SSJ User's Guide

Package `stat`

Tools for Collecting Statistics

Version: December 21, 2006

Contents

Overview

This package provides elementary tools for collecting statistics and computing confidence intervals. The base class `StatProbe` implements common methods needed for all probes. Its extension class `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. 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 in this package extend the `Observable` class of Java, which provides the basic support for 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. An `Observable` object in Java maintains a list of registered `Observer` objects, and broadcasts information to all its registered observers whenever appropriate. For the `StatProbe` class, this happens whenever a new observation is given to the probe. Any object that implements the interface `Observer` can register as an observer.

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 specific types 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.

`StatProbe` extends `Observable` from package `java.util`, allowing statistical probes to register `Observer` objects. 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.

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

public abstract class StatProbe extends Observable
```

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.

`public double max()`
Returns the largest value taken by the variable since the last initialization of this probe.

`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`).

`abstract public double average();`
Returns the average for this collector.

`abstract public String report();`

Returns a string containing a report for this statistical collector.

`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 void setCollecting (boolean b)`

Turns ON or OFF the statistical 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.

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 `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 will also transmit the new information to the registered observers by invoking the methods `setChanged` and `notifyObservers (new Double (x))` inherited from `Observable`.

`public int numberObs()`

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

`public double variance()`

Returns the variance of the observations since the last initialization.

`public double standardDeviation()`

Returns the standard deviation of the observations since the last initialization.

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

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 formatConfidenceIntervalStudent (double level, int d)`

Similar to `confidenceIntervalStudent`, but returns the confidence interval in a formatted string of the form "`95% confidence interval for mean: (32.431, 32.487)`", using d decimal digits of accuracy.

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

An alias for `formatConfidenceIntervalStudent`.

`public String formatCIStudent (double level)`

An alias for `formatConfidenceIntervalStudent`.

`public String report()`

Returns a formatted string that contains a report on this probe.

`public String reportAndConfidenceIntervalStudent (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 `formatConfidenceIntervalStudent`), using d decimal digits of accuracy.

`public String reportAndConfidenceIntervalStudent (double level)`

Same as `reportAndConfidenceIntervalStudent`(level, 3).

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

An alias for `reportAndConfidenceIntervalStudent`.

`public String reportAndCIStudent (double level)`

An alias for `reportAndConfidenceIntervalStudent`.

`public Object clone()`

Clone this object.

TallyStore

This class is a variant of `Tally`, but 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 `getArray` method.

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

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 is an `Observable` object 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()`
Construct a new `TallyStore` statistical probe.

`public TallyStore (int initialCapacity)`
Construct a new `TallyStore` statistical probe with given initial capacity for its associated array.

`public TallyStore (DoubleArrayList a)`
Construct a new `TallyStore` statistical probe with given associated array. This array must be empty.

Methods

`public DoubleArrayList getArray()`
Returns the `DoubleArrayList` object that contains the observations for this probe.

`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.

`public Object clone()`
Clone this object.

References

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