SSJ User's Guide

Package `markovchain`

Structures for the Simulation of Simple Markov chains

Version: February 6, 2006

This package offers predefined tools for the simulation of simple Markov chains with Monte Carlo and randomized quasi-Monte Carlo methods. It was developed in part to experiment with the array-RQMC method studied in [1, 2]. In this method, an array of n copies of a Markov chain are simulated in parallel, using RQMC at each step of the chain to induce negative dependence across the copies to reduce the variance of the global average of some performance measure.

Contents

MarkovChain 2
MarkovChainComparable 4
MarkovChainComparableStop 5
MarkovChainDouble 6
MarkovChainDoubleStop 8
ArrayOfComparableChains 9
ArrayOfComparableChainsStop 11
ArrayOfDoubleChains 12
ArrayOfDoubleChainsStop 13
LeftScrambledSobolSequence 14
LeftScrambledFaureSequence 15

MarkovChain

This class defines a generic Markov chain and provides basic tools to simulate it for a given number of steps and recover the performance measure. Chains can be cloned, so one can simulate many replicates in parallel. In a concrete subclass, it suffices to implement the three abstract methods `initialState`, `nextStep`, and `getPerformance` to get things going.

```
package umontreal.iro.lecuyer.markovchain;

public abstract class MarkovChain implements Cloneable
```

Abstract methods

`public abstract void initialState ()`

Sets the Markov chain to its (deterministic) initial state and initializes the collectors for the performance measure, ready to start a simulation.

`public abstract void nextStep (RandomStream stream)`

Simulates one more step of the chain, from its current state, using `stream` for the randomness.

`public abstract double getPerformance ()`

Returns the performance measure (total or average cost or gain) so far, for the current simulation run.

Other methods

`public Object clone () throws CloneNotSupportedException`

Returns a clone of the chain.

`public void simulSteps (int numSteps, RandomStream stream)`

Starts a new simulation run and simulates `numSteps` steps of the Markov chain, using the given `stream`.

```
public void simulRuns (int n, int numSteps, RandomStream stream,
                       Tally statRuns)
```

Performs `n` simulation runs of the chain, for `numSteps` steps per run, using the given `stream`. The statistics on the performance for the `n` runs are placed in `statRuns`.

```
public void simulRunsWithSubstreams (int n, int numSteps,
                           RandomStream stream, Tally statRuns)
```

Same as `simulRuns`, except that the stream is first reset to its initial seed and then reset to the next substream after each run.

```
public void simulRQMC (PointSet p, int m, int numSteps,
                       RandomStream noise, Tally statReps)
```

Performs `m` independent replicates of n simulation runs of the chain, using the points of the RQMC point set `p`, where n is the number of points in `p`. Each run goes for `numSteps` steps. For each replicate, a randomization is added to the point set using the stream `noise`, an iterator is created, and each run uses a different substream of this iterator (i.e., a different point). The statistics on the performance for the `m` independent replications are placed in `statReps`.

```
public String simulRunsFormat (int n, int numSteps, RandomStream stream,
                                Tally statRuns)
```

Same as `simulRuns` but also returns the results as a formatted string.

```
public String simulRunsWithSubstreamsFormat (int n, int numSteps,
          RandomStream stream, Tally statRuns)
```

Same as `simulRunsWithSubstreams` but also returns the results as a formatted string.

```
public String simulRQMCFormat (PointSet p, int m, int numSteps,
          RandomStream noise, Tally statReps)
```

Same as `simulRQMC` but also returns the results as a formatted string.

```
public String testImprovementRQMC (PointSet p, int m, int numSteps,
          RandomStream noise, double varMC, Tally statReps)
```

Similar to `simulRQMCFormat`, but also gives the variance improvement factor with respect to MC, assuming that `varMC` gives the variance per run for MC.

```
public String formatResults (Tally stat)
```

Returns a string that contains the mean, the variance, and a 90% confidence interval for `stat`.

```
public String formatResultsRQMC (Tally stat, int numPoints)
```

Returns a string that contains the mean, the variance multiplied by `numPoints`, and a 90% confidence interval for `stat`.

MarkovChainComparable

A subclass of Markov chain for which there is a total ordering between the states. A list of Markov chains can then be sorted according to their states at a given step, using `sort`.

```
package umontreal.iro.lecuyer.markovchain;

public abstract class MarkovChainComparable extends MarkovChain
        implements Comparable
```

Methods

`public abstract int compareTo (Object other)`

If the current state of this chain is less than that of `other`, returns -1; if it is equal, returns 0, otherwise returns 1.

MarkovChainComparableStop

A generic Markov chain which can stop at a (possibly random) stopping time. When the stopping time is reached, the `compareTo` method should conclude that the state of this chain is larger than that of any other chain that has not yet reached its stopping time (so the chain should be placed in a special state considered larger than any other state).

Each chain has a boolean indicator `stopped` that should be set to false before starting the simulation and to `true` (in the `nextStep` method) when the stopping time is reached.

```
package umontreal.iro.lecuyer.markovchain;


public abstract class MarkovChainComparableStop extends MarkovChainComparable
```

Methods

`public void simulSteps (int numSteps, RandomStream stream)`

Starts a new simulation run and stops whenever `numSteps` are done or the stopping time is reached (the first of these two events), using the given `stream`.

`public void simulSteps (RandomStream stream)`

Starts a new simulation run and simulates until the stopping time is reached, using the given `stream`. Same as `simulSteps (Integer.MAX_VALUE, stream)`

MarkovChainDouble

A special kind of Markov chain whose state space is a subset of the real numbers. Methods are provided to initialize the chain, advance by a number of steps, and get statistics on the performance. The method `nextStepDouble` makes it possible to simulate several copies of this chain in parallel without cloning and without maintaining the state of the chain in a local variable. The states can be maintained in an external vector and at each step, one passes the current state to the method `nextStepDouble`, which returns the next state. This is exploited in the implementation of `ArrayOfDoubleChains`.

The methods `initialState`, `nextStep`, `getPerformance` and `compareTo`, which are abstract in `MarkovChainComparable`, all have a default implementation that uses internal variables to memorize the *state*, *step number* and *total number of steps*. They update and use these variables by invoking the `initialStateDouble`, `nextStepDouble`, and `getPerformance(numSteps)` methods implemented in subclasses. This is used internally by other methods in `MarkovChain`. On the other hand, the abstract methods specified in the present class do not necessarily update these variables.

```
package umontreal.iro.lecuyer.markovchain;


public abstract class MarkovChainDouble extends MarkovChainComparable
```

Abstract Methods

```
public abstract double initialStateDouble ()
```

Same as `initialState()` but also returns the initial state.

```
public abstract double nextStepDouble (int step, double s,
                                       RandomStream stream)
```

Advances the chain by one step, from state `s` to the next state and using the current `stream`, assuming that we are at step `step`. Saves and returns the new state, but not necessarily the step number. The first call should be with `step = 0`.

```
public abstract void initStats ()
```

Initializes the statistics (collector) for this chain.

```
public abstract double getPerformance (int numSteps)
```

Returns the performance measure accumulated so far, which may depend on the number of steps.

Other Methods

`public double simulStepsDouble (int numSteps, RandomStream stream)`

After invoking `InitStats` starts a new simulation run, simulates `numSteps` steps of the Markov chain using the given `stream`, and returns the state. The `simulSteps` method does the same, but returns nothing.

MarkovChainDoubleStop

Defines a generic Markov chain over the real numbers, where the number of steps of the chain is a (possibly random) stopping time. When the stopping time is reached, the state of the chain is set to `Double.POSITIVE_INFINITY`.

```
package umontreal.iro.lecuyer.markovchain;


public abstract class MarkovChainDoubleStop extends MarkovChainDouble
```

Methods

`public double getPerformance ()`

Can be used when the performance does not depend on the number of steps.

`public double simulStepsDouble (int numSteps, RandomStream stream)`

Starts a new simulation run and stops whenever `numSteps` are done or the stopping time is reached (the first of these two events), using the given `stream`.

`public void simulSteps (int numSteps, RandomStream stream)`

Same as `simulStepsDouble (numSteps, stream)`, but returns nothing.

`public void simulSteps (RandomStream stream)`

Starts a new simulation run and simulates until the stopping time is reached, using the given `stream`. Same as `simulSteps (Integer.MAX_VALUE, stream)`.

```
public void simulRQMC (PointSet p, int m, int numSteps,
                       RandomStream noise, Tally statReps)
```

Same as `MarkovChain.simulRQMC`, except the simulation of any given chain stops whenever it reaches its stopping time.

ArrayOfComparableChains

Permits one to simulate an array of comparable Markov chains using the array-RQMC method of [2], where n copies of the chain are simulated in parallel, sorted in increasing order of their states at each step of the chain, and where the transitions of the n chains at any given step are determined from the n points of a d-dimensional RQMC point set, where d is the number of uniforms required at each step of the chain.

```
package umontreal.iro.lecuyer.markovchain;

public class ArrayOfComparableChains
```

Constructor

`public ArrayOfComparableChains (MarkovChainComparable baseChain)`

Creates an array of the comparable chain `baseChain`. The method `makeCopies` must be called to make the copies.

Methods

`public void makeCopies (int n)`

Creates `n` copies (clones) of the chain `baseChain` and put them in a list, ready for the array RQMC simulation.

```
public double simulArrayRQMC (PointSet p, int numSteps,
                              RandomStream noise)
```

Simulates the n copies of the chain, `numSteps` steps for each copy, using point set `p`, where n is the current number of copies (clones) of the chain and is *assumed* to equal the number of points in `p`. At each step, the points are randomized using `noise`. The dimension of `p` must be at least as large as the number of uniforms required to simulate one step of the chain. Returns the average performance per run.

```
public String simulReplicatesArrayRQMC (PointSet p, int m, int numSteps,
          RandomStream noise, Tally statReps)
```

Performs `m` independent replications of an array-RQMC simulation as in `simulArrayRQMC`. The statistics on the `m` corresponding averages are collected in `statReps` and the results are also returned in a string.

```
public String simulReplicatesArrayRQMC2 (PointSet p, int m, int numSteps,
          PointSet p2, RandomStream noise, Tally statReps)
```

Similar to `simulReplicatesArrayRQMC`, except that an iterator on a randomized version of `p2` is used in place of the `noise` stream to randomize `p` at the different steps of the chain. The dimension of `p2` must equal the number of uniforms required to randomize `p`, and its number of points must equal the number of steps in the chain. One point of `p2` is

used to randomize `p` at each step. The stream `noise` is used to randomize `p2` between the independent replications.

```
public String testImprovementArrayRQMC (PointSet p, int m, int numSteps,
          RandomStream noise, double varMC,
          Tally statReps)
```

Similar to `simulReplicatesArrayRQMC`, but also gives the variance improvement factor with respect to MC, assuming that `varMC` gives the variance per run for MC.

```
public String testImprovementArrayRQMC2 (PointSet p, int m, int numSteps,
          PointSet p2, RandomStream noise, double varMC,
          Tally statReps)
```

Similar to `simulReplicatesArrayRQMC2`, but also gives the variance improvement factor with respect to MC, assuming that `varMC` gives the variance per run for MC.

ArrayOfComparableChainsStop

Similar to `ArrayOfComparableChains`, except that each chain stops whenever it reaches its stopping time. Simulation and sorting (at each step) is continued only for the chains that have not yet reached their stopping time. This is pursued until either all chains have stopped or the maximum number of steps has been reached.

```
package umontreal.iro.lecuyer.markovchain;

public class ArrayOfComparableChainsStop extends ArrayOfComparableChains
```

Constructor

```
public ArrayOfComparableChainsStop (MarkovChainComparableStop baseChain)
```

Methods

```
public double simulArrayRQMC (PointSet p, int numSteps,
                              RandomStream noise)
```

The simulation of each chain stops whenever it reaches its stopping time or `numSteps` steps. To simulate each copy of the chain until the stopping time is reached, it suffices to set `numSteps = Integer.MAX_VALUE`.

ArrayOfDoubleChains

Similar to `ArrayOfComparableChains`, except that instead of working with n clones of a `MarkovChain`, we use a *single* `MarkovChainDouble` object for all the chains. The states of the chains are maintained in an array of real numbers (`double`) and the `nextStepDouble` method is used to advance each chain by one step. The performance measure is assumed to be additive over all steps of all copies of the chain. The sum is cumulated in a *single* accumulator for all copies of the chain, updated at each step of each copy.

```
package umontreal.iro.lecuyer.markovchain;

public class ArrayOfDoubleChains extends ArrayOfComparableChains
```

Constructor

`public ArrayOfDoubleChains (MarkovChainDouble baseChain)`

Creates a virtual array for the chain `baseChain`. The method `makeCopies` must be called to make the copies.

Methods

`public void makeCopies (int n)`

Creates the vector of states for `n` copies of the base chain.

ArrayOfDoubleChainsStop

Similar to `ArrayOfDoubleChains`, except that each chain stops whenever it reaches its stopping time. When simulating the array of chains, any chain that has reached its stopping time is put in the state ∞. Simulation and sorting (at each step) is continued only for the chains that have not yet reached their stopping time. This is pursued until either all chains have stopped or the maximum number of steps has been reached.

```
package umontreal.iro.lecuyer.markovchain;

public class ArrayOfDoubleChainsStop extends ArrayOfDoubleChains
```

Constructor

```
public ArrayOfDoubleChainsStop (MarkovChainDoubleStop baseChain)
```

Methods

```
public double simulArrayRQMC (PointSet p, int numSteps,
                              RandomStream noise)
```

Here, the simulation of a chain stops whenever it reaches its stopping time or `numSteps` steps. To simulate each copy of the chain until the stopping time is reached, it suffices to set `numSteps = Integer.MAX_VALUE`.

LeftScrambledSobolSequence

A Sobol sequence randomized by a left matrix scramble followed by a digital random shift.

```
package umontreal.iro.lecuyer.markovchain;

public class LeftScrambledSobolSequence extends SobolSequence
```

Constructor

`public LeftScrambledSobolSequence (int k, int w, int dim)`

Same as `SobolSequence(k, w, dim)`, except that its `randomize` method will do a left matrix scramble followed by a random digital shift.

LeftScrambledFaureSequence

A Faure sequence randomized by a left matrix scramble followed by a digital random shift.

```
package umontreal.iro.lecuyer.markovchain;

public class LeftScrambledFaureSequence extends FaureSequence
```

Constructor

`public LeftScrambledFaureSequence (int b, int k, int r, int w, int dim)`

Same as `FaureSequence(b, k, r, w, dim)`, except that its `randomize` method will do a left matrix scramble followed by a random digital shift.

References

[1] C. Lécot and B. Tuffin. Quasi-Monte Carlo methods for estimating transient measures of discrete time Markov chains. In H. Niederreiter, editor, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 329–343, Berlin, 2004. Springer-Verlag.

[2] P. L'Ecuyer, C. Lécot, and B. Tuffin. Randomized quasi-Monte Carlo simulation of Markov chains with an ordered state space. In H. Niederreiter and D. Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, 2005. To appear.