MarmoteCore : créer et manipuler des chaînes de Markov en C++.

Emmanuel Hyon

Ecole d'été du GDR R.O.
6 juillet 2016

Courtesy Laurent Chusseau

Outline

Le projet Marmote

Projet ANR 2013-2017 (en cours)

MARkovian MOdeling Tools and Environments. (MARMOTE)

Objectifs

- Developper logiciel pour les problèmes Markoviens
  - qui permette à un scientifique d'avoir à sa disposition un environnement de modélisation.
  - qui puisse mettre à disposition de tout un chacun un ensemble d'algorithmes et de techniques avancées de manipulations.
- ouvert et qui puisse accepter les contributions ;
- orienté composants ;
- une possibilité de manipuler des modèles formels ;

La suite Marmote

Une suite qui rassemble différents logiciels

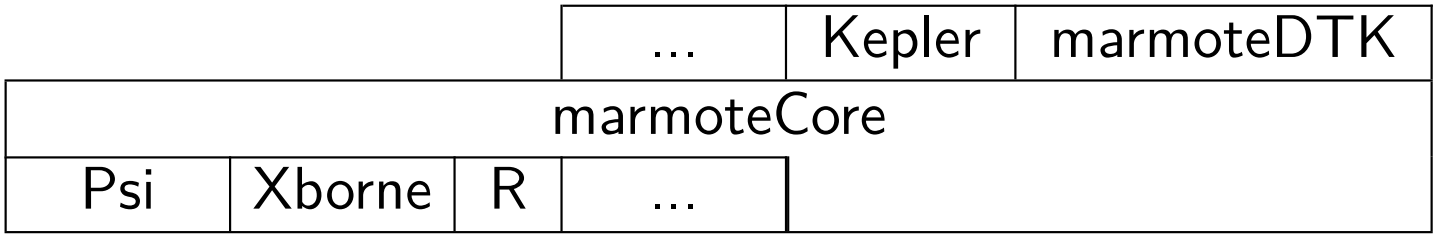
- Un noyau : `marmotecore`
- Un ensemble d'applis illustratives : `marmoteapplis`
- Un environnement basique en ligne `MarmoteOnline`
- Une suite de logiciels associés :
  - Psi3 : Simulation, Simulation Parfaite, (INRIA/POLARIS Grenoble) ;
    `http://psi.gforge.inria.fr/dokuwiki/doku.php`
  - XBORNE : Solutions qui fournit des bornes pour les modèles reliés aux chaînes de Makov (CM), (UVSQ/DAVID) ;

Concurrent GreatSPN, Tangram, ERS...

MarmoteCore

`marmotecore` est une bibliothèque en `C++` de fonctions et d'objets,

- qui s'intègrent facilement dans des codes C++,
- qui offrent également des méthodes natives de manipulation
- qui agissent comme une couche intermediaire entre des logiciels de workflow scientifique et des logiciels de calcul



Pour utiliser Marmote

Des liens (valables juillet 2016).

- Le site du projet ANR
  `https://wiki.inria.fr/MARMOTE/Accueil`.
- La forge du projet
  `http://marmotecore.gforge.inria.fr/`
  qui contient
  - une archive *testing*
  - une documentation.
- Un site tutoriel pour programmeur
  `http://www-sysdef.lip6.fr/~hyon/Marmote/home.php`
  qui contient
  - plusieurs exemples introductifs expliqués
  - un version du code *stable*.

Outline

les classes de bases

`marmotecore` est basé sur 4 classes principales et leurs dérivées

- Chaîne de Markov : `markovChain` et classes derivées,
- Transitions : `transitionStructure` et classes derivées : `sparseMatrix`, `multiDimHomTransition`, `eventMixture` ;
- Espace d'état : `marmoteSet` et classes derivées : `marmoteInterval`, `marmoteBox` ;
- Distributions de probabilités : `Distribution` et classes derivées :
  `dirac`, `Bernoulli`, `discrete`, `geometric`, `uniform`.

Un système de production suisse

On imagine un système de production qui peut être dans 3 états (“décrit” dans un documentaire de 3 minutes qui se trouve sur *youtube*)

etat 1 La vache actionne la chaîne d'assemblage

etat 2 Le chien apporte le lait dans la cuve

etat 3 La marmotte met le chocolat dans le papier d'alu

Evolution du système

Toutes les heures le système change d'état.

- Si la vache actionne la chaîne alors à l'heure suivante
  - La vache actionne la chaîne avec une probabilité : 0.25.
  - Le chien apporte le lait avec une probabilité : 0.5.
  - La marmotte met le chocolat avec une probabilité : 0.25,
- Si le chien apporte le lait alors à l'heure suivante
  - La vache actionne la chaîne avec une probabilité : 0.2.
  - Le chien apporte le lait avec une probabilité : 0.4.
  - La marmotte met le chocolat avec une probabilité : 0.4,
- Si la marmotte met le chocolat dans le papier d'alu alors à l'heure suivante
  - La vache actionne la chaîne avec une probabilité : 0.3
  - Le chien apporte le lait avec une probabilité : 0.4
  - La marmotte met le chocolat avec une probabilité : 0.3

Une chaîne de Markov

L'espace d'état est de dimension 3 : V, C, M.
L'évolution du système est décrite par une CM en temps discret.
De matrice de transition P

$$P = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.2 & 0.4 & 0.4 \\ 0.3 & 0.4 & 0.3 \end{bmatrix}$$

De loi initiale $\mu = (1, 0, 0)$ ou $\mu = (1/3, 1/3, 1/3)$.

Question

Comment savoir si j'aurais la chance de voir la marmotte mettre le chocolat dans le papier ?

Connaître la trajectoire de la chaine

Un code

Les codes des exemples de cette présentation sont disponibles
`http://www-sysdef.lip6.fr/~hyon/Marmote/compile.php`
Ici c'est l'exemple `MarmoteEx1.cpp`

La démarche (méthode de création de la chaine à la main)

1. Inclure les bibiothèques
2. Créer la CM
3. Créer la loi initiale
4. Créer et remplir la matrice de transition
5. Affecter la matrice et la loi initiale à la chaine
6. Lancer la méthode qui calcule la trajectoire

Le code en marmotecore

Dans http ://www-sysdef.lip6.fr/~hyon/Marmote/compile.php

Le code de MarmotEx1.cpp

```
#include "markovChain.h"
#include "Distribution/discreteDistribution.h"
#include "transitionStructure/sparseMatrix.h"

int main( int argc, char** argv ) {
   markovChain* c1= new markovChain(3,DISCRETE);
   discreteDistribution* initiale=
              new discreteDistribution(3,etat,probas);
   sparseMatrix *P= new sparseMatrix(3);
   P->addToEntry(0,0,0.25);
   ...
   c1->setInitDistribution(initiale);
   c1->setGenerator(P);
   simulationResult* simRes1 =
              c1->simulateChainDT( n, true, true, false );
   simRes1->writeTrajectory(stdout,"standard");
   delete c1; ....}
```

Fonctionnalités simples

Les exemples

`http://www-sysdef.lip6.fr/~hyon/Marmote/compile.php`

MarmotEx1.cpp	`simulateChainDT()` ex1 50 1 0 0
MarmotEx1bis.cpp	`transientDistributionDT(0,n)` ex1bis 50 1 0 0
MarmotEx2.cpp	`transientDistributionDT(0,n)` ex2 n p1 p2 p3 ... p8

```
//Calcul de la probabilite transiente à etape n
Distribution* staDis1 = c1->transientDistributionDT(0,n);
//Affichage chaine sur sortie standard
c1->write(stdout,false);
Affichage distribution
staDis1->write( stdout, -4);
```

Outline

Création de chaîne à partir d'un fichier

Deuxième manière de créer une chaine : par lecture d'un fichier.
`marmotecore` a des méthodes pour différents format de fichiers

Le code de `aMarmotEx1.cpp`

```
// creation de la chaine a partir du fichier rw1d
// format ERS, chaine concrete, pas de paramètres en plus
markovChain* c2 =
            new markovChain("Ers", NULL, 0,"rw1d", false);
// Ecriture de la chaine au format Ers dans le fichier
example1.mcl
c2->write("Ers","example1");
```

Fichier `rw1d.mcl`

```
discrete sparse
10
....
4 3 3.000000e-01 ....
stop
0
```

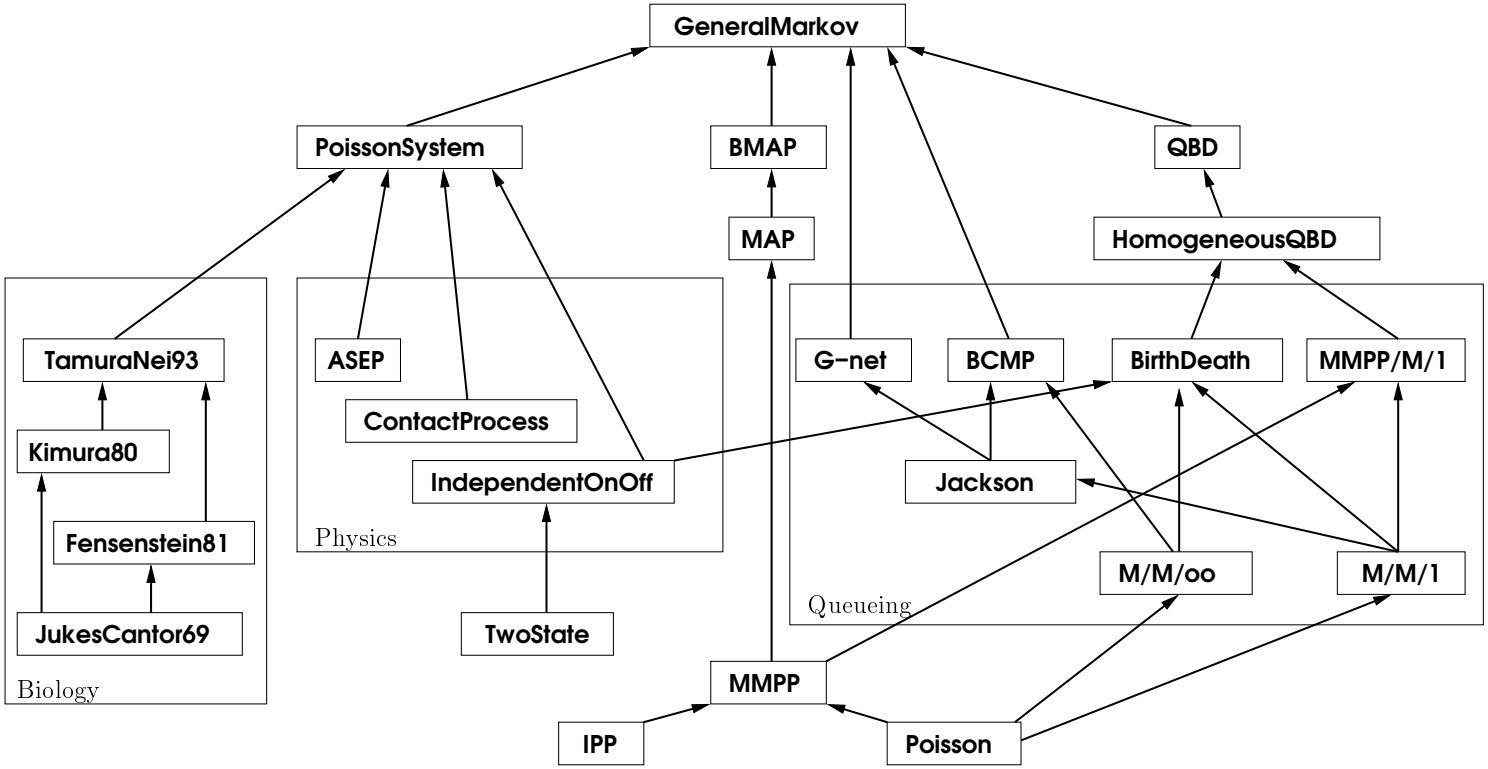
Un ensemble de méthode pour un même objetif

Le code de MarmotEx3.cpp

```
int main( int argc, char** argv ) {
   double etat[3]={0,1,2};
   uniformDistribution* initialeRandom=
                        new uniformDistribution(0,1);
   probas[0] = initialeRandom->sample(); ...
   somme= probas[0]+probas[1]+probas[2];
   probas[0] =probas[0]/somme; ...
   markovChain* c1= new markovChain(3,DISCRETE);
   discreteDistribution* initiale=
                        new discreteDistribution(3,etat,probas);
   sparseMatrix *P= new sparseMatrix(3);
   P->addToEntry(0,0,0.25); ...
   c1->setInitDistribution(initiale);
   c1->setGenerator(P);
   Distribution* staDis1 = c1->stationaryDistribution(false);
   Distribution* staDis2 =
c1->stationaryDistribution_power(100,0.00001,initiale,false);
   ....}
```

Zoo de Markov Continuous Time

Une chaine peut être spécialisée selon le modèle



Zoo de Markov et classe dérivée

On utilise cette décomposition et cette spécialisation en couplant avec un modèle objet.

Ainsi la classe `markovChain` a pour classes derivées :

- `felsenstein81`,
- `homogeneous1DRandomWalk`,
- `homogeneousMultiDRandomWalk`,
- `homogeneous1DBirthDeath` ;

Ce qui offre à la fois

- une généricité
- une spécialisation avec des méthodes (de représentation et de calcul) dédiées.

Application à la marche aléatoire

Marche aléatoire homogène 1D

- taille 10
- probabilité de sauter en $x-1$ vaut 0.3
- probabilité de sauter en $x+1$ vaut 0.4

Code de `iMarmotEx1.cpp`

```
// creating a random walk with size with 10 0.3 0.4
homogeneous1DRandomWalk* m1 =
          new homogeneous1DRandomWalk(10,0.4,0.3);
// computing stationary distribution
Distribution d1=m1->stationaryDistribution();
m1->write(stdout,false);
m1->write("Ers","example1");
```

Application à la marche aléatoire II

Avantage de la spécialisation

Représentation

- homogeneous1DRandomWalk utilise 3 paramètres
- MC avec `sparseMatrix` doit encoder au plus 3 entrées par état, dans notre exemple 28 données.
  Fichier `rw1d.mcl`.
- MC sans `sparseMatrix` tableau 10×10.
  Fichier `example1.mcl`.

Calcul

On peut utiliser des résultats formels de la littérature et éviter des calculs, *i.e.* on a Résolution exacte.

Ici, la distribution stationnaire est une distribution géométrique (tronquée dans le cas fini).

PSI

Le simulateur psi, dispo à
`http://psi.gforge.inria.fr/dokuwiki/doku.php`.

Permet de générer des trajectoires de CM de grande tailles par différentes méthodes (Monte Carlo, Perfect Sampling...)

- Psi représente les chaînes par des matrices creuses. Intégration en cours dans marmotecore.
- Psi3 représente les chaînes en utilisant une approche par événements.
- un langage évolué de modélisation

Markov Decision Process

En cours d'intégration dans marmotecore.
Framework qui permet de calculer les solutions optimales des MDP.

Processus de décision Markovien ou Chaine de Markov Contrôlée

- un modèle de programmation dynamique stochastique
- un ensemble de contrôle $a \in \{\mathcal{A}\}$ qui déterminent
  - un coût dans un état donné $c(x, a)$,
  - une probabilité $\mathbb{P}(y|x, a)$.
- on cherche une règle de décision qui donne la décision optimale en chaque état.
- une fois la règle calculée, le processus stochastique est une chaine de Markov.