
# Programmer avec MARMOTE

# 1 Créer une chaîne de Markov

# Création

Trois familles de possibilités :

1. lire le générateur (et l'espace d'états) dans un fichier
2. utiliser un classe prédéfinie
3. créer le générateur « à la main »

# Création/Read

## Code de création

```
markovChain* c4 = new markovChain( "Xborne", NULL, 0, "rw1d", true );
markovChain* c5 = new markovChain( "PSI", NULL, 0, "rw1d", true );
markovChain* c6 = new markovCHain( "Ers", NULL, 0, "rw1d", false );
```

Chaînes virtuelles :

$\Longrightarrow$ formats PSI1/MARCA, ERS, Xborne/C, Xborne/Rii

Chaînes concrètes :

$\Longrightarrow$ formats ERS, MARCA

# Interface read

```
/**
* @brief Constructor for Markov chains from files in various formats.
* In the abstract form: just stores the name(s) of the files that
* define the mode. In the non-abstract (concrete) form: the chain is
* instantiated in the memory with a concrete transition structure.
* @param format the format or language in which the model is specified
* @param param[] is the list of parameters
* @param nbreParam the size of param
* @param modelName name of the model, usually the prefix of various
files
* @param isAbstract specifies if the chain is abstract or not
* @return Markov Chain
*/
markovChain(string format, string param[], int nbreParam,
string modelName, bool isAbstract);
```

# Création/Use

Utilisation directe d'une des classes de CM pré-programmées

- felsenstein81
- homogeneous1DRandomWalk
- homogeneousMultiDRandomWalk
- homogeneous1DBirthDeath
- ...

# Création/Make

Typiquement en deux étapes :

1. créer un objet `transitionStructure`
2. créer la chaîne de Markov à partir de cet objet

## Code de création

```
sparseMatrix* gen = makeGenerator( sp, N, E1, E2, M, nu );
markovChain* myMC = new markovChain( gen );
```

# Création avec espace d'états

Les objets de type Set sont utiles pour fabriquer le générateur :

```
sparseMatrix* makeGenerator(layeredStateSpace* 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), somRate );
    gen->addToEntry( idx, idx, -somRate );
    ...

    sp->nextState( stateBuffer );
    idx++;
  } while (!sp->isZero(stateBuffer));
}
```

# 2
# Calculer sur une chaîne de Markov

# Méthodes de calcul disponibles pour `MarkovChain`

## Simulation Monte Carlo (forward)

```
virtual simulationResult* simulateChain(double, bool, bool, bool, bool);
virtual simulationResult* simulateChainDT(int, bool, bool, bool);
virtual simulationResult* simulateChainCT(double, bool, bool, bool,
bool);
 simulationResult* simulatePSI(int, bool, bool, bool);
```

## Calcul de distribution stationnaire

```
virtual Distribution* stationaryDistribution(bool);
virtual Distribution* stationaryDistributionCT(bool);
virtual Distribution* stationaryDistributionDT(bool);
Distribution* stationaryDistributionGhtLD();
Distribution* stationaryDistributionSOR();
```

## Échantillonage de la distribution stationnaire

```
simulationResult * stationaryDistributionSample (int nbSamples)
```

# Ctd.

## Temps d'atteinte

```
Distribution * hittingTimeDistribution (int iState, bool
*hitSetIndicator);
int * simulateHittingTime (int iState, bool *hittingSet, int nbSamples,
int tMax);
double * averageHittingTime (bool *hitSetIndicator);
double * averageHittingTimeDT (bool *hitSetIndicator);
double * averageHittingTimeDT_iterative (bool *hitSetIndicator);
```