-background Method* 44b⟩≡ (43c)

```
(defmeth slave-proto :redraw-background ()
  "Method args: ()
   Redraws the background of the screen"
   (send self :start-buffering)
   (call-next-method)
   (unless (slot-value 'survival-plot)
        (send self :redraw-statistics))
   (send self :buffer-to-screen))
```

Defines:
`:redraw-background`, never used.
Uses `:redraw-statistics` 45, `slave-proto` 43b, and `statistics` 7 50.

5.3 The `:redraw-statistics` Method

45 ⟨*The Slave :redraw-statistics Method* 45⟩≡ (43c)

```
(defmeth slave-proto :redraw-statistics ()
  "Method args: ()
   Redraws the statistics on the screen"
  (let* ((master (slot-value 'master))
         (ascent (send self :text-ascent))
         (descent (send self :text-descent))
         (skip (+ ascent descent))
         (em (send self :text-width "m"))
         (en (send self :text-width "n"))
         (canvas-width (send self :canvas-width))
         (y 0)
         (stats-labels (send master :statistics-labels))
         (name-width (max (mapcar #'(lambda(x) (send self :text-width x))
                                  stats-labels)))
         (value-strings (mapcar #'(lambda(x y)
                                   (format nil x y))
                              (send master :statistics-print-formats)
                              (coerce (send master :statistics) 'list)))
         (value-width (max (mapcar #'(lambda(x)
                                       (send self :text-width x))
                                   value-strings)))
         (x2 (- canvas-width value-width em))
         (x1 (- x2 name-width em)))
      (dotimes (i (length stats-labels))
        (let ((sl (select stats-labels i))
              (sv (select value-strings i)))
          (setf y (+ y skip))
          (send self :draw-text sl x1 y 0 0)
          (send self :draw-text sv x2 y 0 0)))))
```

Defines:
  `:redraw-statistics`, used in chunk 44b.

Uses `slave-proto` 43b, `:statistics` 41a, `statistics` 7 50, `:statistics-labels` 41c, `statistics-labels` 7, `:statistics-print-formats` 41b, and `statistics-print-formats` 7.

5.4 The :print-summary Method

This method basically calculates the mean and variance of beta distributions.

46a ⟨*The Slave :print-summary Method* 46a⟩≡ (43c)

```
(defmeth slave-proto :print-summary ()
  "Method args: ()
 Prints the mean and variance of the Law of F(t)."
   (if (slot-value 'survival-plot)
        (let* ((master (slot-value 'master))
               (x (send master :slot-value 'expectation-abscissae))
               (y (send master :slot-value 'expectation-ordinates))
               (stddevs (send master :slot-value
                                'standard-deviation-ordinates)))
           (format t "~%Table of E(S(t))~%")
           (format t "------------------------------------~%")
           (format t "Time t (months)   E(S(t))   SD(S(t))~%")
           (format t "------------------------------------~%")
           (dotimes (i (length x))
              (format t "~14d      ~5,3f   ~5,3f~%" (aref x i)
                         (aref y i) (aref stddevs i))))
      (let* ((master (slot-value 'master))
             (stats (send master :slot-value 'statistics))
             (stat-print-formats (send master :slot-value
                         'statistics-print-formats))
             (hypers (send master :current-hyperparameter-values))
             (alpha (aref hypers 0))
             (t-value-index (truncate (aref hypers 3))))
         (format t (strcat "Mean of S(~d):      "
                         (first stat-print-formats) "~%")
                t-value-index (elt stats 0))
         (format t (strcat "Std. Dev. of S(~d): "
                         (second stat-print-formats) "~%")
                t-value-index (elt stats 1)))))
```

Defines:
  :print-summary, used in chunk 40a.
Uses :current-hyperparameter-values 15a, current-hyperparameter-values 7, and slave-proto 43b.

5.5 The :close Method

46b ⟨*The Slave :close Method* 46b⟩≡ (43c)

```
(defmeth slave-proto :close ()
    (ok-or-cancel-dialog "Please use the master to quit"))
```

Uses :close 42c and slave-proto 43b.

5.6 Defaults for Slave

47a ⟨*Defaults for Slave* 47a⟩≡ (43b)

```
(defvar *default-slave-plot-size* '(300 265))
(defvar *default-slave-plot-stops* 51)
(defvar *default-statistics-labels*
        '("E(S(t))" "SD(S(t))" "Eff. Sample Size"))
(defvar *default-statistics-print-formats*
        '("~5,3f" "~5,3f" "~5,0f"))
```

Defines:
  `*default-slave-plot-size*`, never used.
  `*default-slave-plot-stops*`, never used.
  `*default-statistics-labels*`, used in chunk 19a.
  `*default-statistics-print-formats*`, used in chunk 19a.
Uses `statistics` 7 50, `statistics-labels` 7, and `statistics-print-formats` 7.

6 The C Programs

In designing the C programs, we assume one thing—that no compaction is done during the lifetime of the master object. This allows us to refrain from passing pointers to data arrays each time a routine is called. Instead, all required pointers could be passed once when the master object is created and they remain fixed for the life of the master. Current versions of `Lisp-Stat` don't do memory compaction or move objects around.

Here's our copyright for C programs.

47b ⟨*C Copyright* 47b⟩≡ (48a)

```
/**
***  $Revision: 1.34 $ of $Date: 1998/07/02 17:16:33 $
***
***  Copyright (C) 1994, 1995, 1998. Doss and Narasimhan
***
***  Hani J. Doss (doss@stat.ohio-state.edu) and
***  B. Narasimhan (naras@stat.stanford.edu)
***
***  This program is free software; you can redistribute it and/or modify
***  it under the terms of the GNU General Public License as published by
***  the Free Software Foundation; either version 2 of the License, or
***  (at your option) any later version.
***
***  This program is distributed in the hope that it will be useful,
***  but WITHOUT ANY WARRANTY; without even the implied warranty of
***  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
***  GNU General Public License for more details.
***
***  You should have received a copy of the GNU General Public License
***  along with this program; if not, write to the Free Software
***  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
**/
```

Our C routines follow.

48a ⟨*C Routines* 48a⟩≡
⟨*C Copyright* 47b⟩
⟨*Header files* 48b⟩
⟨*Beta functions* 49⟩
⟨*Global Variables* 50⟩
⟨*Address Initialization Routine* 51⟩
⟨*The Density Routine* 52⟩
⟨*The LogProbability Routine* 53⟩
⟨*The LogLikelihood Routine* 54a⟩
⟨*The hthetaOverHmix Routine* 54b⟩
⟨*The compute-log-hmix Routine* 55⟩
⟨*The calcWeights Routine* 56⟩
⟨*The computeStatistics Method* 57a⟩
⟨*The computeLawOfFOfT Method* 58a⟩
⟨*The computeMeanOfFBarOfT Method* 59b⟩
⟨*The consolidateComputation Routine* 60⟩

Here are the headers files we will need. Note the declaration for the prototype of the `gamma` function that comes with `Lisp-Stat`. This function is not declared `static` in the `Lisp-Stat` source and is therefore available to us.

48b ⟨*Header files* 48b⟩≡ (48a)
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

6.1 Beta Functions

We need two other functions, `betdens` and `logbeta` are indeed declared `static` in the `Lisp-Stat` source. Therefore, a copy of these routines is needed. In `Lisp-Stat`, the $\ln\Gamma$ function is known as `mygamma`.

49 ⟨*Beta functions* 49⟩≡ (48a)

```
extern double mygamma(double a);

static double logbeta (double a, double b)
{
  static double da = 0.0, db = 0.0, lbeta = 0.0;

  if (a != da || b != db) { /* cache most recent call */
    da = a; db = b;
    lbeta = mygamma(da) + mygamma(db) - mygamma(da + db);
  }
  return(lbeta);
}

static double betadens (double x, double a, double b)
{
  if (x <= 0.0 || x >= 1.0)
     return 0.0;
  return exp(log(x) * (a - 1) + log(1 - x) * (b - 1) - logbeta(a, b));
}
```

Defines:
`betadens`, used in chunk 58a.
`logbeta`, never used.
`mygamma`, used in chunks 52 and 54b.

6.2 Global Variables

We begin with some global variables for all these C routines. Note that these are declared `static`.

50 ⟨*Global Variables* 50⟩≡ (48a)

```
static double *summaryData, *hyperValuesUsedInMarkovChains,
        *currentHyperValues, *importanceWeights,
        *logMixtureDensity,
        *logConstantsOfProportionality;

static double *statistics;

static int numberOfMarkovChains, numberOfPoints,
        numberOfPointsForConstants,
        numberOfDataValues, numberOfMonths, numberOfDataValues,
        *indicatorCounts;
#define LOW_LOG_PROBABILITY -1e10;
```

Defines:
- `currentHyperValues`, used in chunks 51, 54b, and 57–60.
- `hyperValuesUsedinMarkovChains`, never used.
- `importanceWeights`, used in chunks 51 and 56–58.
- `indicatorCounts`, used in chunks 51 and 58b.
- `logConstantsOfProportionality`, used in chunks 51 and 55.
- `logMixtureDensity`, used in chunks 51, 54b, and 55.
- `LOW_LOG_PROBABILITY`, used in chunk 53.
- `numberOfDataValues`, used in chunks 51 and 59a.
- `numberOfMarkovChains`, used in chunks 51 and 53–58.
- `numberOfMonths`, used in chunks 51, 58b, and 60.
- `numberOfPoints`, used in chunks 51 and 55–58.
- `numberOfPointsForConstants`, used in chunks 51 and 54a.
- `statistics`, used in chunks 19a, 24b, 39–41, 44, 45, 47a, 51, 56, 60, and 61.
- `summaryData`, used in chunks 51, 52, 54b, and 58b.

6.3 Initialization Routine

This initialization routine gathers addresses and stores them a static storage area. On the Windows platform, we also need to export the routines defined here.

51 ⟨*Address Initialization Routine* 51⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void initializeAddress (double *a, double *b,
             double *c, double *d, double *e, double *ff, int *g, int *h,
             int *i, int *j, int *k, double *l, int *m);
#endif

/*
 * Initialization routine, must be called as soon as the slots
 * are created in the master :isnew method.
*/
void initializeAddress (argSummaryData, argHyperValuesUsedinMarkovChains,
        argCurrentHyperValues, argImportanceWeights,
        argLogMixtureDensity, argLogConstantsOfProportionality,
        argNumberOfMarkovChains, argNumberOfPoints,
        argNumberOfMonths, argNumberOfDataValues,
        argIndicatorCounts, argStatistics, argNumberOfPointsForConstants)
    double *argSummaryData, *argHyperValuesUsedinMarkovChains,
        *argCurrentHyperValues, *argImportanceWeights,
        *argLogMixtureDensity, *argLogConstantsOfProportionality;
    double *argStatistics;
    int *argNumberOfMarkovChains, *argNumberOfPoints,
        *argNumberOfMonths, *argNumberOfDataValues, *argIndicatorCounts,
        *argNumberOfPointsForConstants;
{
    summaryData = argSummaryData;
    hyperValuesUsedInMarkovChains = argHyperValuesUsedinMarkovChains;
    currentHyperValues = argCurrentHyperValues;
    importanceWeights = argImportanceWeights;
    logMixtureDensity = argLogMixtureDensity;
    logConstantsOfProportionality = argLogConstantsOfProportionality;
    indicatorCounts = argIndicatorCounts;

    numberOfMarkovChains = *argNumberOfMarkovChains;
    numberOfPoints = *argNumberOfPoints;
    numberOfMonths = *argNumberOfMonths;
    numberOfDataValues = *argNumberOfDataValues;
    statistics = argStatistics;
    numberOfPointsForConstants = *argNumberOfPointsForConstants;
}
```

Defines:
`initializeAddress`, used in chunk 23a.

Uses `currentHyperValues` 50, `importanceWeights` 50, `indicatorCounts` 50, `:isnew` 18b 44a, `logConstantsOfProportionality` 50, `logMixtureDensity` 50, `numberOfDataValues` 50, `numberOfMarkovChains` 50, `numberOfMonths` 50, `numberOfPoints` 50, `numberOfPointsForConstants` 50, `statistics` 7 50, and `summaryData` 50.

6.4 The C Equivalent of :f Method

This routine calculates the j-th posterior density at a point. See :f method for master-proto. The formulas used for this and the next three methods all follow the development in [3].

52 ⟨*The Density Routine* 52⟩≡ (48a)

```
double f (argJ, argEta, argIndex)
     int argJ, argIndex;
     double *argEta;
{
   double nj, numberOfDistinctXS = summaryData[2 * argIndex],
          theta = summaryData[2 * argIndex + 1], alpha, shape, scale;
   int index = argJ * 3;

   if (argJ == 0)
      nj = -1.0;
   else
      nj = argEta[argJ - 1];
   alpha = hyperValuesUsedInMarkovChains[index];
   shape = hyperValuesUsedInMarkovChains[index + 1];
   scale = hyperValuesUsedInMarkovChains[index + 2];
   return exp(numberOfDistinctXS * log(alpha) +
              shape * log(scale) + (shape - 1.0) * log(theta) -
              scale * theta - mygamma(shape) + nj);
}
```

Defines:
f, used in chunks 53 and 55.
Uses mygamma 49 and summaryData 50.

6.5 The C Equivalent of :logp Method

This routine calculates the probability that a given point comes from a particular Markov chain. See :logp method for master-proto.

53 ⟨*The LogProbability Routine* 53⟩≡ (48a)

```
double logp (argJ, argEta, argIndex)
     int argJ, argIndex;
     double *argEta;
{
   double tmp, tmp1, sum = 0;
   int k;

   for (k = 0; k < numberOfMarkovChains; k++) {
      tmp = f(k, argEta, argIndex);
      if (k == argJ)
         tmp1 = tmp;
      sum += tmp;
   }
   if (tmp1 > 0.0)
      return log(tmp1 / sum);
   else
      return LOW_LOG_PROBABILITY;
}
```

Defines:
logp, used in chunk 54a.
Uses f 52, LOW_LOG_PROBABILITY 50, and numberOfMarkovChains 50.

6.6 The C Equivalent of `:loglik` Method

The routine that calculates the log-likelihood. See `:loglik` method for `master-proto`.

54a ⟨*The LogLikelihood Routine* 54a⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void logLikelihood (double *a, double *b);
#endif

void logLikelihood (argEta, result)
     double *argEta, *result;
{
   double sum = 0.0;

   int i, j, index = 0;

   for (j = 0; j < numberOfMarkovChains; j++) {
      for (i = 0; i < numberOfPointsForConstants; i++) {
         sum += logp(j, argEta, index);
         index++;
      }
   }
   *result = sum;
}
```

Defines:
`loglik`, used in chunks 17c and 23c.
Uses `logp` 53, `numberOfMarkovChains` 50, and `numberOfPointsForConstants` 50.

6.7 The C Equivalent of `:htheta-over-hmix` Method

See `:htheta-over-hmix` method for `master-proto`.

54b ⟨*The hthetaOverHmix Routine* 54b⟩≡ (48a)

```
double hthetaOverHmix (argIndex)
     int argIndex;
{
   double numberOfDistinctXS, theta, alpha, shape, scale;

   numberOfDistinctXS = summaryData[2 * argIndex];
   theta = summaryData[2 * argIndex + 1];
   alpha = currentHyperValues[0];
   shape = currentHyperValues[1];
   scale = currentHyperValues[2];

   return exp(numberOfDistinctXS * log(alpha) +
                 shape * log(scale) + (shape - 1.0) * log(theta) -
                 scale * theta - mygamma(shape)
                 - logMixtureDensity[argIndex]);
}
```

Defines:
`hthetaOverHmix`, used in chunk 56.
Uses `currentHyperValues` 50, `logMixtureDensity` 50, `mygamma` 49, and `summaryData` 50.

6.8 The C Equivalent of `:compute-log-hmix` Method

See `:compute-log-hmix` method for `master-proto`.

55 ⟨*The compute-log-hmix Routine* 55⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void computeLogHmix ();
#endif
void computeLogHmix ()
{
   int i, k, mn;
   double sum;
   mn = numberOfMarkovChains * numberOfPoints;

   for (i = 0; i < mn; i++) {
        sum = 0.0;
        for (k = 0; k < numberOfMarkovChains; k++) {
            sum += f(k, logConstantsOfProportionality, i);
        }
        logMixtureDensity[i] = log(sum);
   }
}
```

Defines:
`computeLogHmix`, used in chunk 17d.
Uses `f` 52, `logConstantsOfProportionality` 50, `logMixtureDensity` 50, `numberOfMarkovChains` 50, and `numberOfPoints` 50.

6.9 The C Equivalent of `:calc-weights` Method

See `:calc-weights` method for `master-proto`.

56 ⟨*The calcWeights Routine* 56⟩≡ (48a)
```
#ifdef _Windows
__declspec(dllexport) void calcWeights ();
#endif

void calcWeights ()
{
   int i, j, index = 0, sampleSize;
   double tmp, sum = 0.0, meanWeight, sumCenteredWeightsSquared;

   for (j = 0; j < numberOfMarkovChains; j++) {
        for (i = 0; i < numberOfPoints; i++) {
            tmp = hthetaOverHmix(index);
            importanceWeights[index] = tmp;
            sum += tmp;
            index++;
        }
   }
   index = 0;
   sampleSize = numberOfMarkovChains * numberOfPoints;
   meanWeight = 1.0 / sampleSize;
   sumCenteredWeightsSquared = 0.0;
   for (j = 0; j < numberOfMarkovChains; j++) {
        for (i = 0; i < numberOfPoints; i++) {
            importanceWeights[index] /= sum;
            tmp = importanceWeights[index] - meanWeight;
            sumCenteredWeightsSquared += tmp * tmp;
            index++;
        }
   }
   tmp = sumCenteredWeightsSquared / (sampleSize - 1);
   statistics[2] = floor(sampleSize /
                         (1 +  sampleSize * sampleSize * tmp));
}
```
Defines:
`calcWeights`, used in chunks 18a and 60.

Uses `hthetaOverHmix` 54b, `importanceWeights` 50, `numberOfMarkovChains` 50, `numberOfPoints` 50, and `statistics` 7 50.

6.10 The C Equivalent of :compute-statistics Method

See :compute-statistics method for slave-proto. In computing the mean and variance of the mixture of Beta densities, we have to use the facts $E(X) = E(E(X|W))$ and $V(X) = E(V(X|W)) + V(E(X|W))$. In our case, $F(t)$ is a Beta random variable X_i with probability w_i, the normalized importance weight.

57a ⟨*The computeStatistics Method* 57a⟩≡ (48a)
```
#ifdef _Windows
__declspec(dllexport) void computeStatistics (double *a);
#endif

void computeStatistics (argStatistics)
     double *argStatistics;
{
  double alpha = currentHyperValues[0],
         expectation = 0.0,
         expectationOfConditionalSquaredMean = 0.0,
         expectationOfVariance = 0.0,
         tmp,
         sumCenteredWeightsSquared = 0,
         betaMean, betaVariance,
         betaParam1, betaParam2;
  int i, j, index = 0, tValueIndex = (int) currentHyperValues[3];
  ⟨Compute Mean and Variance in C 57b⟩
  argStatistics[0] = 1.0 - expectation;
  argStatistics[1] = sqrt(expectationOfConditionalSquaredMean -
                          expectation * expectation);
}
```
Defines:
computeStatistics, never used.
Uses currentHyperValues 50.

57b ⟨*Compute Mean and Variance in C* 57b⟩≡ (57a 59b)
```
for (j = 0; j < numberOfMarkovChains; j ++) {
   for (i = 0; i < numberOfPoints; i++) {
      betaParam1 = ⟨C Expression for first beta parameter 58b⟩;
      betaParam2 = ⟨C Expression for second beta parameter 59a⟩;
      betaMean = betaParam1 / (betaParam1 + betaParam2);
      tmp = importanceWeights[index] * betaMean;
      betaVariance = betaMean * betaParam2 /
           ((betaParam1 + betaParam2) * (betaParam1 + betaParam2 + 1));
      expectation += tmp;
      expectationOfVariance += importanceWeights[index] * betaVariance;
      expectationOfConditionalSquaredMean += tmp * betaMean;
      index++;
   }
}
```
Uses importanceWeights 50, numberOfMarkovChains 50, and numberOfPoints 50.

6.11 The C Equivalent of `:compute-law-of-f-of-t` Method

See `:compute-law-of-f-of-t` method for `slave-proto`. Note the difference in that some parameters need to be passed in this C routine unlike the Lisp routine.

This method basically calculates for equally spaced values of u in $[0, 1]$ the weighted sum

$$\sum_{g=1}^{mn} w_g^{(\kappa)} \beta\Big(\alpha_{\theta^{(g)}}^{(\kappa)}(t) + \textstyle\sum_{i=1}^{n} \delta_{X_i^{(g)}}(t), M_{\theta^{(g)}}^{(\kappa)} + n - \alpha_{\theta^{(g)}}^{(\kappa)}(t) - \sum_{i=1}^{n} \delta_{X_i^{(g)}}(t)\Big)(u),$$

where $\beta(a, b)(u)$ denotes the beta density with parameters a and b evaluated at u.

58a ⟨*The computeLawOfFOfT Method* 58a⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void computeLawOfFOfT (int *n, double *a, double *b);
#endif

void computeLawOfFOfT (int *n, double *x, double *y)
{
  double alpha, sum, betaParam1, betaParam2;
  int i, j, k, index, tValueIndex;

  alpha = currentHyperValues[0];
  tValueIndex = (int) currentHyperValues[3];
  for (k = 0; k < *n; k++) {
     sum = 0.0;
     index = 0;
     for (j = 0; j < numberOfMarkovChains; j++) {
         for (i = 0; i < numberOfPoints; i++) {
            betaParam1 = ⟨C Expression for first beta parameter 58b⟩;
            betaParam2 = ⟨C Expression for second beta parameter 59a⟩;
            sum += importanceWeights[index] *
                     betadens(x[k], betaParam1, betaParam2);
            index++;
         }
     }
     y[k] = sum;
  }
}
```

Defines:
`computeLawOfFOfT`, used in chunk 60.
Uses `betadens` 49, `currentHyperValues` 50, `importanceWeights` 50, `numberOfMarkovChains` 50, and `numberOfPoints` 50.

Here are the formulas for the two beta density parameters. The first one is of course

$$\alpha_{\theta^{(g)}}^{(\kappa)}(t) + \textstyle\sum_{i=1}^{n} \delta_{X_i^{(g)}}(t),$$

with the understanding that α indicates degree of concentration around the exponential family for theta.

The expression for the first beta parameter follows.

58b ⟨*C Expression for first beta parameter* 58b⟩≡ (57b 58a)

```
(indicatorCounts[index * numberOfMonths + tValueIndex] +
 alpha * (1 - exp(-tValueIndex * summaryData[2*index + 1])))
```

Uses `indicatorCounts` 50, `numberOfMonths` 50, and `summaryData` 50.

The second one is

$$M_{\theta^{(g)}}^{(\kappa)} + n - \alpha_{\theta^{(g)}}^{(\kappa)}(t) - \sum_{i=1}^{n} \delta_{X_i^{(g)}}(t)$$

which in C is given by the snippet below.

59a ⟨*C Expression for second beta parameter* 59a⟩≡ (57b 58a)

```
(alpha + numberOfDataValues - betaParam1)
```

Uses `numberOfDataValues` 50.

6.12 The C Equivalent of `:compute-mean-of-fbar-of-t` Method

See `:compute-mean-of-fbar-of-t` method for `slave-proto`. This method basically calculates for several values of t, the quantity $E(S(t))$. The values of t for which this is done is the same as the list of values used in the slider for t.

59b ⟨*The computeMeanOfFBarOfT Method* 59b⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void computeMeanOfFBarOfT (int *n, double *a,
                              double *b, double *c);
#endif

void computeMeanOfFBarOfT (n, x, y, standardDeviations)
     double *x, *y, *standardDeviations;
     int *n;
{
  double alpha = currentHyperValues[0],
        expectation = 0.0,
        expectationOfConditionalSquaredMean = 0.0,
        expectationOfVariance = 0.0,
        tmp,
        sumCenteredWeightsSquared = 0,
        betaMean, betaVariance,
        betaParam1, betaParam2;
  int i, j, k, index = 0, tValueIndex;

  y[0] = 1.0;
  standardDeviations[0] = 0.0;
  for (k = 1; k < *n; k++) {
     index = 0;
     expectation = 0.0;
     expectationOfConditionalSquaredMean = 0.0;
     expectationOfVariance = 0.0;
     tValueIndex = (int) (x[k]);
     ⟨Compute Mean and Variance in C 57b⟩
     y[k] = 1 - expectation;
     standardDeviations[k] = sqrt(expectationOfConditionalSquaredMean -
                        expectation * expectation);
  }
}
```

Defines:
`computeMeanOfFBarOfT`, used in chunk 60.
Uses `currentHyperValues` 50.

6.13 The :consolidate-computation Method in C

This method consolidates all computation by first computing importance weights, then computing all the statistics in one shot.

60 ⟨*The consolidateComputation Routine* 60⟩≡ (48a)

```
#ifdef _Windows
__declspec(dllexport) void consolidateComputation (int *n,
        double *a, double *b, double *c, double *d, double *e,
        double *ff);
#endif

void consolidateComputation (n, argStatistics, argDensityAbscissae,
                argDensityOrdinates,
                argExpectationAbscissae, argExpectationOrdinates,
                argStandardDeviationOrdinates)
     double *argStatistics, *argDensityAbscissae, *argDensityOrdinates,
        *argExpectationAbscissae, *argExpectationOrdinates,
        *argStandardDeviationOrdinates;
     int *n;
{
  int k;

  calcWeights();
  computeLawOfFOfT(n, argDensityAbscissae, argDensityOrdinates);
  computeMeanOfFBarOfT(&numberOfMonths,
        argExpectationAbscissae, argExpectationOrdinates,
        argStandardDeviationOrdinates);
  k = (int) currentHyperValues[3];
  statistics[0] = argExpectationOrdinates[k];
  statistics[1] = argStandardDeviationOrdinates[k];
}
```

Defines:
`consolidateComputation`, used in chunk 39a.

Uses `calcWeights` 56, `computeLawOfFOfT` 58a, `computeMeanOfFBarOfT` 59b, `currentHyperValues` 50, `numberOfMonths` 50, and `statistics` 7 50.

7 Installation Information

We present some information on how to compile and use the software on various platforms. This is presented as a README file.

61 ⟨*README file* 61⟩≡

```
Copyright (C) Hani J. Doss and B. Narasimhan
--------------------------------------------

This file provides information on installing and using
BSA (Bayesian Sensitivity Analysis) software. We expect any
serious user of the software to read our paper that has now
appeared in

@InBook{doss:nara:1998b,
  author =       {Hani J. Doss and B. Narasimhan},
  title =        {Practical Nonparametric and Semiparametric
                  Bayesian Statistics},
  chapter =      {Dynamic Display of Changing Posterior in Bayesian
                  Survival Analysis},
  publisher =    {Springer},
  year =         {1998},
  editor =       {Dipak Dey, Peter M\"uller, Debajyoti Sinha},
  number =       {133},
  series =       {Lecture Notes in Statistics},
  pages =        {63--87}
}

We have included a copy of this paper (ddg.ps.gz) in the subdirectory ddg.


Requirements:

 A) XLisp-Stat 3.53-5 or higher. Freely available from
     ftp://ftp.stat.umn.edu/pub/xlispstat/current

 B) Windows or Unix. Windows includes 95 and NT.
                 (Mac version is in development.)

 C) On Unix, you will also need a C compiler.

Step 1.
------

You probably received the entire package as a compressed archive
named bsa.tgz. On Unix, the contents of the archive may be
extracted into a directory called bsa by executing the commands:

    gunzip -c bsa.tgz | tar -xvf -

If you are using GNU tar, this can be done in one shot via:
```

```
    tar -xvzf bsa.tgz

To extract the files on Windows, you need to use an extractor
like WinZip. See http://www.winzip.com for more details.

Step 2
------

On Windows, you skip this step. Macs are not supported yet because
the authors don't know how to create a dynamic shared library.

On Unix, you need to configure the software to your environment.
Change to the bsa directory and type

   ./configure
   make

This will compile the lisp files and create a shared library for your
platform.

Step 3
------

Start using the program. In Unix, the command

  xlispstat BreastCancerRadiationOnly

or

  xlispstat BreastCancerRadiationChemo

will fire up the readymade examples.

On Windows, you fire up Lisp-Stat and load the files

    BreastCancerRadiationOnly.lsp
               or
    BreastCancerRadiationChemo.lsp

to proceed.

If you are inclined to use commands instead of mouse-clicks, you can
send ``messages'' to the master object in the listener window as
shown in the following examples:

  (send BreastCancerRadiationOnly-master :current-hyperparameter-values
                                        '(1 4 150 12))

  (send BreastCancerRadiationOnly-master :print-all-statistics)

  (send BreastCancerRadiationOnly-master :slot-value
```

```
      'bsa::importance-weights)
```

Note how the master-objects names are related to the lisp file names and how internal slot-names have to be prefixed by the package name.

For dealing with a new problem, we provide a few points regarding the software. A number of inputs are required for running the program. These are discussed in detail in the literate program (bsa.ps) under the section titled ''Introduction.'' For convenience we repeat the details here. This excerpted part is indented two spaces for easy reference.

First note that the software only does sensitivity analysis. No general facility is provided for generating observations from Markov chains. Indeed, since the range of models for which MCMC methods are applicable are large and such methods most likely involve problem-specific issues, it is our opinion that building such a supertool, if it is at all possible, is a non-trivial task. However, the Fortran program used in generating the output for our example is included along with this software and can be used for models similar to ours. Of course, any appropriate method may be used to generate the samples as long as the output is available in a form usable by our software. The requirements on the data that can be used with our software are spelt out below.

Corresponding to each Markov chain output, there must be two files with the extensions ".in" (input file) and ".out" (output file). For example, "mc1.in" and "mc1.out".

The input file must have the following structure. The first four items in the file can be anything, string or number, either on a single line or any conceivable combination of lines. The next three items *must* be the shape of the Gamma distribution on theta, the scale of the Gamma distribution on theta---the parametrization for shape a and scale b is proportional to x^{a-1} exp(-bx)---and M(R). The next three values values following these quantities can be anything, but the one following it should be the number of data points, that is, the number of sets or intervals. In the Fortran program we use -99 is used to denote infinity. Nothing else is read from the input file.

The output file must have the following structure for each data point generated by the Markov chain. The value of theta must be followed by the number of distinct values of the data points, which must be followed by a frequency table of the actual data value and the corresponding frequency. The layout of the values on lines does not matter as long as at least a single white space delimits values. If this structure is violated, errors will result. A peek at the data files included with this software will help the reader.

It is assumed that a proper installation of XLisp-Stat is available.

```
For a new problem, you probably have several Markov chain output files
although even one should work. (In the latter case, reweighting reduces
to simple Importance Sampling.)

   a) It is best to create a new directory for your problem and have
      your data files there. For example, the directory "BreastCancer"
      contains relevant data files for our Breast Cancer data.

   b) The only files you actually need to run the program are:
        1) Either one of bsa.fsl or bsa.lsp
        2) Either one of utility.fsl or utility.lsp
        3) Either one of call-by-reference.fsl or call-by-reference.lsp
              and
        4) the shared library libbsa.so or libbsa.sl as the case
           may be. On Windows, instead of the shared library,
           we need the whole subdirectory "win".

           Copy these files/directories to where you have the data
           files and work there.
        5) The file new-problem.lsp.

Invoke Lisp-Stat and load the file named "new-problem.lsp".

The first time (and first time only), the following inputs will be
needed.

Inputs
------
1) An indentifier for uniquely identifying the run. Use a meaningful
   name here. Let us assume this is BreastCancer (the default)
   in the discussion below.

2) The number of Markov chain outputs that you want to use for
   reweighting. Must be >= 1, with 1 denoting straight Importance
   Sampling.

3) The names of the files containing output from Markov chains,
   *without the extensions*. The software will automatically
   tag on the extensions .in and .out when looking for files.

4) The number of points per chain to use in the dynamic reweighting.
   Thus if you specify 50 and have 8 chains, then 50 points from each
   of the eight chains (= 400) will be used.

5) An initial guess for maximizing the log-quasilikelihood which
   will provide an estimate of the constants of proportionality.

6) The range between which you want to vary the hyperparameters. If
   you use only one chain, then you *must* specify the range. Otherwise,
   the range will be a single point. If you specified many chains,
   the default settings for each hyperparameter will be the minimum and
```

```
   maximum values from values used in all Markov chains. The number of
   stops should be an odd number if you want to hit the middle of the
   interval.

7) The number of points to use in estimating the constants of
   proportionality. If you use all of the data, the estimation can
   take a while. It is almost always better to go with the default
   or less. (If you are really interested in using more points,
   then start off with 10, and use the estimates thus obtained
   to start your larger optimization. This will save you a lot of
   time.)

   Once you specify this, the maximization will take place. This is a
   good point to go refill your coffee cup.

   After the estimation, two files are created so that you are not
   bombarded with questions in subsequent explorations. For example,

       BreastCancer.lsp   (and)
       BreastCancer.run

   For repeating the exploration next time, you only need to load
   the file BreastCancer.lsp into XLisp-Stat. This will bypass
   all the inputs we discussed above except for the question about
   ranges. The file BreastCancer.run contains pre-processed
   information for faster loading and will be used when
   BreastCancer.lsp is loaded.

   All files are text files and can be viewed with a text viewer.

Examples for Breast Cancer Data
-------------------------------

The two files

     BreastCancerRadiationOnly.lsp (and)
     BreastCancerRadiationChemo.lsp

and the corresponding run files are provided for experimentation.
These exist in the main directory "bsa" itself and concern the
dataset on two treatments described in the paper

  Dynamic Display of Changing Posterior in
                  Bayesian Survival Analysis
               by
  Hani J. Doss and B. Narasimhan

By default, they use 50 points each and 8 Markov chains. We wish to
note that an earlier version of the software was used to produce the
results in the paper and subsequently a bug was found. This does not
change any of the conclusions of the paper but the numbers shown in
```

```
table 1.1 in the paper are off from the actual values obtained using
the software. A replacement is provided in the file newtable.tex
and shows that the agreement between estimates obtained by reweighting
and those obtained by actual runs of Markov chains is, if anything,
better that what the original table indicated.

Just load the lisp files into XLisp-Stat to do the dynamic exploration.
If you have a sufficiently fast machine, you can use more points.

To completely reproduce our work from scratch, you need to use the
data files in the subdirectory "BreastCancer".

The authors may be contacted via email at:

  doss@stat.ohio-state.edu (Hani J. Doss)
  naras@stat.Stanford.EDU  (B. Narasimhan)

Note on the program itself
--------------------------

The programs are written in a literate style using the Noweb literate
programming tools. We provide two utility packages that one can use
independently of the program: utility.lsp and call-by reference.lsp.
The former contains functions we have found useful in writing Lispstat
programs; the latter implements a call-by-reference glue between
Lispstat and C.

Enjoy!
```
Uses `:current-hyperparameter-values` 15a, `current-hyperparameter-values` 7, `importance-weights` 7, `:print-all-statistics` 40a, and `statistics` 7 50.

8 Improvements needed

- Make sure that the when a parameter hasn't changed then the hyperparameter range should not bomb. Right now, we've hardwired it. See the method `:hyperparameter-ranges` or follow it.
- Add ability for user to specify a sequence of values in the hyperparameter space along which snapshots may be saved. Thus reruns can be shown like animated weather maps on TV. Actually this can be done now as follows. Assume that `hyperparameters-list` is a list of hyperparameter points (that is, four-tuples), and master-object is the master object for the problem.

```
(dolist (hyperparameters hyperparameters-list)
   (send master-object :current-hyperparameter-values hyperparameters))
```

- Clean up all the code that assumes that the time is an index varying from 0 to `number-of-months`. This has to be done carefully since it plays a role in many places.

9 Acknowledgement

This research was supported by Air Force Office of Scientific Research Grant F49620-94-1-0028.

We are indebted to Fred W. Huffer for the Fortran program `dirichlet` that comes with this software. All we did was modify it slightly to suit our data.

10 Index of Code Chunks

This list is generated automatically. The numeral is that of the first definition of the chunk.

11 Index of Identifiers

Here is a list of the identifiers used, and where they appear. Underlined entries indicate the place of definition. This index is generated automatically.]

References

[1] Hani J. Doss and B. Narasimhan. Bayesian Poisson regression: Sensitivity analysis through dynamic graphics. Technical report, Penn State Erie, The Behrend College, 1994.

[2] Hani J. Doss and B. Narasimhan. Dynamic display of changing posterior in Bayesian survival analysis. In *Practical Nonparametric and Semiparametric Bayesian Statistics*, D. Dey, P. Müller, and D. Sinha, eds, 1998. Springer-Verlag, N. Y.

[3] C. J. Geyer. Estimating normalizing constants and reweighting mixtures in Markov chain Monte Carlo. Technical report, School of Statistics, University of Minnesota, 1994.

[4] Donald E. Knuth. *Literate Programming*. Center for Study of Language and Information, Stanford University, 1992.

[5] A. Kong, J. S. Liu, and W. H. Wong. Sequential imputations and Bayesian missing data problems. *J. Amer. Statist. Assoc.*, 1994.

[6] Christopher Lee. Literate programming—propaganda and tools. World Wide Web, 1994. Available from `http://www.ius.cs.cmu.edu/help/Programming/literate.html`.

[7] Norman Ramsey. Literate programming simplified. *IEEE Software*, 1994.

[8] Norman Ramsey. *The Noweb Hacker's Guide*, 1994. Manual for Noweb.

[9] Norman Ramsey. The noweb home page. World Wide Web, 1995. On the Web at `http://www.cs.virginia.edu/~nr/noweb/`.

[10] Luke Tierney. *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics*. John Wiley & Sons, 1990.