

The markovchain Package: A Package for Easily Handling Discrete Markov Chains in R

Giorgio Alfredo Spedicato, Tae Seung Kang, Sai Bhargav Yalamanchi, Deepak Yadav, Ignacio Córdón

Abstract

The **markovchain** package aims to fill a gap within the R framework providing S4 classes and methods for easily handling discrete time Markov chains, homogeneous and simple inhomogeneous ones as well as continuous time Markov chains. The S4 classes for handling and analysing discrete and continuous time Markov chains are presented, as well as functions and method for performing probabilistic and statistical analysis. Finally, some examples in which the package's functions are applied to Economics, Finance and Natural Sciences topics are shown.

Keywords: discrete time Markov chains, continuous time Markov chains, transition matrices, communicating classes, periodicity, first passage time, stationary distributions.

1. Introduction

Markov chains represent a class of stochastic processes of great interest for the wide spectrum of practical applications. In particular, discrete time Markov chains (DTMC) permit to model the transition probabilities between discrete states by the aid of matrices. Various R packages deal with models that are based on Markov chains:

- **msm** ([Jackson 2011](#)) handles Multi-State Models for panel data.
- **mcmcR** ([Geyer and Johnson 2013](#)) implements Monte Carlo Markov Chain approach.
- **hmm** ([Himmelman and www.linhi.com 2010](#)) fits hidden Markov models with covariates.
- **mstate** fits 'Multi-State Models based on Markov chains for survival analysis ([de Wreede, Fiocco, and Putter 2011](#)).

Nevertheless, the R statistical environment ([R Core Team 2013](#)) seems to lack a simple package that coherently defines S4 classes for discrete Markov chains and allows to perform probabilistic analysis, statistical inference and applications. For the sake of completeness, **markovchain** is the second package specifically dedicated to DTMC analysis, being **DTMCPack** ([Nicholson 2013](#)) the first one. Notwithstanding, **markovchain** package ([Spedicato 2017](#)) aims to offer more flexibility in handling DTMC than other existing solutions, providing S4 classes for both homogeneous and semi-homogeneous Markov chains as well as methods suited to perform statistical and probabilistic analysis.

The **markovchain** package depends on the following R packages: **expm** ([Goulet, Dutang, Maechler, Firth, Shapira, Stadelmann, and expm-developers@lists.R-forge.R-project.org 2013](#))

to perform efficient matrices powers; **igraph** (Csardi and Nepusz 2006) to perform pretty plotting of **markovchain** objects and **matlab** (Roebuck 2011), that contains functions for matrix management and calculations that emulate those within MATLAB environment. Moreover, other scientific softwares provide functions specifically designed to analyze DTMC, as Mathematica 9 (Wolfram Research 2013a).

The paper is structured as follows: Section 2 briefly reviews mathematics and definitions regarding DTMC, Section 3 discusses how to handle and manage Markov chain objects within the package, Section 4 and Section 5 show how to perform probabilistic and statistical modelling, while Section 6 presents some applied examples from various fields analyzed by means of the **markovchain** package.

2. Review of core mathematical concepts

2.1. General Definitions

A DTMC is a sequence of random variables $X_1, X_2, \dots, X_n, \dots$ characterized by the Markov property (also known as memoryless property, see Equation 1). The Markov property states that the distribution of the forthcoming state X_{n+1} depends only on the current state X_n and doesn't depend on the previous ones $X_{n-1}, X_{n-2}, \dots, X_1$.

$$Pr(X_{n+1} = x_{n+1} | X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = Pr(X_{n+1} = x_{n+1} | X_n = x_n). \quad (1)$$

The set of possible states $S = \{s_1, s_2, \dots, s_r\}$ of X_n can be finite or countable and it is named the state space of the chain.

The chain moves from one state to another (this change is named either 'transition' or 'step') and the probability p_{ij} to move from state s_i to state s_j in one step is named transition probability:

$$p_{ij} = Pr(X_1 = s_j | X_0 = s_i). \quad (2)$$

The probability of moving from state i to j in n steps is denoted by $p_{ij}^{(n)} = Pr(X_n = s_j | X_0 = s_i)$.

A DTMC is called time-homogeneous if the property shown in Equation 3 holds. Time homogeneity implies no change in the underlying transition probabilities as time goes on.

$$Pr(X_{n+1} = s_j | X_n = s_i) = Pr(X_n = s_j | X_{n-1} = s_i). \quad (3)$$

If the Markov chain is time-homogeneous, then $p_{ij} = Pr(X_{k+1} = s_j | X_k = s_i)$ and $p_{ij}^{(n)} = Pr(X_{n+k} = s_j | X_k = s_i)$, where $k > 0$.

The probability distribution of transitions from one state to another can be represented into a transition matrix $P = (p_{ij})_{i,j}$, where each element of position (i, j) represents the transition probability p_{ij} . E.g., if $r = 3$ the transition matrix P is shown in Equation 4

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}. \quad (4)$$

The distribution over the states can be written in the form of a stochastic row vector x (the term stochastic means that $\sum_i x_i = 1, x_i \geq 0$): e.g., if the current state of x is s_2 , $x = (0 \ 1 \ 0)$. As a consequence, the relation between $x^{(1)}$ and $x^{(0)}$ is $x^{(1)} = x^{(0)}P$ and, recursively, we get $x^{(2)} = x^{(0)}P^2$ and $x^{(n)} = x^{(0)}P^n$, $n > 0$.

DTMC are explained in most theory books on stochastic processes, see [Brémaud \(1999\)](#) and [Dobrow \(2016\)](#) for example. Valuable references online available are: [Konstantopoulos \(2009\)](#), [Snell \(1999\)](#) and [Bard \(2000\)](#).

2.2. Properties and classification of states

A state s_j is said accessible from state s_i (written $s_i \rightarrow s_j$) if a system starting in state s_i has a positive probability to reach the state s_j at a certain point, i.e., $\exists n > 0 : p_{ij}^n > 0$. If both $s_i \rightarrow s_j$ and $s_j \rightarrow s_i$, then s_i and s_j are said to communicate.

A communicating class is defined to be a set of states that communicate. A DTMC can be composed by one or more communicating classes. If the DTMC is composed by only one communicating class (i.e., if all states in the chain communicate), then it is said irreducible. A communicating class is said to be closed if no states outside of the class can be reached from any state inside it.

If $p_{ii} = 1$, s_i is defined as absorbing state: an absorbing state corresponds to a closed communicating class composed by one state only.

The canonical form of a DTMC transition matrix is a matrix having a block form, where the closed communicating classes are shown at the beginning of the diagonal matrix.

A state s_i has period k_i if any return to state s_i must occur in multiples of k_i steps, that is $k_i = \gcd\{n : Pr(X_n = s_i | X_0 = s_i) > 0\}$, where $\gcd$ is the greatest common divisor. If $k_i = 1$ the state s_i is said to be aperiodic, else if $k_i > 1$ the state s_i is periodic with period k_i . Loosely speaking, s_i is periodic if it can only return to itself after a fixed number of transitions $k_i > 1$ (or multiple of k_i), else it is aperiodic.

If states s_i and s_j belong to the same communicating class, then they have the same period k_i . As a consequence, each of the states of an irreducible DTMC share the same periodicity. This periodicity is also considered the DTMC periodicity. It is possible to classify states according to their periodicity. Let $T^{x \rightarrow x}$ is the number of periods to go back to state x knowing that the chain starts in x .

- A state x is recurrent if $P(T^{x \rightarrow x} < +\infty) = 1$ (equivalently $P(T^{x \rightarrow x} = +\infty) = 0$). In addition:
 1. A state x is null recurrent if in addition $E(T^{x \rightarrow x}) = +\infty$.
 2. A state x is positive recurrent if in addition $E(T^{x \rightarrow x}) < +\infty$.
 3. A state x is absorbing if in addition $P(T^{x \rightarrow x} = 1) = 1$.
- A state x is transient if $P(T^{x \rightarrow x} < +\infty) < 1$ (equivalently $P(T^{x \rightarrow x} = +\infty) > 0$).

It is possible to analyze the timing to reach a certain state. The first passage time (or hitting time) from state s_i to state s_j is the number T_{ij} of steps taken by the chain until it arrives for the first time to state s_j , given that $X_0 = s_i$. The probability distribution of T_{ij} is defined by Equation 5

$$h_{ij}^{(n)} = \Pr(T_{ij} = n) = \Pr(X_n = s_j, X_{n-1} \neq s_j, \dots, X_1 \neq s_j | X_0 = s_i) \quad (5)$$

and can be found recursively using Equation 6, given that $h_{ij}^{(n)} = p_{ij}$.

$$h_{ij}^{(n)} = \sum_{k \in S - \{s_j\}} p_{ik} h_{kj}^{(n-1)}. \quad (6)$$

A commonly used quantity related to h is its average value, i.e. the *mean first passage time* (also expected hitting time), namely $\bar{h}_{ij} = \sum_{n=1 \dots \infty} n h_{ij}^{(n)}$.

If in the definition of the first passage time we let $s_i = s_j$, we obtain the first recurrence time $T_i = \inf\{n \geq 1 : X_n = s_i | X_0 = s_i\}$. We could also ask ourselves which is the *mean recurrence time*, an average of the mean first recurrence times:

$$r_i = \sum_{k=1}^{\infty} k \cdot P(T_i = k)$$

Revisiting the definition of recurrence and transience: a state s_i is said to be recurrent if it is visited infinitely often, i.e., $\Pr(T_i < +\infty | X_0 = s_i) = 1$. On the opposite, s_i is called transient if there is a positive probability that the chain will never return to s_i , i.e., $\Pr(T_i = +\infty | X_0 = s_i) > 0$.

Given a time homogeneous Markov chain with transition matrix P , a stationary distribution z is a stochastic row vector such that $z = z \cdot P$, where $0 \leq z_j \leq 1 \forall j$ and $\sum_j z_j = 1$.

If a DTMC $\{X_n\}$ is irreducible and aperiodic, then it has a limit distribution and this distribution is stationary. As a consequence, if P is the $k \times k$ transition matrix of the chain and $z = (z_1, \dots, z_k)$ is the unique eigenvector of P such that $\sum_{i=1}^k z_i = 1$, then we get

$$\lim_{n \rightarrow \infty} P^n = Z, \quad (7)$$

where Z is the matrix having all rows equal to z . The stationary distribution of $\{X_n\}$ is represented by z .

A matrix A is called primitive if all of its entries are strictly positive, and we write it $A > 0$. If the transition matrix P for a DTMC has some primitive power, i.e. it exists $m > 0 : P^m > 0$, then the DTMC is said to be regular. In fact being regular is equivalent to being irreducible and aperiodic. All regular DTMCs are irreducible. The counterpart is not true.

Given two absorbing states s_A (source) and s_B (sink), the *committor probability* $q_j^{(AB)}$ is the probability that a process starting in state s_i is absorbed in state s_B (rather than s_A) (Noé, Schütte, Vanden-Eijnden, Reich, and Weigl 2009). It can be computed via

$$q_j^{(AB)} = \sum_{k \ni A, B} P_{jk} q_k^{(AB)} \quad \text{with} \quad q_A^{(AB)} = 0 \quad \text{and} \quad q_B^{(AB)} = 1 \quad (8)$$

Note we can also define the hitting probability from i to j as the probability of ever reaching the state j if our initial state is i :

$$h_{i,j} = \Pr(T_{ij} < \infty) = \sum_{n=0}^{\infty} h_{ij}^{(n)} \quad (9)$$

In a DTMC with finite set of states, we know that a transient state communicates at least with one recurrent state. If the chain starts in a transient element, once it hits a recurrent state, it is going to be caught in its recurrent state, and we cannot expect it would go back to the initial state. Given a transient state i we can define the *absorption probability* to the recurrent state j as the probability that the first recurrent state that the Markov chain visits (and therefore gets absorbed by its recurrent class) is j , f_i^*j . We can also define the *mean absorption time* as the mean number of steps the transient state i would take until it hits any recurrent state, b_i .

2.3. A short example

Consider the following numerical example. Suppose we have a DTMC with a set of 3 possible states $S = \{s_1, s_2, s_3\}$. Let the transition matrix be:

$$P = \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{bmatrix}. \quad (10)$$

In P , $p_{11} = 0.5$ is the probability that $X_1 = s_1$ given that we observed $X_0 = s_1$ is 0.5, and so on. It is easy to see that the chain is irreducible since all the states communicate (it is made by one communicating class only).

Suppose that the current state of the chain is $X_0 = s_2$, i.e., $x^{(0)} = (0\ 1\ 0)$, then the probability distribution of states after 1 and 2 steps can be computed as shown in Equations (11) and (12).

$$x^{(1)} = (0\ 1\ 0) \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{bmatrix} = (0.15\ 0.45\ 0.4). \quad (11)$$

$$x^{(n)} = x^{(n-1)}P \rightarrow (0.15\ 0.45\ 0.4) \begin{bmatrix} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{bmatrix} = (0.2425\ 0.3725\ 0.385). \quad (12)$$

If we were interested in the probability of being in the state s_3 in the second step, then $Pr(X_2 = s_3 | X_0 = s_2) = 0.385$.

3. The structure of the package

3.1. Creating markovchain objects

The package is loaded within the R command line as follows:

```
R> library("markovchain")
```

The `markovchain` and `markovchainList` S4 classes ([Chambers 2008](#)) are defined within the **markovchain** package as displayed:

```
Class "markovchain" [package "markovchain"]
```

Slots:

Name:	states	byrow	transitionMatrix	name
Class:	character	logical	matrix	character

```
Class "markovchainList" [package "markovchain"]
```

Slots:

Name:	markovchains	name
Class:	list	character

The first class has been designed to handle homogeneous Markov chain processes, while the latter (which is itself a list of `markovchain` objects) has been designed to handle semi-homogeneous Markov chains processes.

Any element of `markovchain` class is comprised by following slots:

1. **states**: a character vector, listing the states for which transition probabilities are defined.
2. **byrow**: a logical element, indicating whether transition probabilities are shown by row or by column.
3. **transitionMatrix**: the probabilities of the transition matrix.
4. **name**: optional character element to name the DTMC.

The `markovchainList` objects are defined by following slots:

1. **markovchains**: a list of `markovchain` objects.
2. **name**: optional character element to name the DTMC.

The `markovchain` objects can be created either in a long way, as the following code shows

```

R> weatherStates <- c("sunny", "cloudy", "rain")
R> byRow <- TRUE
R> weatherMatrix <- matrix(data = c(0.70, 0.2, 0.1,
+                                0.3, 0.4, 0.3,
+                                0.2, 0.45, 0.35), byrow = byRow, nrow = 3,
+                                dimnames = list(weatherStates, weatherStates))
R> mcWeather <- new("markovchain", states = weatherStates, byrow = byRow,
+                   transitionMatrix = weatherMatrix, name = "Weather")

```

or in a shorter way, displayed below

```

R> mcWeather <- new("markovchain", states = c("sunny", "cloudy", "rain"),
+                   transitionMatrix = matrix(data = c(0.70, 0.2, 0.1,
+                                0.3, 0.4, 0.3,
+                                0.2, 0.45, 0.35), byrow = byRow, nrow = 3),
+                   name = "Weather")

```

When `new("markovchain")` is called alone, a default Markov chain is created.

```
R> defaultMc <- new("markovchain")
```

The quicker way to create `markovchain` objects is made possible thanks to the implemented `initialize` S4 method that checks that:

- the `transitionMatrix`, either of class `matrix` or `Matrix`, to be a transition matrix, i.e., all entries to be probabilities and either all rows or all columns to sum up to one.
- the columns and rows names of `transitionMatrix` to be defined and to coincide with `states` vector slot.

The `markovchain` objects can be collected in a list within `markovchainList` S4 objects as following example shows.

```

R> mcList <- new("markovchainList", markovchains = list(mcWeather, defaultMc),
+               name = "A list of Markov chains")

```

3.2. Handling markovchain objects

Table 1 lists which methods handle and manipulate `markovchain` objects.

The examples that follow shows how operations on `markovchain` objects can be easily performed. For example, using the previously defined matrix we can find what is the probability distribution of expected weather states in two and seven days, given the actual state to be cloudy.

```

R> initialState <- c(0, 1, 0)
R> after2Days <- initialState * (mcWeather * mcWeather)
R> after7Days <- initialState * (mcWeather ^ 7)
R> after2Days

```

Method	Purpose
<code>*</code>	Direct multiplication for transition matrices.
<code>^</code>	Compute the power <code>markovchain</code> of a given one.
<code>[</code>	Direct access to the elements of the transition matrix.
<code>==</code>	Equality operator between two transition matrices.
<code>!=</code>	Inequality operator between two transition matrices.
<code>as</code>	Operator to convert <code>markovchain</code> objects into <code>data.frame</code> and <code>table</code> object.
<code>dim</code>	Dimension of the transition matrix.
<code>names</code>	Equal to <code>states</code> .
<code>names<-</code>	Change the <code>states</code> name.
<code>name</code>	Get the name of <code>markovchain</code> object.
<code>name<-</code>	Change the name of <code>markovchain</code> object.
<code>plot</code>	<code>plot</code> method for <code>markovchain</code> objects.
<code>print</code>	<code>print</code> method for <code>markovchain</code> objects.
<code>show</code>	<code>show</code> method for <code>markovchain</code> objects.
<code>sort</code>	<code>sort</code> method for <code>markovchain</code> objects, in terms of their states.
<code>states</code>	Name of the transition states.
<code>t</code>	Transposition operator (which switches <code>byrow</code> 'slot value and modifies the transition matrix coherently).

Table 1: **markovchain** methods for handling `markovchain` objects.

```
      sunny cloudy  rain
[1,]  0.39  0.355 0.255
```

```
R> round(after7Days, 3)
```

```
      sunny cloudy  rain
[1,] 0.462  0.319 0.219
```

A similar answer could have been obtained defining the vector of probabilities as a column vector. A column - defined probability matrix could be set up either creating a new matrix or transposing an existing `markovchain` object thanks to the `t` method.

```
R> initialState <- c(0, 1, 0)
R> after2Days <- (t(mcWeather) * t(mcWeather)) * initialState
R> after7Days <- (t(mcWeather) ^ 7) * initialState
R> after2Days
```

```
      [,1]
sunny  0.390
cloudy 0.355
rain   0.255
```

```
R> round(after7Days, 3)
```

```

      [,1]
sunny  0.462
cloudy 0.319
rain    0.219

```

The initial state vector previously shown can not necessarily be a probability vector, as the code that follows shows:

```

R> fvals<-function(mchain,initialstate,n) {
+   out<-data.frame()
+   names(initialstate)<-names(mchain)
+   for (i in 0:n)
+   {
+     iteration<-initialstate*mchain^(i)
+     out<-rbind(out,iteration)
+   }
+   out<-cbind(out, i=seq(0,n))
+   out<-out[,c(4,1:3)]
+   return(out)
+ }
R> fvals(mchain=mcWeather,initialstate=c(90,5,5),n=4)

```

```

      i    sunny    cloudy    rain
1 0 90.00000  5.00000  5.00000
2 1 65.50000 22.25000 12.25000
3 2 54.97500 27.51250 17.51250
4 3 50.23875 29.88063 19.88062
5 4 48.10744 30.94628 20.94628

```

Basic methods have been defined for `markovchain` objects to quickly get states and transition matrix dimension.

```

R> states(mcWeather)

[1] "sunny" "cloudy" "rain"

R> names(mcWeather)

[1] "sunny" "cloudy" "rain"

R> dim(mcWeather)

[1] 3

```

Methods are available to set and get the name of `markovchain` object.

```
R> name(mcWeather)
```

```
[1] "Weather"
```

```
R> name(mcWeather) <- "New Name"
```

```
R> name(mcWeather)
```

```
[1] "New Name"
```

Also it is possible to alphabetically sort the transition matrix:

```
R> markovchain:::sort(mcWeather)
```

```
New Name
```

```
A 3 - dimensional discrete Markov Chain defined by the following states:
```

```
cloudy, rain, sunny
```

```
The transition matrix (by rows) is defined as follows:
```

	cloudy	rain	sunny
cloudy	0.40	0.30	0.3
rain	0.45	0.35	0.2
sunny	0.20	0.10	0.7

A direct access to transition probabilities is provided both by `transitionProbability` method and `"["` method.

```
R> transitionProbability(mcWeather, "cloudy", "rain")
```

```
[1] 0.3
```

```
R> mcWeather[2,3]
```

```
[1] 0.3
```

The transition matrix of a `markovchain` object can be displayed using `print` or `show` methods (the latter being less verbose). Similarly, the underlying transition probability diagram can be plotted by the use of `plot` method (as shown in Figure 1) which is based on **igraph** package (Csardi and Nepusz 2006). `plot` method for `markovchain` objects is a wrapper of `plot.igraph` for `igraph` S4 objects defined within the **igraph** package. Additional parameters can be passed to `plot` function to control the network graph layout. There are also **diagram** and **DiagrammeR** ways available for plotting as shown in Figure 2. The `plot` function also uses `communicatingClasses` function to separate out states of different communicating classes. All states that belong to one class have same color.

```
R> print(mcWeather)
```

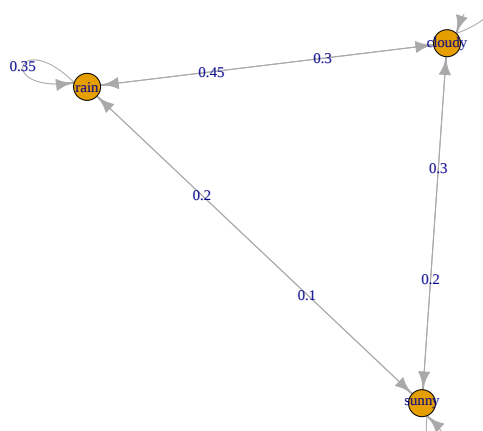


Figure 1: Weather example. Markov chain plot

	sunny	cloudy	rain
sunny	0.7	0.20	0.10
cloudy	0.3	0.40	0.30
rain	0.2	0.45	0.35

```
R> show(mcWeather)
```

New Name

A 3 - dimensional discrete Markov Chain defined by the following states:

sunny, cloudy, rain

The transition matrix (by rows) is defined as follows:

	sunny	cloudy	rain
sunny	0.7	0.20	0.10
cloudy	0.3	0.40	0.30
rain	0.2	0.45	0.35

Caricamento pacchetto: 'igraph'

I seguenti oggetti sono mascherati da 'package:stats':

decompose, spectrum

Il seguente oggetto è mascherato da 'package:base':

union

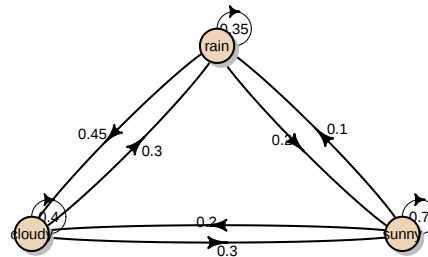


Figure 2: Weather example. Markov chain plot with diagram

Caricamento del pacchetto richiesto: `shape`

Import and export from some specific classes is possible, as shown in Figure 3 and in the following code.

```

R> mcDf <- as(mcWeather, "data.frame")
R> mcNew <- as(mcDf, "markovchain")
R> mcDf

      t0    t1 prob
1 sunny sunny 0.70
2 sunny cloudy 0.20
3 sunny  rain 0.10
4 cloudy sunny 0.30
5 cloudy cloudy 0.40
6 cloudy  rain 0.30
7  rain sunny 0.20
8  rain cloudy 0.45
9  rain  rain 0.35

R> mcIgraph <- as(mcWeather, "igraph")

R> if (requireNamespace("msm", quietly = TRUE)) {
+   require(msm)
+   Q <- rbind ( c(0, 0.25, 0, 0.25),
+                 c(0.166, 0, 0.166, 0.166),
+                 c(0, 0.25, 0, 0.25),

```

```

+           c(0, 0, 0, 0) )
+ cavmsm <- msm(state ~ years, subject = PTNUM, data = cav, qmatrix = Q, death = 4)
+ msmMc <- as(cavmsm, "markovchain")
+ msmMc
+   } else {
+     message("msm unavailable")
+   }

```

Caricamento del pacchetto richiesto: msm

Unnamed Markov chain

A 4 - dimensional discrete Markov Chain defined by the following states:

State 1, State 2, State 3, State 4

The transition matrix (by rows) is defined as follows:

	State 1	State 2	State 3	State 4
State 1	0.853958721	0.08836953	0.01475543	0.04291632
State 2	0.155576908	0.56663284	0.20599563	0.07179462
State 3	0.009903994	0.07853691	0.65965727	0.25190183
State 4	0.000000000	0.00000000	0.00000000	1.00000000

from etm (now archived as of September 2020):

```

R> if (requireNamespace("etm", quietly = TRUE)) {
+ library(etm)
+ data(sir.cont)
+ sir.cont <- sir.cont[order(sir.cont$id, sir.cont$time), ]
+ for (i in 2:nrow(sir.cont)) {
+   if (sir.cont$id[i]==sir.cont$id[i-1]) {
+     if (sir.cont$time[i]==sir.cont$time[i-1]) {
+       sir.cont$time[i-1] <- sir.cont$time[i-1] - 0.5
+     }
+   }
+ }
+ tra <- matrix(ncol=3,nrow=3,FALSE)
+ tra[1, 2:3] <- TRUE
+ tra[2, c(1, 3)] <- TRUE
+ tr.prob <- etm::etm(sir.cont, c("0", "1", "2"), tra, "cens", 1)
+ tr.prob
+ etm2mc<-as(tr.prob, "markovchain")
+ etm2mc
+   } else {
+     message("etm unavailable")
+   }

```

Unnamed Markov chain

A 3 - dimensional discrete Markov Chain defined by the following states:

Import – Export from and to markovchain objects

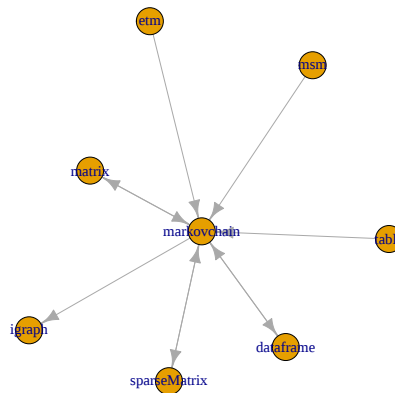


Figure 3: The markovchain methods for import and export

```

0, 1, 2
The transition matrix (by rows) is defined as follows:
      0      1      2
0 0.0000000 0.5000000 0.5000000
1 0.5000000 0.0000000 0.5000000
2 0.3333333 0.3333333 0.3333333

```

Warning: ‘graph.formula()’ was deprecated in igraph 2.1.0.
 i Please use ‘graph_from_literal()’ instead.
 This warning is displayed once per session.
 Call ‘lifecycle::last_lifecycle_warnings()’ to see where this warning was generated.

Coerce from `matrix` method, as the code below shows, represents another approach to create a `markovchain` method starting from a given squared probability matrix.

```

R> myMatr<-matrix(c(.1,.8,.1,.2,.6,.2,.3,.4,.3), byrow=TRUE, ncol=3)
R> myMc<-as(myMatr, "markovchain")
R> myMc

```

Unnamed Markov chain

```

A 3 - dimensional discrete Markov Chain defined by the following states:
s1, s2, s3
The transition matrix (by rows) is defined as follows:
      s1  s2  s3
s1 0.1 0.8 0.1
s2 0.2 0.6 0.2
s3 0.3 0.4 0.3

```

Semi-homogeneous Markov chains can be created with the aid of `markovchainList` object. The example that follows arises from health insurance, where the costs associated to patients in a Continuous Care Health Community (CCHC) are modeled by a semi-homogeneous Markov Chain, since the transition probabilities change by year. Methods explicitly written for `markovchainList` objects are: `print`, `show`, `dim` and `[`.

```
R> stateNames = c("H", "I", "D")
R> Q0 <- new("markovchain", states = stateNames,
+           transitionMatrix = matrix(c(0.7, 0.2, 0.1, 0.1, 0.6, 0.3, 0, 0, 1),
+           byrow = TRUE, nrow = 3), name = "state t0")
R> Q1 <- new("markovchain", states = stateNames,
+           transitionMatrix = matrix(c(0.5, 0.3, 0.2, 0, 0.4, 0.6, 0, 0, 1),
+           byrow = TRUE, nrow = 3), name = "state t1")
R> Q2 <- new("markovchain", states = stateNames,
+           transitionMatrix = matrix(c(0.3, 0.2, 0.5, 0, 0.2, 0.8, 0, 0, 1),
+           byrow = TRUE, nrow = 3), name = "state t2")
R> Q3 <- new("markovchain", states = stateNames,
+           transitionMatrix = matrix(c(0, 0, 1, 0, 0, 1, 0, 0, 1),
+           byrow = TRUE, nrow = 3), name = "state t3")
R> mcCCRC <- new("markovchainList", markovchains = list(Q0, Q1, Q2, Q3),
+           name = "Continuous Care Health Community")
R> print(mcCCRC)
```

Continuous Care Health Community list of Markov chain(s)

Markovchain 1

state t0

A 3 - dimensional discrete Markov Chain defined by the following states:

H, I, D

The transition matrix (by rows) is defined as follows:

	H	I	D
H	0.7	0.2	0.1
I	0.1	0.6	0.3
D	0.0	0.0	1.0

Markovchain 2

state t1

A 3 - dimensional discrete Markov Chain defined by the following states:

H, I, D

The transition matrix (by rows) is defined as follows:

	H	I	D
H	0.5	0.3	0.2
I	0.0	0.4	0.6
D	0.0	0.0	1.0

Markovchain 3

state t2

A 3 - dimensional discrete Markov Chain defined by the following states:

```

H, I, D
The transition matrix (by rows) is defined as follows:
  H   I   D
H 0.3 0.2 0.5
I 0.0 0.2 0.8
D 0.0 0.0 1.0

Markovchain 4
state t3
A 3 - dimensional discrete Markov Chain defined by the following states:
H, I, D
The transition matrix (by rows) is defined as follows:
  H I D
H 0 0 1
I 0 0 1
D 0 0 1

```

It is possible to perform direct access to `markovchainList` elements, as well as to determine the number of `markovchain` objects by which a `markovchainList` object is composed.

```

R> mcCCRC[[1]]

state t0
A 3 - dimensional discrete Markov Chain defined by the following states:
H, I, D
The transition matrix (by rows) is defined as follows:
  H   I   D
H 0.7 0.2 0.1
I 0.1 0.6 0.3
D 0.0 0.0 1.0

R> dim(mcCCRC)

[1] 4

```

The `markovchain` package contains some data found in the literature related to DTMC models (see Section 6. Table 2 lists datasets and tables included within the current release of the package.

Finally, Table 3 lists the demos included in the demo directory of the package.

4. Probability with markovchain objects

The `markovchain` package contains functions to analyse DTMC from a probabilistic perspective. For example, the package provides methods to find stationary distributions and identifying absorbing and transient states. Many of these methods come from MATLAB listings that have been ported into R. For a full description of the underlying theory and algorithm the

Dataset	Description
blanden	Mobility across income quartiles, Jo Blanden and Machin (2005) .
craigsendi	CD4 cells, B. A. Craig and A. A. Sendi (2002) .
kullback	raw transition matrices for testing homogeneity, Kullback, Kupperman, and Ku (1962) .
preproglucacon	Preproglucacon DNA basis, P. J. Avery and D. A. Henderson (1999) .
rain	Alofi Island rains, P. J. Avery and D. A. Henderson (1999) .
holson	Individual states trajectories.
sales	Sales of six beverages in Hong Kong Ching, Ng, and Fung (2008) .

Table 2: The **markovchain** `data.frame` and `table`.

R Code File	Description
bard.R	Structural analysis of Markov chains from Bard PPT.
examples.R	Notable Markov chains, e.g., The Gambler Ruin chain.
quickStart.R	Generic examples.
extractMatrices.R	Generic examples.

Table 3: The **markovchain** demos.

interested reader can overview the original MATLAB listings, [Feres \(2007\)](#) and [Montgomery \(2009\)](#).

Table 4 shows methods that can be applied on **markovchain** objects to perform probabilistic analysis.

4.1. Conditional distributions

The conditional distribution of weather states, given that current day's weather is sunny, is given by following code.

```
R> conditionalDistribution(mcWeather, "sunny")

sunny cloudy    rain
    0.7     0.2     0.1
```

4.2. Stationary states

A stationary (steady state, or equilibrium) vector is a probability vector such that Equation 13 holds

$$\begin{aligned} 0 &\leq \pi_j \leq 1 \\ \sum_{j \in S} \pi_j &= 1 \\ \pi \cdot P &= \pi \end{aligned} \tag{13}$$

Steady states are associated to P eigenvalues equal to one. We could be tempted to compute them solving the eigenvalues and eigenvectors of the matrix and taking real parts (since if $u + iv$ is a eigenvector, for the matrix P , then $Re(u + iv) = u$ and $Im(u + iv) = v$ are eigenvectors) and normalizing by the vector sum, this carries some concerns:

Method	Returns
<code>absorbingStates</code>	the absorbing states of the transition matrix, if any.
<code>steadyStates</code>	the vector(s) of steady state(s) in matrix form.
<code>entropyRate</code>	entropy rate of an irreducible Markov chain.
<code>meanFirstPassageTime</code>	matrix or vector of mean first passage times.
<code>kemenyConstant</code>	stationary-weighted mean hitting time for an irreducible chain.
<code>meanRecurrenceTime</code>	vector of mean number of steps to return to each recurrent state
<code>hittingProbabilities</code>	matrix of hitting probabilities for a Markov chain.
<code>meanAbsorptionTime</code>	expected number of steps for a transient state to be absorbed by any recurrent class
<code>absorptionProbabilities</code>	probabilities of transient states of being absorbed by each recurrent state
<code>committorAB</code>	committor probabilities
<code>communicatingClasses</code>	list of communicating classes. s_j , given actual state s_i .
<code>canonicForm</code>	the transition matrix into canonic form.
<code>is.accessible</code>	checks whether a state j is reachable from state i .
<code>is.irreducible</code>	checks whether a DTMC is irreducible.
<code>is.regular</code>	checks whether a DTMC is regular.
<code>is.stochasticallyMonotone</code>	checks whether a DTMC (or a Matrix) is stochastically monotone.
<code>period</code>	the period of an irreducible DTMC.
<code>recurrentClasses</code>	list of recurrent communicating classes.
<code>transientClasses</code>	list of transient communicating classes.
<code>recurrentStates</code>	the recurrent states of the transition matrix.
<code>transientStates</code>	the transient states of the transition matrix, if any.
<code>summary</code>	DTMC summary.

Table 4: **markovchain** methods: probabilistic operations.

1. If $u, v \in \mathbb{R}^n$ are linearly independent eigenvectors associated with the eigenvalue 1, $u + iv$, $u + iu$ are also linearly independent eigenvectors, and their real parts coincide. Clearly if we took real parts, we would be losing an eigenvector, because we cannot know in advance if the underlying algorithm to compute the eigenvectors is going to output something similar to what we described. We should be agnostic to the underlying eigenvector computation algorithm.
2. Imagine the identity P of dimensions 2×2 . Its eigenvectors associated to the 1 eigenvalue are $u = (1, 0)$ and $v = (0, 1)$. However, the underlying algorithm to compute eigenvectors could return $(1, -2)$ and $(-2, 1)$ instead, that are linear combinations of the aforementioned ones, and therefore eigenvectors. Normalizing by their sum, we would get: $(-1, 2)$ and $(2, -1)$, which obviously are not probability measures. Again, we should be agnostic to the underlying eigen computation algorithm.
3. Algorithms to compute eigenvalues and eigenvectors are computationally expensive: they are iterative, and we cannot predict a fixed number of iterations for them. Moreover, each iteration takes $\mathcal{O}(m^2)$ or $\mathcal{O}(m^3)$ algorithmic complexity, with m the number of states.

We are going to use that every irreducible DTMC has a unique steady state, that is, if M is the matrix for an irreducible DTMC (all states communicate with each other), then it exists a unique $v \in \mathbb{R}^m$ such that:

$$v \cdot M = v, \quad \sum_{i=1}^m v_i = 1$$

Also, we'll use that a steady state for a DTMC assigns 0 to the transient states. The canonical form of a (by row) stochastic matrix looks alike:

$$\left(\begin{array}{c|c|c|c|c} M_1 & 0 & 0 & \dots & 0 \\ \hline 0 & M_2 & 0 & \dots & 0 \\ \hline 0 & 0 & M_3 & \dots & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline A_1 & A_2 & A_3 & \dots & R \end{array} \right)$$

where M_i corresponds to irreducible sub-chains, the blocks A_i correspond to the transitions from transient states to each of the recurrent classes and R are the transitions from the transient states to themselves.

Also, we should note that a Markov chain has exactly the same name of steady states as recurrent classes. Therefore, we have coded the following algorithm ¹:

1. Identify the recurrent classes $[C_1, \dots, C_l]$ with **recurrentClasses** function.
2. Take each class C_i , compute the sub-matrix corresponding to it M_i .
3. Solve the system $v \cdot C_i = v$, $\sum_{j=1}^{|C_i|} v_j = 1$ which has a unique solution, for each $i = 1, \dots, l$.
4. Map each state v_i to the original order in P and assign a 0 to the slots corresponding to transient states in the matrix.

The result is returned in matrix form.

```
R> steadyStates(mcWeather)

      sunny    cloudy    rain
[1,] 0.4636364 0.3181818 0.2181818
```

It is possible for a Markov chain to have more than one stationary distribution, as the gambler ruin example shows.

```
R> gamblerRuinMarkovChain <- function(moneyMax, prob = 0.5) {
+   m <- markovchain:::zeros(moneyMax + 1)
+   m[1,1] <- m[moneyMax + 1, moneyMax + 1] <- 1
+   states <- as.character(0:moneyMax)
+   rownames(m) <- colnames(m) <- states
+ }
```

¹We would like to thank Prof. Christophe Dutang for his contributions to the development of this method. He coded a first improvement of the original **steadyStates** method and we could not have reached the current correctness without his previous work

```

+
+   for(i in 2:moneyMax){
+     m[i,i-1] <- 1 - prob
+     m[i, i + 1] <- prob
+   }
+
+   new("markovchain", transitionMatrix = m,
+       name = paste("Gambler ruin", moneyMax, "dim", sep = " "))
+ }
R>
R> mcGR4 <- gamblerRuinMarkovChain(moneyMax = 4, prob = 0.5)
R> steadyStates(mcGR4)

```

```

      0 1 2 3 4
[1,] 0 0 0 0 1
[2,] 1 0 0 0 0

```

4.3. Entropy rate

For a stationary first-order Markov chain, the entropy rate is the conditional entropy of the next state given the current state:

$$H(X) = H(X_{t+1} | X_t) = - \sum_i \pi_i \sum_j p_{ij} \log_b p_{ij},$$

where π is the stationary distribution and $0 \log(0)$ is interpreted as zero. With $b = 2$ the result is measured in bits per transition; with $b = e$ it is measured in nats per transition ([Cover and Thomas 2006](#)). The method `entropyRate` uses the unique stationary distribution of a finite irreducible chain. Aperiodicity is not required; reducible chains are rejected because the result can depend on the chosen stationary distribution ([Streliaoff, Crutchfield, and Hubler 2007](#)).

```

R> entropyRate(mcWeather)

[1] 1.366258

R> entropyRate(mcWeather, base = exp(1))

[1] 0.9470176

```

A deterministic chain has entropy rate zero, whereas a chain whose rows are all uniform on n states has the maximum value $\log_b(n)$.

4.4. Classification of states

Absorbing states are determined by means of `absorbingStates` method.

```
R> absorbingStates(mcGR4)
```

```
[1] "0" "4"
```

```
R> absorbingStates(mcWeather)
```

```
character(0)
```

The key function in methods which need knowledge about communicating classes, recurrent states, transient states, is `.commclassKernel`, which is a modification of Tarjan's algorithm from [Tarjan \(1972\)](#). This `.commclassKernel` method gets a transition matrix of dimension n and returns a list of two items:

1. `classes`, an matrix whose (i, j) entry is `true` if s_i and s_j are in the same communicating class.
2. `closed`, a vector whose i -th entry indicates whether the communicating class to which i belongs is closed.

These functions are used by two other internal functions on which the `summary` method for `markovchain` objects works.

The example matrix used in [Feres \(2007\)](#) well exemplifies the purpose of the function.

```
R> P <- markovchain:::zeros(10)
R> P[1, c(1, 3)] <- 1/2;
R> P[2, 2] <- 1/3; P[2,7] <- 2/3;
R> P[3, 1] <- 1;
R> P[4, 5] <- 1;
R> P[5, c(4, 5, 9)] <- 1/3;
R> P[6, 6] <- 1;
R> P[7, 7] <- 1/4; P[7,9] <- 3/4;
R> P[8, c(3, 4, 8, 10)] <- 1/4;
R> P[9, 2] <- 1;
R> P[10, c(2, 5, 10)] <- 1/3;
R> rownames(P) <- letters[1:10]
R> colnames(P) <- letters[1:10]
R> probMc <- new("markovchain", transitionMatrix = P,
+               name = "Probability MC")
R> summary(probMc)
```

Probability MC Markov chain that is composed by:

Closed classes:

a c

b g i

f

Recurrent classes:

{a,c},{b,g,i},{f}

Transient classes:

{d,e},{h},{j}

The Markov chain is not irreducible

The absorbing states are: f

All states that pertain to a transient class are named “transient” and a specific method has been written to elicit them.

```
R> transientStates(probMc)
```

```
[1] "d" "e" "h" "j"
```

`canonicForm` method that turns a Markov chain into its canonic form, reordering the states to have first the recurrent classes and then the transient states.

```
R> probMcCanonic <- canonicForm(probMc)
```

```
R> probMc
```

Probability MC

A 10 - dimensional discrete Markov Chain defined by the following states:

a, b, c, d, e, f, g, h, i, j

The transition matrix (by rows) is defined as follows:

	a	b	c	d	e	f	g	h	i	j
a	0.5	0.0000000	0.50	0.0000000	0.0000000	0	0.0000000	0.00	0.0000000	0.0000000
b	0.0	0.3333333	0.00	0.0000000	0.0000000	0	0.6666667	0.00	0.0000000	0.0000000
c	1.0	0.0000000	0.00	0.0000000	0.0000000	0	0.0000000	0.00	0.0000000	0.0000000
d	0.0	0.0000000	0.00	0.0000000	1.0000000	0	0.0000000	0.00	0.0000000	0.0000000
e	0.0	0.0000000	0.00	0.3333333	0.3333333	0	0.0000000	0.00	0.3333333	0.0000000
f	0.0	0.0000000	0.00	0.0000000	0.0000000	1	0.0000000	0.00	0.0000000	0.0000000
g	0.0	0.0000000	0.00	0.0000000	0.0000000	0	0.2500000	0.00	0.7500000	0.0000000
h	0.0	0.0000000	0.25	0.2500000	0.0000000	0	0.0000000	0.25	0.0000000	0.2500000
i	0.0	1.0000000	0.00	0.0000000	0.0000000	0	0.0000000	0.00	0.0000000	0.0000000
j	0.0	0.3333333	0.00	0.0000000	0.3333333	0	0.0000000	0.00	0.0000000	0.3333333

```
R> probMcCanonic
```

Probability MC

A 10 - dimensional discrete Markov Chain defined by the following states:

a, c, b, g, i, f, d, e, h, j

The transition matrix (by rows) is defined as follows:

	a	c	b	g	i	f	d	e	h	j
a	0.5	0.50	0.0000000	0.0000000	0.0000000	0	0.0000000	0.0000000	0.00	0.0000000
c	1.0	0.00	0.0000000	0.0000000	0.0000000	0	0.0000000	0.0000000	0.00	0.0000000
b	0.0	0.00	0.3333333	0.6666667	0.0000000	0	0.0000000	0.0000000	0.00	0.0000000
g	0.0	0.00	0.0000000	0.2500000	0.7500000	0	0.0000000	0.0000000	0.00	0.0000000
i	0.0	0.00	1.0000000	0.0000000	0.0000000	0	0.0000000	0.0000000	0.00	0.0000000

```
f 0.0 0.00 0.0000000 0.0000000 0.0000000 1 0.0000000 0.0000000 0.00 0.0000000
d 0.0 0.00 0.0000000 0.0000000 0.0000000 0 0.0000000 1.0000000 0.00 0.0000000
e 0.0 0.00 0.0000000 0.0000000 0.3333333 0 0.3333333 0.3333333 0.00 0.0000000
h 0.0 0.25 0.0000000 0.0000000 0.0000000 0 0.2500000 0.0000000 0.25 0.2500000
j 0.0 0.00 0.3333333 0.0000000 0.0000000 0 0.0000000 0.3333333 0.00 0.3333333
```

The function `is.accessible` permits to investigate whether a state s_j is accessible from state s_i , that is whether the probability to eventually reach s_j starting from s_i is greater than zero.

```
R> is.accessible(object = probMc, from = "a", to = "c")
```

```
[1] TRUE
```

```
R> is.accessible(object = probMc, from = "g", to = "c")
```

```
[1] FALSE
```

In Section 2.2 we observed that, if a DTMC is irreducible, all its states share the same periodicity. Then, the `period` function returns the periodicity of the DTMC, provided that it is irreducible. The example that follows shows how to find if a DTMC is reducible or irreducible by means of the function `is.irreducible` and, in the latter case, the method `period` is used to compute the periodicity of the chain.

```
R> E <- matrix(0, nrow = 4, ncol = 4)
R> E[1, 2] <- 1
R> E[2, 1] <- 1/3; E[2, 3] <- 2/3
R> E[3,2] <- 1/4; E[3, 4] <- 3/4
R> E[4, 3] <- 1
R>
R> mcE <- new("markovchain", states = c("a", "b", "c", "d"),
+           transitionMatrix = E,
+           name = "E")
R> is.irreducible(mcE)
```

```
[1] TRUE
```

```
R> period(mcE)
```

```
[1] 2
```

The example Markov chain found in Mathematica web site ([Wolfram Research 2013b](#)) has been used, and is plotted in Figure 4.

```
R> mathematicaMatr <- markovchain:::zeros(5)
R> mathematicaMatr[1,] <- c(0, 1/3, 0, 2/3, 0)
R> mathematicaMatr[2,] <- c(1/2, 0, 0, 0, 1/2)
```

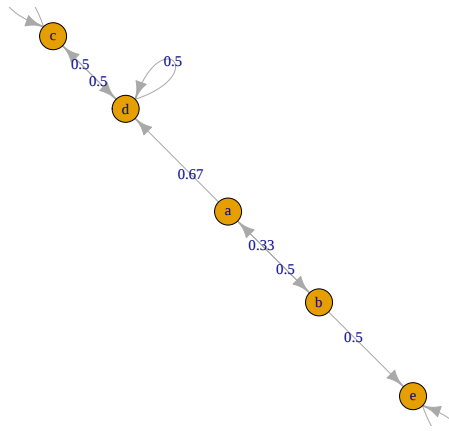


Figure 4: Mathematica 9 example. Markov chain plot.

```

R> mathematicaMatr[3,] <- c(0, 0, 1/2, 1/2, 0)
R> mathematicaMatr[4,] <- c(0, 0, 1/2, 1/2, 0)
R> mathematicaMatr[5,] <- c(0, 0, 0, 0, 1)
R> statesNames <- letters[1:5]
R> mathematicaMc <- new("markovchain", transitionMatrix = mathematicaMatr,
+                       name = "Mathematica MC", states = statesNames)

```

Mathematica MC Markov chain that is composed by:

Closed classes:

c d

e

Recurrent classes:

{c,d},{e}

Transient classes:

{a,b}

The Markov chain is not irreducible

The absorbing states are: e

4.5. First passage time distributions and means

Feres (2007) provides code to compute first passage time (within $1, 2, \dots, n$ steps) given the initial state to be i . The MATLAB listings translated into R on which the `firstPassage` function is based are:

```

R> .firstpassageKernel <- function(P, i, n){
+   G <- P

```

```

+   H <- P[i,]
+   E <- 1 - diag(size(P)[2])
+   for (m in 2:n) {
+     G <- P %*% (G * E)
+     H <- rbind(H, G[i,])
+   }
+   return(H)
+ }

```

We conclude that the probability for the *first* rainy day to be the third one, given that the current state is sunny, is given by:

```

R> firstPassagePdF <- firstPassage(object = mcWeather, state = "sunny",
+                                n = 10)
R> firstPassagePdF[3, 3]

```

```
[1] 0.121
```

To compute the *mean* first passage times, i.e. the expected number of days before it rains given that today is sunny, we can use the `meanFirstPassageTime` function:

```

R> meanFirstPassageTime(mcWeather)

      sunny   cloudy    rain
sunny 0.000000 4.285714 6.666667
cloudy 3.725490 0.000000 5.000000
rain   4.117647 2.857143 0.000000

```

indicating e.g. that the average number of days of sun or cloud before rain is 6.67 if we start counting from a sunny day, and 5 if we start from a cloudy day. Note that we can also specify one or more destination states:

```

R> meanFirstPassageTime(mcWeather, "rain")

      sunny   cloudy
6.666667 5.000000

```

The implementation follows the matrix solutions by (Grinstead and Snell 2006). We can check the result by averaging the first passage probability density function:

```

R> firstPassagePdF.long <- firstPassage(object = mcWeather, state = "sunny", n = 100)
R> sum(firstPassagePdF.long[, "rain"] * 1:100)

```

```
[1] 6.666664
```

4.6. Kemeny's constant

For a finite irreducible chain with stationary distribution π , let m_{ij} denote the mean time to hit state j from state i , using $m_{ii} = 0$. Kemeny's constant is

$$K = \sum_j \pi_j m_{ij}.$$

Remarkably, K does not depend on the starting state i . Equivalently, if $Z = (I - P + \mathbf{1}\pi^T)^{-1}$, then $K = \text{tr}(Z) - 1$ (Kemeny and Snell 1960). The function `kemenyConstant` uses this formula and requires irreducibility, but not aperiodicity. If mean return times $m_{jj} = 1/\pi_j$ are placed on the diagonal instead, the weighted sum is $K + 1$.

```
R> kemenyConstant(mcWeather)
```

```
[1] 2.818182
```

```
R> piWeather <- as.numeric(steadyStates(mcWeather))
R> meanFirstPassageTime(mcWeather) %*% piWeather
```

```
      [,1]
sunny 2.818182
cloudy 2.818182
rain  2.818182
```

The second result has one entry for each possible starting state; apart from numerical rounding, all entries equal Kemeny's constant.

4.7. Mean recurrence time

The `meanRecurrenceTime` method gives the first mean recurrence time (expected number of steps to go back to a state if it was the initial one) for each recurrent state in the transition probabilities matrix for a DTMC. Let's see an example:

```
R> meanRecurrenceTime(mcWeather)
```

```
      sunny  cloudy  rain
2.156863  3.142857  4.583333
```

Another example, with not all of its states being recurrent:

```
R> recurrentStates(probMc)
```

```
[1] "a" "b" "c" "f" "g" "i"
```

```
R> meanRecurrenceTime(probMc)
```

	f	b	g	i	a	c
	1.000000	2.555556	2.875000	3.833333	1.500000	3.000000

4.8. Absorption probabilities and mean absorption time

We are going to use the Drunkard's random walk from (Grinstead and Snell 2006). We have a drunk person walking through the street. Each move the person does, if they have not arrived to either home (corner 1) or to the bar (corner 5) could be to the left corner or to the right one, with equal probability. In case of arrival to the bar or to home, the person stays there.

```
R> drunkProbs <- markovchain::zeros(5)
R> drunkProbs[1,1] <- drunkProbs[5,5] <- 1
R> drunkProbs[2,1] <- drunkProbs[2,3] <- 1/2
R> drunkProbs[3,2] <- drunkProbs[3,4] <- 1/2
R> drunkProbs[4,3] <- drunkProbs[4,5] <- 1/2
R>
R> drunkMc <- new("markovchain", transitionMatrix = drunkProbs)
R> drunkMc
```

Unnamed Markov chain

A 5 - dimensional discrete Markov Chain defined by the following states:

1, 2, 3, 4, 5

The transition matrix (by rows) is defined as follows:

	1	2	3	4	5
1	1.0	0.0	0.0	0.0	0.0
2	0.5	0.0	0.5	0.0	0.0
3	0.0	0.5	0.0	0.5	0.0
4	0.0	0.0	0.5	0.0	0.5
5	0.0	0.0	0.0	0.0	1.0

Recurrent (in fact absorbing states) are:

```
R> recurrentStates(drunkMc)
```

```
[1] "1" "5"
```

Transient states are the rest:

```
R> transientStates(drunkMc)
```

```
[1] "2" "3" "4"
```

The probability of either being absorbed by the bar or by the sofa at home are:

```
R> absorptionProbabilities(drunkMc)
```

```

      1      5
2 0.75 0.25
3 0.50 0.50
4 0.25 0.75

```

which means that the probability of arriving home / bar is inversely proportional to the distance to each one.

But we also would like to know how much time does the person take to arrive there, which can be done with `meanAbsorptionTime`:

```

R> meanAbsorptionTime(drunkMc)

2 3 4
3 4 3

```

Thus, the expected time to absorption is three steps from states 2 or 4 and four steps from state 3.

Fundamental matrix

For an absorbing chain, let Q be the transition submatrix among transient states. The fundamental matrix is

$$N = (I - Q)^{-1} = I + Q + Q^2 + \dots \quad (14)$$

Its (i, j) entry is the expected number of visits to transient state j before absorption when starting from transient state i , including the initial visit. The expected absorption-time vector is $N\mathbf{1}$, while absorption probabilities are obtained from NR , where R contains transitions from transient to absorbing states ([Grinstead and Snell 2006](#)).

```

R> fundamentalMatrix(drunkMc)

      2 3      4
2 1.5 1 0.5
3 1.0 2 1.0
4 0.5 1 1.5

```

4.9. Committed probability

The committed probability tells us the probability to reach a given state before another given. Suppose that we start in a cloudy day, the probabilities of experiencing a rainy day before a sunny one is 0.5:

```

R> committedAB(mcWeather, 3, 1)

sunny cloudy  rain
0.0      0.5    1.0

```

4.10. Hitting probabilities

For states i and j , the hitting probability h_{ij} is the probability of ever visiting j when the chain starts in i . For $i = j$, the package uses the return probability after at least one transition, rather than the trivial time-zero probability. The values are the minimal non-negative solution of the first-step equations

$$h_{ij} = p_{ij} + \sum_{k \neq j} p_{ik} h_{kj}. \quad (15)$$

`hittingProbabilities` first analyses the directed graph of positive transitions. This identifies probabilities that are structurally zero or one, without discarding very small positive transitions. Remaining probabilities are computed from a convergent Neumann series using a relative residual stopping rule. This is important for nearly absorbing chains: a small transition probability may imply a long waiting time, but the probability of eventually taking that transition can still be exactly one.

```
R> M <- markovchain::zeros(5)
R> M[1,1] <- M[5,5] <- 1
R> M[2,1] <- M[2,3] <- 1/2
R> M[3,2] <- M[3,4] <- 1/2
R> M[4,2] <- M[4,5] <- 1/2
R>
R> hittingTest <- new("markovchain", transitionMatrix = M)
R> hittingProbabilities(hittingTest)
```

	1	2	3	4	5
1	1.0	0.000	0.000	0.0000000	0.0
2	0.8	0.375	0.500	0.3333333	0.2
3	0.6	0.750	0.375	0.6666667	0.4
4	0.4	0.500	0.250	0.1666667	0.6
5	0.0	0.000	0.000	0.0000000	1.0

For the irreducible finite chain `mcWeather`, every state is eventually visited with probability one:

```
R> hittingProbabilities(mcWeather)
```

	sunny	cloudy	rain
sunny	1	1	1
cloudy	1	1	1
rain	1	1	1

4.11. Stochastic Monotonicity

A transition matrix P is stochastically monotone if the probability of moving to higher states is non-decreasing with respect to the starting state. In other words, row i is stochastically

dominated by row j whenever $i < j$. This property is fundamental in many applied models, such as credit ratings or social mobility, because it guarantees that the Markov chain preserves the initial ordering of states over time.

For instance, in an income mobility model, if you start in a higher income class, your probability of moving to or staying in a high income class should ideally be greater than or equal to someone starting in a lower income class.

The `is.stochasticallyMonotone` method allows verifying this property for any `markovchain` object:

```
R> # Define the state space for income classes
R> incomeStates <- c("low", "mid", "high")
R>
R> # Example 1: A stochastically monotone transition matrix
R> # Higher starting income leads to higher probabilities of reaching/staying in higher br
R> monotoneMatrix <- matrix(c(0.6, 0.3, 0.1,
+                             0.3, 0.5, 0.2,
+                             0.1, 0.4, 0.5),
+                             byrow = TRUE, nrow = 3,
+                             dimnames = list(incomeStates, incomeStates))
R>
R> mcIncomeMonotone <- new("markovchain", states = incomeStates,
+                           transitionMatrix = monotoneMatrix,
+                           name = "Monotone Income MC")
R>
R> is.stochasticallyMonotone(mcIncomeMonotone)
```

```
[1] TRUE
```

```
R> # Example 2: A non-monotone transition matrix
R> # Here, a person in the "mid" class has an unusually high chance of dropping
R> # to "low" (0.7), which is worse than a person already in the "low" class (0.5).
R> # This violates the stochastic monotonicity property.
R> notMonotoneMatrix <- matrix(c(0.5, 0.3, 0.2,
+                               0.7, 0.2, 0.1,
+                               0.1, 0.4, 0.5),
+                               byrow = TRUE, nrow = 3,
+                               dimnames = list(incomeStates, incomeStates))
R>
R> mcIncomeNotMonotone <- new("markovchain", states = incomeStates,
+                             transitionMatrix = notMonotoneMatrix,
+                             name = "Non-Monotone Income MC")
R>
R> is.stochasticallyMonotone(mcIncomeNotMonotone)
```

```
[1] FALSE
```

5. Statistical analysis

Table 5 lists the functions and methods implemented within the package which help to fit, simulate and predict DTMC.

Function	Purpose
<code>markovchainFit</code>	Function to return fitted Markov chain for a given sequence.
<code>predict</code>	Method to calculate predictions from <code>markovchain</code> or <code>markovchainList</code> objects.
<code>rmarkovchain</code>	Function to sample from <code>markovchain</code> or <code>markovchainList</code> objects.

Table 5: The **markovchain** statistical functions.

5.1. Simulation

Simulating a random sequence from an underlying DTMC is quite easy thanks to the function `rmarkovchain`. The following code generates a year of weather states according to `mcWeather` underlying stochastic process.

```
R> weathersOfDays <- rmarkovchain(n = 220, object = mcWeather, t0 = "sunny")
R> weathersOfDays[1:30]
```

```
[1] "cloudy" "rain"   "rain"   "rain"   "cloudy" "rain"   "rain"   "cloudy"
[9] "sunny"  "rain"   "cloudy" "cloudy" "cloudy" "rain"   "rain"   "rain"
[17] "cloudy" "rain"   "rain"   "rain"   "rain"   "cloudy" "cloudy" "sunny"
[25] "sunny"  "sunny"  "sunny"  "sunny"  "cloudy" "sunny"
```

Similarly, it is possible to simulate one or more sequences from a semi-homogeneous Markov chain, as the following code (applied on CCHC example) exemplifies.

```
R> patientStates <- rmarkovchain(n = 5, object = mcCCRC, t0 = "H",
+                               include.t0 = TRUE)
R> patientStates[1:10,]
```

```
iteration values
1          1      H
2          1      H
3          1      D
4          1      D
5          1      D
6          2      H
7          2      H
8          2      D
9          2      D
10         2      D
```

Two advanced parameters are available to the `rmarkovchain` method which help users decide which implementation to use. There are four options available : R, R in parallel, C++ and C++ in parallel. Two boolean parameters `useRcpp` and `parallel` will decide which implementation will be used. Default is `useRcpp = TRUE` and `parallel = FALSE` i.e. C++ implementation. The C++ implementation is generally faster than the R implementation. If you have multicore processors then you can take advantage of `parallel` parameter by setting it to `TRUE`. When both `useRcpp = TRUE` and `parallel = TRUE` the parallelization has been carried out using `RcppParallel` package (Allaire, Francois, Ushey, Vandenbrouck, Geelhard, and Intel 2016).

5.2. Lumpability and aggregation of states

In some applications the original state space is too detailed, and it is useful to aggregate several micro-states into a smaller number of macro-states. A Markov chain is said to be strongly lumpable with respect to a partition of the state space if, for every pair of macro-states, the total probability of moving from a micro-state in the source block to the destination block is the same for all micro-states in the source block (Kemeny and Snell 1960). Under this condition, the aggregated process is again Markovian and its transition matrix is well defined.

The function `is.lumpable` checks exact strong lumpability for a user-supplied partition. The function `lump` returns the aggregated Markov chain. By default, `lump` requires exact lumpability; setting `force = TRUE` performs an approximate weighted aggregation.

The following example is the classical Land of Oz weather chain. We aggregate `rainy` and `snowy` into a single `Bad_Weather` macro-state, leaving `nice` as a separate macro-state.

```
R> states_oz <- c("rainy", "nice", "snowy")
R>
R> P_oz <- matrix(
+   c(0.5, 0.25, 0.25,
+     0.5, 0.00, 0.5,
+     0.25, 0.25, 0.5),
+   byrow = TRUE,
+   nrow = 3,
+   dimnames = list(states_oz, states_oz)
+ )
R>
R> mcOz <- new(
+   "markovchain",
+   states = states_oz,
+   transitionMatrix = P_oz,
+   name = "Land of Oz"
+ )
R>
R> partition_oz <- list(
+   Bad_Weather = c("rainy", "snowy"),
+   Nice_Weather = c("nice")
+ )
```

```

R>
R> is.lumpable(mcOz, partition_oz)

[1] TRUE

R> lumped_oz <- lump(mcOz, partition_oz)
R> lumped_oz

```

Land of Oz (Lumped)

A 2 - dimensional discrete Markov Chain defined by the following states:

Bad_Weather, Nice_Weather

The transition matrix (by rows) is defined as follows:

	Bad_Weather	Nice_Weather
Bad_Weather	0.75	0.25
Nice_Weather	1.00	0.00

The lumped chain has two states. From either rainy or snowy, the probability of moving to nice is 0.25, so the transition probability from Bad_Weather to Nice_Weather is also 0.25. The remaining probability, 0.75, stays inside the Bad_Weather macro-state.

If the exact lumpability condition is not satisfied, lump stops by default:

```

R> P_bad <- matrix(
+   c(0.5, 0.3, 0.2,
+     0.7, 0.2, 0.1,
+     0.1, 0.4, 0.5),
+   byrow = TRUE,
+   nrow = 3,
+   dimnames = list(c("A1", "A2", "B"), c("A1", "A2", "B")))
R>
R> mcBad <- new(
+   "markovchain",
+   states = c("A1", "A2", "B"),
+   transitionMatrix = P_bad
+ )
R>
R> partition_bad <- list(
+   Group_A = c("A1", "A2"),
+   Group_B = c("B")
+ )
R>
R> is.lumpable(mcBad, partition_bad)

```

```
[1] FALSE
```

```
R> try(lump(mcBad, partition_bad))
```

Error in lump(mcBad, partition_bad) :

The Markov chain is not exactly lumpable. Use force = TRUE to perform an approximate wei

For exploratory analyses, approximate lumping can be forced:

```
R> lump(mcBad, partition_bad, force = TRUE)
```

Unnamed Markov chain (Lumped)

A 2 - dimensional discrete Markov Chain defined by the following states:

Group_A, Group_B

The transition matrix (by rows) is defined as follows:

	Group_A	Group_B
Group_A	0.8389831	0.1610169
Group_B	0.5000000	0.5000000

When force = TRUE, the aggregated transition probabilities are computed using stationary weights when available. If a macro-state has zero stationary mass, an arithmetic average over its micro-states is used as a fallback. This produces a valid coarse transition matrix, but it should be interpreted as an approximation: the aggregated process is not guaranteed to be exactly Markovian unless the strong lumpability condition holds.

The helper `autoLump` provides a heuristic way to discover macro-states using spectral clustering. It should be regarded as an exploratory approximation, not as a test of exact lumpability.

```
R> rc <- c("AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D")
```

```
R>
```

```
R> creditMatrix <- matrix(
+   c(90.81, 8.33, 0.68, 0.06, 0.08, 0.02, 0.01, 0.01,
+     0.70, 90.65, 7.79, 0.64, 0.06, 0.13, 0.02, 0.01,
+     0.09, 2.27, 91.05, 5.52, 0.74, 0.26, 0.01, 0.06,
+     0.02, 0.33, 5.95, 85.93, 5.30, 1.17, 1.12, 0.18,
+     0.03, 0.14, 0.67, 7.73, 80.53, 8.84, 1.00, 1.06,
+     0.01, 0.11, 0.24, 0.43, 6.48, 83.46, 4.07, 5.20,
+     0.21, 0.00, 0.22, 1.30, 2.38, 11.24, 64.86, 19.79,
+     0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 100.00),
+   nrow = 8,
+   byrow = TRUE,
+   dimnames = list(rc, rc)
+ ) / 100
```

```
R>
```

```
R> creditMc <- new(
+   "markovchain",
+   states = rc,
+   transitionMatrix = creditMatrix,
+   name = "Credit ratings"
+ )
R>
```

```
R> set.seed(123)
R> auto_lumped <- autoLump(creditMc, k = 3)
R>
R> auto_lumped$partition
R> auto_lumped$lumped_chain
```

To keep the vignette fast, the example uses a small transition matrix. In larger applications, the automatically discovered partition should be inspected and validated substantively before being used for modelling or forecasting.

5.3. Estimation

A time-homogeneous Markov chain can be fitted from observed state sequences. The current version of **markovchain** implements four fitting methods: maximum likelihood (MLE), MLE with Laplace smoothing, bootstrap estimation and maximum a posteriori (MAP) estimation. Equation 16 shows the maximum likelihood estimator of the transition probability p_{ij} . Here n_{ij} is the number of observed transitions $X_t = s_i, X_{t+1} = s_j$ and $n_i = \sum_{u=1}^k n_{iu}$ is the number of observed outgoing transitions from state s_i :

$$\hat{p}_{ij}^{MLE} = \frac{n_{ij}}{n_i}. \quad (16)$$

For a fixed origin state s_i , the counts $(n_{i1}, \dots, n_{ik})$ are treated as a multinomial sample with row total $n_i = \sum_{u=1}^k n_{iu}$. The **standardError** returned for each marginal transition probability is therefore the usual binomial standard error in Equation (17).

$$SE_{ij} = \sqrt{\frac{\hat{p}_{ij}^{MLE}(1 - \hat{p}_{ij}^{MLE})}{n_i}} \quad (17)$$

```
R> weatherFittedMLE <- markovchainFit(data = weathersOfDays, method = "mle", name = "Weather")
R> weatherFittedMLE$estimate
```

Weather MLE

A 3 - dimensional discrete Markov Chain defined by the following states:

cloudy, rain, sunny

The transition matrix (by rows) is defined as follows:

	cloudy	rain	sunny
cloudy	0.3947368	0.35526316	0.2500000
rain	0.4909091	0.38181818	0.1272727
sunny	0.2045455	0.07954545	0.7159091

```
R> weatherFittedMLE$standardError
```

	cloudy	rain	sunny
cloudy	0.05606853	0.05489838	0.04966996
rain	0.06740884	0.06550962	0.04493923
sunny	0.04299929	0.02884480	0.04807467

The Laplace smoothing approach is a smoothed version of the MLE in which each count n_{ij} is replaced by $n_{ij} + \alpha$ (Equation 18). The smoothing parameter α must be non-negative; $\alpha = 0$ gives the unsmoothed MLE.

$$\hat{p}_{ij}^{LS} = \frac{n_{ij} + \alpha}{\sum_{u=1}^k (n_{iu} + \alpha)} \quad (18)$$

```
R> weatherFittedLAPLACE <- markovchainFit(data = weathersOfDays,
+                                       method = "laplace", laplacian = 0.01,
+                                       name = "Weather LAPLACE")
R> weatherFittedLAPLACE$estimate
```

Weather LAPLACE

A 3 - dimensional discrete Markov Chain defined by the following states:

cloudy, rain, sunny

The transition matrix (by rows) is defined as follows:

	cloudy	rain	sunny
cloudy	0.3947126	0.35525450	0.2500329
rain	0.4908232	0.38179175	0.1273851
sunny	0.2045893	0.07963194	0.7157787

The confidence interval option is enabled by default; set `confint = FALSE` to reduce computation time when intervals are not needed. Both MLE and Laplace fitting are based on `createSequenceMatrix`, which returns the raw transition-count matrix.

```
R> createSequenceMatrix(stringchar = weathersOfDays)
```

	cloudy	rain	sunny
cloudy	30	27	19
rain	27	21	7
sunny	18	7	63

`stringchar` could contain NA values, and the transitions containing NA would be ignored.

If a state is observed only as the last element of a sequence, or is supplied through `possibleStates` but has no observed outgoing transition, its row has zero total count. In this case the point estimate follows the package convention of assigning a uniform row, $p_{\beta j} = 1/k$. For MLE confidence intervals, such a row carries no empirical information, so the corresponding cell-wise intervals are uninformative.

Create sequence matrix can also be used to obtain raw count transition matrices from a given $n * 2$ matrix as the following example shows:

```
R> myMatr<-matrix(c("a","b","b","a","a","b","b","b","b","a","a","a","b","a"),ncol=2)
R> createSequenceMatrix(stringchar = myMatr,toRowProbs = TRUE)
```

	a	b
a	0.6666667	0.3333333
b	0.5000000	0.5000000

A bootstrap estimation approach is also available to assess the variability of $\hat{p}_{ij}$. The implementation in **markovchain** proceeds as follows:

1. generate bootstrap state sequences using the transition probabilities estimated from the original data. The number of bootstrap samples is controlled by the **nboot** argument of **markovchainFit**;
2. apply MLE estimation to each bootstrap sequence; the bootstrap estimates are saved in the **bootStrapSamples** component of the returned list;
3. compute the bootstrap transition estimate as the row-normalized average of the bootstrap MLE estimates. The reported **standardError** is the empirical standard deviation of the bootstrap estimates for each transition probability.

```
R> weatherFittedBOOT <- markovchainFit(data = weathersOfDays,
+                                     method = "bootstrap", nboot = 10)
R> weatherFittedBOOT$estimate
```

BootStrap Estimate

A 3 - dimensional discrete Markov Chain defined by the following states:

cloudy, rain, sunny

The transition matrix (by rows) is defined as follows:

	cloudy	rain	sunny
cloudy	0.4039584	0.34401538	0.2520262
rain	0.4619274	0.42383613	0.1142365
sunny	0.2139972	0.09615059	0.6898522

```
R> weatherFittedBOOT$standardError
```

	cloudy	rain	sunny
cloudy	0.07466146	0.06249693	0.05123581
rain	0.03943952	0.06057494	0.04185424
sunny	0.05526397	0.02727959	0.04988424

The bootstrapping process can be done in parallel thanks to **RcppParallel** package ([Allaire et al. 2016](#)). The parallelized implementation is recommended when the data sample size or the required number of bootstrap runs is high.

```
R> weatherFittedBOOTParallel <- markovchainFit(data = weathersOfDays,
+                                               method = "bootstrap", nboot = 200,
+                                               parallel = TRUE)
R> weatherFittedBOOTParallel$estimate
R> weatherFittedBOOTParallel$standardError
```

The parallel bootstrapping uses all the available cores on a machine by default. However, it is also possible to tune the number of threads used. Note that this should be done in R before calling the **markovchainFit** function. For example, the following code will set the number of threads to 4.

```
R> RcppParallel::setNumThreads(2)
```

For more details, please refer to **RcppParallel** web site.

For all the fitting methods, the `logLikelihood` (Skuriat-Olechnowska 2005) denoted in Equation 19 is provided.

$$LLH = \sum_{i,j} n_{ij} * \log(p_{ij}) \quad (19)$$

where n_{ij} is the entry of the frequency matrix and p_{ij} is the entry of the transition probability matrix.

```
R> weatherFittedMLE$logLikelihood
```

```
[1] -203.3674
```

```
R> weatherFittedBOOT$logLikelihood
```

```
[1] -203.7924
```

Confidence matrices of estimated parameters are available as well. The `confidenceInterval` component contains two matrices, `lowerEndpointMatrix` and `upperEndpointMatrix`. The confidence level is 0.95 by default and can be changed through the `confidencelevel` argument of `markovchainFit`.

For MLE fits, **markovchain** reports cell-wise Wilson score intervals for the marginal transition probabilities (Brown, Cai, and DasGupta 2001). Compared with the simple Wald interval $\hat{p}_{ij} \pm zSE_{ij}$, Wilson intervals have better small-sample behaviour and do not collapse to a zero-width interval when a transition is never observed or always observed. With $z = \Phi^{-1}(1 - \alpha/2)$, $n_i = \sum_u n_{iu}$, $\hat{p}_{ij} = n_{ij}/n_i$, and confidence level $1 - \alpha$, the interval is

$$LowerEndpoint_{ij} = \frac{\hat{p}_{ij} + z^2/(2n_i) - z\sqrt{\hat{p}_{ij}(1 - \hat{p}_{ij})/n_i + z^2/(4n_i^2)}}{1 + z^2/n_i}, \quad (20)$$

$$UpperEndpoint_{ij} = \frac{\hat{p}_{ij} + z^2/(2n_i) + z\sqrt{\hat{p}_{ij}(1 - \hat{p}_{ij})/n_i + z^2/(4n_i^2)}}{1 + z^2/n_i}. \quad (21)$$

If no outgoing transition is available for a state, the point estimate follows the package convention of a uniform row, while the interval is left maximally uninformative. These intervals are cell-wise marginal intervals, not simultaneous multinomial confidence regions for a full row or matrix. For simultaneous multinomial intervals, use `multinomialConfidenceIntervals`. For bootstrap fits, intervals use the empirical standard deviation of the bootstrap transition-probability estimates and the two-sided normal quantile corresponding to `confidencelevel`. For MAP fits, credible intervals are equal-tailed beta posterior intervals, obtained from the inverse beta distribution (Strelloff *et al.* 2007).

```
R> weatherFittedMLE$confidenceInterval
```

NULL

```
R> weatherFittedBOOT$confidenceInterval
```

```
$confidenceLevel
```

```
[1] 0.95
```

```
$lowerEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.2576246	0.22152365	0.15160591
rain	0.3846273	0.30511143	0.03220371
sunny	0.1056818	0.04268359	0.59208089

```
$upperEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.5502921	0.4665071	0.3524466
rain	0.5392274	0.5425608	0.1962693
sunny	0.3223126	0.1496176	0.7876235

A special function, `multinomialConfidenceIntervals`, has been written in order to obtain multinomial wise confidence intervals. The code has been based on and Rcpp translation of package's **MultinomialCI** functions Villacorta (2012) that were themselves based on the Sison and Glaz (1995) paper.

```
R> multinomialConfidenceIntervals(transitionMatrix =
+   weatherFittedMLE$estimate@transitionMatrix,
+   countsTransitionMatrix = createSequenceMatrix(weathersOfDays))
```

```
$confidenceLevel
```

```
[1] 0.95
```

```
$lowerEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.2894737	0.2500000	0.1447368
rain	0.3636364	0.2545455	0.0000000
sunny	0.1250000	0.0000000	0.6363636

```
$upperEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.5262267	0.4867530	0.3814898
rain	0.6329951	0.5239042	0.2693588
sunny	0.3049268	0.1799268	0.8162905

The functions for fitting DTMC have mostly been rewritten in C++ using Rcpp Eddelbuettel (2013) since version 0.2.

It is also possible to fit a DTMC object from `matrix` or `data.frame` objects as shown in following code.

```
R> data(holson)
R> singleMc<-markovchainFit(data=holson[,2:12],name="holson")
```

The same applies for `markovchainList` (output length has been limited).

```
R> mcListFit<-markovchainListFit(data=holson[,2:6],name="holson")
R> mcListFit$estimate
```

```
holson  list of Markov chain(s)
Markovchain  1
Unnamed Markov chain
A 3 - dimensional discrete Markov Chain defined by the following states:
1, 2, 3
The transition matrix (by rows) is defined as follows:
      1      2      3
1 0.94609164 0.05390836 0.00000000
2 0.26356589 0.62790698 0.1085271
3 0.02325581 0.18604651 0.7906977
```

```
Markovchain  2
Unnamed Markov chain
A 3 - dimensional discrete Markov Chain defined by the following states:
1, 2, 3
The transition matrix (by rows) is defined as follows:
      1      2      3
1 0.9323410 0.0676590 0.00000000
2 0.2551724 0.5103448 0.2344828
3 0.0000000 0.0862069 0.9137931
...
```

Finally, given a `list` object, it is possible to fit a `markovchain` object or to obtain the raw transition matrix.

```
R> c1<-c("a","b","a","a","c","c","a")
R> c2<-c("b")
R> c3<-c("c","a","a","c")
R> c4<-c("b","a","b","a","a","c","b")
R> c5<-c("a","a","c",NA)
R> c6<-c("b","c","b","c","a")
R> mylist<-list(c1,c2,c3,c4,c5,c6)
R> mylistMc<-markovchainFit(data=mylist)
R> mylistMc
```

```
$estimate
MLE Fit
A 3 - dimensional discrete Markov Chain defined by the following states:
```

```

a, b, c
The transition matrix (by rows) is defined as follows:
      a      b      c
a 0.4 0.2000000 0.4000000
b 0.6 0.0000000 0.4000000
c 0.5 0.3333333 0.1666667

$standardError
      a      b      c
a 0.1549193 0.1264911 0.1549193
b 0.2190890 0.0000000 0.2190890
c 0.2041241 0.1924501 0.1521452

$confidenceLevel
[1] 0.95

$lowerEndpointMatrix
      a      b      c
a 0.1681803 0.05668215 0.16818033
b 0.2307243 0.00000000 0.11762077
c 0.1876163 0.09677141 0.03005337

$upperEndpointMatrix
      a      b      c
a 0.6873262 0.5098375 0.6873262
b 0.8823792 0.4344825 0.7692757
c 0.8123837 0.7000067 0.5635028

The same works for markovchainFitList.

R> markovchainListFit(data=mylist)

$estimate
  list of Markov chain(s)
Markovchain 1
Unnamed Markov chain
A 3 - dimensional discrete Markov Chain defined by the following states:
a, b, c
The transition matrix (by rows) is defined as follows:
      a      b      c
a 0.5 0.5 0.0
b 0.5 0.0 0.5
c 1.0 0.0 0.0

Markovchain 2
Unnamed Markov chain

```

A 3 - dimensional discrete Markov Chain defined by the following states:
...

If any transition contains NA, it will be ignored in the results as the above example showed.

5.4. Prediction

The n -step forward predictions can be obtained using the `predict` methods explicitly written for `markovchain` and `markovchainList` objects. The prediction is the mode of the conditional distribution of X_{t+1} given $X_t = s_j$, being s_j the last realization of the DTMC (homogeneous or semi-homogeneous).

Predicting from a markovchain object

The 3-days forward predictions from `markovchain` object can be generated as follows, assuming that the last two days were respectively “cloudy” and “sunny”.

```
R> predict(object = weatherFittedMLE$estimate, newdata = c("cloudy", "sunny"),
+         n.ahead = 3)

[1] "sunny" "sunny" "sunny"
```

Predicting from a markovchainList object

Given an initial two years health status, the 5-year ahead prediction of any CCRC guest is

```
R> predict(mcCCRC, newdata = c("H", "H"), n.ahead = 5)

[1] "H" "D" "D"
```

The prediction has stopped at time sequence since the underlying semi-homogeneous Markov chain has a length of four. In order to continue five years ahead, the `continue=TRUE` parameter setting makes the `predict` method keeping to use the last `markovchain` in the sequence list.

```
R> predict(mcCCRC, newdata = c("H", "H"), n.ahead = 5, continue = TRUE)

[1] "H" "D" "D" "D" "D"
```

5.5. Statistical Tests

In this section, we describe the statistical tests: assessing the Markov property (`verifyMarkovProperty`), the order (`assessOrder`), the time-homogeneity (`assessStationarity`) of a Markov chain sequence, and the divergence tests for empirically estimated transition matrices (`verifyEmpiricalToTheoretical`, `verifyHomogeneity`). Most of such tests are based on the χ^2 statistics; the Pearson and likelihood-ratio (G) versions of each test are provided, together with a Monte Carlo option

(`method = "simulation"`) that avoids the asymptotic χ^2 approximation altogether by simulating the null distribution of the G statistic directly from the fitted model. Test results are returned as objects of class `htest`, following R's standard convention for hypothesis tests, so that they print, and can be manipulated, like the output of `stats::chisq.test`. Relevant references are [Kullback *et al.* \(1962\)](#) and [Anderson and Goodman \(1957\)](#).

All such tests have been designed for small samples, since it is easy to detect departures from the Markov property as long as the sample size increases. The Monte Carlo option is particularly useful in this regime, since the asymptotic χ^2 approximation can be unreliable when expected counts are small; the simulated p -value is computed as $(1 + \#\{G^* \geq G\})/(B+1)$ over B parametric replicates drawn under the null hypothesis, which is a finite-sample-valid Monte Carlo test ([Davison and Hinkley 1997](#)). The accuracy of the statistical inference functions for very small samples is nonetheless an area of ongoing work.

Assessing the Markov property of a Markov chain sequence

The `verifyMarkovProperty` function verifies whether the Markov property holds for the given chain. The test implemented in the package looks at triplets of successive observations. If $x_1, x_2, \dots, x_N$ is a set of observations and n_{ijk} is the number of times t ($1 \leq t \leq N-2$) such that $x_t = i, x_{t+1} = j, x_{t+2} = k$, then, if the Markov property holds, n_{ijk} is expected to be close to $n_{ij}\hat{p}_{jk} = n_{ij}n_{jk}/n_j$, where $n_{ij} = \sum_k n_{ijk}$, $n_{jk} = \sum_i n_{ijk}$ and $n_j = \sum_k n_{jk}$. The corresponding Pearson and likelihood-ratio statistics are

$$X^2 = \sum_i \sum_j \sum_k \frac{(n_{ijk} - n_{ij}\hat{p}_{jk})^2}{n_{ij}\hat{p}_{jk}}, \quad G^2 = 2 \sum_i \sum_j \sum_k n_{ijk} \log \frac{n_{ijk}}{n_{ij}\hat{p}_{jk}},$$

both asymptotically $\chi^2(q)$ under the null hypothesis ([Anderson and Goodman 1957](#)). The number of degrees of freedom q is obtained by summing, over each present state j , the degrees of freedom of the corresponding (past $\times$ future) contingency table restricted to the past and future states that were actually observed together with j : $q = \sum_j (s_j - 1)(r_j - 1)$, where s_j is the number of past states i with $n_{ij} > 0$ and r_j is the number of future states k with $n_{jk} > 0$.

```
R> sample_sequence <- c("a", "b", "a", "a", "a", "a", "b", "a", "b", "a",
+                       "b", "a", "a", "b", "b", "b", "a")
R> verifyMarkovProperty(sample_sequence, method = "G")
```

Likelihood-ratio test

```
data: sample_sequence
G-squared = 0.59916, df = 2, p-value = 0.7411
```

Burkett, Scherer and Todd (2019) ([Burkett, Scherer, and Todd 2019](#)) study recurrent and persistent financial-market states. One of their published tests considers the transition into state 1 and reports the following counts, for which a standard Pearson test of independence gives a non-significant result:

```
R> burkett_counts <- matrix(
+   c(127, 59,
```

```

+      11,  7,
+      36, 20,
+      9,  4),
+      nrow = 4,
+      byrow = TRUE,
+      dimnames = list(
+        c("previous state 1", "previous state 2",
+          "previous state 4", "previous state 5"),
+        c("transition to state 1", "transition elsewhere")
+      )
+    )
R> chisq.test(burkett_counts, correct = FALSE)

```

Warning in `chisq.test(burkett_counts, correct = FALSE)`: L'approssimazione al Chi-quadrato potrebbe essere inesatta

Pearson's Chi-squared test

```

data:  burkett_counts
X-squared = 0.63611, df = 3, p-value = 0.8881

```

The published contingency table gives a non-significant Pearson test ($\chi^2 \approx 0.64$, 3 degrees of freedom, $p \approx 0.89$). It is included as an independent literature example of a non-rejection; it is not claimed to be numerically identical to `verifyMarkovProperty`, since the published test is a cell-wise test for a particular transition rather than the package's global first-order Markov-property test.

Assessing the order of a Markov chain sequence

The `assessOrder` function checks whether the given chain is of first order or of second order. For each possible present state, we construct a contingency table of the frequency of the future state for each past-to-present state transition, as shown in Table 6, and performs a Pearson χ^2 test of independence between past and future conditionally on the present state, summing the resulting statistics over all present states.

past	present	future	future
		a	b
a	a	2	2
b	a	2	2

Table 6: Contingency table to assess the order for the present state a.

As for `verifyMarkovProperty`, degrees of freedom are summed over present states using only the past and future states actually observed together with that present state, $q = \sum_j (s_j - 1)(r_j - 1)$, rather than a fixed $k(k - 1)^2$; the two coincide whenever every state is observed as both a past and a future state for every present state, and only differ – correctly reducing q

– when some transitions are unobserved. If the p-value is greater than the given significance level, we cannot reject the hypothesis that the sequence is of first order.

```
R> data(rain)
R> assessOrder(rain$rain)
```

Pearson's Chi-squared test for Markov order

```
data: rain$rain
X-squared = 26.096, df = 12, p-value = 0.0104
```

Assessing the stationarity and time-homogeneity of a Markov chain sequence

The `assessStationarity` function assesses whether the transition probabilities of the given chain change over time, i.e. whether

$$p_{ij}(t) = p_{ij} \text{ for all } t. \quad (22)$$

For each state, the observed sequence is split into `nblocks` contiguous blocks and a (block $\times$ destination state) contingency table of one-step-ahead transition counts is built, as shown in Table 7; a Pearson χ^2 test of homogeneity across blocks is then performed and the resulting statistics summed across states, with degrees of freedom adapted, state by state, to the blocks and destinations that were actually observed.

time (t)	probability of transition to a	probability of transition to b
1	0	1
2	0	1
.	.	.
.	.	.
.	.	.
16	0.44	0.56

Table 7: Contingency table to assess the stationarity of the state a.

```
R> assessStationarity(rain$rain, 10)
```

Pearson's Chi-squared test for time-homogeneity

```
data: rain$rain
X-squared = 110.02, df = 54, p-value = 1.05e-05
```

Tran and Carmichael (2012) ([Tran and Carmichael 2012](#)) apply likelihood-ratio tests to transition probabilities in a real contractor-payment application, and provide an example where the null hypothesis of time-homogeneous transition probabilities is rejected. Their application

also motivates explicit handling of *structural zeros*, i.e. transitions that are impossible under the state-space model, as opposed to *sampling zeros*, i.e. transitions that are possible but were simply not observed. The package allows structural zeros to be specified through the `structural.zeros` argument, so that they are excluded from the contingency tables rather than treated as evidence of homogeneity:

```
R> structural.zeros <- matrix(FALSE, 3, 3,
+                             dimnames = list(c("a", "b", "c"),
+                             c("a", "b", "c")))
R> structural.zeros["a", "a"] <- TRUE
R>
R> structural_sequence <- c(
+   "a", "b", "a", "c", "a", "b", "c", "a",
+   "b", "c", "b", "a", "c", "b", "a", "c",
+   "a", "b", "c", "a", "b", "c", "b", "a"
+ )
R>
R> assessStationarity(structural_sequence, nblocks = 4,
+                     structural.zeros = structural.zeros)
```

Pearson's Chi-squared test for time-homogeneity

```
data:  structural_sequence
X-squared = 1.7639, df = 15, p-value = 1
```

Divergence tests for empirically estimated transition matrices

This section discusses tests developed to verify whether:

1. An empirical transition matrix is consistent with a theoretical one.
2. Two or more empirical transition matrices belong to the same DTMC.

The first test is implemented by the `verifyEmpiricalToTheoretical` function. Denoting by f_{ij} the raw empirical transition count and by P_{ij} the theoretical (fully specified) transition probability, expected counts are constructed row-wise as $E_{ij} = f_{i+}P_{ij}$, where $f_{i+} = \sum_j f_{ij}$, and

$$G^2 = 2 \sum_{i=1}^r \sum_{j=1}^r f_{ij} \ln \frac{f_{ij}}{f_{i+}P_{ij}} \sim \chi^2(\sum_i (k_i - 1)),$$

where k_i is the number of states reachable with positive probability from i under the theoretical matrix; since P is fully specified rather than estimated, no degrees of freedom are lost to parameter estimation. The following example is taken from [Kullback et al. \(1962\)](#):

```

R> kullback_counts <- matrix(
+   c(51, 11, 8,
+     12, 31, 9,
+     6, 11, 10),
+   nrow = 3,
+   byrow = TRUE,
+   dimnames = list(c("a", "b", "c"), c("a", "b", "c")))
+ )
R>
R> kullback_P <- matrix(
+   c(5/8, 1/4, 1/8,
+     1/4, 1/2, 1/4,
+     1/4, 3/8, 3/8),
+   nrow = 3,
+   byrow = TRUE,
+   dimnames = list(c("a", "b", "c"), c("a", "b", "c")))
+ )
R>
R> kullback_mc <- new("markovchain", states = c("a", "b", "c"),
+   transitionMatrix = kullback_P)
R>
R> verifyEmpiricalToTheoretical(kullback_counts, kullback_mc, method = "G")

```

Likelihood-ratio test

```

data: kullback_counts
G-squared = 6.5184, df = 6, p-value = 0.3677

```

Kullback et al. report $G^2 = 6.551795$, $df = 6$ and $p = 0.3642899$ for this table. The historical `markovchain` implementation reproduced this published value by using column totals when constructing expected counts. For the row-wise multinomial null implemented by the current package – the standard construction for this kind of test, since each row of a transition matrix is itself a separate multinomial sample – expected counts are $E_{ij} = f_{i+}P_{ij}$, which gives $G^2 = 6.518384$ with the same $df = 6$ and the same non-rejection of the null hypothesis. The small numerical discrepancy with the published value is therefore a correction of the historical implementation rather than a change in the inferential conclusion.

The second test is implemented by the `verifyHomogeneity` function, inspired by (Kullback *et al.* 1962, section 9). Assuming that $i = 1, 2, \dots, s$ DTMC samples are available and that the cardinality of the state space is r , it verifies whether the s chains belong to the same unknown one. Pooling counts across samples for each state i and comparing each sample's destination distribution to the pooled one, the resulting statistic is asymptotically χ^2 with degrees of freedom summed, state by state, over the samples and destinations actually observed for that state. The following example generates two independent sequences from the same chain, for which the homogeneity hypothesis should not be rejected:

```

R> homogeneity_P <- matrix(

```

```

+   c(.70, .20, .10,
+     .20, .50, .30,
+     .10, .30, .60),
+   nrow = 3,
+   byrow = TRUE,
+   dimnames = list(c("a", "b", "c"), c("a", "b", "c"))
+ )
R>
R> homogeneity_mc <- new("markovchain", states = c("a", "b", "c"),
+   transitionMatrix = homogeneity_P)
R>
R> set.seed(123)
R> x1 <- rmarkovchain(500, homogeneity_mc, t0 = "a")
R> x2 <- rmarkovchain(500, homogeneity_mc, t0 = "a")
R>
R> verifyHomogeneity(list(x1, x2), method = "G")

```

Likelihood-ratio test

```

data: list(x1, x2)
G-squared = 1.8881, df = 6, p-value = 0.9297

```

For sparse data, the asymptotic χ^2 approximation underlying any of the tests above should be interpreted with care; the `method = "simulation"` option provides a Monte Carlo alternative that does not rely on the asymptotic approximation. `## Continuous-Time Markov Chains`

Intro

The **markovchain** package provides functionality for continuous time Markov chains (CTMCs). CTMCs are a generalization of discrete time Markov chains (DTMCs) in that we allow time to be continuous. We assume a finite state space S (for an infinite state space wouldn't fit in memory). We can think of CTMCs as Markov chains in which state transitions can happen at any time.

More formally, we would like our CTMCs to satisfy the following two properties:

- The Markov property - let $F_{X(s)}$ denote the information about X up to time s . Let $j \in S$ and $s \leq t$. Then, $P(X(t) = j | F_{X(s)}) = P(X(t) = j | X(s))$.
- Time homogeneity - $P(X(t) = j | X(s) = k) = P(X(t - s) = j | X(0) = k)$.

If both the above properties are satisfied, it is referred to as a time-homogeneous CTMC. If a transition occurs at time t , then $X(t)$ denotes the new state and $X(t) \neq X(t-)$.

Now, let $X(0) = x$ and let T_x be the time a transition occurs from this state. We are interested in the distribution of T_x . For $s, t \geq 0$, it can be shown that $P(T_x > s+t | T_x > s) = P(T_x > t)$

This is the memoryless property that only the exponential random variable exhibits. Therefore, the exponential distribution is therefore the relevant distribution, and each state $s \in S$

has an exponential holding parameter $\lambda(s)$. Since $ET_x = \frac{1}{\lambda(x)}$, the higher the rate $\lambda(x)$, the smaller the expected time of transitioning out of the state x .

However, specifying this parameter alone for each state would only paint an incomplete picture of our CTMC. To see why, consider a state x that may transition to either state y or z . The holding parameter enables us to predict when a transition may occur if we start off in state x , but tells us nothing about which state will be next.

To this end, we also need transition probabilities associated with the process, defined as follows (for $y \neq x$) - $p_{xy} = P(X(T_s) = y | X(0) = x)$. Note that $\sum_{y \neq x} p_{xy} = 1$. Let Q denote this transition matrix ($Q_{ij} = p_{ij}$). What is key here is that T_x and the state y are independent random variables. Let's define $\lambda(x, y) = \lambda(x)p_{xy}$

We now look at Kolmogorov's backward equation. Let's define $P_{ij}(t) = P(X(t) = j | X(0) = i)$ for $i, j \in S$. The backward equation is given by (it can be proved) $P_{ij}(t) = \delta_{ij}e^{-\lambda(i)t} + \int_0^t \lambda(i)e^{-\lambda(i)s} \sum_{k \neq i} Q_{ik}P_{kj}(t-s)ds$. Basically, the first term is non-zero if and only if $i = j$ and represents the probability that the first transition from state i occurs after time t . This would mean that at t , the state is still i . The second term accounts for any transitions that may occur before time t and denotes the probability that at time t , when the smoke clears, we are in state j .

This equation can be represented compactly as follows $P'(t) = AP(t)$ where A is the *generator* matrix.

$$A(i, j) = \begin{cases} \lambda(i, j) & \text{if } i \neq j \\ -\lambda(i) & \text{else.} \end{cases}$$

Observe that the sum of each row is 0. A CTMC can be completely specified by the generator matrix.

Stationary Distributions

The following theorem guarantees the existence of a unique stationary distribution for CTMCs. Note that $X(t)$ being irreducible and recurrent is the same as $X_n(t)$ being irreducible and recurrent.

Suppose that $X(t)$ is irreducible and recurrent. Then $X(t)$ has an invariant measure η , which is unique up to multiplicative factors. Moreover, for each $k \in S$, we have

$$\eta_k = \frac{\pi_k}{\lambda(k)}$$

where π is the unique invariant measure of the embedded discrete time Markov chain X_n . Finally, η satisfies

$$0 < \eta_j < \infty, \forall j \in S$$

and if $\sum_i \eta_i < \infty$ then η can be normalized to get a stationary distribution.

Estimation

Let the data set be $D = \{(s_0, t_0), (s_1, t_1), \dots, (s_{N-1}, t_{N-1})\}$ where $N = |D|$. Each s_i is a state from the state space S and during the time $[t_i, t_{i+1}]$ the chain is in state s_i . Let the

parameters be represented by $\theta = \{\lambda, P\}$ where λ is the vector of holding parameters for each state and P the transition matrix of the embedded discrete time Markov chain.

Then the probability is given by

$$Pr(D|\theta) \propto \lambda(s_0)e^{-\lambda(s_0)(t_1-t_0)} Pr(s_1|s_0) \cdot \dots \cdot \lambda(s_{N-2})e^{-\lambda(s_{N-2})(t_{N-1}-t_{N-2})} Pr(s_{N-1}|s_{N-2})$$

Let $n(j|i)$ denote the number of $i \rightarrow j$ transitions in D , and $n(i)$ the number of times s_i occurs in D . Let $t(s_i)$ denote the total time the chain spends in state s_i .

Then the MLEs are given by

$$\hat{\lambda}(s) = \frac{n(s)}{t(s)}, Pr(\hat{j}|i) = \frac{n(j|i)}{n(i)}$$

Expected Hitting Time

The package provides a function **ExpectedTime** to calculate average hitting time from one state to another. Let the final state be j , then for every state $i \in S$, where S is the set of all states and holding time $q_i > 0$ for every $i \neq j$. Assuming the conditions to be true, expected hitting time is equal to minimal non-negative solution vector p to the system of linear equations:

$$\begin{cases} p_k = 0 & k = j \\ -\sum_{l \in I} q_{kl} p_k = 1 & k \neq j \end{cases} \quad (23)$$

Probability at time t

The package provides a function **probabilityatT** to calculate probability of every state according to given **ctmc** object. Here we use Kolmogorov's backward equation $P(t) = P(0)e^{tQ}$ for $t \geq 0$ and $P(0) = I$. Here $P(t)$ is the transition function at time t . The value $P(t)[i][j]$ at time $P(t)$ describes the probability of the state at time t to be equal to j if it was equal to i at time $t = 0$. It takes care of the case when **ctmc** object has a generator represented by columns. If initial state is not provided, the function returns the whole transition matrix $P(t)$.

Examples

To create a CTMC object, you need to provide a valid generator matrix, say Q . The CTMC object has the following slots - states, generator, by row, name (look at the documentation object for further details). Consider the following example in which we aim to model the transition of a molecule from the σ state to the σ^* state. When in the former state, if it absorbs sufficient energy, it can make the jump to the latter state and remains there for some time before transitioning back to the original state. Let us model this by a CTMC:

```
R> energyStates <- c("sigma", "sigma_star")
R> byRow <- TRUE
R> gen <- matrix(data = c(-3, 3,
```

```
+           1, -1), nrow = 2,
+           byrow = byRow, dimnames = list(energyStates, energyStates))
R> molecularCTMC <- new("ctmc", states = energyStates,
+           byrow = byRow, generator = gen,
+           name = "Molecular Transition Model")
```

To generate random CTMC transitions, we provide an initial distribution of the states. This must be in the same order as the dimnames of the generator. The output can be returned either as a list or a data frame.

```
R> statesDist <- c(0.8, 0.2)
R> rctmc(n = 3, ctmc = molecularCTMC, initDist = statesDist, out.type = "df", include.T0 = FALSE)

      states      time
1 sigma_star 0.0625779551143447
2      sigma  0.39382385184532
3 sigma_star  0.685406352703829
```

n represents the number of samples to generate. There is an optional argument T for `rctmc`. It represents the time of termination of the simulation. To use this feature, set n to a very high value, say `Inf` (since we do not know the number of transitions before hand) and set T accordingly.

```
R> statesDist <- c(0.8, 0.2)
R> rctmc(n = Inf, ctmc = molecularCTMC, initDist = statesDist, T = 2)

[[1]]
[1] "sigma"      "sigma_star"

[[2]]
[1] 0.0000000 0.2738256
```

To obtain the stationary distribution simply invoke the `steadyStates` function

```
R> steadyStates(molecularCTMC)

      sigma sigma_star
[1,]  0.25      0.75
```

For fitting, use the `ctmcFit` function. It returns the MLE values for the parameters along with the confidence intervals.

```
R> data <- list(c("a", "b", "c", "a", "b", "a", "c", "b", "c"),
+           c(0, 0.8, 2.1, 2.4, 4, 5, 5.9, 8.2, 9))
R> ctmcFit(data)
```

```

$estimate
An object of class "ctmc"
Slot "states":
[1] "a" "b" "c"

Slot "byrow":
[1] TRUE

Slot "generator":
      a      b      c
a -0.9090909  0.6060606  0.3030303
b  0.3225806 -0.9677419  0.6451613
c  0.3846154  0.3846154 -0.7692308

Slot "name":
[1] ""

$errors
$errors$dtmcConfidenceInterval
$errors$dtmcConfidenceInterval$confidenceLevel
[1] 0.95

$errors$dtmcConfidenceInterval$lowerEndpointMatrix
      a      b      c
a 0.00000000 0.20765960 0.06149194
b 0.06149194 0.00000000 0.20765960
c 0.09453121 0.09453121 0.00000000

$errors$dtmcConfidenceInterval$upperEndpointMatrix
      a      b      c
a 0.5614970 0.9385081 0.7923404
b 0.7923404 0.5614970 0.9385081
c 0.9054688 0.9054688 0.6576198

$errors$lambdaConfidenceInterval
$errors$lambdaConfidenceInterval$lowerEndpointVector
[1] 0 0 0

$errors$lambdaConfidenceInterval$upperEndpointVector
[1] 1.937805 2.062825 1.835311

```

One approach to obtain the generator matrix is to apply the `logm` function from the **expm** package on a transition matrix. Numeric issues arise, see [Israel, Rosenthal, and Wei \(2001\)](#). For example, applying the standard `method` ('Higham08') on `mcWeather` raises an error, whilst the alternative method (eigenvalue decomposition) is OK. The following code estimates the

generator matrix of the mcWeather transition matrix.

```
R> mcWeatherQ <- expm::logm(mcWeather@transitionMatrix,method='Eigen')
R> mcWeatherQ
```

```
      sunny      cloudy      rain
sunny -0.863221  2.428723 -1.565502
cloudy 4.284592 -20.116312 15.831720
rain   -4.414019 24.175251 -19.761232
```

Therefore, the “half - day” transition probability for mcWeather DTMC is

```
R> mcWeatherHalfDayTM <- expm::expm(mcWeatherQ*.5)
R> mcWeatherHalfDay <- new("markovchain",transitionMatrix=mcWeatherHalfDayTM,name="Half Day")
R> mcWeatherHalfDay
```

Half Day Weather Transition Matrix

A 3 - dimensional discrete Markov Chain defined by the following states:

sunny, cloudy, rain

The transition matrix (by rows) is defined as follows:

```
      sunny      cloudy      rain
sunny 0.81598647 0.1420068 0.04200677
cloudy 0.21970167 0.4401492 0.34014916
rain   0.07063048 0.5146848 0.41468476
```

The **ctmcd** package ([Pfeuffer 2017](#)) provides various functions to estimate the generator matrix (GM) of a CTMC process using different methods. The following code provides a way to join **markovchain** and **ctmcd** computations.

```
R> if(requireNamespace(package='ctmcd', quietly = TRUE)) {
+ require(ctmcd)
+ require(expm)
+ #defines a function to transform a GM into a TM
+ gm_to_markovchain<-function(object, t=1) {
+   if(!(class(object) %in% c("gm","matrix","Matrix"))){
+     stop("Error! Expecting either a matrix or a gm object")
+   }
+   if ( class(object) %in% c("matrix","Matrix")) generator_matrix<-object else generator_matrix<-gm_to_matrix(object)
+   #must add importClassesFrom("markovchain",markovchain) in the NAMESPACE
+   #must add importFrom(expm, "expm")
+   transitionMatrix<-expm(generator_matrix*t)
+   out<-as(transitionMatrix,"markovchain")
+   return(out)
+ }
+ #loading ctmcd dataset
+ data(tm_abs)
+ gm0=matrix(1,8,8) #initializing
```

```

+ diag(gm0)=0
+ diag(gm0)=-rowSums(gm0)
+ gm0[8,]=0
+ gmem=gm(tm_abs,te=1,method="EM",gmguess=gm0) #estimating GM
+ mc_at_2=gm_to_markovchain(object=gmem, t=2) #converting to TM at time 2
+ } else {
+   warning('package ctmc unavailable')
+ }

```

Caricamento del pacchetto richiesto: ctmc

Caricamento del pacchetto richiesto: expm

Caricamento pacchetto: 'expm'

Il seguente oggetto è mascherato da 'package:Matrix':

expm

5.6. Pseudo - Bayesian Estimation

Hu, Kiesel, and Perraudin (2002) show an empirical quasi-Bayesian method to estimate transition matrices, given an empirical $\hat{P}$ transition matrix (estimated using the classical approach) and an a priori estimate Q . In particular, each row of the matrix is estimated using the linear combination $\alpha \cdot Q + (1 - \alpha) \cdot P$, where α is defined for each row as Equation 24 shows

$$\begin{cases} \hat{\alpha}_i = \frac{\hat{K}_i}{v(i) + \hat{K}_i} \\ \hat{K}_i = \frac{v(i)^2 - \sum_j Y_{ij}^2}{\sum_j (Y_{ij} - v(i) * q_{ij})^2} \end{cases} \quad (24)$$

The following code returns the pseudo Bayesian estimate of the transition matrix:

```

R> pseudoBayesEstimator <- function(raw, apriori){
+   v_i <- rowSums(raw)
+   K_i <- numeric(nrow(raw))
+   sumSquaredY <- rowSums(raw^2)
+   #get numerator
+   K_i_num <- v_i^2 - sumSquaredY
+   #get denominator
+   VQ <- matrix(0, nrow= nrow(apriori), ncol=ncol(apriori))
+   for (i in 1:nrow(VQ)) {
+     VQ[i,]<-v_i[i]*apriori[i,]
+   }
+
+   K_i_den<-rowSums((raw - VQ)^2)

```

```

+
+   K_i <- K_i_num/K_i_den
+
+   #get the alpha vector
+   alpha <- K_i / (v_i+K_i)
+
+   #empirical transition matrix
+   Emp<-raw/rowSums(raw)
+
+   #get the estimate
+   out<-matrix(0, nrow= nrow(raw),ncol=ncol(raw))
+   for (i in 1:nrow(out)) {
+     out[i,]<-alpha[i]*apriori[i,]+(1-alpha[i])*Emp[i,]
+   }
+   return(out)
+ }

```

We then apply it to the weather example:

```

R> trueMc<-as(matrix(c(0.1, .9,.7,.3),nrow = 2, byrow = 2),"markovchain")
R> aprioriMc<-as(matrix(c(0.5, .5,.5,.5),nrow = 2, byrow = 2),"markovchain")
R>
R> smallSample<-rmarkovchain(n=20,object = trueMc)
R> smallSampleRawTransitions<-createSequenceMatrix(stringchar = smallSample)
R> pseudoBayesEstimator(
+   raw = smallSampleRawTransitions,
+   apriori = aprioriMc@transitionMatrix
+ ) - trueMc@transitionMatrix

           s1          s2
s1 -0.1000000  0.1000000
s2  0.0806874 -0.0806874

R> biggerSample<-rmarkovchain(n=60,object = trueMc)
R> biggerSampleRawTransitions<-createSequenceMatrix(stringchar = biggerSample)
R> pseudoBayesEstimator(
+   raw = biggerSampleRawTransitions,
+   apriori = aprioriMc@transitionMatrix
+ ) - trueMc@transitionMatrix

           s1          s2
s1 -0.1000000  0.1000000
s2 -0.1842829  0.1842829

R> bigSample<-rmarkovchain(n=100,object = trueMc)
R> bigSampleRawTransitions<-createSequenceMatrix(stringchar = bigSample)

```

```

R> pseudoBayesEstimator(
+   raw = bigSampleRawTransitions,
+   apriori = aprioriMc@transitionMatrix
+ ) - trueMc@transitionMatrix

           s1           s2
s1 -0.02491803 0.02491803
s2 -0.01493931 0.01493931

```

5.7. Bayesian Estimation

The **markovchain** package provides functionality for maximum a posteriori (MAP) estimation of the chain parameters (at the time of writing this document, only first order models are supported) by Bayesian inference. It also computes the probability of observing a new data set, given a (different) data set. This vignette provides the mathematical description for the methods employed by the package.

Notation and set-up

The data is denoted by D , the model parameters (transition matrix) by θ . The object of interest is $P(\theta|D)$ (posterior density). $\mathcal{A}$ represents an alphabet class, each of whose members represent a state of the chain. Therefore

$$D = s_0 s_1 \dots s_{N-1}, s_t \in \mathcal{A}$$

where N is the length of the data set. Also,

$$\theta = \{p(s|u), s \in \mathcal{A}, u \in \mathcal{A}\}$$

where $\sum_{s \in \mathcal{A}} p(s|u) = 1$ for each $u \in \mathcal{A}$.

Our objective is to find θ which maximizes the posterior. That is, if our solution is denoted by $\hat{\theta}$, then

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} P(\theta|D)$$

where the search space is the set of right stochastic matrices of dimension $|\mathcal{A}| \times |\mathcal{A}|$.

$n(u, s)$ denotes the number of times the word us occurs in D and $n(u) = \sum_{s \in \mathcal{A}} n(u, s)$. The hyper-parameters are similarly denoted by $\alpha(u, s)$ and $\alpha(u)$ respectively.

Methods

Given D , its likelihood conditioned on the observed initial state in D is given by

$$P(D|\theta) = \prod_{s \in \mathcal{A}} \prod_{u \in \mathcal{A}} p(s|u)^{n(u,s)}$$

Conjugate priors are used to model the prior $P(\theta)$. The reasons are two fold:

1. Exact expressions can be derived for the MAP estimates, expectations and even variances
2. Model order selection/comparison can be implemented easily (available in a future release of the package)

The hyper-parameters determine the form of the prior distribution, which is a product of Dirichlet distributions

$$P(\theta) = \prod_{u \in \mathcal{A}} \left\{ \frac{\Gamma(\alpha(u))}{\prod_{s \in \mathcal{A}} \Gamma(\alpha(u, s))} \prod_{s \in \mathcal{A}} p(s|u)^{\alpha(u, s)-1} \right\}$$

where $\Gamma(\cdot)$ is the Gamma function. The hyper-parameters are specified using the `hyperparam` argument in the `markovchainFit` function. If this argument is not specified, then a default value of 1 is assigned to each hyper-parameter resulting in the prior distribution of each chain parameter to be uniform over $[0, 1]$.

Given the likelihood and the prior as described above, the evidence $P(D)$ is simply given by

$$P(D) = \int P(D|\theta)P(\theta)d\theta$$

which simplifies to

$$P(D) = \prod_{u \in \mathcal{A}} \left\{ \frac{\Gamma(\alpha(u))}{\prod_{s \in \mathcal{A}} \Gamma(\alpha(u, s))} \frac{\prod_{s \in \mathcal{A}} \Gamma(n(u, s) + \alpha(u, s))}{\Gamma(\alpha(u) + n(u))} \right\}$$

Using Bayes' theorem, the posterior now becomes (thanks to the choice of conjugate priors)

$$P(\theta|D) = \prod_{u \in \mathcal{A}} \left\{ \frac{\Gamma(n(u) + \alpha(u))}{\prod_{s \in \mathcal{A}} \Gamma(n(u, s) + \alpha(u, s))} \prod_{s \in \mathcal{A}} p(s|u)^{n(u, s) + \alpha(u, s) - 1} \right\}$$

Since this is again a product of Dirichlet distributions, the marginal distribution of a particular parameter $P(s|u)$ of our chain is given by

$$P(s|u) \sim \text{Beta}(n(u, s) + \alpha(u, s), n(u) + \alpha(u) - n(u, s) - \alpha(u, s))$$

Thus, the MAP estimate $\hat{\theta}$ is given by

$$\hat{\theta} = \left\{ \frac{n(u, s) + \alpha(u, s) - 1}{n(u) + \alpha(u) - |\mathcal{A}|}, s \in \mathcal{A}, u \in \mathcal{A} \right\}$$

The function also returns the expected value, given by

$$\text{E}_{\text{post}} p(s|u) = \left\{ \frac{n(u, s) + \alpha(u, s)}{n(u) + \alpha(u)}, s \in \mathcal{A}, u \in \mathcal{A} \right\}$$

The variance is given by

$$\text{Var}_{\text{post}} p(s|u) = \frac{n(u, s) + \alpha(u, s)}{(n(u) + \alpha(u))^2} \frac{n(u) + \alpha(u) - n(u, s) - \alpha(u, s)}{n(u) + \alpha(u) + 1}$$

The square root of this quantity is the standard error, which is returned by the function. The credible intervals are constructed as equal-tailed intervals by computing the inverse of the beta integral.

Predictive distribution

Given the old data set, the probability of observing new data is $P(D'|D)$ where D' is the new data set. Let $m(u, s), m(u)$ denote the corresponding counts for the new data. Then,

$$P(D'|D) = \int P(D'|\theta)P(\theta|D)d\theta$$

We already know the expressions for both quantities in the integral and it turns out to be similar to evaluating the evidence

$$P(D'|D) = \prod_{u \in \mathcal{A}} \left\{ \frac{\Gamma(\alpha(u))}{\prod_{s \in \mathcal{A}} \Gamma(\alpha(u, s))} \frac{\prod_{s \in \mathcal{A}} \Gamma(n(u, s) + m(u, s) + \alpha(u, s))}{\Gamma(\alpha(u) + n(u) + m(u))} \right\}$$

Choosing the hyper-parameters

The hyper parameters model the shape of the parameters' prior distribution. These must be provided by the user. The package offers functionality to translate a given prior belief transition matrix into the hyper-parameter matrix. It is assumed that this belief matrix corresponds to the mean value of the parameters. Since the relation

$$E_{\text{prior}}p(s|u) = \frac{\alpha(u, s)}{\alpha(u)}$$

holds, the function accepts as input the belief matrix as well as a scaling vector (serves as a proxy for $\alpha(\cdot)$) and proceeds to compute $\alpha(\cdot, \cdot)$.

Alternatively, the function accepts a data sample and infers the hyper-parameters from it. Since the mode of a parameter (with respect to the prior distribution) is proportional to one less than the corresponding hyper-parameter, we set

$$\alpha(u, s) - 1 = m(u, s)$$

where $m(u, s)$ is the $u \rightarrow s$ transition count in the data sample. This is regarded as a 'fake count' which helps $\alpha(u, s)$ to reflect knowledge of the data sample.

Usage and examples

```
R> weatherStates <- c("sunny", "cloudy", "rain")
R> byRow <- TRUE
R> weatherMatrix <- matrix(data = c(0.7, 0.2, 0.1,
+                               0.3, 0.4, 0.3,
+                               0.2, 0.4, 0.4),
+                           byrow = byRow, nrow = 3,
+                           dimnames = list(weatherStates, weatherStates))
```

```
R> mcWeather <- new("markovchain", states = weatherStates,
+                   byrow = byRow, transitionMatrix = weatherMatrix,
+                   name = "Weather")
R> weathersOfDays <- rmarkovchain(n = 365, object = mcWeather, t0 = "sunny")
```

For the purpose of this section, we shall continue to use the weather of days example introduced in the main vignette of the package (reproduced above for convenience).

Let us invoke the fit function to estimate the MAP parameters with 92% confidence bounds and hyper-parameters as shown below, based on the first 200 days of the weather data. Additionally, let us find out what the probability is of observing the weather data for the next 165 days. The usage would be as follows

```
R> hyperMatrix<-matrix(c(1, 1, 2,
+                       3, 2, 1,
+                       2, 2, 3),
+                      nrow = 3, byrow = TRUE,
+                      dimnames = list(weatherStates,weatherStates))
R> markovchainFit(weathersOfDays[1:200], method = "map",
+                confidencelevel = 0.92, hyperparam = hyperMatrix)
```

\$estimate

Bayesian Fit

A 3 - dimensional discrete Markov Chain defined by the following states:

cloudy, rain, sunny

The transition matrix (by rows) is defined as follows:

	cloudy	rain	sunny
cloudy	0.2923077	0.33846154	0.3692308
rain	0.5365854	0.31707317	0.1463415
sunny	0.2277228	0.03960396	0.7326733

\$expectedValue

	cloudy	rain	sunny
cloudy	0.2941176	0.33823529	0.3676471
rain	0.5227273	0.31818182	0.1590909
sunny	0.2307692	0.04807692	0.7211538

\$standardError

	cloudy	rain	sunny
cloudy	0.05485322	0.05695564	0.05804580
rain	0.07445856	0.06943297	0.05452441
sunny	0.04111711	0.02087731	0.04376245

\$confidenceInterval

\$confidenceInterval\$confidenceLevel

[1] 0.92

```
$confidenceInterval$lowerEndpointMatrix
```

```
      cloudy      rain      sunny
cloudy 0.2024315 0.24192322 0.26880381
rain   0.3917409 0.20261227 0.07519196
sunny  0.1624958 0.01813559 0.64169687
```

```
$confidenceInterval$upperEndpointMatrix
```

```
      cloudy      rain      sunny
cloudy 0.3941775 0.4411289 0.4718760
rain   0.6522800 0.4452100 0.2643798
sunny  0.3061880 0.0898865 0.7947391
```

```
$logLikelihood
```

```
[1] -169.945
```

```
R> predictiveDistribution(weathersOfDays[1:200],
+                          weathersOfDays[201:365], hyperparam = hyperMatrix)
```

```
[1] -154.6281
```

The results should not change after permuting the dimensions of the matrix.

```
R> hyperMatrix2<- hyperMatrix[c(2,3,1), c(2,3,1)]
R> markovchainFit(weathersOfDays[1:200], method = "map",
+                 confidencelevel = 0.92, hyperparam = hyperMatrix2)
```

```
$estimate
```

```
Bayesian Fit
```

```
A 3 - dimensional discrete Markov Chain defined by the following states:
```

```
cloudy, rain, sunny
```

```
The transition matrix (by rows) is defined as follows:
```

```
      cloudy      rain      sunny
cloudy 0.2923077 0.33846154 0.3692308
rain   0.5365854 0.31707317 0.1463415
sunny  0.2277228 0.03960396 0.7326733
```

```
$expectedValue
```

```
      cloudy      rain      sunny
cloudy 0.2941176 0.33823529 0.3676471
rain   0.5227273 0.31818182 0.1590909
sunny  0.2307692 0.04807692 0.7211538
```

```
$standardError
```

	cloudy	rain	sunny
cloudy	0.05485322	0.05695564	0.05804580
rain	0.07445856	0.06943297	0.05452441
sunny	0.04111711	0.02087731	0.04376245

```
$confidenceInterval
```

```
$confidenceInterval$confidenceLevel
```

```
[1] 0.92
```

```
$confidenceInterval$lowerEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.2024315	0.24192322	0.26880381
rain	0.3917409	0.20261227	0.07519196
sunny	0.1624958	0.01813559	0.64169687

```
$confidenceInterval$upperEndpointMatrix
```

	cloudy	rain	sunny
cloudy	0.3941775	0.4411289	0.4718760
rain	0.6522800	0.4452100	0.2643798
sunny	0.3061880	0.0898865	0.7947391

```
$logLikelihood
```

```
[1] -169.945
```

```
R> predictiveDistribution(weathersOfDays[1:200],
+                          weathersOfDays[201:365], hyperparam = hyperMatrix2)
```

```
[1] -154.6281
```

Note that the predictive probability is very small. However, this can be useful when comparing model orders. Suppose we have an idea of the (prior) transition matrix corresponding to the expected value of the parameters, and have a data set from which we want to deduce the MAP estimates. We can infer the hyper-parameters from this known transition matrix itself, and use this to obtain our MAP estimates.

```
R> inferHyperparam(transMatr = weatherMatrix, scale = c(10, 10, 10))
```

```
$scaledInference
```

	cloudy	rain	sunny
cloudy	4	3	3
rain	4	4	2
sunny	2	1	7

Alternatively, we can use a data sample to infer the hyper-parameters.

```
R> inferHyperparam(data = weathersOfDays[1:15])
```

```
$dataInference
      cloudy rain sunny
cloudy      2   3    2
rain        3   3    3
sunny       2   2    3
```

In order to use the inferred hyper-parameter matrices, we do

```
R> hyperMatrix3 <- inferHyperparam(transMatr = weatherMatrix,
+                                 scale = c(10, 10, 10))
R> hyperMatrix3 <- hyperMatrix3$scaledInference
R> hyperMatrix4 <- inferHyperparam(data = weathersOfDays[1:15])
R> hyperMatrix4 <- hyperMatrix4$dataInference
```

Now we can safely use `hyperMatrix3` and `hyperMatrix4` with `markovchainFit` (in the `hyperparam` argument).

Supposing we don't provide any hyper-parameters, then the prior is uniform. This is the same as maximum likelihood.

```
R> data(preproglucacon)
R> preproglucacon <- preproglucacon[[2]][1:200]
R> MLEest <- markovchainFit(preproglucacon, method = "mle")
R> MAPest <- markovchainFit(preproglucacon, method = "map")
R> MLEest$estimate
```

MLE Fit

A 4 - dimensional discrete Markov Chain defined by the following states:

A, C, G, T

The transition matrix (by rows) is defined as follows:

	A	C	G	T
A	0.4084507	0.07042254	0.2112676	0.3098592
C	0.3809524	0.14285714	0.0952381	0.3809524
G	0.3414634	0.07317073	0.1707317	0.4146341
T	0.3181818	0.15151515	0.2424242	0.2878788

```
R> MAPest$estimate
```

Bayesian Fit

A 4 - dimensional discrete Markov Chain defined by the following states:

A, C, G, T

The transition matrix (by rows) is defined as follows:

	A	C	G	T
A	0.4084507	0.07042254	0.2112676	0.3098592
C	0.3809524	0.14285714	0.0952381	0.3809524
G	0.3414634	0.07317073	0.1707317	0.4146341
T	0.3181818	0.15151515	0.2424242	0.2878788

6. Applications

This section shows applications of DTMC in various fields.

6.1. Weather forecasting

Markov chains provide a simple model to predict the next day's weather given the current meteorological condition. The first application herewith shown is the “Land of Oz example” from J. G. Kemeny, J. L. Snell, and G. L. Thompson (1974), the second is the “Alofi Island Rainfall” from P. J. Avery and D. A. Henderson (1999).

Land of Oz

The Land of Oz is acknowledged not to have ideal weather conditions at all: the weather is snowy or rainy very often and, once more, there are never two nice days in a row. Consider three weather states: rainy, nice and snowy. Let the transition matrix be as in the following:

```
R> mcWP <- new("markovchain", states = c("rainy", "nice", "snowy"),
+           transitionMatrix = matrix(c(0.5, 0.25, 0.25,
+                                     0.5, 0, 0.5,
+                                     0.25, 0.25, 0.5), byrow = T, nrow = 3))
```

Given that today it is a nice day, the corresponding stochastic row vector is $w_0 = (0, 1, 0)$ and the forecast after 1, 2 and 3 days are given by

```
R> W0 <- t(as.matrix(c(0, 1, 0)))
R> W1 <- W0 * mcWP; W1
```

```
      rainy nice snowy
[1,]  0.5    0   0.5
```

```
R> W2 <- W0 * (mcWP ^ 2); W2
```

```
      rainy nice snowy
[1,] 0.375 0.25 0.375
```

```
R> W3 <- W0 * (mcWP ^ 3); W3
```

```
      rainy   nice   snowy
[1,] 0.40625 0.1875 0.40625
```

As can be seen from w_1 , if in the Land of Oz today is a nice day, tomorrow it will rain or snow with probability 1. One week later, the prediction can be computed as

```
R> W7 <- W0 * (mcWP ^ 7)
R> W7
```

```

      rainy      nice      snowy
[1,] 0.4000244 0.1999512 0.4000244

```

The steady state of the chain can be computed by means of the `steadyStates` method.

```

R> q <- steadyStates(mcWP)
R> q

```

```

      rainy nice snowy
[1,]   0.4   0.2   0.4

```

Note that, from the seventh day on, the predicted probabilities are substantially equal to the steady state of the chain and they don't depend from the starting point, as the following code shows.

```

R> R0 <- t(as.matrix(c(1, 0, 0)))
R> R7 <- R0 * (mcWP ^ 7); R7

```

```

      rainy      nice      snowy
[1,] 0.4000244 0.2000122 0.3999634

```

```

R> S0 <- t(as.matrix(c(0, 0, 1)))
R> S7 <- S0 * (mcWP ^ 7); S7

```

```

      rainy      nice      snowy
[1,] 0.3999634 0.2000122 0.4000244

```

Alofi Island Rainfall

Alofi Island daily rainfall data were recorded from January 1st, 1987 until December 31st, 1989 and classified into three states: “0” (no rain), “1-5” (from non zero until 5 mm) and “6+” (more than 5mm). The corresponding dataset is provided within the **markovchain** package.

```

R> data("rain", package = "markovchain")
R> table(rain$rain)

```

```

  0 1-5 6+
548 295 253

```

The underlying transition matrix is estimated as follows.

```

R> mcAlofi <- markovchainFit(data = rain$rain, name = "Alofi MC")$estimate
R> mcAlofi

```

Alofi MC

A 3 - dimensional discrete Markov Chain defined by the following states:

0, 1-5, 6+

The transition matrix (by rows) is defined as follows:

	0	1-5	6+
0	0.6605839	0.2299270	0.1094891
1-5	0.4625850	0.3061224	0.2312925
6+	0.1976285	0.3122530	0.4901186

The long term daily rainfall distribution is obtained by means of the `steadyStates` method.

```
R> steadyStates(mcAlofi)
```

	0	1-5	6+
[1,]	0.5008871	0.2693656	0.2297473

6.2. Finance and Economics

Other relevant applications of DTMC can be found in Finance and Economics.

Finance

Credit ratings transitions have been successfully modeled with discrete time Markov chains. Some rating agencies publish transition matrices that show the empirical transition probabilities across credit ratings. The example that follows comes from **CreditMetrics** R package (Wittmann 2007), carrying Standard & Poor's published data.

```
R> rc <- c("AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D")
R> creditMatrix <- matrix(
+   c(90.81, 8.33, 0.68, 0.06, 0.08, 0.02, 0.01, 0.01,
+     0.70, 90.65, 7.79, 0.64, 0.06, 0.13, 0.02, 0.01,
+     0.09, 2.27, 91.05, 5.52, 0.74, 0.26, 0.01, 0.06,
+     0.02, 0.33, 5.95, 85.93, 5.30, 1.17, 1.12, 0.18,
+     0.03, 0.14, 0.67, 7.73, 80.53, 8.84, 1.00, 1.06,
+     0.01, 0.11, 0.24, 0.43, 6.48, 83.46, 4.07, 5.20,
+     0.21, 0, 0.22, 1.30, 2.38, 11.24, 64.86, 19.79,
+     0, 0, 0, 0, 0, 0, 0, 100
+   )/100, 8, 8, dimnames = list(rc, rc), byrow = TRUE)
```

It is easy to convert such matrices into `markovchain` objects and to perform some analyses

```
R> creditMc <- new("markovchain", transitionMatrix = creditMatrix,
+   name = "S&P Matrix")
R> absorbingStates(creditMc)
```

```
[1] "D"
```

Economics

For a recent application of **markovchain** in Economic, see [Jacob \(2014\)](#).

A dynamic system generates two kinds of economic effects ([Bard 2000](#)):

1. those incurred when the system is in a specified state, and
2. those incurred when the system makes a transition from one state to another.

Let the monetary amount of being in a particular state be represented as a m-dimensional column vector c^S , while let the monetary amount of a transition be embodied in a C^R matrix in which each component specifies the monetary amount of going from state i to state j in a single step. Henceforth, Equation (25) represents the monetary of being in state i.

$$c_i = c_i^S + \sum_{j=1}^m C_{ij}^R p_{ij}. \quad (25)$$

Let $\bar{c} = [c_i]$ and let e_i be the vector valued 1 in the initial state and 0 in all other, then, if f_n is the random variable representing the economic return associated with the stochastic process at time n , Equation (26) holds:

$$E[f_n(X_n) | X_0 = i] = e_i P^n \bar{c}. \quad (26)$$

The following example assumes that a telephone company models the transition probabilities between customer/non-customer status by matrix P and the cost associated to states by matrix M .

```
R> statesNames <- c("customer", "non customer")
R> P <- markovchain::zeros(2); P[1, 1] <- .9; P[1, 2] <- .1; P[2, 2] <- .95; P[2, 1] <- .
R> rownames(P) <- statesNames; colnames(P) <- statesNames
R> mcP <- new("markovchain", transitionMatrix = P, name = "Telephone company")
R> M <- markovchain::zeros(2); M[1, 1] <- -20; M[1, 2] <- -30; M[2, 1] <- -40; M[2, 2] <-
```

If the average revenue for existing customer is +100, the cost per state is computed as follows.

```
R> c1 <- 100 + conditionalDistribution(mcP, state = "customer") %*% M[1,]
R> c2 <- 0 + conditionalDistribution(mcP, state = "non customer") %*% M[2,]
```

For an existing customer, the expected gain (loss) at the fifth year is given by the following code.

```
R> as.numeric((c(1, 0)* mcP ^ 5) %*% (as.vector(c(c1, c2))))

[1] 48.96009
```

6.3. Actuarial science

Markov chains are widely applied in the field of actuarial science. Two classical applications are policyholders' distribution across Bonus Malus classes in Motor Third Party Liability (MTPL) insurance (Section 6.3.1) and health insurance pricing and reserving (Section 6.3.2).

MPTL Bonus Malus

Bonus Malus (BM) contracts grant the policyholder a discount (enworsen) as a function of the number of claims in the experience period. The discount (enworsen) is applied on a premium that already allows for known (a priori) policyholder characteristics (Denuit, Maréchal, Pitrebois, and Walhin 2007) and it usually depends on vehicle, territory, the demographic profile of the policyholder, and policy coverage deep (deductible and policy limits). Since the proposed BM level depends on the claim on the previous period, it can be modeled by a discrete Markov chain. A very simplified example follows. Assume a BM scale from 1 to 5, where 4 is the starting level. The evolution rules are shown in Equation 27:

$$bm_{t+1} = \max(1, bm_t - 1) * (\tilde{N} = 0) + \min(5, bm_t + 2 * \tilde{N}) * (\tilde{N} \geq 1). \quad (27)$$

The number of claim $\tilde{N}$ is a random variable that is assumed to be Poisson distributed.

```
R> getBonusMalusMarkovChain <- function(lambda) {
+   bmMatr <- markovchain::zeros(5)
+   bmMatr[1, 1] <- dpois(x = 0, lambda)
+   bmMatr[1, 3] <- dpois(x = 1, lambda)
+   bmMatr[1, 5] <- 1 - ppois(q = 1, lambda)
+
+   bmMatr[2, 1] <- dpois(x = 0, lambda)
+   bmMatr[2, 4] <- dpois(x = 1, lambda)
+   bmMatr[2, 5] <- 1 - ppois(q = 1, lambda)
+
+   bmMatr[3, 2] <- dpois(x = 0, lambda)
+   bmMatr[3, 5] <- 1 - dpois(x=0, lambda)
+
+   bmMatr[4, 3] <- dpois(x = 0, lambda)
+   bmMatr[4, 5] <- 1 - dpois(x = 0, lambda)
+
+   bmMatr[5, 4] <- dpois(x = 0, lambda)
+   bmMatr[5, 5] <- 1 - dpois(x = 0, lambda)
+   stateNames <- as.character(1:5)
+   out <- new("markovchain", transitionMatrix = bmMatr,
+             states = stateNames, name = "BM Matrix")
+   return(out)
+ }
```

Assuming that the a-priori claim frequency per car-year is 0.05 in the class (being the class the group of policyholders that share the same common characteristics), the underlying BM transition matrix and its underlying steady state are as follows.

Now we build a function that returns the transition during the $t + 1$ th year, assuming that the subject has attained year t .

```
R> possibleStates<-c("A","I","D")
R> getMc4Age<-function(age) {
+   transitionsAtAge<-ltcDemo[ltcDemo$age==age,]
+
+   myTransMatr<-matrix(0, nrow=3,ncol = 3,
+                        dimnames = list(possibleStates, possibleStates))
+   myTransMatr[1,1]<-transitionsAtAge$pAA[1]
+   myTransMatr[1,2]<-transitionsAtAge$pAI[1]
+   myTransMatr[1,3]<-transitionsAtAge$pAD[1]
+   myTransMatr[2,2]<-transitionsAtAge$pII[1]
+   myTransMatr[2,3]<-transitionsAtAge$pID[1]
+   myTransMatr[3,3]<-1
+
+   myMc<-new("markovchain", transitionMatrix = myTransMatr,
+             states = possibleStates,
+             name = paste("Age",age,"transition matrix"))
+
+   return(myMc)
+ }
```

Cause transitions are not homogeneous across ages, we use a `markovchainList` object to describe the transition probabilities for a guy starting at age 100.

```
R> getFullTransitionTable<-function(age){
+   ageSequence<-seq(from=age, to=120)
+   k=1
+   myList=list()
+   for ( i in ageSequence) {
+     mc_age_i<-getMc4Age(age = i)
+     myList[[k]]<-mc_age_i
+     k=k+1
+   }
+   myMarkovChainList<-new("markovchainList", markovchains = myList,
+                          name = paste("TransitionsSinceAge", age, sep = ""))
+   return(myMarkovChainList)
+ }
R> transitionsSince100<-getFullTransitionTable(age=100)
```

We can use such transition for simulating ten life trajectories for a guy that begins “active” (A) aged 100:

```
R> rmarkovchain(n = 10, object = transitionsSince100,
+              what = "matrix", t0 = "A", include.t0 = TRUE)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]
[1,]	"A"	"A"	"A"	"A"	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[2,]	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[3,]	"A"	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[4,]	"A"	"A"	"A"	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[5,]	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[6,]	"A"	"A"	"A"	"A"	"A"	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[7,]	"A"	"I"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[8,]	"A"	"A"	"A"	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[9,]	"A"	"A"	"A"	"I"	"I"	"I"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[10,]	"A"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"

	[,14]	[,15]	[,16]	[,17]	[,18]	[,19]	[,20]	[,21]	[,22]
[1,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[2,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[3,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[4,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[5,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[6,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[7,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[8,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[9,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"
[10,]	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"	"D"

Let us consider 100 simulated life trajectories for a healthy individual aged 80. We can compute the expected time the individual is disabled, starting from the active state at age 80.

```
R> transitionsSince80<-getFullTransitionTable(age=80)
R> lifeTrajectories<-rmarkovchain(n=1e2, object=transitionsSince80,
+                               what="matrix",t0="A",include.t0=TRUE)
R> temp<-matrix(0,nrow=nrow(lifeTrajectories),ncol = ncol(lifeTrajectories))
R> temp[lifeTrajectories=="I"]<-1
R> expected_period_disabled<-mean(rowSums((temp)))
R> expected_period_disabled
```

```
[1] 1.18
```

Assuming that the health insurance will pay a benefit of 12000 per year disabled and that the real interest rate is 0.02, we can compute the lump-sum premium at 80.

```
R> mean(rowMeans(12000*temp%*( matrix((1+0.02)^~seq(from=0, to=ncol(temp)-1))))))

[1] 11954.07
```

6.4. Sociology

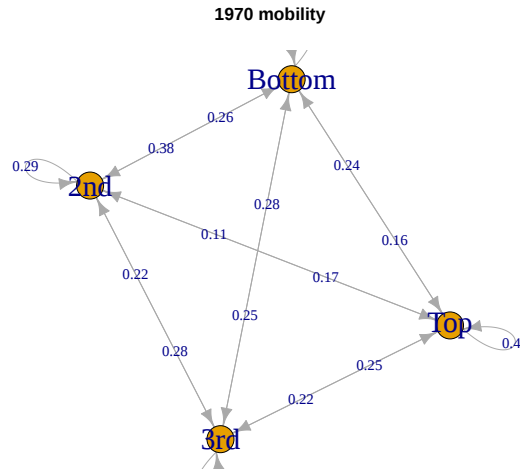


Figure 5: 1970 UK cohort mobility data.

Markov chains have been actively used to model progressions and regressions between social classes. The first study was performed by [Glass and Hall \(1954\)](#), while a more recent application can be found in [Jo Blanden and Machin \(2005\)](#). The table that follows shows the income quartile of the father when the son was 16 (in 1984) and the income quartile of the son when aged 30 (in 2000) for the 1970 cohort.

```
R> data("blanden")
R> mobilityMc <- as(blanden, "markovchain")
R> mobilityMc
```

Unnamed Markov chain

A 4 - dimensional discrete Markov Chain defined by the following states:
Bottom, 2nd, 3rd, Top

The transition matrix (by rows) is defined as follows:

	2nd	3rd	Bottom	Top
Bottom	0.2900000	0.2200000	0.3800000	0.1100000
2nd	0.2772277	0.2574257	0.2475248	0.2178218
3rd	0.2626263	0.2828283	0.2121212	0.2424242
Top	0.1700000	0.2500000	0.1600000	0.4200000

The underlying transition graph is plotted in Figure 5.

The steady state distribution is computed as follows. Since transition across quartiles are shown, the probability function is evenly 0.25.

```
R> round(steadyStates(mobilityMc), 2)
```

	Bottom	2nd	3rd	Top
[1,]	0.25	0.25	0.25	0.25

6.5. Genetics and Medicine

This section contains two examples: the first shows the use of Markov chain models in genetics, the second shows an application of Markov chains in modelling diseases' dynamics.

Genetics

P. J. Avery and D. A. Henderson (1999) discusses the use of Markov chains in model Preproglucacon gene protein bases sequence. The `preproglucacon` dataset in **markovchain** contains the dataset shown in the package.

```
R> data("preproglucacon", package = "markovchain")
```

It is possible to model the transition probabilities between bases as shown in the following code.

```
R> mcProtein <- markovchainFit(preproglucacon$preproglucacon,
+                             name = "Preproglucacon MC")$estimate
R> mcProtein
```

Preproglucacon MC

A 4 - dimensional discrete Markov Chain defined by the following states:

A, C, G, T

The transition matrix (by rows) is defined as follows:

	A	C	G	T
A	0.3585271	0.1434109	0.16666667	0.3313953
C	0.3840304	0.1558935	0.02281369	0.4372624
G	0.3053097	0.1991150	0.15044248	0.3451327
T	0.2844523	0.1819788	0.17667845	0.3568905

Medicine

Discrete-time Markov chains are also employed to study the progression of chronic diseases. The following example is taken from B. A. Craig and A. A. Sendi (2002). Starting from six month follow-up data, the maximum likelihood estimation of the monthly transition matrix is obtained. This transition matrix aims to describe the monthly progression of CD4-cell counts of HIV infected subjects.

```
R> craigSendiMatr <- matrix(c(682, 33, 25,
+                             154, 64, 47,
+                             19, 19, 43), byrow = T, nrow = 3)
R> hivStates <- c("0-49", "50-74", "75-UP")
R> rownames(craigSendiMatr) <- hivStates
R> colnames(craigSendiMatr) <- hivStates
R> craigSendiTable <- as.table(craigSendiMatr)
R> mcM6 <- as(craigSendiTable, "markovchain")
R> mcM6@name <- "Zero-Six month CD4 cells transition"
R> mcM6
```

Zero-Six month CD4 cells transition

A 3 - dimensional discrete Markov Chain defined by the following states:

0-49, 50-74, 75-UP

The transition matrix (by rows) is defined as follows:

	0-49	50-74	75-UP
0-49	0.9216216	0.04459459	0.03378378
50-74	0.5811321	0.24150943	0.17735849
75-UP	0.2345679	0.23456790	0.53086420

As shown in the paper, the second passage consists in the decomposition of $M_6 = V \cdot D \cdot V^{-1}$ in order to obtain M_1 as $M_1 = V \cdot D^{1/6} \cdot V^{-1}$.

```
R> eig <- eigen(mcM6@transitionMatrix)
```

```
R> D <- diag(eig$values)
```

```
R> V <- eig$vectors
```

```
R> V %*% D %*% solve(V)
```

	[,1]	[,2]	[,3]
[1,]	0.9216216	0.04459459	0.03378378
[2,]	0.5811321	0.24150943	0.17735849
[3,]	0.2345679	0.23456790	0.53086420

```
R> d <- D ^ (1/6)
```

```
R> M <- V %*% d %*% solve(V)
```

```
R> mcM1 <- new("markovchain", transitionMatrix = M, states = hivStates)
```

7. Discussion, issues and future plans

The **markovchain** package has been designed in order to provide easily handling of DTMC and communication with alternative packages.

The package has known several improvements in the recent years: many functions added, porting the software in Rcpp **Rcpp** package (Eddelbuettel 2013) and many methodological improvements that have improved the software reliability.

8. Acknowledgments

The package was selected for Google Summer of Code 2015 support. The authors wish to thank Michael Cole, Tobi Gutman and Mildenerger Thoralf for their suggestions and bug checks. A final thanks also to Dr. Simona C. Minotti and Dr. Mirko Signorelli for their support in drafting this version of the vignettes.

References

- Allaire J, Francois R, Ushey K, Vandenbrouck G, Geelnard M, Intel (2016). *RcppParallel: Parallel Programming Tools for 'Rcpp'*. R package version 4.3.19, URL <https://rcppcore.github.io/RcppParallel/>.
- Anderson TW, Goodman LA (1957). “Statistical inference about Markov chains.” *The Annals of Mathematical Statistics*, pp. 89–110.
- B A Craig, A A Sendi (2002). “Estimation of the Transition Matrix of a Discrete-Time Markov Chain.” *Health Economics*, **11**, 33–42.
- Bard JF (2000). “Lecture 12.5 - Additional Issues Concerning Discrete-Time Markov Chains.”
- Brémaud P (1999). “Discrete-Time Markov Models.” In *Markov Chains*, pp. 53–93. Springer.
- Brown LD, Cai TT, DasGupta A (2001). “Interval Estimation for a Binomial Proportion.” *Statistical Science*, **16**(2), 101–133. doi:10.1214/ss/1009213286.
- Burkett MW, Scherer WT, Todd A (2019). “Are financial market states recurrent and persistent?” *Cogent Economics and Finance*, **7**(1), 1622171. doi:10.1080/23322039.2019.1622171.
- Chambers J (2008). *Software for Data Analysis: Programming with R*. Statistics and computing. Springer-Verlag. ISBN 9780387759357.
- Ching WK, Ng MK, Fung ES (2008). “Higher-order multivariate Markov chains and their applications.” *Linear Algebra and its Applications*, **428**(2), 492–507.
- Cover TM, Thomas JA (2006). *Elements of Information Theory*. 2 edition. Wiley-Interscience, Hoboken, NJ. doi:10.1002/047174882X.
- Csardi G, Nepusz T (2006). “The **igraph** Software Package for Complex Network Research.” *InterJournal, Complex Systems*, 1695. URL <https://igraph.sourceforge.net/>.
- Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press. doi:10.1017/CB09780511802843.
- De Angelis, Paolo, Di Falco, L (2016). *Assicurazioni sulla salute: caratteristiche, modelli attuariali e basi tecniche*. Il Mulino. Il Mulino. ISBN 9788815260840. URL <https://books.google.it/books?id=D56bjgEACAAJ>.
- de Wreede LC, Fiocco M, Putter H (2011). “**mstate**: An R Package for the Analysis of Competing Risks and Multi-State Models.” *Journal of Statistical Software*, **38**(7), 1–30. URL <https://www.jstatsoft.org/v38/i07/>.
- Denuit M, Maréchal X, Pitrebois S, Walhin JF (2007). *Actuarial modelling of claim counts: Risk classification, credibility and bonus-malus systems*. Wiley.
- Deshmukh S (2012). *Multiple Decrement Models in Insurance: An Introduction Using R*. SpringerLink : Bücher. Springer-Verlag. ISBN 9788132206590.

- Dobrow RP (2016). *Introduction to Stochastic Processes with R*. John Wiley & Sons.
- Eddelbuettel D (2013). *Seamless R and C++ Integration with **Rcpp***. Springer-Verlag, New York. ISBN 978-1-4614-6867-7.
- Feres R (2007). “Notes for Math 450 MATLAB Listings for Markov Chains.”
- Geyer CJ, Johnson LT (2013). *mcmc: Markov Chain Monte Carlo*. R package version 0.9-2, URL <https://CRAN.R-project.org/package=mcmc>.
- Glass D, Hall JR (1954). “Social Mobility in Great Britain: A Study in Intergenerational Change in Status.” In *Social Mobility in Great Britain*. Routledge and Kegan Paul.
- Goulet V, Dutang C, Maechler M, Firth D, Shapira M, Stadelmann M, expm-developers@listsR-forgeR-projectorg (2013). **expm**: *Matrix Exponential*. R package version 0.99-1, URL <https://CRAN.R-project.org/package=expm>.
- Grinstead CM, Snell LJ (2006). *Grinstead and Snell’s Introduction to Probability*. Version dated 4 July 2006 edition. American Mathematical Society.
- Himmelmann SSDDL, wwwlinhicom (2010). **HMM**: *HMM - Hidden Markov Models*. R package version 1.0, URL <https://CRAN.R-project.org/package=HMM>.
- Hu YT, Kiesel R, Perraudin W (2002). “The estimation of transition matrices for sovereign credit ratings.” *Journal of Banking and Finance*, **26**(7), 1383–1406. ISSN 03784266. doi: [10.1016/S0378-4266\(02\)00268-6](https://doi.org/10.1016/S0378-4266(02)00268-6).
- Israel RB, Rosenthal JS, Wei JZ (2001). “Finding generators for Markov chains via empirical transition matrices, with applications to credit ratings.” *Mathematical finance*, **11**(2), 245–265.
- J G Kemeny, J L Snell, G L Thompson (1974). *Introduction to Finite Mathematics*. Prentice Hall.
- Jackson CH (2011). “Multi-State Models for Panel Data: The **msm** Package for R.” *Journal of Statistical Software*, **38**(8), 1–29. URL <https://www.jstatsoft.org/v38/i08/>.
- Jacob I (2014). “Is R Cost Effective?” Electronic. Presented on Manchester R Meeting.
- Jo Blanden PG, Machin S (2005). “Intergenerational Mobility in Europe and North America.” *Technical report*, Center for Economic Performances. URL <https://cep.lse.ac.uk/about/news/IntergenerationalMobility.pdf>.
- Kemeny JG, Snell JL (1960). *Finite Markov Chains*. D. Van Nostrand, Princeton, NJ.
- Konstantopoulos T (2009). “Markov Chains and Random Walks.” Lecture notes.
- Kullback S, Kupperman M, Ku H (1962). “Tests for Contingency Tables and Markov Chains.” *Technometrics*, **4**(4), 573–608.
- Montgomery J (2009). “Communication Classes.”

- Nicholson W (2013). *DTMCPack: Suite of Functions Related to Discrete-Time Discrete-State Markov Chains*. R package version 0.1-2, URL <https://CRAN.R-project.org/package=DTMCPack>.
- Noé F, Schütte C, Vanden-Eijnden E, Reich L, Weigl TR (2009). “Constructing the equilibrium ensemble of folding pathways from short off-equilibrium simulations.” *Proceedings of the National Academy of Sciences*, **106**(45), 19011–19016. ISSN 0027-8424, 1091-6490. doi:10.1073/pnas.0905466106.
- P J Avery, D A Henderson (1999). “Fitting Markov Chain Models to Discrete State Series.” *Applied Statistics*, **48**(1), 53–61.
- Pfeuffer M (2017). *ctmcd: Estimating the Parameters of a Continuous-Time Markov Chain from Discrete-Time Data*. R package version 1.1, URL <https://CRAN.R-project.org/package=ctmcd>.
- R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Roebuck P (2011). *matlab: MATLAB emulation package*. R package version 0.8.9, URL <https://CRAN.R-project.org/package=matlab>.
- Sison CP, Glaz J (1995). “Simultaneous confidence intervals and sample size determination for multinomial proportions.” *Journal of the American Statistical Association*, **90**(429), 366–369.
- Skuriat-Olechnowska M (2005). *Statistical inference and hypothesis testing for Markov chains with Interval Censoring*. diploma thesis, Delft University of Technology.
- Snell L (1999). “Probability Book: chapter 11.” URL https://web.archive.org/web/20150721052835/http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter11.pdf.
- Spedicato GA (2017). “Discrete Time Markov Chains with R.” *The R Journal*. URL <https://journal.r-project.org/articles/RJ-2017-036/index.html>.
- Strelioff CC, Crutchfield JP, Hubler AW (2007). “Inferring Markov Chains: Bayesian Estimation, Model Comparison, Entropy Rate, and Out-of-Class Modeling.” *Physical Review E*, **76**(1), 011106. doi:10.1103/PhysRevE.76.011106.
- Tarjan R (1972). “Depth-First Search and Linear Graph Algorithms.” *SIAM Journal on Computing*, **1**(2), 146–160. doi:10.1137/0201010. URL <https://doi.org/10.1137/0201010>.
- Tran H, Carmichael DG (2012). “Stationarity of the transition probabilities in the Markov chain formulation of owner payments on projects.” *ANZIAM Journal*, **53**, C69–C89. doi:10.21914/anziamj.v53i0.5111.
- Villacorta PJ (2012). *MultinomialCI: Simultaneous confidence intervals for multinomial proportions according to the method by Sison and Glaz*. R package version 1.0, URL <https://CRAN.R-project.org/package=MultinomialCI>.

Wittmann A (2007). *CreditMetrics: Functions for Calculating the CreditMetrics Risk Model*. R package version 0.0-2.

Wolfram Research I (2013a). *Mathematica*. Wolfram Research, Inc., ninth edition.

Wolfram Research I (2013b). “Structural Properties of Finite Markov Processes.” URL <https://www.wolfram.com/mathematica/new-in-9/markov-chains-and-queues/structural-properties-of-finite-markov-processes.html>.

Affiliation:

Giorgio Alfredo Spedicato
Ph.D C.Stat FCAS, FSA, CSPA Unipol Group
Via Firenze 11 Paderno Dugnano 20037 Italy
E-mail: spedygiorgio@gmail.com
URL: www.statisticaladvisor.com

Tae Seung Kang
Ph.D student, Computer & Information Science & Engineering
University of Florida Gainesville, FL, USA
E-mail: tskang3@gmail.com

Sai Bhargav Yalamanchi
B-Tech student, Electrical Engineering
Indian Institute of Technology, Bombay Mumbai - 400 076, India
E-mail: bhargavcoolboy@gmail.com

Deepak Yadav
B-Tech student, Computer Science and Engineering
Indian Institute of Technology, Varanasi Uttar Pradesh - 221 005, India
E-mail: deepakyadav.iitbhu@gmail.com

Ignacio Cordon
Software Engineer
Madrid (Madrid), Spain
E-mail: nacho.cordon.castillo@gmail.com