

CLASP

Common Lisp Analytical Statistics Package

October 9, 1993

CLASP programming interface

- **Data Structures**
- **Modules**
- **Functions**

CLASP data structures

data

All data classes in CLASP have DATA as a superclass

- name: The name the user will refer to the data by
- description: A textual history of the data

variable (data)

A variable is a collection of semantically related values

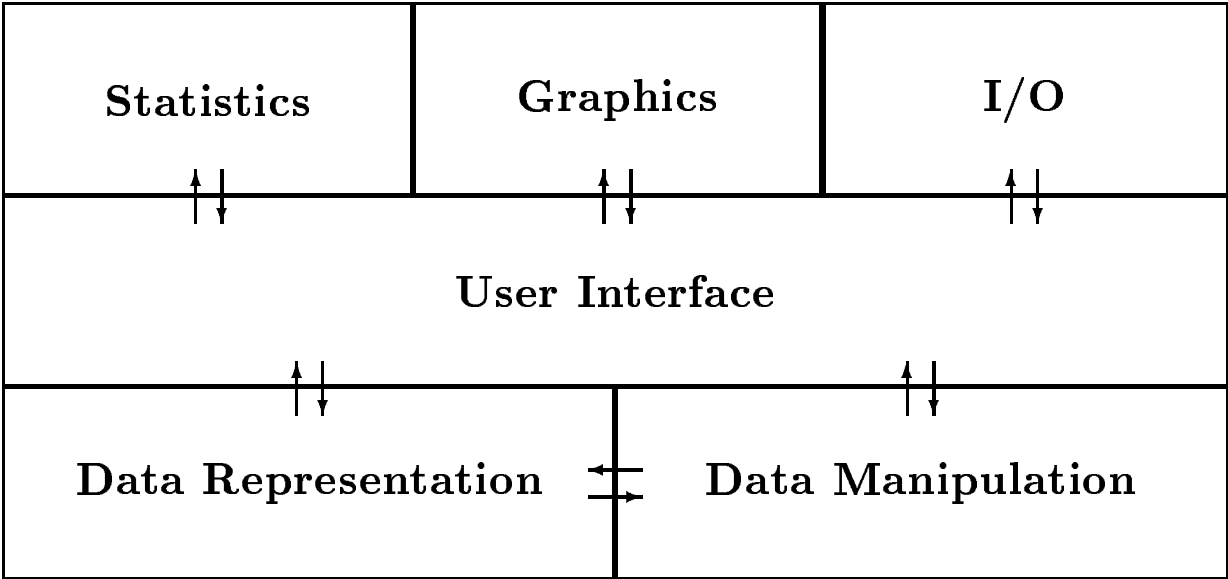
- value: The values in the variable
- dataset: The dataset to which the variable belongs

dataset (data)

A dataset is a collection of variables

- variables: List of the variables in the dataset
- rows: Number of rows in the dataset

CLASP modules



Data Representation Functions

In general, data-representation functions deal with the data objects defined in CLASP. With a few exceptions (primarily the slot-accessors), anywhere a dataset or variable is a function parameter, it is also legal to use a string or symbol with the name of that dataset or variable.

Some commonly used CLASP functions

- Creating datasets
 - make-dataset-from-columns *name data variable-names*
 - make-dataset-from-rows *name data variable-names*
- Accessing CLASP data objects
 - get-dataset *dataset-name*
 - get-variable *variable-name* *Optional dataset*
- Deleting CLASP data objects
 - delete-dataset *dataset*
 - delete-variable *variable*

Data access functions

- Getting data from a dataset
 - `dataset-to-rows` *dataset :rows (where-clause)*
:columns (variable-list)
 - `dataset-to-columns` *dataset :rows (where-clause)*
:columns (variable-list)
- Getting data from a variable
 - `variable-value` *:where (where-clause)*
:order-by variable

I/O Functions

There are four I/O functions,

load-dataset *filename*

save-dataset *dataset filename*

import-dataset *ℰkey separator include-labels-p*

export-dataset *ℰkey separator include-labels-p*

Load-dataset and save-dataset use CLASP format. In a CLASP format file, the first line is a double-quoted string containing the name of the dataset, the next *n* lines are double-quoted strings with the names of each variable, and the next *m* lines are rows of the dataset, each row being a list of values, and the order must be the same as the variable names in the header.

Import-dataset and export-dataset are for use with non-clasp format data files.

The data appear in columns separated by an optional separator character (default “,”). If include-labels-p is t, then the first line of the file will contain the names of the variables.

Data Manipulation Functions

- `partition-dataset` *dataset partition-clause*
Optional variables

This creates a new dataset containing all the rows of *dataset* which satisfy the conditions in *partition-clause*. If *variables* is a list of variables from *dataset*, then only those variables will be carried over into the new dataset, otherwise all variables will. If the partition clause contains an `(.on. variable)` expression then the values of variable will be mapped over and a new dataset will be created for each unique value.

- `merge-datasets` *datasets*

This creates a new dataset which has as its variables the union of all the variables in *datasets*, plus an extra variable, `SOURCE`. For every dataset *d* in *datasets*, each of *d*'s rows contributes one row to the new dataset, and for that row, the value of all the variables in *d* are copied into the new dataset, and any variables in the new dataset that don't exist in *d* are assigned the value `nil`. Finally, the variable `SOURCE` in the new dataset is assigned the name of *d*, the dataset from which that row came.

Applying a function to data

`create-new-column-from-function` *dataset expression*
‘expression’ must be a valid lisp expression, it may contain the names of variables in ‘dataset’. The rows of ‘dataset’ are mapped over, substituting the values of the variables in ‘expression’, after substitution, ‘expression’ is evaluated. This produces a new column of data which is added as a variable to ‘dataset’. For example, the following would produce the ratio of “Nodes Searched” to “Runtime” in the dataset “My Data”, and add it as a new variable.

```
(create-new-column-from-function  
  'my-data '(/ nodes-searched runtime))
```

Statistical Functions

All statistical functions operate on sequences.

Wherever it is possible/makes sense, they take a set of standard keyword arguments including *:start*, *:end* and *:key*. These are used to determine what part of the sequence is processed and how to get at elements of the sequence.

For instance,

```
(mean '((a 3) (g 5) (c 7) (a 4) (i 3))
      :start 1 :end 3 :key \#'second)
```

$\Rightarrow 5\frac{1}{3}$

Special case statistical functions

The following is a list of things to be aware of:

- **-from-summaries statistics**

`correlation-from-summaries`

`confidence-interval-t-summaries`

`confidence-interval-z-summaries`

These are more efficient versions of the `correlation`, `confidence-interval-t` and `confidence-interval-z` functions. They do their computations directly from various summary statistics, so if you have those summary statistics lying around, you should call the `-summaries` version of the functions.

- **Special cases for arguments**

`interquartile-range` doesn't recognize *`:start`*, *`:end`* and *`:key`* keywords

`t-test` and its variants (`-one-sample` and `-matched`) take an extra argument, *`tails`*, which must be `:positive`, `:negative` or `:both`

- **Chi Square**

For chi-square analyses, there are `-counts` versions of the functions which are the moral equivalent of `-summaries` versions of other statistical functions.

Special case statistical functions

- Anova

`anova-one-way-variables` should be used if you want to pass in the sequences for the anova

`anova-one-way-groups` should be called if you have the data in grouped form, see the manual for a description

The same holds for `anova-two-way-variables` and `anova-two-way-groups`.

`anova-two-way-variables-unequal-cell-sizes` should be used if you have data with unequal cell sizes

- Linear Regression

There are six versions of linear-regression

`linear-regression-minimal`

`linear-regression-minimal-summaries`

`linear-regression-brief`

`linear-regression-brief-summaries`

`linear-regression-verbose`

`linear-regression-verbose-summaries`

`minimal` returns just the slope and intercept. `brief` returns the slope, intercept, r^2 , the standard-error of the slope and the significance. `verbose` also returns an anova table.

An example using lisp and clasp together

```
;;; Maps across the values of prediction-point and
;;; prediction-threshold, extracting the appropriate
;;; rows for prediction and actual and calculating the
;;; average prediction error for each condition
(defun summary ()
  ;; loop through independent variable values
  (loop for pt in '(.6 .75 .9)
    append
      (loop for pp in '(2 5 7)
        collect
          ;; extract appropriate rows for prediction and actual
          (let
            ((prediction
              (variable-value
                (get-variable 'predicted-queue-length-port-1)
                :where
                '(.and. (==. prediction-point ,pp)
                      (==. prediction-threshold ,pt)))))
              (actual
                (variable-value
                  (get-variable 'ships-queued-port-1)
                  :where
                  '(.and. (==. prediction-point ,pp)
                        (==. prediction-threshold ,pt)))))))
```

```

;; collect the independent variables and the
;; prediction error as calculated from prediction
;; and actual
(list pt pp
      (summarize-prediction-error
       prediction actual pp))))))

;;; Since the system attempts to predict what the
;;; queue length will be lag days in advance,
;;; lagged-correlation is a measure of how accurate
;;; the prediction was.
(defun summarize-prediction-error
  (predicted-sequence actual-sequence prediction-lag)
  (unless (and predicted-sequence actual-sequence
                prediction-lag)
    (return-from summarize-prediction-error ':missing))
  (lagged-correlation predicted-sequence
                       actual-sequence prediction-lag))

```

;;; Here's what a typical clasp session might look
;;; like using the functions defined above.

```
(load-dataset 'port-state-demo.clasp')
(setf new-data summary)
(make-dataset-from-rows 'compressed-port-data'
  new-data
  '(('trial'
      'prediction-threshold'
      'eta-variance-multiplier'
      'prediction-point'
      'prediction-accuracy)))
```