# marmoteCore:
# a Markov modeling platform

Alain Jean-Marie, Inria

Valuetools 2017
Venezia, 5 December 2017

# Contents

# 1 Motivation and Objectives

# Markov modeling

Markov chains on discrete state spaces are useful in many areas of science and engineering:

Operations Research : queueing theory, Markov decision processes, random graphs (*e.g.* PERT), ...

BioInformatics : random sequences, random trees, ...

BioMaths : random population models, epidemic models, ...

Physics : interacting particle models, magnetism, lasers, network science (Erdös-Rényi, pref. attachment, ...), ...

yet...

- scientists outside Stochastic Operations Research do not identify a "Markov" software library that suits their needs;
- even within SOR, Markov modelers continue to do ad-hoc development.

# Markov modeling tools

Wouldn't it be nice if tools for Stochastic Operations Research would reach the maturity of tools for Deterministic OR (Mathematical Programming, Linear Programming, (M)ILP, ...)?

# The `marmoteCore` roadmap

Guiding ideas:

- develop a software base focusing on Markov chains *per se*
- providing an API in several languages
- allowing the construction of complex models
- providing access to advanced solution methods

`marmoteCore` is the prototype of such a system.

It was realized thanks to the funding of the ANR (project MARMOTE ANR-12-MONU-0019) and the contributions of Issam Rabhi, Hlib Mykhailenko, Emmanuel Hyon.

(Courtesy Laurent Chusseau)

# Architecture

Target architecture in three layers:

- Bottom: solution methods
- Middle: marmoteCore API, construction of models, handling of data, algorithms, results
- Top: User models/applications, GUI & workflow management

<table>
<tr><td rowspan="2">Specific user models</td><td rowspan="2">High-level models</td><td rowspan="2" colspan="2">...</td><td colspan="3">GUI:</td></tr>
<tr><td>Kepler</td><td>DTK</td><td>...</td></tr>
<tr><td colspan="7">marmoteCore</td></tr>
<tr><td>Psi</td><td>Xborne</td><td>R</td><td>Scilab</td><td>...</td><td colspan="2"></td></tr>
</table>

Choice of an object-oriented language: `C++`

# 3 Inside marmoteCore

# Abstractions of `marmoteCore`

The programming model is based on just 4 main abstractions (implemented as *classes* in C++)

- Markov chains: `MarkovChain` and derived classes
- Transitions: `TransitionStructure` and derived classes
- State Spaces: `MarmoteSet`
- Probability distributions: `Distribution` and derived classes

# Markov modeling, in practice

Markov modeling usually consists in

- constructing Markov models:
  - specify state space
  - specify transitions, probabilities/rates
- analyzing them:
  - determine qualitive properties: structure, ergodicity, stability ...
  - compute metrics related with probabilities/distributions, frequencies, times, durations ...

# State spaces in marmoteCore

marmoteCore provides the MarmoteSet interface with some standard state space implementations:

- **MarmoteInterval** a simple 1-dimensional discrete interval, possibly infinite
- **MarmoteBox** cartesian products of intervals
- **BinarySequence** sequences of bits
- **Simplex** sequences of integers with given total sum
- **BinarySimplex** sequences of bits with given count of ones

# Marmote sets, ctd

Implementing new sets is easy: it just requires providing the minimal interface:

## Required methods for MarmoteSets

```
virtual long int Cardinal();
virtual int Index(int* buffer);
virtual void DecodeState(int index, int* buffer);
```

`Index()` converts a state (buffer) into an integer, `DecodeState()` does the converse.
Other functions may help state space exploration:

## Useful methods for MarmoteSets

```
void FirstState(int* buffer);
void NextState(int* buffer);
bool IsFirst(int* buffer);
```

# Transition Structures

The object that describes direct transitions between states and their weight.

## What a `TransitionStructure` $T$ should know how to do

- evaluate the $(i,j)$ entry $T_{ij}$
- (continuous time) evaluate the transition rate out of $i$
- identify the distribution of transitions from $i$
- evaluate the action on a measure
  $\pi' = \pi T$
- evaluate the action on a value
  $v' = Tv$

plus some other things...

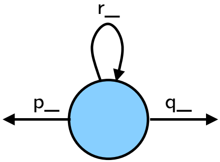
# Non-matrix implementation of transitions

Typical implementation of `TransitionStructure` will be a (sparse) matrix but...

Example of a transition structure on an infinite state space: the random walk.

Implementation of `getEntry()` for the 1-D random walk

```
getEntry(int i, int j)
if ( i == j-1 ) return p_;
else if ( i == j+1 ) return q_;
else if ( i == j ) return r_;
else return 0.0;
```




$\Longrightarrow$ possibility of making simulations.

# Creation of a transition structure with a state space

Objects of type MarmoteSet are useful to create the generator:

```
SparseMatrix* makeGenerator(AdHocStateSpace* sp, ...  )  {

  SparseMatrix* gen = new SparseMatrix( sp->Cardinal() );

  int stateBuffer[5];
  sp->FirstState(stateBuffer);
  int idx = 0;
  do {
    ...
    // destination state stored in nextBuffer
    nextBuffer[0] = MIN( stateBuffer[0] + 1, someBound );
    ...
    gen->addToEntry( idx, sp->Index(nextBuffer), someRate );
    gen->addToEntry( idx, idx, -someRate );
    ...
    sp->NextState(stateBuffer);
    idx++;
  } while (!sp->IsFirst(stateBuffer));
}
```

# The Markov Chain object

The `markovChain` object is just a container for its state space and transition structure.

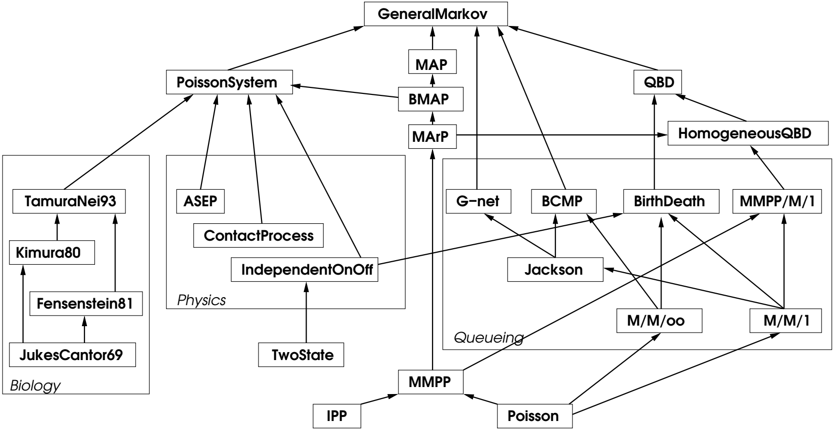
## Attributes of markovChain

```
timeType type_;
MarmoteSet* state_space_;
TransitionStructure* generator_;
DiscreteDistribution* init_distribution_;
```

What is more interesting is the possibility to organize families of `markovChain` objects in a hierarchy following the inclusion relation. Exploit the principle: The more structure (the fewer parameters), the deeper the analysis

# Markov Zoo, continuous time

A hierarchy of Markov models $\Longleftrightarrow$ C++ classes



# Available Solution Methods for `markovChain`

- Structural analysis
- Monte Carlo Simulation (forward)
- Exact sampling from the stationary distribution (backwards)
- Computation of the stationary distribution (various methods)
- Computation of transient distributions
- Hitting times (distribution, average)

# Exploiting the hierarchy/structure

Reimplementation with direct solution methods for specific chains:

- `Homogeneous1DRandomWalk`

```
DiscreteDistribution* TransientDistribution(int t, int nMax);
GeometricDistribution* StationaryDistribution();
SimulationResult* SimulateChain(long int tMax, ...);
```

- `Felsenstein81`

```
DiscreteDistribution* TransientDistribution(double);
DiscreteDistribution* StationaryDistribution();
Distribution* HittingTime(int iState, bool *hitSetIndicator);
double* AverageHittingTime(bool *hitSetIndicator);
SimulationResult* SimulateChain(double tMax, ...);
```

# 4
# Using 3rd party tools

# Interface with R tools

Just one example of interaction with external tools.

Interfacing with R is possible with the Rcpp C++ library.
In current marmoteCore

- Structural analysis, computation of stationary distribution
  → interface with R's package markovchain (maintainer: G.A. Spedicato)
- Sampling from probability distributions
  → for PoissonDistribution
- Computation of transient distributions
  → interface with R code (L. Cerdà-Alabern, Valuetools 2013)

# 5 Conclusion

# As a conclusion

A large todo list

- interfaces with R, `scilab`, projected: Python
- addition of solution methods
- more interface formats
- more models in the hierarchy, *e.g.* QBDs
- application packages: Markov Decision Processes
- ...

but already operational.

An open development

- in need of users/testers
- in need of contributors

`http://marmotecore.gforge.inria.fr`