13 Approximate Bayesian Computations

This section describes a set of tools for computing approximate posterior moments and marginal densities in XLISP-STAT. The definitions needed are in the file `bayes.lsp` on the distribution disk. This file is not loaded automatically at start up; you should load it now, using the **Load** item on the **File** menu or the `load` command, to carry out the calculations in this section. The material in this section is somewhat more advanced as it assumes you are familiar with the basic concepts of Bayesian inference.

The functions described in this section can be used to compute first and second order approximations to posterior moments and saddlepoint-like approximations to one dimensional marginal posterior densities. The approximations, based primarily on the results in [18,19,20], assume the posterior density is smooth and dominated by a single mode. The implementation is experimental and may change in a number of ways in the near future.

Let's start with a simple example, a data set used to study the relation between survival time (in weeks) of leukemia patients and white blood cell count recorded for the patients at their entry into the study [9, Example U]. The data consists of two groups of patients classified as AG positive and AG negative. The data for the 17 AG positive patients, contained in the file `leukemia.lsp` in the `Data` folder on the distribution disk, can be entered as

```
(def wbc-pos (list 2300 750 4300 2600 6000 10500 10000 17000 5400 7000
                   9400 32000 35000 100000 100000 52000 100000))
(def times-pos (list 65 156 100 134 16 108 121 4 39 143 56 26 22 1 1 5 65))
```

A high white blood cell count indicates a more serious stage of the disease and thus a lower chance of survival.

A model often used for this data assumes the survival times are exponentially distributed with a mean that is log linear in the logarithm of the white blood cell count. For convenience I will scale the white blood cell counts by 10000. That is, the mean survival time for a patient with white blood cell count $\log(WBC_i)$ is

$$\mu_i = \theta_0 \exp\{-\theta_1 x_i\},$$

with $x_i = \log(WBC_i/10000)$. The log likelihood function is thus given by

$$\sum_{i=1}^{n} \theta_1 x_i - n\log(\theta_0) - \frac{1}{\theta_0}\sum_{i=1}^{n} y_i e^{\theta_1 x_i},$$

with y_i representing the survival times. After computing the transformed WBC variable as

```
(def transformed-wbc-pos (- (log wbc-pos) (log 10000)))
```

the log likelihood can be computed using the function

```
(defun llik-pos (theta)
  (let* ((x transformed-wbc-pos)
         (y times-pos)
         (theta0 (select theta 0))
         (theta1 (select theta 1))
         (t1x (* theta1 x)))
    (- (sum t1x)
       (* (length x) (log theta0))
       (/ (sum (* y (exp t1x)))
          theta0))))
```

I will look at this problem using a vague, improper prior distribution that is constant over the range $\theta_i > 0$; thus the log posterior density is equal to the log likelihood constructed above, up to an additive constant. The first step is to construct a Bayes model object using the function `bayes-model`. This function takes a function for computing the log posterior density and an initial guess for the posterior mode, computes the posterior mode by an iterative method starting with the initial guess, prints a short summary of the information in the posterior distribution, and returns a model object. We can use the function `llik-pos` to compute the log posterior density, so all we need is an initial estimate for the posterior mode. Since the model we are using assumes a linear relationship between the logarithm of the mean survival time and the transformed WBC variable a linear regression of the logarithms of the survival times on `transformed-wbc-pos` should provide reasonable estimates. The linear regression gives

```
> (regression-model transformed-wbc-pos (log times-pos))

Least Squares Estimates:

Constant                  3.54234   (0.302699)
Variable 0              -0.817801   (0.214047)

R Squared:                 0.4932
Sigma hat:                 1.23274
Number of cases:                17
Degrees of freedom:             15
```

so reasonable initial estimates of the mode are $\hat{\theta}_0 = \exp(3.5)$ and $\hat{\theta}_1 = 0.8$. Now we can use these estimates in the `bayes-model` function:

```
> (def lk (bayes-model #'llik-pos (list (exp 3.5) .8)))
maximizing...
Iteration 0.
Criterion value = -90.8662
Iteration 1.
Criterion value = -85.4065
Iteration 2.
Criterion value = -84.0944
Iteration 3.
Criterion value = -83.8882
Iteration 4.
Criterion value = -83.8774
Iteration 5.
Criterion value = -83.8774
Iteration 6.
Criterion value = -83.8774
Reason for termination: gradient size is less than gradient tolerance.


First Order Approximations to Posterior Moments:

Parameter 0                56.8489 (13.9713)
Parameter 1               0.481829 (0.179694)

#<Object: 1565592, prototype = BAYES-MODEL-PROTO>
>
```

It is possible to suppress the summary information by supplying `NIL` as the value of the `:print` keyword argument.

The summary printed by `bayes-model` gives first order approximations to the posterior means and standard deviations of the parameters. That is, the means are approximated by the elements of the posterior mode and the standard deviations by the square roots of the diagonal elements of the inverse of the negative Hessian matrix of the log posterior at the mode. These approximations can also be obtained from the model by sending it the `:1stmoments` message:

```
> (send lk :1stmoments)
((56.8489 0.481829) (13.9713 0.179694))
```

The result is a list of two lists, the means and the standard deviations. A slower but more accurate second order approximation is available as well. It can be obtained using the message `:moments`:

```
> (send lk :moments)
((65.3085 0.485295) (17.158 0.186587))
```

Both these messages allow you to compute moments for individual parameters or groups of parameters by specifying an individual parameter index or a list of indices:

```
> (send lk :moments 0)
((65.3085) (17.158))
> (send lk :moments (list 0 1))
((65.3085 0.485295) (17.158 0.186587))
```

The first and second order approximations to the moments of θ_0 are somewhat different; in particular the mean appears to be somewhat larger than the mode. This suggests that the posterior distribution of this parameter is skewed to the right. We can confirm this by looking at a plot of the approximate marginal posterior density. The message `:margin1` takes a parameter index, and a sequence of points at which to evaluate the density and returns as its value a list of the supplied sequence and the approximate density values at these points. This list can be given to `plot-lines` to produce a plot of the marginal density:

```
> (plot-lines (send lk :margin1 0 (rseq 30 120 30)))
#<Object: 1623804, prototype = SCATTERPLOT-PROTO>
```

The result is shown in Figure 20 and does indeed show some skewness to the right.

In addition to examining individual parameters it is also possible to look at the posterior distribution of smooth functions of the parameters.[29] For example, you might want to ask what information the data contains about the probability of a patient with $WBC = 50000$ surviving a year or more. This probability is given by

$$\frac{1}{\mu(x)} e^{-52/\mu(x)},$$

with

$$\mu(x) = \theta_0 e^{-\theta_1 x}$$

and $x = \log(5)$, and can be computed by the function

```
(defun lk-sprob (theta)
  (let* ((time 52.0)
         (x (log 5))
         (mu (* (select theta 0) (exp (- (* (select theta 1) x))))))
    (exp (- (/ time mu)))))
```

[29] The approximation methods assume these functions are twice continuously differentiable; thus they can not be indicator functions.

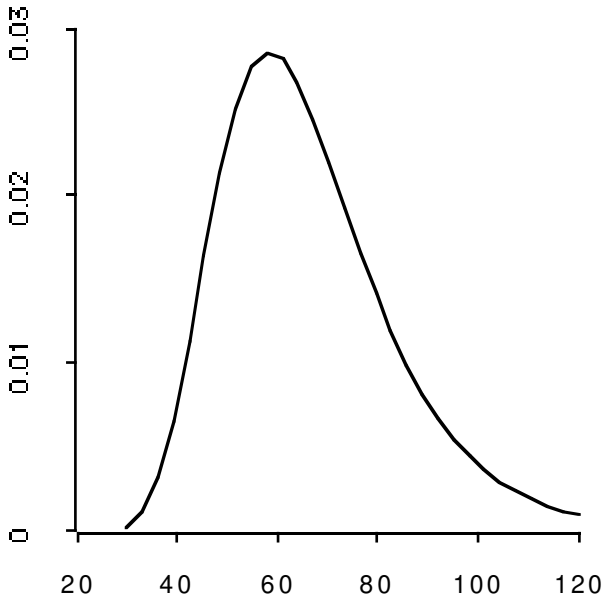

Figure 20: Plot of marginal posterior density for θ_0.

This function can be given to the `:1stmoments`, `:moments` and `:margin1` methods to approximate the posterior moments and marginal posterior density of this function. For the moments the results are

```
> (send lk :1stmoments #'lk-sprob)
((0.137189) (0.0948248))
> (send lk :moments #'lk-sprob)
((0.184275) (0.111182))
```

with the difference again suggesting some positive skewness, and the marginal density produced by the expression

```
(plot-lines (send lk :margin1 #'lk-sprob (rseq .01 .8 30)))
```

is shown in Figure 21. Based on this picture the data suggests that this survival probability is almost certainly below 0.5, but it is difficult to make a more precise statement than that.

The functions described in this section are based on the optimization code described in the previous section. By default derivatives are computed numerically. If you can compute derivatives yourself you can have your log posterior function return a list of the function value and the gradient or a list of the function value, the gradient and the Hessian matrix.

Exercises

1. To be able to think about prior distributions for the two parameters in this problem we need to try to understand what the parameters represent. The parameter θ_0 is fairly easy to understand: it is the mean survival time for patients with $WBC = 10000$. The parameter θ_0 is a little more difficult to think about. In represents the approximate percent difference in mean survival time for patients with WBC differing by one percent. Because of the minus sign

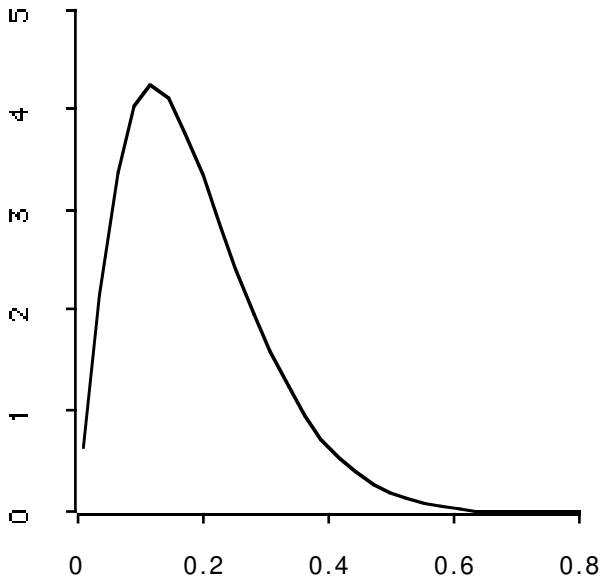

Figure 21: Plot of marginal posterior density of the one year survival probability of a patient with $WBC = 50000$.

in the mean relationship, and the expected inverse relation between WBC and mean survival time, θ_1 is expected to be positive.

Consider an informative prior distribution that assumes the two parameters *a priori* independent, takes $\log(\theta_0)$ to be normally distributed with mean $\log(52)$ and standard deviation $\log(2)$, and θ_1 to be exponentially distributed with mean $\mu = 5$. This prior is designed to represent an opinion that mean survival time at $WBC = 10000$ should be around one year, but that guess could easily be off by a factor of two either way. The percentage change in the mean for a one percent change in WBC should be on the order of one to ten or so. Examine the posterior distribution for this prior and compare your results to the results for the vague prior used above.

2. Construct and examine a posterior distribution for the parameters of the gamma model based on the aircraft data of Section 12.

References

[1] Bates, D. M. and Watts, D. G., (1988), *Nonlinear Regression Analysis and its Applications*, New York: Wiley.

[2] Becker, Richard A., and Chambers, John M., (1984), *S: An Interactive Environment for Data Analysis and Graphics*, Belmont, Ca: Wadsworth.

[3] Becker, Richard A., Chambers, John M., and Wilks, Allan R., (1988), *The New S Language: A Programming Environment for Data Analysis and Graphics*, Pacific Grove, Ca: Wadsworth.

[4] Becker, Richard A., and William S. Cleveland, (1987), "Brushing scatterplots," *Technometrics*, vol. 29, pp. 127-142.

[5] Betz, David, (1985) "An XLISP Tutorial," *BYTE*, pp 221.

[6] Betz, David, (1988), "XLISP: An experimental object-oriented programming language," Reference manual for XLISP Version 2.0.

[7] Chaloner, Kathryn, and Brant, Rollin, (1988) "A Bayesian approach to outlier detection and residual analysis," *Biometrika*, vol. 75, pp. 651-660.

[8] Cleveland, W. S. and McGill, M. E., (1988) *Dynamic Graphics for Statistics*, Belmont, Ca.: Wadsworth.

[9] Cox, D. R. and Snell, E. J., (1981) *Applied Statistics: Principles and Examples*, London: Chapman and Hall.

[10] Dennis, J. E. and Schnabel, R. B., (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, N.J.: Prentice-Hall.

[11] Devore, J. and Peck, R., (1986), *Statistics, the Exploration and Analysis of Data*, St. Paul, Mn: West Publishing Co.

[12] McDonald, J. A., (1982), "Interactive Graphics for Data Analysis," unpublished Ph. D. thesis, Department of Statistics, Stanford University.

[13] Oehlert, Gary W., (1987), "MacAnova User's Guide," Technical Report 493, School of Statistics, University of Minnesota.

[14] Press, Flannery, Teukolsky and Vetterling, (1988), *Numerical Recipes in C*, Cambridge: Cambridge University Press.

[15] Steele, Guy L., (1984), *Common Lisp: The Language*, Bedford, MA: Digital Press.

[16] Stuetzle, W., (1987), "Plot windows," *J. Amer. Statist. Assoc.*, vol. 82, pp. 466 - 475.

[17] Tierney, Luke, (1990) *LISP-STAT: Statistical Computing and Dynamic Graphics in Lisp.* Forthcoming.

[18] Tierney, L. and J. B. Kadane, (1986), "Accurate approximations for posterior moments and marginal densities," *J. Amer. Statist. Assoc.*, vol. 81, pp. 82-86.

[19] Tierney, Luke, Robert E. Kass, and Joseph B. Kadane, (1989), "Fully exponential Laplace approximations to expectations and variances of nonpositive functions," *J. Amer. Statist. Assoc.*, to appear.

[20] Tierney, L., Kass, R. E., and Kadane, J. B., (1989), "Approximate marginal densities for nonlinear functions," *Biometrika*, to appear.

[21] Weisberg, Sanford, (1982), "MULTREG Users Manual," Technical Report 298, School of Statistics, University of Minnesota.

[22] Winston, Patrick H. and Berthold K. P. Horn, (1988), *LISP*, 3rd Ed., New York: Addison-Wesley.

A XLISP-STAT on UNIX Systems

This tutorial has dealt primarily with the Macintosh version of XLISP-STAT. XLISP-STAT is also available on UNIX systems. If it has been installed in a directory in your search path you should be able to start it up by typing

```
xlispstat
```

at the UNIX shell level. There are a few differences between the Macintosh and UNIX versions. On UNIX systems:

- UNIX versions of XLISP-STAT are designed to run on a standard terminal and therefore do not provide parenthesis matching or indentation support. If you use the *GNU* `emacs` editior you can obtain both these features by running XLISP-STAT from within `emacs`. Otherwise, for editing files with `vi` you can use the `-l` flag to get some Lisp editing support.

- To quit from the program type

  ```
  (exit)
  ```

  On most systems you can also quit by typing a *Control-D*.

- You can interrupt a calculation that is taking too long or was started in error by typing a *Control-C*.

- Data and example files are stored in the `Data` and `Examples` subdirectories of the library tree. The functions `load-data` and `load-examples` will look in these directories, so

  ```
  (load-data "tutorial")
  ```

  will load the data sets for the tutorial. Within XLISP-STAT the variable `*default-path*` shows the root directory of the library; you can look there if you want to examine the example files.

- The `require` function can be used to load program modules not loaded at startup. To load the nonlinear regression module, for example, use the expression

  ```
  (require "nonlin")
  ```

On basic UNIX systems the only graphics available are the functions `plot-points` and `plot-lines`. These functions assume you are using a *Tektronix* terminal or emulator.

A.1 XLISP-STAT Under the *X11* Window System

Window based graphics are available in XLISP-STAT on a workstation running the *X11* window system. Graphics under *X11* are fairly similar to Macintosh graphics as documented in this tutorial. A few points to note:

- Plot menus are popped up using a menu button in the top right corner of a plot.

- In plot interaction you can use any of the mouse buttons. Normally clicking any button in a plot unselects all selected points. To extend a selection, or have a rotating plot continue rotating after the button is released, hold down the shift key as you press any mouse button.

- How plot windows are opened in response to a graphics command depends on the window manager you are using. Under `uwm`, the default window manager on many systems, a little corner will appear which you can use to choose the position of your plot window.
- Slider dialog items are the only items that assume a three button mouse. In the central part of the slider the right button increases and the left button decreases the slider value. The middle button drags the thumb indicator.
- Postscript images of plots can be saved by selecting the **Save to File** item on a plot menu. The postscript file can then be printed using a standard printing command.

A.1.1 More Advanced *X11* Features

You can have XLISP-STAT use an alternate display for its graphics by setting the `DISPLAY` environment variable before starting XLISP-STAT. At present this is the only way to set an alternate display. You can specify alternate fonts and a few other options using the *X11* resource management facilities. Resources controlling appearance are

`xlisp*menu*titles:`	*on* for a title on a menu, *off* otherwise
`xlisp*menu*font:`	
`xlisp*dialog*font:`	
`xlisp*graph*font:`	

There are also a few experimental options controlling performance. These are

`xlisp*graph*fastlines:`	*on* to use 0 width lines
`xlisp*graph*fastsymbols:`	*on* to use `DrawPoints` instead of bitmaps
`xlisp*graph*motionsync:`	*on* to use `XSync` during mouse motion

By default all three options are are *on*. That seems to give the best performance on a *Sun 3/50*. It may not be the best choice on other workstations. You can also use the function `x11-options` to change these three options from within XLISP-STAT. The `fastlines` option will not take effect immediately when changed this way but will affect the next plot created. The other two options do take effect immediately.

A.2 XLISP-STAT Under the *SunView* Window System

Window based graphics are also available when XLISP-STAT is run on a *Sun* console running `suntools`. Graphics under `suntools` work like graphics on the Macintosh with the following changes:

- To close or resize plots or dialogs use the frame menu or the standard `suntools` shortcuts (e. g. to resize a plot window drag the frame with the middle mouse button while holding down the *control* key).
- Plot menus are popped up by pressing the right mouse button in the interior of the plot. Check marks do not appear on menu items so it may not always be clear what state an item is in.
- Clicking and dragging on the Macintosh corresponds to clicking and dragging with the left mouse button. Shift-clicking or shift-dragging on the Macintosh (to extend a selection or cause a rotating plot to continue spinning when the button is released) corresponds to using the middle mouse button.
- Postscript images of plots can be saved by selecting the **Save to File** item on a plot menu.

A.3 Running UNIX Commands from XLISP-STAT

The `system` function can be used to run UNIX commands from within XLISP-STAT. This function takes a shell command string as its argument and returns the shell exit code for the command. For example, you can print the date using the UNIX `date` command:

```
> (system "date")
Wed Jul 19 11:06:53 CDT 1989
0
>
```

The return value is 0, indicating successful completion of the UNIX command.

A.4 Dynamic Loading and Foreign Function Calling

Some UNIX implementations of XLISP-STAT also provide a facility to allow you to use your own C functions of FORTRAN subroutines from within XLISP-STAT. The facility, patterned after the approach used in the New *S* language [3], consists of the function `dyn-load` for loading your code into a running XLISP-STAT process and the functions `call-cfun`, `call-fsub` and `call-lfun` for calling subroutines in the loaded code. The `dyn-load` function requires one argument, a string containing the name of a file to be linked and loaded. The file will be linked with standard C libraries before loading. If you need it to be linked with the standard FORTRAN libraries as well you can give the keyword argument `:fortran` the value `T`. Finally, if you need to link other libraries you can supply a string containing the library flags you would specify in a linking command as the value of the keyword argument `:libflags`. For example, to include the library `cmlib` use the string `"-lcmlib"`.[30]

The function `call-cfun` takes a string identifying the C function you want to call, followed by additional arguments for your C function. The additional arguments must be integers, sequences of integers, real numbers or sequences of real numbers. Pointers to `int` or `double` data containing these values will be passed to your routine. After your routine returns the contents of the data referred to by these pointers are copied into lists and `call-cfun` returns a list of these lists.

As an example, suppose the file `foo.c` contains the following C function:

```
foo(n, x, sum)
     int *n;
     double *x, *sum;
{
  int i;

  for (i = 0, *sum = 0.0; i < *n; i++) {
    *sum += x[i];
  }
}
```

After compiling the file to `foo.o` we can load it into XLISP-STAT using the expression

```
(dyn-load "foo.o")
```

We can then call the function `foo` using a list of real numbers as the second argument. The function `float` can be used to coerce numbers to reals:

```
> (call-cfun "foo" 5 (float (iseq 1 5)) 0.0)
((5) (1 2 3 4 5) (15))
```

[30] There may be slight differences in the implementation of `dyn-load` on different systems. The help information for this function should give information that is appropriate for your system.

The third argument to `foo` has been used to return the result.

The function `call-fsub` is used for calling FORTRAN subroutines that have been loaded dynamically. A FORTRAN subroutine analogous to the C function `foo` might be written as

```
      subroutine foo(n, x, sum)
      integer n
      double precision x(n), sum

      integer i

      sum = 0.0
      do 10 i = 1, n
         sum = sum + x(i)
 10   continue
      return
      end
```

After compiling and loading this routine it can be called using `call-fsub`:

```
> (call-fsub "foo" 5 (float (iseq 1 5)) 0.0)
((5) (1 2 3 4 5) (15))
```

Two C functions you may want to call from within your C functions are `xscall_alloc` and `xscall_fail`. The function `xscall_alloc` is like `calloc`, except it insures the allocated memory is garbage collected after the call to `call-cfun` returns. The function `xscall_fail` takes a character string as its argument. It prints the string and signals an error.

The function `call-lfun` can be used to call C functions written using the internal XLISP conventions for obtaining arguments and returning results. This allows you to accept any kinds of arguments. Unfortunately, the source code is the only documentation for the internal calling conventions.

A note of caution may be appropriate at this point. Dynamically loaded code contains only the error checking you build into it. If a function is not called with the proper arguments it will most likely cause XLISP-STAT to crash, losing any variables you have not saved.

At present the number of arguments you can pass to C functions or FORTRAN subroutines using `call-cfun` or `call-fsubr` is limited to 15.

If dynloading is not available on your system you can still recompile XLISP-STAT with files of your own added to the source code. The functions `call-cfun`, `call-fsubr` and `call-lfun` can be used to call your functions or subroutines in this case as well.

B Graphical Interface Tools

One of the characteristic features of the Macintosh user interface is the use of menus and dialogs for interacting with the computer. XLISP-STAT allows you to construct your own menus and dialogs using Lisp commands. This appendix gives a very brief introduction to constructing menus and dialogs; further details will be given in [17]. A few of the explanations and examples in this appendix use Lisp concepts that have not been covered in this tutorial.

B.1 Menus

As an illustration I will outline how to construct a menu for sending some simple messages to a regression model. I will make the convention that there is a *current regression model*, the value of the symbol `*current-model*`.

Menus are created by sending the `:new` message to the menu prototype, `menu-proto`. The method for this message takes a single argument, the menu title. We can use this message to set up our menu:

```
> (setf model-menu (send menu-proto :new "Model"))
#<Object: 4055334, prototype = MENU-PROTO>
```

Macintosh menus can be installed in and removed from the menu bar by sending them the `:install` and `:remove` messages:

```
> (send model-menu :install)
NIL
> (send model-menu :remove)
NIL
```

On other systems menus are *popped up*; this can be accomplished by sending the `:popup` message to a menu. This message requires two arguments, the x and y pixel coordinates of the left top corner of the menu, measured from the left top corner of the screen.

Initially the menu has no items in it. Items are created using the `menu-item-proto` prototype. The initialization method requires one argument, the item's title, and takes several keyword arguments, including

- `:action` – a function to be called when the item is selected
- `:enabled` – true by default
- `:key` – a character to serve as the keyboard equivalent
- `:mark` – `nil` (the default) or `t` to indicate a check mark.

Analogous messages are available for changing these values in existing menu items.

Suppose we would like to be able to use our menu to print a summary of the current model or obtain a residual plot. We can construct two menu items:

```
> (setf summary (send menu-item-proto :new "Summary" :action
                    #'(lambda () (send *current-model* :display))))
#<Object: 4034406, prototype = MENU-ITEM-PROTO>
> (setf plot (send menu-item-proto :new "Plot Residuals" :action
                    #'(lambda () (send *current-model* :plot-residuals))))
#<Object: 3868686, prototype = MENU-ITEM-PROTO>
```

Suppose we have assigned the `bikes2` model of Section 7 to `*current-model*`. You can force an item's action to be invoked by sending it the `:do-action` message from the listener:

```
> (send summary :do-action)

Least Squares Estimates:

Constant               -16.41924   (7.848271)
Variable 0              2.432667   (0.9719628)
Variable 1            -0.05339121   (0.02922567)

R Squared:             0.9477923
Sigma hat:             0.5120859
Number of cases:              10
Degrees of freedom:            7

NIL
```

Ordinarily you will not send this message this way: the system sends this message to the menu item when you select the item from a menu.

To add these items to the menu use the `:append-items` message:

```
> (send model-menu :append-items summary plot)
NIL
```

You can also use the `:append-items` message to add items to a plot menu. The menu associated with a plot can be obtained by sending the plot the `:menu` message with no arguments.

You can enable and disable a menu item with the `:enabled` message:

```
> (send summary :enabled)
T
> (send summary :enabled nil)
NIL
> (send summary :enabled t)
T
```

B.2 Dialogs

Dialogs are similar to menus in that they are based on a dialog prototype and dialog item prototypes. There are, however many more variations. Fortunately most dialogs you need fall into one of several categories and can be produced by custom dialog construction functions.

B.2.1 Modal Dialogs

Modal dialogs are designed to ask specific questions and wait until they receive a response. All other interaction is disabled until the dialog is dismissed – they place the system in *dialog mode*. Six functions are available for producing some standard modal dialogs:

- `(message-dialog` *<string>*`)` – presents a message with an **OK** button; returns `nil` when the button is pressed.
- `(ok-or-cancel-dialog` *<string>*`)` – presents a message with an **OK** and a **Cancel** button; returns `t` or `NIL` according to the button pressed.
- `(choose-item-dialog` *<string>* *<string-list>*`)` – presents a heading and a set of radio buttons for choosing one of the strings. Returns the index of the selected string on **OK** or `nil` on **Cancel**. Example:

```
> (choose-item-dialog "Dependent variable:" '("X" "Y" "Z"))
1
```

- `(choose-subset-dialog` <*string*> <*string-list*>`)` – presents a heading and a set of check boxes for indicating which items to select. Returns a list of the list of selected indices on **OK** or `nil` on **Cancel**. Example:

```
> (choose-subset-dialog "Independent variables:" '("X" "Y" "Z"))
((0 2))
```

- `(get-string-dialog` <*prompt*> `[:initial` <*expr*>`])` – presents a dialog with a prompt, an editable text field, an **OK** and a **Cancel** button. The initial contents of the editable field is empty or the `princ` formated version of <*expr*>. The result is a string or `nil`. Example:

```
> (get-string-dialog "New variable label:" :initial "X")
"Tensile Strength"
```

- `(get-value-dialog` <*prompt*> `[:initial` <*expr*>`])` – like `get-string-dialog`, except
  - the initial value expression is converted to a string with `print` formatting
  - the result is interpreted as a lisp expression and is evaluated
  - the result is a list of the value, or `nil`

On the Macintosh there are two additional dialogs for dealing with files:

- `(open-file-dialog)` – presents a standard **Open File** dialog and returns a file name string or `nil`. Resets the working folder on **OK**.
- `(set-file-dialog prompt)` – presents a standard **Save File** dialog. Returns a file name string or `nil`. Resets the working folder on **OK**.

B.2.2 Modeless Dialogs

Two functions for constructing custom modeless dialogs are available also. They are the functions `interval-slider-dialog` and `sequence-slider-dialog` introduced above in Section 8.

C Selected Listing of XLISP-STAT Functions

C.1 Arithmetic and Logical Functions

(+ &rest numbers) [Function]
Returns the sum of its arguments. With no args, returns 0. Vectorized.

(- number &rest more-numbers) [Function]
Subtracts the second and all subsequent NUMBERs from the first. With one arg, negates it. Vectorized.

(* &rest numbers) [Function]
Returns the product of its arguments. With no args, returns 1. Vectorized.

(/ number &rest more-numbers) [Function]
Divides the first NUMBER (element-wise) by each of the subsequent NUMBERS. With one arg, returns its reciprocal. Vectorized.

(^ base-number power-number) [Function]
Returns BASE-NUMBER raised to the power POWER-NUMBER. Vectorized.

(** base-number power-number) [Function]
Returns BASE-NUMBER raised to the power POWER-NUMBER. Vectorized.

(< &rest numbers) [Function]
Returns T if NUMBERS are in strictly increasing order; NIL otherwise. Vectorized.

(<= &rest numbers) [Function]
Returns T if NUMBERS are in nondecreasing order; NIL otherwise. Vectorized.

(= &rest numbers) [Function]
Returns T if NUMBERS are all equal; NIL otherwise. Vectorized.

(/= &rest numbers) [Function]
Returns T if NUMBERS no two adjacent numbers are equal; NIL otherwise. Vectorized.

(>= &rest numbers) [Function]
Returns T if NUMBERS are in nonincreasing order; NIL otherwise. Vectorized.

(> &rest numbers) [Function]
Returns T if NUMBERS are in strictly decreasing order; NIL otherwise. Vectorized.

(abs number) [Function]
Returns the absolute value or modulus of NUMBER. Vectorized.

(acos number) [Function]
Returns the arc cosine of NUMBER. Vectorized.

(asin number) [Function]
Returns the arc sine of NUMBER. Vectorized.

(atan number) [Function]
Returns the arc tangent of NUMBER. Vectorized.

(ceiling number) [Function]
Returns the smallest integer(s) not less than or NUMBER. Vectorized.

(complex realpart &optional (imagpart 0)) [Function]
Returns a complex number with the given real and imaginary parts.

(conjugate number) [Function]
Returns the complex conjugate of NUMBER.

(cos radians) [Function]
Returns the cosine of RADIANS. Vectorized.

(exp x) [Function]
Calculates e raised to the power x, where e is the base of natural logarithms. Vectorized.

(expt base-number power-number) [Function]
Returns BASE-NUMBER raised to the power POWER-NUMBER. Vectorized.

(float number) [Function]
Converts real number to a floating-point number. If NUMBER is already a float, FLOAT simply returns NUMBER. Vectorized.

(floor number) [Function]
Returns the largest integer(not larger than the NUMBER. Vectorized.

(imagpart number) [Function]
Extracts the imaginary part of NUMBER.

(log number) [Function]
Returns the natural logarithm(s) of NUMBER. Vectorized.

(log-gamma x) [Function]
Returns the log gamma function of X. Vectorized.

(max number &rest more-numbers) [Function]
Returns the greatest of its arguments. Vector reducing

(min number &rest more-numbers) [Function]
Returns the least of its arguments. Vector reducing

(phase number) [Function]
Returns the angle part of the polar representation of a complex number. For non-complex numbers, this is 0.

(pmax &rest items) [Function]
Parallel maximum of ITEMS. Vectorized.

(pmin &rest items) [Function]
Parallel minimum of ITEMS. Vectorized.

(prod &rest number-data) [Function]
Returns the product of all the elements of its arguments. Returns 1 if there are no arguments. Vector reducing.

(random number) [Function]
Generates a uniformly distributed pseudo-random number between zero (inclusive) and NUMBER (exclusive). Vectorized.

(realpart number) [Function]
Extracts the real part of NUMBER.

(rem x y) [Function]
Returns the remainder of dividing x by y. Vectorized.

(round number) [Function]
Rounds NUMBER to nearest integer. Vectorized.

(sin radians) [Function]
Returns the sine of RADIANS. Vectorized.

(sqrt number) [Function]
Returns the square root of NUMBER. Vectorized.

(sum &rest number-data) [Function]
Returns the sum of all the elements of its arguments. Returns 0 if there are no arguments. Vector reducing.

(tan radians) [Function]
Returns the tangent of RADIANS. Vectorized.

(truncate number) [Function]
Returns real NUMBER as an integer, rounded toward 0. Vectorized.

C.2 Constructing and Modifying Compound Data and Variables

(def var form) [Macro]
VAR is not evaluated and must be a symbol. Assigns the value of FORM to VAR and adds VAR to the list *VARIABLES* of def'ed variables. Returns VAR. If VAR is already bound and the global variable *ASK-ON-REDEFINE* is not nil then you are asked if you want to redefine the variable.

(if-else first x y) [Function]
Takes simple or compound data items FIRST, X and Y and returns result of elementswise selecting from X if FIRST is not NIL and from Y otherwise.

(iseq n m) [Function]
Returns a list of consecutive integers from n to m. Examples:
(iseq 3 7) returns (3 4 5 6 7)
(iseq 3 -3) returns (3 2 1 0 -1 -2 -3)

(list &rest args) [Function]
Returns a list of its arguments

(repeat vals times) [Function]
Repeats VALS. If TIMES is a number and VALS is a non-null, non-array atom, a list of length TIMES with all elements eq to VALS is returned. If VALS is a list and TIMES is a number then VALS is appended TIMES times. If TIMES is a list of numbers then VALS must be a list of equal length and the simpler version of repeat is mapped down the two lists. Examples:
(repeat 2 5) returns (2 2 2 2 2)
(repeat '(1 2) 3) returns (1 2 1 2 1 2)
(repeat '(4 5 6) '(1 2 3)) returns (4 5 5 6 6 6)
(repeat '((4) (5 6)) '(2 3)) returns (4 4 5 6 5 6 5 6)

(rseq a b num) [Function]
Returns a list of NUM equally spaced points starting at A and ending at B.

(select a &rest indices) [Function]
A can be a list or an array. If A is a list and INDICES is a single number then the appropriate element of A is returned. If is a list and INDICES is a list of numbers then the sublist of the corresponding elements is returned. If A in an array then the number of INDICES must match the ARRAY-RANK of A. If each index is a number then the appropriate array element is returned. Otherwise the INDICES must all be lists of numbers and the corresponding submatrix of A is returned. SELECT can be used in setf.

(undef symbol) [Function]
If SYMBOL is a defined variable it is unbound and removed from the list of defined variables and returns SYMBOL.

(vector &rest items) [Function]
Returns a vector with ITEMS as elements.

(which x) [Function]
X is an array or a list. Returns a list of the indices where X is not NIL.

C.3 Basic Statistical Functions

(bayes-model logpost mode &key scale data derivstep (verbose t) (quick t) (print t)) [Function]
LOGPOST computes the logposterior density. It should return the function, or a list of the function value and gradient, or a list of the function value, gradient and Hessian. MODE is an initial guess for the mode. SCALE and DERIVSTEP are used for numerical derivatives and scaling. VERBOSE controls printing of iteration information during optimization, PRINT controls printing of summary information. If QUICK is T the summary is based on first order approximations.

(beta-cdf x alpha beta) [Function]
Returns the value of the Beta(ALPHA, BETA) distribution function at X. Vectorized.

(beta-dens x alpha beta) [Function]
Returns the density at X of the Beta(ALPHA, BETA) distribution. Vectorized.

(beta-quant p alpha beta) [Function]
Returns the P-th quantile of the Beta(ALPHA, BETA) distribution. Vectorized.

(beta-rand n a b) [Function]
Returns a list of N beta(A, B) random variables. Vectorized.

(binomial-cdf x n p) [Function]
Returns value of the Binomial(N, P) distribution function at X. Vectorized.

(binomial-pmf k n p) [Function]
Returns value of the Binomial(N, P) pmf function at integer K. Vectorized.

(binomial-quant x n p) [Function]
Returns x-th quantile (left continuous inverse) of Binomial(N, P) cdf. Vectorized.

(binomial-rand k n p) [Function]
Returns list of K draws from the Binomial(N, P) distribution. Vectorized.

(bivnorm-cdf x y r) [Function]
Returns the value of the standard bivariate normal distribution function with correlation R at (X, Y). Vectorized.

(cauchy-cdf x) [Function]
Returns the value of the standard Cauchy distribution function at X. Vectorized.

(cauchy-dens x) [Function]
Returns the density at X of the standard Cauchy distribution. Vectorized.

(cauchy-quant p) [Function]
Returns the P-th quantile(s) of the standard Cauchy distribution. Vectorized.

(cauchy-rand n) [Function]
Returns a list of N standard Cauchy random numbers. Vectorized.

(chisq-cdf x df) [Function]
Returns the value of the Chi-Square(DF) distribution function at X. Vectorized.

(chisq-dens x alpha) [Function]
Returns the density at X of the Chi-Square(DF) distribution. Vectorized.

(chisq-quant p df) [Function]
Returns the P-th quantile of the Chi-Square(DF) distribution. Vectorized.

(chisq-rand n df) [Function]
Returns a list of N Chi-Square(DF) random variables. Vectorized.

(covariance-matrix &rest args) [Function]
Returns the sample covariance matrix of the data columns in ARGS. ARGS may consist of lists, vectors or matrices.

(difference x) [Function]
Returns differences for a sequence X.

(f-cdf x ndf ddf) [Function]
Returns the value of the F(NDF, DDF) distribution function at X. Vectorized.

(f-dens x ndf ddf) [Function]
Returns the density at X of the F(NDF, DDF) distribution. Vectorized.

(f-quant p ndf ddf) [Function]
Returns the P-th quantile of the F(NDF, DDF) distribution. Vectorized.

(f-rand n d) [Function]
Returns a list of N F(NDF, DDF) random variables. Vectorized.

(fivnum number-data) [Function]
Returns the five number summary (min, 1st quartile, medinan, 3rd quartile, max) of the elements X.

(gamma-cdf x alpha) [Function]
Returns the value of the Gamma(alpha, 1) distribution function at X. Vectorized.

(gamma-dens x alpha) [Function]
Returns the density at X of the Gamma(ALPHA, 1) distribution. Vectorized.

(gamma-quant p alpha) [Function]
Returns the P-th quantile of the Gamma(ALPHA, 1) distribution. Vectorized.

(gamma-rand n a) [Function]
Returns a list of N Gamma(A, 1) random variables. Vectorized.

(interquartile-range number-data) [Function]
Returns the interquartile range of the elements of X.

(mean x) [Function]
Returns the mean of the elements x. Vector reducing.

(median x) [Function]
Returns the median of the elements of X.

(newtonmax f start &key scale derivstep (verbose 1) return-derivs) [Function]
Maximizes F starting from START using Newton's method with backtracking. If RETURN-DERIVS is NIL returns location of maximum; otherwise returns list of location, unction value, gradient and hessian at maximum. SCALE should be a list of the typical magnitudes of the parameters. DERIVSTEP is used in numerical derivatives and VERBOSE controls printing of iteration information. COUNT-LIMIT limits the number of iterations

(nelmeadmax f start &key (size 1) (epsilon (sqrt machine-epsilon)) (count-limit 500) (verbose t) alpha beta gamma delta) [Function]
Maximizes F using the Nelder-Mead simplex method. START can be a starting simplex - a list of N+1 points, with N=dimension of problem, or a single point. If start is a single point you should give the size of the initial simplex as SIZE, a sequence of length N. Default is all 1's. EPSILON is the convergence tolerance. ALPHA-DELTA can be used to control the behavior of simplex algorithm.

(normal-cdf x) [Function]
Returns the value of the standard normal distribution function at X. Vectorized.

(normal-dens x) [Function]
Returns the density at X of the standard normal distribution. Vectorized.

(normal-quant p) [Function]
Returns the P-th quantile of the standard normal distribution. Vectorized.

(normal-rand n) [Function]
Returns a list of N standard normal random numbers. Vectorized.

(nreg-model mean-function y theta &key (epsilon 0.0001) (count-limit 20) (print t) parameter-names response-name case-labels weights included (vetbose print)) [Function]
Fits nonlinear regression model with MEAN-FUNCTION and response Y using initial parameter guess THETA. Returns model object.

(numgrad f x &optional scale derivstep) [Function]
Computes the numerical gradient of F at X.

(numhess f x &optional scale derivstep) [Function]
Computes the numerical Hessian matrix of F at X.

(oneway-model data &key (print t)) [Function]
DATA: list of compound-data Example:

(order x) [Function]
Returns a sequence of the indices of elements in the sequence of numbers or strings X in order.

(pmin &rest items) [Function]
Parallel minimum of ITEMS. Vectorized.

(pmax &rest items) [Function]
Parallel maximum of ITEMS. Vectorized.

(poisson-cdf x mu) [Function]
Returns value of the Poisson(MU) distribution function at X. Vectorized.

(poisson-pmf k mu) [Function]
Returns value of the Poisson(MU) pmf function at integer K. Vectorized.

(poisson-quant x mu) [Function]
Returns x-th quantile (left continuous inverse) of Poisson(MU) cdf. Vectorized.

(poisson-rand k mu) [Function]
Returns list of K draws from the Poisson(MU) distribution. Vectorized.

(quantile x p) [Function]
Returns the P-th quantile(s) of sequence X. P can be a number or a sequence.

(rank x) [Function]
Returns a sequence with the elements of the list or array of numbers or strings X replaced by their ranks.

(read-data-columns file cols) [Function]
Reads the data in FILE as COLS columns and returns a list of lists representing the columns.

(read-data-file file) [Function]
Returns a list of all lisp objects in FILE. FILE can be a string or a symbol, in which case the symbol'f print name is used.

(regression-model x y &key (intercept t) (print t) weights included predictor-names response-name case-labels) [Function]
X - list of independent variables or X matrix
Y - dependent variable
INTERCEPT - T to include (default), NIL for no intercept
PRINT - if not NIL print summary information
WEIGHTS - if supplied should be the same length as Y; error variances are
assumed to be inversely proportional to WEIGHTS
PREDICTOR-NAMES
RESPONSE-NAME
CASE-LABELS - sequences of strings or symbols
INCLUDED - if supplied should be the same length as Y, with elements nil
to skip a in computing estimates (but not in residual analysis)
Returns a regression model object. To examine the model further assign the result to a variable and send it messages. Example (data are in file absorbtion.lsp in the sample data directory/folder):
(def m (regression-model (list iron aluminum) absorbtion))
(send m :help)
(send m :plot-residuals)

(sort-data sequence) [Function]
Returns a sequence with the numbers or strings in the sequence X in order.

(standard-deviation x) [Function]
Returns the standard deviation of the elements x. Vector reducing.

(t-cdf x df) [Function]
Returns the value of the T(DF) distribution function at X. Vectorized.

(t-dens x alpha) [Function]
Returns the density at X of the T(DF) distribution. Vectorized.

(t-quant p df) [Function]
Returns the P-th quantile of the T(DF) distribution. Vectorized.

(t-rand n d) [Function]
Returns a list of N T(DF) random variables. Vectorized.

(uniform-rand n) [Function]
Returns a list of N uniform random variables from the range (0, 1). Vectorized.

C.4 Plotting Functions

(boxplot data &key (title "Box Plot")) [Function]
DATA is a sequence, a list of sequences or a matrix. Makes a boxplot of the sequence or a parallel box plot of the sequences in the list or the columns of the matrix.

(boxplot-x x data &key (title "Box Plot")) [Function]
DATA is a list of sequences or a matrix. X is a sequence with as many elements as DATA has elements or columns. Makes a parallel box plot of the sequences in the list or the columns of the matrix vs X.

(close-all-plots) [Function]
Close all plot windos.

(histogram data &key (title "Histogram")) [Function]
Opens a window with a histogram of DATA. TITLE is the window title. The number of bins used can be adjusted using the histogram menu. The histogram can be linked to other plots with the link-views command. Returns a plot object.

(link-views &rest plots) [Function]
Links the argument plots: any change in hiliting or visibility of points in the current plot is propagated to the other plots.

(name-list names &key (title "Name List")) [Function]
NAMES is a number or a list of character strings. Opens a window with a list of the supplied character strings or entries numbered from 0 to NAMES - 1. This display can be linked to plots with the link-views function. Returns a plot object.

(plot-function f xmin xmax &optional (num-points 50)) [Function]
Plots function F of one real variable over the range between xmin and xmax. The function is evaluated at NUM-POINTS points.

(plot-lines x y &key (title "Line Plot") variable-labels type width color) [Function]
Opens a window with a connected line plot of X vs Y, where X and Y are compound number-data. VARIABLE-LABELS, if supplied, should be lists of character strings. TITLE is the window title. The plot can be linked to other plots with the link-views command. Returns a plot object.

(plot-points x y &key (title "Scatter Plot") variable-labels point-labels symbol color) [Function]
Opens a window with a scatter plot of X vs Y, where X and Y are compound number-data. VARIABLE-LABELS and POINT-LABELS, if supplied, should be lists of character strings. TITLE is the window title. The plot can be linked to other plots with the link-views command. Returns a plot object.

(probability-plot data &key (distribution-function (function normal-cdf)) (title "Probability Plot") point-labels) [Function]

(quantile-plot data &key (quantile-function (function normal-quant)) (title "Quantile Plot") point-labels) [Function]

(scatterplot-matrix data &key (title "Spinning Plot") variable-labels point-labels (scale t)) [Function]
DATA is a list of two or more compound number-data objects of equal length. Opens a window with a brushable scatter plot matrix of the elements of DATA. VARIABLE-LABELS and POINT-LABELS, if supplied, should be lists of character strings. TITLE is the window title. If scale is NIL data are assumed to be between -1 and 1.The plot can be linked to other plots with the link-views command. Returns a plot object.

(spin-function f xmin xmax ymin ymax &optional (num-points 6)) [Function]
Rotatable plot of function F of two real variables over the range between [xmin, xmax] x [ymin, ymax]. The function is evaluated at NUM-POINTS points.

(spin-plot data &key (title "Spinning Plot") variable-labels point-labels (scale t)) [Function]
DATA is a list of three compound number-data objects of equal length. Opens a window with a rotating plot of the three elements of DATA. VARIABLE-LABELS and POINT-LABELS, if supplied, should be lists of character strings. TITLE is the window title. If scale is NIL data are assumed to be between -1 and 1. The plot can be linked to other plots with the link-views command. Returns a plot object.

(unlink-views &rest plots) [Function]
Removes links to its arguments. With no arguments removes all links.

C.5 Object Methods

C.5.1 Regression Methods

:basis [Object Method]
Returns the indices of the variables used in fitting the model.

:coef-estimates [Object Method]
Returns the OLS (ordinary least squares) estimates of the regression coefficients. Entries beyond the intercept correspond to entries in basis.

:coef-standard-errors [Object Method]
Returns estimated standard errors of coefficients. Entries beyond the intercept correspond to entries in basis.

:cooks-distances [Object Method]
Computes Cook's distances.

:df [Object Method]
Returns the number of degrees of freedom in the model.

:display [Object Method]
Prints the least squares regression summary. Variables not used in the fit are marked as aliased.

:fit-values [Object Method]
Returns the fitted values for the model.

:included &optional new-included [Object Method]
With no argument, NIL means a case is not used in calculating estimates, and non-nil means it is used. NEW-INCLUDED is a sequence of length of y of nil and t to select cases. Estimates are recomputed.

:intercept &optional new-intercept [Object Method]
With no argument returns T if the model includes an intercept term, nil if not. With an argument NEW-INTERCEPT the model is changed to include or exclude an intercept, according to the value of NEW-INTERCEPT.

:leverages [Object Method]
Returns the diagonal elements of the hat matrix.

:num-cases [Object Method]
Returns the number of cases in the model.

:num-coefs [Object Method]
Returns the number of coefficients in the fit model (including the intercept if the model includes one).

:num-included [Object Method]
Returns the number of cases used in the computations.

:plot-bayes-residuals &optional x-values [Object Method]
Opens a window with a plot of the standardized residuals and two standard error bars for the posterior distribution of the actual deviations from the line. See Chaloner and Brant. If X-VALUES are not supplied the fitted values are used. The plot can be linked to other plots with the link-views function. Returns a plot object.

:plot-residuals &optional x-values [Object Method]
Opens a window with a plot of the residuals. If X-VALUES are not supplied the fitted values are used. The plot can be linked to other plots with the link-views function. Returns a plot object.

:predictor-names &optional (names nil set) [Object Method]
With no argument returns the predictor names. NAMES sets the names.

:r-squared [Object Method]
Returns the sample squared multiple correlation coefficient, R squared, for the regression.

:raw-residuals [Object Method]
Returns the raw residuals for a model.

:residuals [Object Method]
Returns the raw residuals for a model without weights. If the model includes weights the raw residuals times the square roots of the weights are returned.

:sigma-hat [Object Method]
Returns the estimated standard deviation of the deviations about the regression line.

:studentized-residuals [Object Method]
Computes the internally studentized residuals for included cases and externally studentized residuals for excluded cases.

:sum-of-squares [Object Method]
Returns the error sum of squares for the model.

:weights &optional new-w [Object Method]
With no argument returns the weight sequence as supplied to m; NIL means an unweighted model. NEW-W sets the weights sequence to NEW-W and recomputes the estimates.

:x-matrix [Object Method]
Returns the X matrix for the model, including a column of 1's, if appropriate. Columns of X matrix correspond to entries in basis.

:xtxinv [Object Method]
Returns $(X^T X)^{-1}$ or $(X^T W X)^{-1}$.

C.5.2 General Plot Methods

:add-lines lines &key type (draw t) [Object Method]
Adds lines to plot. LINES is a list of sequences, the coordinates of the line starts. TYPE is normal or dashed. If DRAW is true the new lines are added to the screen.

:add-points points &key point-labels (draw t) [Object Method]
Adds points to plot. POINTS is a list of sequences, POINT-LABELS a list of strings. If DRAW is true the new points are added to the screen.

:adjust-to-data &key (draw t) [Object Method]
Sets ranges to the actual range of variables in the original coordinate system. If DRAW is true sends :RESIZE and :REDRAW messages.

:all-points-showing-p [Object Method]

:all-points-unmasked-p [Object Method]

:any-points-selected-p [Object Method]

:apply-transformation a &key draw [Object Method]
Applies matrix A to current transformation. If draw is true the :REDRAW-CONTENT message is sent.

:clear &key (draw t) [Object Method]
Clears the plot data. If DRAW is nil the plot is redrawn; otherwise its current screen image remains unchanged.

:clear-lines &key (draw t) [Object Method]
Removes all lines from the plot. If DRAW is true the :REDRAW-CONTENT message is sent.

:clear-points &key (draw t) [Object Method]
Removes all points from the plot. If DRAW is true the :REDRAW-CONTENT message is sent.

:clear-strings &key (draw t) [Object Method]
Removes all strings from the plot. If DRAW is true the :REDRAW-CONTENT message is sent.

:content-variables &optional xvar yvar [Object Method]
Sets or retrieves the indices of the current content variables.

:do-mouse x y type extend option [Object Method]
Sends appropriate action message for mouse mode to plot.

:drag-grey-rect x y width height [Object Method]
Drags grey rectangle starting at (LIST (- X WIDTH) (- Y HEIGHT) WIDTH HEIGHT) while mouse button is down. Returns the final rectangle. Should be called when the mouse is down.

:erase-selection [Object Method]
Sets selected points states to invisible and sends :ADJUST-SCREEN message.

:fixed-aspect &optional fixed [Object Method]
Sets or retrieves current size adjustment option (true or NIL).

:frame-location &optional left top [Object Method]
Moves window frame to (LEFT TOP) if supplied. Returns list of current left, top. Adjusts for the menu bar.

:frame-size &optional width height [Object Method]
Sets window frame width and size to WIDTH and SIZE if supplied. Returns list of current WIDTH and HEIGHT. Adjusts for the menu bar.

:idle-on &optional on [Object Method]
Sets or returns idling state. On means :do-idle method is sent each pass through the event loop.

:linked &optional on [Object Method]
Sets or retrieves plot's linking state.

:num-lines [Object Method]
Returns the number of line starts in the plot.

:num-points [Object Method]
Returns the number of points in the plot.

:num-strings [Object Method]
Returns the number of strings in the plot.

:num-variables [Object Method]
Returns the number of variables in the plot.

:point-coordinate var point &optional value [Object Method]
Sets or retrieves coordinate for variable VAR and point POINT in the original coordinate system. Vectorized.

:point-hilited point &optional hilited [Object Method]
Sets or returns highlighting status (true or NIL) of POINT. Sends :ADJUST-SCREEN message if states are set. Vectorized.

:point-label point &optional label [Object Method]
Sets or retrieves label of point POINT. Vectorized.

:point-selected point &optional selected [Object Method]
Sets or returns selection status (true or NIL) of POINT. Sends :ADJUST-SCREEN message if states are set. Vectorized.

:point-showing point &optional selected [Object Method]
Sets or returns visibility status (true or NIL) of POINT. Sends :ADJUST-SCREEN message if states are set. Vectorized.

:point-symbol point &optional symbol [Object Method]
Sets or retrieves symbol of point POINT. Vectorized.

:range index &optional low high [Object Method]
Sets or retrieves variable's original coordinate range. Vectorized.

:real-to-screen x y [Object Method]
Returns list of screen coordinates of point (X, Y), in the original coordinate system, based on current content variables.

:redraw [Object Method]
Redraws entire plot.

:redraw-content [Object Method]
Redraws plot's content.

:rotate-2 var1 var2 angle &key (draw t) [Object Method]
Rotates int the plane of variables with indices VAR1 and VAR2 by ANGLE, in radians. sends the :REDRAW-CONTENT message if DRWA is true.

:scale-to-range var low high &key (draw t) [Object Method]
Scales and shifts data to map visible range into specified range. Sends :RESIZE and :REDRAW messages if DRAW is true.

:scaled-range index &optional low high [Object Method]
Sets or retrieves variable's transformed coordinate range. Vectorized.

:screen-to-real x y [Object Method]
Returns list of real coordinates, in the original coordinate system, of screen point (X, Y), based on current content variables.

:selection [Object Method]
Return indices of current selection.

:show-all-points [Object Method]
Sets all point states to normal and sends :ADJUST-SCREEN message

:showing-labels &optional showing [Object Method]
Sets or retrieves current labeling state (true or NIL).

:title &optional string [Object Method]
Sets or retrieves window title.

:transformation &optional a &key (draw t) [Object Method]
Sets or retrieves transformation. A should be a matrix or NIL. If draw is true the :REDRAW-CONTENT message is sent.

:unselect-all-points &key (draw t) [Object Method]
Unselects all points. Sends :ADJUST-SCREEN message if DRAW is true.

:variable-label var &optional label [Object Method]
Sets or returns label for variable with index VAR. Vectorized.

:visible-range var [Object Method]
Returns list of min and max of variable VAR over visible, unmasked points, lines and strings. Vectorized.

:while-button-down fcn &optional (motion-only t) [Object Method]
Calls fcn repeatedly while mouse button is down. FCN should take two arguments, the current x and y coordinates of the mouse. Returns NIL. Should be called when button is already down.

:x-axis &optional showing labeled ticks [Object Method]
Sets or retrieves current acis label state. SHOWING and LABELED should be true or NIL; TICKS should be a number. All three should be supplied for setting a new state. A list of the three properties is returned.

:y-axis &optional showing labeled ticks [Object Method]
Sets or retrieves current acis label state. SHOWING and LABELED should be true or NIL; TICKS should be a number. All three should be supplied for setting a new state. A list of the three properties is returned.

C.5.3 Histogram Methods

:add-points points (draw t) [Object Method]
Adds points to plot. POINTS is a sequence or a list of sequences. If DRAW is true the new points are added to the screen.

:num-bins &optional bins &key (draw t) [Object Method]
Sets or retrieves number of bins in the histogram. Sends :REDRAW-CONTENT message if DRAW is true.

C.5.4 Name List Methods

:add-points points &key point-labels (draw t) [Object Method]
Adds points to plot. POINTS is a number or a list of sequences, POINT-LABELS a list of strings. If DRAW is true the new points are added to the screen.

C.5.5 Scatterplot Methods

:abline a b [Object Method]
Adds the graph of the line A + B x to the plot.

:add-lines lines &key type (draw t) [Object Method]
Adds lines to plot. LINES is a list of sequences, the coordinates of the line starts. TYPE is normal or dashed. If DRAW is true the new lines are added to the screen.

:add-points points &key point-labels (draw t) [Object Method]
Adds points to plot. POINTS is a list of sequences, POINT-LABELS a list of strings. If DRAW is true the new points are added to the screen.

:add-strings locations strings [Object Method]
Adds strings to plot. LOCATIONS is a list of sequences, the coordinates of the strings. If DRAW is true the new lines are added to the screen.

C.5.6 Spin Plot Methods

:abcplane a b c [Object Method]
Adds the graph of the plane A + B x + Cy to the plot.

:add-function [Object Method]
surface of function F over a NUM-POINTS by NUM-POINTS grid on the rectangle [xmin, xmax] x [ymin, ymax]. Passes other keywords to :add-surface method.

:add-surface [Object Method]
a grid surface using sequences X, Y with values in the matrix Z. Z should be (length X) by (length Y).

:angle &optional angle [Object Method]
Sets or retrieves current rotation angle, in radians.

:content-variables &optional xvar yvar [Object Method]
Sets or retrieves the indices of the current content variables.

:depth-cuing &optional cuing [Object Method]
Sets or retrieves depth cuing status (true or NIL).

:do-idle [Object Method]
Sends :ROTATE message.

:rotate [Object Method]
Rotates once in the current plane by the current angle.

:showing-axes &optional cuing [Object Method]
Sets or retrieves axis showing status (true or NIL).

C.6 Some Useful Array and Linear Algebra Functions

(%* a b) [Function]
Returns the matrix product of matrices a and b. If a is a vector it is treated as a row vector; if b is a vector it is treated as a column vector.

(aref array &rest subscripts) [Function]
Returns the element of ARRAY specified by SUBSCRIPTS.

(array-dimension array) [Function]
Returns a list whose elements are the dimensions of ARRAY

(array-dimensions array) [Function]
Returns a list whose elements are the dimensions of ARRAY

(array-in-bounds-p array &rest subscripts) [Function]
Returns T if SUBSCRIPTS are valid subscripts for ARRAY; NIL otherwise.

(array-rank array) [Function]
Returns the number of dimensions of ARRAY.

(array-row-major-index array &rest subscripts) [Function]
Returns the index into the data vector of ARRAY for the element of ARRAY specified by SUBSCRIPTS.

(array-total-size array) [Function]
Returns the total number of elements of ARRAY.

(arrayp x) [Function]
Returns T if X is an array; NIL otherwise.

(bind-columns &rest args) [Function]
The ARGS can be matrices, vectors, or lists. Arguments are bound into a matrix along their columns. Example: (bind-columns #2a((1 2)(3 4)) #(5 6)) returns #2a((1 2 5)(3 4 6))

(bind-rows &rest args) [Function]
The ARGS can be matrices, vectors, or lists. Arguments are bound into a matrix along their rows. Example: (bind-rows #2a((1 2)(3 4)) #(5 6)) returns #2a((1 2)(3 4)(5 6))

(chol-decomp a) [Function]
Modified Cholesky decomposition. A should be a square, symmetric matrix. Computes lower triangular matrix L such that $LL^T = A + D$ where D is a diagonal matrix. If A is strictly positive definite D will be zero. Otherwise D is as small as possible to make $A + D$ numerically strictly positive definite. Returns a list $(L(maxD))$.

(column-list m) [Function]
Returns a list of the columns of M as vectors

(copy-array array) [Function]
Returns a copy of ARRAY with elements eq to the elements of ARRAY.

(copy-list list) [Function]
Returns a new copy of LIST.

(copy-vector vector) [Function]
Returns a copy of VECTOR with elements eq to the elements of VECTOR

(count-elements number &rest more-numbers) [Function]
Returns the number of its arguments. Vector reducing

(cross-product x) [Function]
If X is a matrix returns (matmult (transpose X) X). If X is a vector returns (inner-product X X).

(determinant m) [Function]
Returns the determinant of the square matrix M.

(diagonal x) [Function]
If X is a matrix, returns the diagonal of X. If X is a sequence, returns a diagonal matrix of rank (length X) with diagonal elements eq to the elements of X.

(identity-matrix n) [Function]
Returns the identity matrix of rank N.

(inner-product x y) [Function]
Returns inner product of sequences X and Y.

(inverse m) [Function]
Returns the inverse of the the square matrix M; signals an error if M is ill conditioned or singular

(lu-decomp a) [Function]
A is a square matrix of numbers (real or complex). Computes the LU decomposition of A and returns a list of the form (LU IV D FLAG), where LU is a matrix with the L part in the lower triangle, the U part in the upper triangle (the diagonal entries of L are taken to be 1), IV is a vector describing the row permutation used, D is 1 if the number of permutations is odd, -1 if even, and FLAG is T if A is numerically singular, NIL otherwise. Used bu LU-SOLVE.

(lu-solve lu b) [Function]
LU is the result of (LU-DECOMP A) for a square matrix A, B is a sequence. Returns the solution to the equation Ax = B. Signals an error if A is singular.

(make-list size &key (initial-element nil)) [Function]
Creates and returns a list containing SIZE elements, each of which is initialized to INITIAL-ELEMENT.

(make-sweep-matrix x y &optional weights) [Function]
X is a matrix, Y and WEIGHTS are sequences. Returns the sweep matrix for the (possibly weighted) regression of Y on X.

(map-elements function data &rest more-data) [Function]
FUNCTION must take as many arguments as there are DATA arguments supplied. DATA arguments must either all be sequences or all be arrays of the same shape. The result is of the same type and shape as the first DATA argument, with elements the result of applying FUNCTION elementwise to the DATA arguments

(matmult a b) [Function]
Returns the matrix product of matrices a and b. If a is a vector it is treated as a row vector; if b is a vector it is treated as a column vector.

(matrix dim data) [Function]
returns a matrix of dimensions DIM initialized using sequence DATA in row major order.

(matrixp m) [Function]
Returns T if M is a matrix, NIL otherwise.

(outer-product x y &optional (fcn #'*)) [Function]
Returns the generalized outer product of x and y, using fcn. That is, the result is a matrix of dimension ((length x) (length y)) and the (i j) element of the result is computed as (apply fcn (aref x i) (aref y j)).

(permute-array a p) [Function]
Returns a copy of the array A permuted according to the permutation P.

(qr-decomp a) [Function]
A is a matrix of real numbers with at least as many rows as columns. Computes the QR factorization of A and returns the result in a list of the form (Q R).

(rcondest a) [Function]
Returns an estimate of the reciprocal of the L1 condition number of an upper triangular matrix a.

(row-list m) [Function]
Returns a list of the rows of M as vectors

(solve a b) [Function]
Solves A x = B using LU decomposition and backsolving. B can be a sequence or a matrix.

(split-list list cols) [Function]
Returns a list of COLS lists of equal length of the elements of LIST. Example: (split-list '(1 2 3 4 5 6) 2) returns ((1 2 3) (4 5 6))

(sum &rest number-data) [Function]
Returns the sum of all the elements of its arguments. Returns 0 if there are no arguments. Vector reducing.

(sv-decomp a) [Function]
A is a matrix of real numbers with at least as many rows as columns. Computes the singular value decomposition of A and returns a list of the form (U W V FLAG) where U and V are matrices whose columns are the left and right singular vectors of A and W is the sequence of singular values of A. FLAG is T if the algorithm converged, NIL otherwise.

(sweep-operator a indices &optional tolerances) [Function]
A is a matrix, INDICES a sequence of the column indices to be swept. Returns a list of the swept result and the list of the columns actually swept. (See MULTREG documentation.) If supplied, TOLERANCES should be a list of real numbers the same length as INDICES. An index will only be swept if its pivot element is larger than the corresponding element of TOLERANCES.

(transpose m) [Function]
Returns the transpose of the matrix M.

(vectorp m) [Function]
Returns T if M is a vector, NIL otherwise.

C.7 System Functions

(alloc number) [Function]
Changes number of nodes to allocate in each segment to NUMBER. Returns old number of nodes to allocate.

(call-cfun cfun &rest args) [Function]
CFUN is a string naming a C function. The remaining arguments must be integers, sequences of integers, reals or sequences of reals. CFUN is called with the remaining arguments and a list of the lists of the values of the arguments after the call is returned. Arguments in the call will be pointers to ints or pointers to doubles. Not available on all implementations.

(call-fsub fsub &rest args) [Function]
FSUB is a string naming a FORTRAN subroutine. The remaining arguments must be integers, sequences of integers, reals or sequences of reals. FSUB is called with the remaining arguments and a list of the lists of the values of the arguments after the call is returned. Arguments in the call will be (arrays of) integers or double precision numbers. Not available on all implementations.

(call-lfun lfun &rest args) [Function]
LFUN is a C function written to conform to internal XLISP argument reading and value returning conventions. Applies LFUN to ARGS and returns the result.

(debug) [Function]
Enable breaking on error on.

(dyn-load file &key verbose libflags fortran) [Function]
Links the object file FILE with standard C libraries and loads into the running XLISP-STAT process. If FORTRAN is not NIL also searches standard FORTRAN libraries. LIBFLAGS can be a string used to specify additional libraries, for example

(exit) [Function]
Exits from XLISP.

(expand number) [Function]
Expand memory by adding NUMBER segments. Returns the number of segments.

(gc) [Function]
Forces garbage collection. Returns nil.

(help &optional symbol) [Function]
Prints the documentation associated with SYMBOL. With no argument, this function prints the greeting message to beginners.

(help* string) [Function]
Prints the documentation associated with those symbols whose print names contain STRING as substring. STRING may be a symbol, in which case the print-name of that symbol is used.

(load filename &key (verbose t) (print nil)) [Function]
Loads the file named by FILENAME into XLISP. Returns T if load succeeds, NIL if file does not exist.

(nodebug) [Function]
Disable breaking on error on.

(room) [Function]
Shows memory allocation statistics. Returns nil.

(save file) [Function]
Saves current memory image in FILE.wks. Does not work right with allocated objects.

(variables) [Function]
Prints the names of all def'ed variables

C.8 Some Basic Lisp Functions, Macros and Special Forms

Except where noted these functions should have a significant subset of their Common Lisp functionality as defined in Steele [15].

(and {form}*) [Macro]
Evaluates FORMs in order from left to right. If any FORM evaluates to NIL, returns immediately with the value NIL. Else, returns the value of the last FORM.

(append &rest lists) [Function]
Constructs a new list by concatenating its arguments.

(apply function &rest args) [Function]
Conses all arguments but the last onto the last and applies FUNCTION to the resulting argument list. Last argument must be a list.

(apropos string) [Function]
Prints symbols whose print-names contain STRING as substring. If STRING is a symbol its print name is used.

(apropos-list string) [Function]
Returns, as a list, all symbols whose print-names contain STRING as substring. If STRING is a symbol its print name is used.

(assoc item alist &key (test (function eql)) test-not) [Function]
Returns the first pair in ALIST whose car is equal (in the sense of TEST) to ITEM.

(atom x) [Function]
Returns T if X is not a cons; NIL otherwise.

(boundp symbol) [Function]
Returns T if the global variable named by SYMBOL has a value; NIL otherwise.

(car list) [Function]
Returns the car of LIST. Returns NIL if LIST is NIL.

(case keyform (key | (key*) form*) *) [Function]
Evaluates KEYFORM and tries to find the KEY that is EQL to the value of KEYFORM. If one is found, then evaluates FORMs that follow the KEY and returns the value of the last FORM. If not, simply returns NIL.

(cdr list) [Function]
Returns the cdr of LIST. Returns NIL if LIST is NIL.

(close stream) [Function]
Close file stream STREAM.

(coerce x type) [Function]
Coerces X to an object of the type TYPE.

(cond (test form*) *) [Function]
Evaluates each TEST in order until one evaluates to a non-NIL value. Then evaluates the associated FORMs in order and returns the value of the last FORM. If no forms follow the TEST, then returns the value of the TEST. Returns NIL, if all TESTs evaluate to NIL.

(cons x y) [Function]
Returns a new cons (list node) whose car and cdr are X and Y, respectively.

(consp x) [Function]
Returns T if X is a cons; NIL otherwise.

(defmacro name defmacro-lambda-list [doc] {form}*) [Macro]
Defines a macro as the global definition of the symbol NAME. The complete syntax of a lambda-list is: (var* [&optional var*] [&rest var] [&aux var*]) The doc-string DOC, if supplied, is saved as a FUNCTION doc and can be retrieved by (documentation 'NAME 'function).

(defun name lambda-list [doc] {form}*) [Macro]
Defines a function as the global definition of the symbol NAME. The complete syntax of a lambda-list is: (var* [&optional var*] [&rest var] [&aux var*]) The doc-string DOC, if supplied, is saved as a FUNCTION doc and can be retrieved by (documentation 'NAME 'function).

(do ({(var [init [step]])}*) (endtest {result}*) {tag | statement}*) [Macro]
Creates a NIL block, binds each VAR to the value of the corresponding INIT, and then executes STATEMENTs repeatedly until ENDTEST is satisfied. After each iteration, assigns to each VAR the value of the corresponding STEP. When ENDTEST is satisfied, evaluates RESULTs as a PROGN and returns the value of the last RESULT (or NIL if no RESULTs are supplied). Performs variable bindings and assignments all at once, just like LET does.

(do* ({(var [init [step]])}*) (endtest {result}*) tag | statement*) [Macro]
Just like DO, but performs variable bindings and assignments in serial, just like LET* and SETQ do.

(dolist (var listform [result]) {tag | statement}*) [Macro]
Executes STATEMENTs, with VAR bound to each member of the list value of LISTFORM. Then returns the value of RESULT (which defaults to NIL).

(dotimes (var countform [result]) {tag | statement}*) [Macro]
Executes STATEMENTs, with VAR bound to each number between 0 (inclusive) and the value of COUNTFORM (exclusive). Then returns the value of RESULT (which defaults to NIL).

(elt a &rest indices) [Function]
A can be a list or an array. If A is a list and INDICES is a single number then the appropriate element of A is returned. If is a list and INDICES is a list of numbers then the sublist of the corresponding elements is returned. If A in an array then the number of INDICES must match the ARRAY-RANK of A. If each index is a number then the appropriate array element is returned. Otherwise the INDICES must all be lists of numbers and the corresponding submatrix of A is returned. ELT can be used in setf.

(eq x y) [Function]
Returns T if X and Y are the same identical object; NIL otherwise.

(eql x y) [Function]
Returns T if X and Y are EQ, or if they are numbers of the same type with the same value, or if they are identical strings. Returns NIL otherwise.

(equal x y) [Function]
Returns T if X and Y are EQL or if they are of the same type and corresponding components are EQUAL. Returns NIL otherwise. Arrays must be EQ to be EQUAL.

(equalp x y) [Function]
Returns T if (equal x y), or x, y are numbers and (= x y), or x and y are strings and (string-equal x y).

(first x) [Function]
Equivalent to (CAR X).

(format destination control &rest args) [Function]
Very basic implementation of Common Lisp format function. Only A, S, D, F, E, G, and G can take two.

(funcall function &rest arguments) [Function]
Applies FUNCTION to the ARGUMENTs

(function x) [Special Form]
or #'x If X is a lambda expression, creates and returns a lexical closure of X in the current lexical environment. If X is a symbol that names a function, returns that function.

(getf place indicator &optional default) [Function]
Returns property value of INDICATOR in PLACE, or DEFAULT if not found.

(identity x) [Function]
Simply returns X.

(if test then [else]) [Macro]
If TEST evaluates to non-NIL, then evaluates THEN and returns the result. If not, evaluates ELSE (which defaults to NIL) and returns the result.

(last list) [Function]
Returns the last cons in LIST

(length sequence) [Function]
Returns the length of SEQUENCE.

(let (var | (var [value]) *) form*) [Function]
Initializes VARs, binding them to the values of VALUEs (which defaults to NIL) all at once, then evaluates FORMs as a PROGN.

(let* (var | (var [value]) *) form*) [Function]
Initializes VARs, binding them to the values of VALUEs (which defaults to NIL) from left to right, then evaluates FORMs as a PROGN.

(listp x) [Function]
Returns T if X is either a cons or NIL; NIL otherwise.

(map result-type function sequence &rest more-sequences) [Function]
FUNCTION must take as many arguments as there are sequences provided. RESULT-TYPE must be the either the symbol VECTOR or the symbol LIST. The result is a sequence of the specified type such that the i-th element of the result is the result of applying FUNCTION to the i-th elements of the SEQUENCEs.

(mapc fun list &rest more-lists) [Function]
Applies FUN to successive cars of LISTs. Returns the first LIST.

(mapcar fun list &rest more-lists) [Function]
Applies FUN to successive cars of LISTs and returns the results as a list.

(mapl fun list &rest more-lists) [Function]
Applies FUN to successive cdrs of LISTs. Returns the first LIST.

(maplist fun list &rest more-lists) [Function]
Applies FUN to successive cdrs of LISTs and returns the results as a list.

(member item list &key (test (function eql)) test-not) [Function]
Returns the tail of LIST beginning with the first ITEM.

(not x) [Function]
Returns T if X is NIL; NIL otherwise.

(nth n list) [Function]
Returns the N-th element of LIST, where the car of LIST is the zero-th element.

(nthcdr n list) [Function]
Returns the result of performing the CDR operation N times on LIST.

(null x) [Function]
Returns T if X is NIL; NIL otherwise.

(numberp x) [Function]
Returns T if X is any kind of number; NIL otherwise.

(objectp x) [Function]
Returns T if X is an object, NIL otherwise.

(open fname &key (direction :input)) [Function]
Opens file named by string or symbol FNAME. DIRECTION is :INPUT or :OUTPUT.

(or {form}*) [Macro]
Evaluates FORMs in order from left to right. If any FORM evaluates to non-NIL, quits and returns that value. If the last FORM is reached, returns whatever value it returns.

(prin1 object &optional (stream *standard-output*)) [Function]
Prints OBJECT in the most readable representation. Returns OBJECT.

(princ object &optional (stream *standard-output*)) [Function]
Prints OBJECT without escape characters. Returns OBJECT.

(print object &optional (stream *standard-output*)) [Function]
Outputs a newline character, and then prints OBJECT in the most readable representation. Returns OBJECT.

(prog ({var | (var [init])}*) {tag | statement}*) [Macro]
Binds VARs in parallel, and then executes STATEMENTs.

(prog* ({var | (var [init])}*) {tag | statement}*) [Macro]
Binds VARs sequentially, and then executes STATEMENTs.

(prog1 first {form}*) [Macro]
Evaluates FIRST and FORMs in order, and returns the value of FIRST.

(prog2 first second {forms}*) [Macro]
Evaluates FIRST, SECOND, and FORMs in order, and returns the value of SECOND.

(progn {form}*) [Macro]
Evaluates FORMs in order, and returns whatever the last FORM returns.

(progv symbols values {form}*) [Macro]
Evaluates FORMs in order, with SYMBOLS dynamically bound to VALUES, and returns whatever the last FORM returns.

(provide name) [Function]
Adds NAME to the list of modules.

(quote x) [Special Form]
or 'x Returns X without evaluating it.

(read &optional (stream *standard-input*) (eof-error-p t) (eof-value nil) (recursivep nil)) [Function]
Reads and returns the next object from STREAM.

(reduce function sequence &key initial-value) [Function]
Combines all the elements of SEQUENCE using a binary operation FUNCTION. If INITIAL-VALUE is supplied it is logically placed before SEQUENCE.

(remove item list &key (test (function eql)) test-not) [Function]
Returns a copy of LIST with ITEM removed.

(remove-if test list) [Function]
Returns a copy of LIST with elements satisfying TEST removed.

(remove-if-not test list) [Function]
Returns a copy of LIST with elements not satisfying TEST removed.

(require name) [Function]
Loads module NAME, unless it has already been loaded. If PATH is supplied it is used as the file name; otherwise NAME is used. If file NAME is not in the current directory *default-path* is searched.

(rest x) [Function]
Equivalent to (CDR X).

(return [result]) [Macro]
Returns from the lexically surrounding PROG construct. The value of RESULT, which defaults to NIL, is returned as the value of the PROG construct.

(reverse list) [Function]
Returns a new list containing the same elements as LIST but in reverse order.

(second x) [Function]
Equivalent to (CAR (CDR X)).

(set symbol value) [Function]
Assigns the value of VALUE to the dynamic variable named by SYMBOL (i. e. it changes the global definition of SYMBOL), and returns the value assigned.

(setf {place newvalue}*) [Macro]
Replaces the value in PLACE with the value of NEWVALUE, from left to right. Returns the value of the last NEWVALUE. Each PLACE may be any one of the following: * A symbol that names a variable. * A function call form whose first element is the name of the following functions: nth aref subarray sublist select elt get symbol-value symbol-plist documentation slot-value c?r c??r c???r c????r where '?' stands for either 'a' or 'd'.

(setq {var form}*) [Macro]
VARs are not evaluated and must be symbols. Assigns the value of the first FORM to the first VAR, then assigns the value of the second FORM to the second VAR, and so on. Returns the last value assigned.

(string sym) [Function]
Returns print-name of SYM if SYM is a symbol, or SYM if SYM is a.

(stringp x) [Function]
Returns T if X is a string; NIL otherwise.

(sublis alist tree &key (test (function eql)) test-not) [Function]
Substitutes from ALIST for subtrees of TREE nondestructively.

(subst new old tree &key (test (function eql)) test-not) [Function]
Substitutes NEW for subtrees of TREE that match OLD.

(symbol-name symbol) [Function]
Returns the print name of the symbol SYMBOL.

(symbol-plist symbol) [Function]
Returns the property list of SYMBOL.

(symbol-value symbol) [Function]
Returns the current global value of the variable named by SYMBOL.

(symbolp x) [Function]
Returns T if X is a symbol; NIL otherwise.

(terpri &optional (stream *standard-output*)) [Function]
Outputs a newline character.

(time form) [Macro]
Form is evaluated and its result returned. In addition the time required for the evaluation is printed.

(type-of x) [Function]
Returns the type of X.

(unless test {form}*) [Macro]
If TEST evaluates to NIL evaluates FORMs as a PROGN. If not, returns NIL.

(unwind-protect protected-form {cleanup-form}*) [Macro]
Evaluates PROTECTED-FORM and returns whatever it returned. Guarantees that CLEANUP-FORMs be always evaluated before exiting from the UNWIND-PROTECT form.

(when test {form}*) [Macro]
If TEST evaluates to non-NIL evaluates FORMs as a PROGN. If not, returns NIL.

Index