État des lieux WP3 MARMOTE

1
Les objets

Objets

Les classes ou implémentées

- MarkovChain et descendants
- Distribution et descendants
- `transitionStructure`

Celles qu'il va falloir implémenter

- ensemble d'états (`stateSpace` ou `set`)

L'objet Distribution

Attributs

```
virtual double mean() = 0;
virtual double rate() = 0; // inverse of the mean
virtual double moment(int n) = 0;
double variance();
virtual double laplace(double s) = 0;
virtual double dLaplace(double s) = 0;
virtual double cdf(double x) = 0;
double ccdf(double x)  return 1.0 - cdf(x);;
virtual bool hasMoment(int n) = 0;
virtual Distribution* rescale(double factor) = 0;
virtual Distribution* copy() = 0;
virtual double sample() = 0;
virtual void iidSample(int n, double* s) = 0;
virtual double distanceL1(Distribution*);
virtual bool hasProperty(std::string);
```

Distributions

Distributions implémentées :

- `Distribution/BernoulliDistribution`
- `Distribution/ExponentialDistribution`
- `Distribution/UniformDiscreteDistribution`
- `Distribution/DiracDistribution`
- `Distribution/GeometricDistribution`
- `Distribution/UniformDistribution`
- `Distribution/DiscreteDistribution`

Transition Structures

Methods

```
timeType getType()  return _type; ;
virtual double getEntry(int,int) = 0;
virtual int getNbElts(int) = 0;
virtual int getCol(int,int) = 0;
virtual double getEntryByCol(int,int) = 0;
virtual DiscreteDistribution* getTransDistrib(int) = 0;
// transitions from some state and their probas
bool readEntry(FILE*);
virtual double rowSum(int) = 0;
virtual transitionStructure* uniformize() = 0;
virtual void evaluateMeasure(double*,double*);
virtual void
evaluateMeasure(DiscreteDistribution*,DiscreteDistribution*);
virtual void evaluateValue(double*,double*) = 0;
```

Transition structures

Transition structures implémentée :

- `transitionStructure/sparseMatrix`

En projet :

- transitionStructure/birthDeath
- transitionStructure/matrix

Simulation Result

Attributes

```
timeType _type;
int _stateSpaceSize;
int _trajectorySize;
bool _hasDistrib;
bool _hasTrajectory;
DiscreteDistribution* _distrib;
double* _dates;
double* _increments;
int* _states;
```

Methods

```
simulationResult(int size, timeType t, bool stats);
simulationResult(string format, string modelName, bool
stats);
```

2
Les chaînes de Markov

L'objet chaîne de Markov

Principes :

- Chaînes "concrètes"
  - lecture/création à partir de fichiers aux formats Ers, Marca
    $\Longrightarrow$ en cours : HBF, Xborne transition spec, PSI spec
  - création à partir d'une transitionStructure (e.g. Matrice, mais pas que)
  - sauvegarde format Ers,
    $\Longrightarrow$ en cours : Marca, HBF
- Chaînes "virtuelles" spécifiées par
  - format
  - nom de modèle
  - liste (optionnelle) d'extensions ou de noms de fichiers

  Évaluation "paresseuse"

Attributs

Attributs de MarkovChain

```
timeType _type;
int _stateSpaceSize;
transitionStructure* _generator;
DiscreteDistribution* _initDistribution;

// fields related to abstractness and file names
bool _isAbstract;
int _abstractNbre;
string* _abstract;
string _format;
string _modelName;
```

Les chaînes de Markov implémentées

Hiérarchie de Chaînes de Markov

- MarkovChain (DTMC et CTMC)
- Homogeneous1DRandomWalk
- Homogeneous1DBirthDeath
- Felsenstein81

Méthodes de résolution 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();
```

Utilities

```
virtual MarkovChain* copy();
```

Détails d'implémentation

Pour Xborne : inclusion du code C (non C++-ifié) avec petites modifications de structure :

- Méthode SOR dans `.../XBORNE/SOR/SOR.[ch]`
  $\Longrightarrow$ ajout des méthodes `readModel()` `doSor()` `mainSOR()`

```
void readModel(char* modelName, elmtvector *arc, indexvector *debut,
indexvector *taille, int *size)
double* doSOR(elmtvector arc, indexvector debut, indexvector taille, int
NSommets)
void mainSOR(char* modelName, double **pi, int *NSommets, bool
writeResult)
```

- Méthode gthLD dans `.../XBORNE/gthLD/gthLD.c`

```
void readModel(char* modelName, char* suffix, long double **P, int
*NSommets)
long double* doGTH(long double **P, int NSommets)
void mainGthLD(char * modelName, char * suffix, bool write)
```

Détails, suite

Encapsulation des applications PSI-Matrice : appel à `psi_alias` puis `psi_traj`

Code appel à Psi

```
// call to psi_traj
snprintf(cmd, 1024, "psi_alias -i ./%s.marca -o ./%s",
_abstract[0].c_str(), _abstract[0].c_str());
system(cmd);
snprintf(cmd, 1024, "psi_traj -i ./%s.simu -o ./%s
-s %d -l %d", _abstract[0].c_str(), _abstract[0].c_str(),
(int)rint(_initDistribution->sample()), tMax);
system(cmd);
```

To be done : call to psi_mcmc

Autres

Méthodes de résolution directes pour chaînes spécifiques :

- `Homogeneous1DRandomWalk`

```
DiscreteDistribution* transientDistribution(int t, int nMax) ;
GeometricDistribution* stationaryDistribution() ;
DiscreteDistribution* stationaryDistribution(int nMax) ;
simulationResult* simulateChain(long int tMax, bool stat, bool traj, bool
trace) ;
```

- `Felsenstein81`

```
DiscreteDistribution* transientDistribution(double) ;
DiscreteDistribution* stationaryDistribution() ;
simulationResult* simulateChain(double, bool stat, bool traj, bool incr,
bool trace) ;
```