SSJ User's Guide

Package `stat`

Tools for Collecting Statistics

Version: September 29, 2015

Contents

Overview of package `stat`

This package provides elementary tools for collecting statistics and computing confidence intervals. The base class `StatProbe` implements common methods needed by all probes. Its subclass `Tally` collects data as a sequence of observations $X_1, X_2, \ldots$, and computes sample averages, sample standard deviations, and confidence intervals based on the normality assumption. [1] The class `TallyStore` is similar, but it also stores the individual observations in a list implemented as a `DoubleArrayList`, a class imported from the COLT library. This permits one to compute more quantities and to use the methods provided by COLT for computing descriptive statistics.

The class `Accumulate`, in package `simevents`, computes integrals and averages with respect to time. This class is in package `simevents` because its operation depends on the simulation clock.

All classes that represent statistical probes support the *observer* design pattern, well-known in software engineering [1]. This pattern facilitates the separation of data generation (by the simulation program) from data processing (for statistical reports and displays). This can be very helpful in particular in large simulation programs or libraries, where different objects may need to process the same data in different ways. A statistical probe maintains a list of registered `ObservationListener` objects, and broadcasts information to all its registered observers whenever appropriate. Any object that implements the interface `ObservationListener` can register as an observer. For an example, see the program `QueueObs` in the directory `examples`.

Subpackages of package `stat` provide matrices of `Tally`'s and lists of `Tally`'s.

[1] From Richard: Toujours basé sur la normale?

ObservationListener

Represents an object that can listen to observations broadcast by statistical probes.

```
package umontreal.iro.lecuyer.stat;

public interface ObservationListener
```

```
   public void newObservation (StatProbe probe, double x);
```

Receives the new observation `x` broadcast by `probe`.

StatProbe

The objects of this class are *statistical probes* or *collectors*, which are elementary devices for collecting statistics. Each probe collects statistics on a given variable. The subclasses `Tally`, `TallyStore`, and `Accumulate` (from package `simevents`) implement two kinds of probes, for the case of successive observations $X_1, X_2, X_3, \ldots$, and for the case of a variable whose value evolves in time, respectively.

Each instance of `StatProbe` contains a list of `ObservationListener` that can listen to individual observations. When a probe is updated, i.e., receives a new statistical observation, it broadcasts this new data to all registered observers. The broadcasting of observations to registered observers can be turned ON or OFF at any time. It is initially OFF by default and should stay OFF when there are no registered observers, to avoid unnecessary overhead.

The data collection by the statistical probe itself can also be turned ON or OFF. By default, it is initially ON. We can turn it OFF, for example, if we want to use the statistical probe only to pass data to the observers, and do not need it to store any information.

In the simplest programs, collection is ON, broadcast is OFF, and the overall stats are accessed via the methods `min`, `max`, `sum`, `average`, ... of the collector.

```
package umontreal.iro.lecuyer.stat;

public abstract class StatProbe
```

Methods

`abstract public void init();`

Initializes the statistical collector.

`public void setName (String name)`

Sets the name of this statistical collector to `name`.

`public String getName()`

Returns the name associated with this probe, or `null` if no name was specified upon construction.

`public double min()`

Returns the smallest value taken by the variable since the last initialization of this probe. This returns `Double.POSITIVE_INFINITY` if the probe was not updated since the last initialization.

`public double max()`

Returns the largest value taken by the variable since the last initialization of this probe. This returns `Double.NEGATIVE_INFINITY` if the probe was not updated since the last initialization.

`public double sum()`
Returns the sum cumulated so far for this probe. The meaning of this sum depends on the subclass (e.g., `Tally` or `Accumulate`). This returns 0 if the probe was not updated since the last initialization.

`abstract public double average();`
Returns the average for this collector. This returns `Double.NaN` if the probe was not updated since the last initialization.

`abstract public String report();`
Returns a string containing a report for this statistical collector. The contents of this report depends on the statistical probe as well as on the parameters set by the user through probe-specific methods.

`abstract public String shortReport();`
Formats and returns a short, one-line report about this statistical probe. This line is composed of whitespace-separated fields which must correspond to the column names given by `shortReportHeader`(). This report should not contain any end-of-line character, and does not include the name of the probe. Its contents depends on the statistical probe as well as on the parameters set by the user through probe-specific methods.

`abstract public String shortReportHeader();`
Returns a string containing the name of the values returned in the report strings. The returned string must depend on the type of probe and on the reporting options only. It must not depend on the observations received by the probe. This can be used as header when printing several reports. For example,

```
System.out.println (probe1.shortReportHeader());
System.out.println (probe1.getName() + " " + probe1.shortReport());
System.out.println (probe2.getName() + " " + probe2.shortReport());
...
```

Alternatively, one can use `report (String,StatProbe[])` to get a report with aligned probe names.

`public static String report (String globalName, StatProbe[] probes)`
Formats short reports for each statistical probe in the array `probes` while aligning the probes' names. This method first formats the given global name. It then determines the maximum length ℓ of the names of probes in the given array. The first line of the report is composed of $\ell + 3$ spaces followed by the string returned by `shortReportHeader` called on the first probe in `probes`. Each remaining line corresponds to a statistical probe; it contains the probe's name followed by the contents returned by `shortReport`. Note that this method assumes that `probes` contains no `null` element.

```
public static String report (String globalName,
                             Iterable<? extends StatProbe> probes)
```

Equivalent to `report`, except that `probes` is an `Iterable` object instead of an array. Of course, the iterator returned by `probes` should enumerate the statistical probes to include in the report in a consistent and sensible order.

`public boolean isBroadcasting()`

Determines if this statistical probe is broadcasting observations to registered observers. The default is `false`.

`public void setBroadcasting (boolean b)`

Instructs the probe to turn its broadcasting ON or OFF. The default value is OFF.

Warning: To avoid useless overhead and performance degradation, broadcasting should never be turned ON when there are no registered observers.

`public boolean isCollecting()`

Determines if this statistical probe is collecting values. The default is `true`.

`public void setCollecting (boolean b)`

Turns ON or OFF the collection of statistical observations. The default value is ON. When statistical collection is turned OFF, observations added to the probe are passed to the registered observers if broadcasting is turned ON, but are not counted as observations by the probe itself.

`public void addObservationListener (ObservationListener l)`

Adds the observation listener `l` to the list of observers of this statistical probe.

`public void removeObservationListener (ObservationListener l)`

Removes the observation listener `l` from the list of observers of this statistical probe.

`public void clearObservationListeners()`

Removes all observation listeners from the list of observers of this statistical probe.

`public void notifyListeners (double x)`

Notifies the observation `x` to all registered observers if broadcasting is ON. Otherwise, does nothing.

Tally

A subclass of `StatProbe`. This type of statistical collector takes a sequence of real-valued observations $X_1, X_2, X_3, \ldots$ and can return the average, the variance, a confidence interval for the theoretical mean, etc. Each call to `add` provides a new observation. When the broadcasting to observers is activated, the method `add` will also pass this new information to its registered observers. This type of collector does not memorize the individual observations, but only their number, sum, sum of squares, maximum, and minimum. The subclass `TallyStore` offers a collector that memorizes the observations.

```
package umontreal.iro.lecuyer.stat;

public class Tally extends StatProbe implements Cloneable
```

Constructors

`public Tally()`

Constructs a new unnamed `Tally` statistical probe.

`public Tally (String name)`

Constructs a new `Tally` statistical probe with name `name`.

Methods

`public void add (double x)`

Gives a new observation `x` to the statistical collector. If broadcasting to observers is activated for this object, this method also transmits the new information to the registered observers by invoking the method `notifyListeners`.

`public int numberObs()`

Returns the number of observations given to this probe since its last initialization.

```
@Override
public double sum()
```

`public double average()`

Returns the average value of the observations since the last initialization.

`public double variance()`

Returns the sample variance of the observations since the last initialization. This returns `Double.NaN` if the tally contains less than two observations.

`public double standardDeviation()`

Returns the sample standard deviation of the observations since the last initialization. This returns `Double.NaN` if the tally contains less than two observations.

`public void confidenceIntervalNormal (double level, double[] centerAndRadius)`

Computes a confidence interval on the mean. Returns, in elements 0 and 1 of the array object `centerAndRadius[]`, the center and half-length (radius) of a confidence interval on the true mean of the random variable X, with confidence level `level`, assuming that the n observations given to this collector are independent and identically distributed (i.i.d.) copies of X, and that n is large enough for the central limit theorem to hold. This confidence interval is computed based on the statistic

$$Z = \frac{\bar{X}_n - \mu}{S_{n,x}/\sqrt{n}}$$

where n is the number of observations given to this collector since its last initialization, $\bar{X}_n =$ `average()` is the average of these observations, $S_{n,x} =$ `standardDeviation()` is the empirical standard deviation. Under the assumption that the observations of X are i.i.d. and n is large, Z has the standard normal distribution. The confidence interval given by this method is valid *only if* this assumption is approximately verified.

`public void confidenceIntervalStudent (double level, double[] centerAndRadius)`

Computes a confidence interval on the mean. Returns, in elements 0 and 1 of the array object `centerAndRadius[]`, the center and half-length (radius) of a confidence interval on the true mean of the random variable X, with confidence level `level`, assuming that the observations given to this collector are independent and identically distributed (i.i.d.) copies of X, and that X has the normal distribution. This confidence interval is computed based on the statistic

$$T = \frac{\bar{X}_n - \mu}{S_{n,x}/\sqrt{n}}$$

where n is the number of observations given to this collector since its last initialization, $\bar{X}_n =$ `average()` is the average of these observations, $S_{n,x} =$ `standardDeviation()` is the empirical standard deviation. Under the assumption that the observations of X are i.i.d. and normally distributed, T has the Student distribution with $n-1$ degrees of freedom. The confidence interval given by this method is valid *only if* this assumption is approximately verified, or if n is large enough so that $\bar{X}_n$ is approximately normally distributed.

`public String formatCINormal (double level, int d)`

Similar to `confidenceIntervalNormal`. Returns the confidence interval in a formatted string of the form

"`95% confidence interval for mean (normal):  (32.431, 32.487)`",

using d fractional decimal digits.

`public String formatCINormal (double level)`

Equivalent to `formatCINormal (level, 3)`.

`public String formatCIStudent (double level, int d)`

Similar to `confidenceIntervalStudent`. Returns the confidence interval in a formatted string of the form

"`95% confidence interval for mean (student): (32.431, 32.487)`",
using d fractional decimal digits.

`public String formatCIStudent (double level)`
Equivalent to `formatCIStudent (level, 3)`.

`public void confidenceIntervalVarianceChi2 (double level, double[] interval)`
Computes a confidence interval on the variance. Returns, in elements 0 and 1 of array `interval`, the left and right boundaries $[I_1, I_2]$ of a confidence interval on the true variance σ^2 of the random variable X, with confidence level `level`, assuming that the observations given to this collector are independent and identically distributed (i.i.d.) copies of X, and that X has the normal distribution. This confidence interval is computed based on the statistic $\chi^2_{n-1} = (n-1)S_n^2/\sigma^2$ where n is the number of observations given to this collector since its last initialization, and $S_n^2 =$ `variance()` is the empirical variance of these observations. Under the assumption that the observations of X are i.i.d. and normally distributed, χ^2_{n-1} has the chi-square distribution with $n-1$ degrees of freedom. Given the `level` $= 1-\alpha$, one has $P[\chi^2_{n-1} < x_1] = P[\chi^2_{n-1} > x_2] = \alpha/2$ and $[I_1, I_2] = [(n-1)S_n^2/x_2,\ (n-1)S_n^2/x_1]$.

`public String formatCIVarianceChi2 (double level, int d)`
Similar to `confidenceIntervalVarianceChi2`. Returns the confidence interval in a formatted string of the form
"`95.0% confidence interval for variance (chi2): ( 510.642, 519.673 )`",
using d fractional decimal digits.

`public String report()`
Returns a formatted string that contains a report on this probe.

`public String report (double level, int d)`
Returns a formatted string that contains a report on this probe with a confidence interval level `level` using d fractional decimal digits.

`public String shortReport()`
Formats and returns a short statistical report for this tally. The returned single-line report contains the minimum value, the maximum value, the average, and the standard deviation, in that order, separated by three spaces. If the number of observations is shown in the short report, a column containing the number of observations in this tally is added.

`public String reportAndCIStudent (double level, int d)`
Returns a formatted string that contains a report on this probe (as in `report`), followed by a confidence interval (as in `formatCIStudent`), using d fractional decimal digits.

`public String reportAndCIStudent (double level)`
Same as `reportAndCIStudent (level, 3)`.

`public double getConfidenceLevel()`
Returns the level of confidence for the intervals on the mean displayed in reports. The default confidence level is 0.95.

`public void setConfidenceLevel (double level)`

Sets the level of confidence for the intervals on the mean displayed in reports.

`public void setConfidenceIntervalNone()`

Indicates that no confidence interval needs to be printed in reports formatted by `report`, and `shortReport`. This restores the default behavior of the reporting system.

`public void setConfidenceIntervalNormal()`

Indicates that a confidence interval on the true mean, based on the central limit theorem, needs to be included in reports formatted by `report` and `shortReport`. The confidence interval is formatted using `formatCINormal`.

`public void setConfidenceIntervalStudent()`

Indicates that a confidence interval on the true mean, based on the normality assumption, needs to be included in reports formatted by `report` and `shortReport`. The confidence interval is formatted using `formatCIStudent`.

`public void setShowNumberObs (boolean showNumObs)`

Determines if the number of observations must be displayed in reports. By default, the number of observations is displayed.

`public Tally clone()`

Clones this object.

TallyStore

This class is a variant of `Tally` for which the individual observations are stored in a list implemented as a `DoubleArrayList`. The class `DoubleArrayList` is imported from the COLT library and provides an efficient way of storing and manipulating a list of real-valued numbers in a dynamic array. The `DoubleArrayList` object used to store the values can be either passed to the constructor or created by the constructor, and can be accessed via the `getDoubleArrayList` method.

The same counters as in `Tally` are maintained and are used by the inherited methods. One must access the list of observations to compute quantities not supported by the methods in `Tally`, and/or to use methods provided by the COLT package.

Never add or remove observations directly on the `DoubleArrayList` object, because this would put the counters of the `TallyStore` object in an inconsistent state.

There are two potential reasons for using a `TallyStore` object instead of directly using a `DoubleArrayList` object: (a) it can broadcast observations and (b) it maintains a few additional counters that may speed up some operations such as computing the average.

```
package umontreal.iro.lecuyer.stat;

public class TallyStore extends Tally
```

Constructors

`public TallyStore()`

Constructs a new `TallyStore` statistical probe.

`public TallyStore (String name)`

Constructs a new `TallyStore` statistical probe with name `name`.

`public TallyStore (int capacity)`

Constructs a new `TallyStore` statistical probe with given initial capacity `capacity` for its associated array.

`public TallyStore (String name, int capacity)`

Constructs a new `TallyStore` statistical probe with name `name` and given initial capacity `capacity` for its associated array.

`public TallyStore (DoubleArrayList a)`

Constructs a new `TallyStore` statistical probe with given associated array. This array must be empty.

Methods

`public double[] getArray()`

Returns the observations stored in this probe.

`public DoubleArrayList getDoubleArrayList()`

Returns the `DoubleArrayList` object that contains the observations for this probe. **WARNING:** In previous releases, this function was named `getArray`.

`public void quickSort()`

Sorts the elements of this probe using the `quicksort` from Colt.

`public double covariance (TallyStore t2)`

Returns the sample covariance of the observations contained in this tally, and the other tally `t2`. Both tallies must have the same number of observations. This returns `Double.NaN` if the tallies do not contain the same number of observations, or if they contain less than two observations.

`public TallyStore clone()`

Clones this object and the array which stores the observations.

`public String toString()`

Returns the observations stored in this object as a `String`.

TallyHistogram

This class is an extension of `Tally` which gives a more detailed view of the observations statistics. The individual observations are assumed to fall into different bins (boxes) of equal width on an interval. The total number of observations falling into the bins are kept in an array of counters. This is useful, for example, if one wish to build a histogram from the observations. One must access the array of bin counters to compute quantities not supported by the methods in `Tally`.

Never add or remove observations directly on the array of bin counters because this would put the `Tally` counters in an inconsistent state.

```
package umontreal.iro.lecuyer.stat;

public class TallyHistogram extends Tally
```

Constructors

`public TallyHistogram(double a, double b, int s)`

Constructs a `TallyHistogram` statistical probe. Divide the interval $[a, b]$ into s bins of equal width and initializes a counter to 0 for each bin. Whenever an observation falls into a bin, the bin counter is increased by 1. There are two extra bins (and counters) that count the number of observations x that fall outside the interval $[a, b]$: one for those $x < a$, and the other for those $x > b$.

`public TallyHistogram (String name, double a, double b, int s)`

Constructs a new `TallyHistogram` statistical probe with name `name`.

Methods

`public void init (double a, double b, int s)`

Initializes this object. Divide the interval $[a, b]$ into s bins of equal width and initializes all counters to 0.

`public void add (double x)`

Gives a new observation x to the statistical collectors. Increases by 1 the bin counter in which value x falls. Values that fall outside the interval $[a, b]$ are added in extra bin counter bin[0] if $x < a$, and in bin$[s + 1]$ if $x > b$.

`public int[] getCounters()`

Returns the bin counters. Each counter contains the number of observations that fell in its corresponding bin. The counters bin$[i]$, $i = 1, 2, \ldots, s$ contain the number of observations that fell in each subinterval of $[a, b]$. Values that fell outside the interval $[a, b]$ were added in extra bin counter bin[0] if $x < a$, and in bin$[s + 1]$ if $x > b$. There are thus $s + 2$ counters.

`public int getNumBins()`

Returns the number of bins s dividing the interval $[a, b]$. Does not count the two extra bins for the values of $x < a$ or $x > b$.

`public double getA()`

Returns the left boundary a of interval $[a, b]$.

`public double getB()`

Returns the right boundary b of interval $[a, b]$.

`public TallyHistogram clone()`

Clones this object and the array which stores the counters.

`public String toString()`

Returns the bin counters as a `String`.

Overview of package `stat.list`

Provides support for lists of statistical probes. Sometimes, a simulator computes several related performance measures such as the quality of service for different call types in a phone call center, the waiting times of different types of customers, the average number of pieces of different types a machine processes, etc. A list of statistical probes, in contrast with an ordinary array, can be resized. Since a list of statistical probes implements the Java `List` interface, one can iterate over each probe, e.g., to set reporting options. In addition to an ordinary list, a list of probes provides facilities to get a vector of averages, a vector of sums, and to create reports.

In the Java programming language, a list is usually constructed empty, and filled with items. Lists of statistical probes can be constructed this generic way, or created using factory methods that automatically construct the probes.

`ListOfStatProbes` is the base class for lists of statistical probes. It can hold a list of any `StatProbe` subclass, and provides the basic facilities to obtain an array of sums, an array of averages, etc. Subclasses provide probe-specific functionalities for adding vectors of observations, computing sample covariances, etc. `ListOfTallies` is used to contain `Tally` instances. A subclass, `ListOfTalliesWithCovariance`, is provided to add support for covariance computation without storing observations.

All classes in this package representing lists of probes support the observer design pattern similarly to the classes in package `stat`. A list of statistical probes maintains a list of registered `ArrayOfObservationListener` objects, and broadcasts information to all its registered observers when it receives a new vector of observations. Any object that implements the interface `ArrayOfObservationListener` can register as an observer.

ArrayOfObservationListener

Represents an object that can listen to observations broadcast by lists of statistical probes.

```
package umontreal.iro.lecuyer.stat.list;


public interface ArrayOfObservationListener

   public void newArrayOfObservations (ListOfStatProbes<?> listOfProbes,
                                       double[] x);
```

Receives the new array of observations `x` broadcast by the list of statistical probes `listOfProbes`.

ListOfStatProbes

Represents a list of statistical probes that can be managed simultaneously. Each element of this list is a `StatProbe` instance which can be obtained and manipulated.

When constructing a list of statistical probes, one specifies the concrete subclass of the `StatProbe` objects in it. One then creates an empty list of probes, and fills it with statistical probes. If the list is not intended to be modified, one can then use the `setUnmodifiable` to prevent any change in the contents of the list.

Each list of statistical probes can have a global name describing the contents of its elements, and local names associated with each individual probe. For example, a list of statistical probes for the waiting times can have the global name `Waiting times` while the individual probes have local names `type 1`, `type 2`, etc. These names are used for formatting reports.

Facilities are provided to fill arrays with sums, averages, etc. obtained from the individual statistical probes. Methods are also provided to manipulate the contents of the list. However, one should always call `init` immediately after adding or removing statistical probes in the list.

```
package umontreal.iro.lecuyer.stat.list;


public class ListOfStatProbes<E extends StatProbe>
                                 implements Cloneable, List<E>, RandomAccess
```

Constructors

`public ListOfStatProbes()`

Constructs an empty list of statistical probes.

`public ListOfStatProbes (String name)`

Constructs an empty list of statistical probes with name `name`.

Methods

`public String getName()`

Returns the global name of this list of statistical probes.

`public void setName (String name)`

Sets the global name of this list to `name`.

`public boolean isModifiable()`

Determines if this list of statistical probes is modifiable, i.e., if probes can be added or removed. Any list of statistical probes is modifiable by default, until one calls the `setUnmodifiable` method.

`public void setUnmodifiable()`

Forbids any future modification to this list of statistical probes. After this method is called, any attempt to modify the list results in an exception. Setting a list unmodifiable can be useful if some data structures are defined depending on the probes in the list.

`public void init()`

Initializes this list of statistical probes by calling `init` on each element.

`public void sum (double[] s)`

For each probe in the list, computes the sum by calling `sum`, and stores the results into the array `s`. This method throws an exception if the size of `s` mismatches with the size of the list.

`public void average (double[] a)`

For each probe in this list, computes the average by calling `average`, and stores the results into the array `a`. This method throws an exception if the size of `s` mismatches with the size of the list.

`public boolean isCollecting()`

Determines if this list of statistical probes is collecting values. Each probe of the list could or could not be collecting values. The default is `true`.

`public void setCollecting (boolean c)`

Sets the status of the statistical collecting mechanism to `c`. A `true` value turns statistical collecting ON, a `false` value turns it OFF.

`public boolean isBroadcasting()`

Determines if this list of statistical probes is broadcasting observations to registered observers. The default is `false`.

`public void setBroadcasting (boolean b)`

Sets the status of the observation broadcasting mechanism to `b`. A `true` value turns broadcasting ON, a `false` value turns it OFF.

`public void addArrayOfObservationListener (ArrayOfObservationListener l)`

Adds the observation listener `l` to the list of observers of this list of statistical probes.

`public void removeArrayOfObservationListener (ArrayOfObservationListener l)`

Removes the observation listener `l` from the list of observers of this list of statistical probes.

`public void clearArrayOfObservationListeners()`

Removes all observation listeners from the list of observers of this list of statistical probes.

`public void notifyListeners (double[] x)`

Notifies the observation `x` to all registered observers if broadcasting is ON. Otherwise, does nothing.

`public String report()`

Formats a report for each probe in the list of statistical probes. The returned string is constructed by using `StatProbe.report (getName(), this)`.

`public ListOfStatProbes<E> clone()`

Clones this object. This makes a shallow copy of this list, i.e., this does not clone all the probes in the list. The created clone is modifiable, even if the original list is unmodifiable.

ListOfTallies

Represents a list of tally statistical collectors. Each element of the list is an instance of `Tally`, and a vector of observations can be added with the `add` method. This class defines factory methods to fill a newly-constructed list with `Tally` or `TallyStore` instances.

```
package umontreal.iro.lecuyer.stat.list;

public class ListOfTallies<E extends Tally> extends ListOfStatProbes<E>
```

Constructors

`public ListOfTallies()`

Constructs a new empty list of tallies.

`public ListOfTallies (String name)`

Constructs a new empty list of tallies with name `name`.

Methods

`public static ListOfTallies<Tally> createWithTally (int size)`

This factory method constructs and returns a list of tallies with `size` instances of `Tally`.

`public static ListOfTallies<TallyStore> createWithTallyStore (int size)`

This factory method constructs and returns a list of tallies with `size` instances of `TallyStore`.

`public void add (double[] x)`

Adds the observation `x[i]` in tally `i` of this list, for `i = 0,..., size() - 1`. No observation is added if the value is `Double.NaN`, or if collecting is turned OFF. If broadcasting is ON, the given array is notified to all registered observers. The given array `x` not being stored by this object, it can be freely used and modified after the call to this method.

`public int numberObs()`

Assuming that each tally in this list contains the same number of observations, returns the number of observations in tally 0, or 0 if this list is empty.

`public boolean areAllNumberObsEqual()`

Tests that every tally in this list contains the same number of observations. This returns `true` if and only if all tallies have the same number of observations, or if this list is empty. If observations are always added using the `add` method from this class, and not `add` from `Tally`, this method always returns `true`.

`public void average (double[] r)`

Computes the average for each tally in this list, and stores the averages in the array `r`. If the tally `i` has no observation, the `Double.NaN` value is stored in the array, at index `i`.

`public void variance (double[] v)`

For each tally in this list, computes the sample variance, and stores the variances into the array `v`. If, for some tally `i`, there are not enough observations for estimating the variance, `Double.NaN` is stored in the array.

`public void standardDeviation (double[] std)`

For each tally in this list, computes the sample standard deviation, and stores the standard deviations into the array `std`. This is equivalent to calling `variance` and performing a square root on every element of the filled array.

`public double covariance (int i, int j)`

Returns the empirical covariance of the observations in tallies with indices `i` and `j`. If $x_1, \ldots, x_n$ represent the observations in tally `i` whereas $y_1, \ldots, y_n$ represent the observations in tally `j`, then the covariance is given by

$$S_{X,Y} = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \bar{X}_n)(y_k - \bar{Y}_n) = \frac{1}{n-1}\left(\sum_{k=1}^{n} x_k y_k - \frac{1}{n}\sum_{k=1}^{n} x_k \sum_{r=1}^{n} y_r\right).$$

This returns `Double.NaN` if the tallies do not contain the same number of observations, or if they contain less than two observations. This method throws an exception if the underlying tallies are not capable of storing observations, i.e. if the tallies are not TallyStores. The `ListOfTalliesWithCovariance` subclass provides an alternative implementation of this method which does not require the observations to be stored.

`public double correlation (int i, int j)`

Returns the empirical correlation between the observations in tallies with indices `i` and `j`. If the tally `i` contains a sample of the random variate X and the tally `j` contains a sample of Y, this corresponds to

$$\mathrm{Cor}(X,Y) = \mathrm{Cov}(X,Y)/\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}.$$

This method uses `covariance` to obtain an estimate of the covariance, and `variance` in class `Tally` to obtain the sample variances.

`public void covariance (DoubleMatrix2D c)`

Constructs and returns the sample covariance matrix for the tallies in this list. The given $d \times d$ matrix `c`, where $d =$ `size()`, is filled with the computed sample covariances. Element `c.get (i, j)` corresponds to the result of `covariance (i, j)`.

`public void correlation (DoubleMatrix2D c)`

Similar to `covariance` for computing the sample correlation matrix.

`public ListOfTallies<E> clone()`

Clones this object. This makes a shallow copy of this list, i.e., this does not clone all the tallies in the list. The created clone is modifiable, even if the original list is unmodifiable.

ListOfTalliesWithCovariance

Extends `ListOfTallies` to add support for the computation of the sample covariance between each pair of elements in a list, without storing all observations. This list of tallies contains internal structures to keep track of $\bar{X}_{n,i}$ for $i = 0, \ldots, d-1$, and $\sum_{k=0}^{n-1}(X_{i,k} - \bar{X}_{k,i})(X_{j,k} - \bar{X}_{k,j})/n$, for $i = 0, \ldots, d-2$ and $j = 1, \ldots, d-1$, with $j > i$. Here, $\bar{X}_{n,i}$ is the ith component of $\bar{\mathbf{X}}_n$, the average vector, and $\bar{X}_{0,i} = 0$ for $i = 0, \ldots, d-1$. The value $X_{i,k}$ corresponds to the ith component of the kth observation $\mathbf{X}_k$. These sums are updated every time a vector is added to this list, and are used to estimate the covariances.

Note: the size of the list of tallies must remain fixed because of the data structures used for computing sample covariances. As a result, the first call to `init` makes this list unmodifiable.

Note: for the sample covariance to be computed between a pair of tallies, the number of observations in each tally should be the same. It is therefore recommended to always add complete vectors of observations to this list. Moreover, one must use the `add` method in this class to add vectors of observations for the sums used for covariance estimation to be updated correctly. Failure to use this method, e.g., adding observations to each individual tally in the list, will result in an incorrect estimate of the covariances, unless the tallies in the list can store observations. For example, the following code, which adds the vector `v` in the list of tallies `list`, works correctly only if the list contains instances of `TallyStore`:

```
for (int i = 0; i < v.length; i++)
   list.get (i).add (v[i]);
```

But the following code is always correct:

```
list.add (v);
```

```
package umontreal.iro.lecuyer.stat.list;


public class ListOfTalliesWithCovariance<E extends Tally>
       extends ListOfTallies<E>
```

Constructors

`public ListOfTalliesWithCovariance()`

Creates an empty list of tallies with covariance support. One must fill the list with tallies, and call `init` before adding any observation.

`public ListOfTalliesWithCovariance (String name)`

Creates an empty list of tallies with covariance support and name `name`. One must fill the list with tallies, and call `init` before adding any observation.

Methods

`public static ListOfTalliesWithCovariance<Tally> createWithTally (int size)`

This factory method constructs and returns a list of tallies with `size` instances of `Tally`.

`public static ListOfTalliesWithCovariance<TallyStore> createWithTallyStore (int size)`

This factory method constructs and returns a list of tallies with `size` instances of `TallyStore`.

`public void add (double[] x)`

Adds a new vector of observations `x` to this list of tallies, and updates the internal data structures computing averages, and sums of products. One must use this method instead of adding observations to individual tallies to get a covariance estimate.

`public ListOfTalliesWithCovariance<E> clone()`

Clones this object. This clones the list of tallies and the data structures holding the sums of products but not the tallies comprising the list. The created clone is modifiable, even though the original list is unmodifiable.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, second edition, 1998.