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

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

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 (Himmelmann 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 non-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,
The markovchain package

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 2013b).

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 $X_1, X_2, \ldots, X_n, \ldots$ 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 $X_{n-1}, X_{n-2}, \ldots, X_1$.

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

The set of possible states $S = \{s_1, s_2, \ldots, s_r\}$ 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:

$$p_{ij} = Pr(X_1 = s_j | X_0 = s_i).$$ \hspace{1cm} (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.

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

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}. \hspace{1cm} (4)$$
The distribution over the states can be written in the form of a stochastic row vector $x$ (the term stochastic means that $\sum x_i = 1, x_i \geq 0$): 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 started in state $s_i$ has a positive probability to reach the state $s_j$ at a certain point, i.e., $\exists n > 0: p^n_{ij} > 0$. 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 multiplies of $k_i$ steps, that is $k_i = \gcd \{ n > 0: Pr(X_n = s_i | X_0 = s_i) > 0 \}$, 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 $T^{s \rightarrow x}$ is the number of periods to go back to state $x$ knowing that the chain starts in $x$.

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

- A state $x$ is transient if $P(T^{s \rightarrow x} < +\infty) < 1$ (equivalently $P(T^{s \rightarrow x} = +\infty) > 0$).

It is possible to analyze the timing to reach a certain state. The first passage 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

$$h_{ij}(n) = Pr(T_{ij} = n) = Pr(X_n = s_j, X_{n-1} \neq s_j, \ldots, X_1 \neq s_j | X_0 = s_i) \quad (5)$$
and can be found recursively using Equation 6, given that \( h_{ij}^{(n)} = p_{ij} \).

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

If in the definition of the first passage time we let \( s_i = s_j \), we obtain the first return time \( T_i = \inf\{n \geq 1 : X_n = s_i | X_0 = s_i\} \). 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 \( \sum z_j = 1 \).

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 \( z = (z_1, \ldots, z_k) \) is the eigenvector of \( P \) such that \( \sum_{i=1}^k z_i = 1 \), then we get

\[
\lim_{n \to \infty} P^n = Z, \tag{7}
\]

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

### 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

\[
P = \begin{bmatrix}
0.5 & 0.2 & 0.3 \\
0.15 & 0.45 & 0.4 \\
0.25 & 0.35 & 0.4 \\
\end{bmatrix}. \tag{8}
\]

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 9 and 10.

\[
x^{(1)} = (0 1 0) \begin{bmatrix}
0.5 & 0.2 & 0.3 \\
0.15 & 0.45 & 0.4 \\
0.25 & 0.35 & 0.4 \\
\end{bmatrix} = (0.15 0.45 0.4). \tag{9}
\]

\[
x^{(n)} = x^{(n-1)} P \rightarrow (0.15 0.45 0.4) \begin{bmatrix}
0.5 & 0.2 & 0.3 \\
0.15 & 0.45 & 0.4 \\
0.25 & 0.35 & 0.4 \\
\end{bmatrix} = (0.2425 0.3725 0.385). \tag{10}
\]

If, f.e., we are interested in the probability of reaching the state \( s_3 \) in two steps, 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:

<table>
<thead>
<tr>
<th>Name:</th>
<th>states</th>
<th>byrow</th>
<th>transitionMatrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class:</td>
<td>character</td>
<td>logical</td>
<td>matrix</td>
</tr>
</tbody>
</table>

Name: name
Class: 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 non-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
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
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
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` 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
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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Direct multiplication for transition matrices.</td>
</tr>
<tr>
<td>[</td>
<td>Direct access to the elements of the transition matrix.</td>
</tr>
<tr>
<td>==</td>
<td>Equality operator between two transition matrices.</td>
</tr>
<tr>
<td>as</td>
<td>Operator to convert markovchain objects into data.frame and table object.</td>
</tr>
<tr>
<td>dim</td>
<td>Dimension of the transition matrix.</td>
</tr>
<tr>
<td>names</td>
<td>Equal to states.</td>
</tr>
<tr>
<td>names&lt;-</td>
<td>Change the states name.</td>
</tr>
<tr>
<td>name</td>
<td>Get the name of markovchain object.</td>
</tr>
<tr>
<td>name&lt;-</td>
<td>Change the name of markovchain object.</td>
</tr>
<tr>
<td>plot</td>
<td>plot method for markovchain objects.</td>
</tr>
<tr>
<td>print</td>
<td>print method for markovchain objects.</td>
</tr>
<tr>
<td>show</td>
<td>show method for markovchain objects.</td>
</tr>
<tr>
<td>sort</td>
<td>sort method for markovchain objects.</td>
</tr>
<tr>
<td>states</td>
<td>Name of the transition states.</td>
</tr>
<tr>
<td>t</td>
<td>Transposition operator (which switches byrow slot value and modifies the transition matrix coherently).</td>
</tr>
</tbody>
</table>

Table 1: markovchain methods for handling markovchain objects.

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

  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
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
```
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)
+ }
> fvals(mchain=mcWeather,initialstate=c(90,5,5),n=4)
```

```
      i  sunny  cloudy  rain
 1 0  90.0000  5.0000  5.0000
 2 1  65.5000 22.2500 12.2500
 3 2  54.9750 27.5125 17.5125
 4 3  50.2388 29.8807 19.8807
 5 4  48.1074 30.9463 20.9463
```

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

```R
> states(mcWeather)
[1] "sunny" "cloudy" "rain"
> names(mcWeather)
[1] "sunny" "cloudy" "rain"
> dim(mcWeather)
[1] 3
Methods are available to set and get the name of `markovchain` object.

```r
R> name(mcWeather)
[1] "Weather"
```

```r
R> name(mcWeather) <- "New Name"
R> name(mcWeather)
[1] "New Name"
```

Also it is possible to alphabetically sort the transition matrix:

```r
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
R> transitionProbability(mcWeather, "cloudy", "rain")
[1] 0.3
```

```r
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 laconic). 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 colour.

```r
R> print(mcWeather)
```
The markovchain package

Weather transition matrix

Figure 1: Weather example. Markov chain plot.

```r
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
Figure 2: Weather example. Markov chain plot with diagram. plot(mcWeather, package="diagram", box.size = 0.04)
Import and export from some specific classes is possible, as shown in Figure 3 and in the following code.

```r
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   7 rain sunny 0.20
8   rain cloudy 0.45
9   rain   rain 0.35
```

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

R> require(msm)
R> 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, 1) )
R> cavmsm <- msm(state ~ years, subject = PTNUM, data = cav, qmatrix = Q, death = 4)
R> msmMc <- as(cavmsm, "markovchain")
R> msmMc

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

R> library(etm)
R> data(sir.cont)
R> sir.cont <- sir.cont[order(sir.cont$id, sir.cont$time), ]
R> 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]) {
++     
+   
+   if (sir.cont$id[i]!=sir.cont$id[i-1]) {
+     if (sir.cont$time[i]!=sir.cont$time[i-1]) {
++     
+   }
++   
+   }


```r
+ sir.cont$time[i-1] <- sir.cont$time[i-1] - 0.5
+ }
+ }
+ }
R> tra <- matrix(ncol=3,nrow=3,FALSE)
R> tra[1, 2:3] <- TRUE
R> tra[2, c(1, 3)] <- TRUE
R> tr.prob <- etm(sir.cont, c("0", "1", "2"), tra, "cens", 1)
R> tr.prob

Multistate model with 2 transient state(s)
and 1 absorbing state(s)

Possible transitions:
from to
0 1
0 2
1 0
1 2

Estimate of P(1, 183)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Estimate of cov(P(1, 183))

```
```text
0 0 0 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0
1 0 0 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0
2 0 0 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0
1 1 0 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0
2 1 0 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0
0 2 0 0 0 0 0 0 -2.864030e-20 -1.126554e-19 0
1 2 0 0 0 0 0 -4.785736e-20 2.710505e-19 0
2 2 0 0 0 0 0 0.0000000e+00 0.0000000e+00 0

R> etm2mc<-'as(tr.prob, "markovchain")
R> etm2mc

Unnamed Markov chain
A 3-dimensional discrete Markov Chain defined by the following states:
0, 1, 2
The transition matrix (by rows) is defined as follows:
```
0 0 1 2
0 0 0.000000 0.500000 0.500000
```
**Import – Export from and to markovchain objects**

![Diagram of import and export methods](image)

**Figure 3:** The *markovchain* methods for import and export.

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
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
  - s3
Non-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 modelled by a non-homogeneous Markov Chain, since the transition probabilities change by year. Methods explicitly written for `markovchainList` objects are: `print`, `show`, `dim` and `[`.

```r
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:

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>I</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.7</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>I</td>
<td>0.1</td>
<td>0.6</td>
<td>0.3</td>
</tr>
<tr>
<td>D</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>I</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>I</td>
<td>0.0</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>D</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```
The Markovchain package

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.
### Dataset Description

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blanden</td>
<td>Mobility across income quartiles, Jo Blanden and Machin (2005).</td>
</tr>
<tr>
<td>craigsendi</td>
<td>CD4 cells, B. A. Craig and A. A. Sendi (2002).</td>
</tr>
<tr>
<td>kullback</td>
<td>raw transition matrices for testing homogeneity, Kullback, Kupperman, and</td>
</tr>
<tr>
<td>holson</td>
<td>Individual states trajectories.</td>
</tr>
<tr>
<td>sales</td>
<td>Sales of six beverages in Hong Kong.</td>
</tr>
<tr>
<td>Ching, Ng, and Fung (2008).</td>
<td></td>
</tr>
</tbody>
</table>

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

### R Code File Description

<table>
<thead>
<tr>
<th>R Code File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bard.R</td>
<td>Structural analysis of Markov chains from Bard PPT.</td>
</tr>
<tr>
<td>examples.R</td>
<td>Notable Markov chains, e.g., The Gambler Ruin chain.</td>
</tr>
<tr>
<td>quickStart.R</td>
<td>Generic examples.</td>
</tr>
<tr>
<td>extractMatrices.R</td>
<td>Generic examples.</td>
</tr>
</tbody>
</table>

Table 3: The **markovchain** demos.
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 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>absorbingStates</td>
<td>the absorbing states of the transition matrix, if any.</td>
</tr>
<tr>
<td>steadyStates</td>
<td>the vector(s) of steady state(s) in matrix form.</td>
</tr>
<tr>
<td>communicatingClasses</td>
<td>list of communicating classes.</td>
</tr>
<tr>
<td>s_j, given actual state s_i.</td>
<td></td>
</tr>
<tr>
<td>canonicForm</td>
<td>the transition matrix into canonic form.</td>
</tr>
<tr>
<td>is.accessible</td>
<td>verification if a state j is reachable from state i.</td>
</tr>
<tr>
<td>is.irreducible</td>
<td>verification whether a DTMC is irreducible.</td>
</tr>
<tr>
<td>period</td>
<td>the period of an irreducible DTMC.</td>
</tr>
<tr>
<td>recurrentClasses</td>
<td>list of recurrent classes.</td>
</tr>
<tr>
<td>steadyStates</td>
<td>the vector(s) of steady state(s) in matrix form.</td>
</tr>
<tr>
<td>summary</td>
<td>DTMC summary.</td>
</tr>
<tr>
<td>transientStates</td>
<td>the transient states of the transition matrix, if any.</td>
</tr>
</tbody>
</table>

Table 4: `markovchain` methods: statistical operations.

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

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

sunny cloudy rain
0.7 0.2 0.1

A stationary (steady state) vector is a probability vector such that Equation 11

\[
\begin{align*}
0 & \leq \pi_j \leq 1 \\
\sum_{j \in S} \pi_j & = 1 \\
\pi \ast P & = \pi
\end{align*}
\] (11)

Steady states are associated to \( P \) eigenvalues equal to one. Therefore the steady states vector can be identified by the following:

1. decompose the transition matrix in eigenvalues and eigenvectors;
2. consider only eigenvectors corresponding to eigenvalues equal to one;
3. normalize such eigenvalues so that the sum of their components is one.
Numeric issue (negative values) can arise when the Markov Chain contains more closed classes. If negative values are found in the initial solution, the above described algorithm is performed on the submatrix corresponding to recurrent $P$ classes. Another vignette in the package focuses on this issue.

The result is returned in matrix form.

```r
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
R> gamblerRuinMarkovChain <- function(moneyMax, prob = 0.5) {
+   require(matlab)
+   matr <- zeros(moneyMax + 1)
+   states <- as.character(seq(from = 0, to = moneyMax, by = 1))
+   rownames(matr) = states; colnames(matr) = states
+   matr[1,1] = 1; matr[moneyMax + 1,moneyMax + 1] = 1
+   for(i in 2:moneyMax)
+     { matr[i,i-1] = 1 - prob; matr[i, i + 1] = prob }
+   out <- new("markovchain",
+     transitionMatrix = matr,
+     name = paste("Gambler ruin", moneyMax, "dim", sep = " ")
+   )
+   return(out)
+ }
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
```

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

```r
R> absorbingStates(mcGR4)

[1] "0" "4"
R> absorbingStates(mcWeather)

class(0)
```

The key function used within Feres (2007) (and `markovchain`'s derived functions) is `.commclassKernel`, that is called below.
The \texttt{.commclassKernel} function gets a transition matrix of dimension \( n \) and return a list of two items:

1. \( C \), an adjacency matrix showing for each state \( s_j \) (in the row) which states lie in the same communicating class of \( s_j \) (flagged with 1).

2. \( v \), a binary vector indicating whether the state \( s_j \) is transient (0) or not (1).

These functions are used by two other internal functions on which the \texttt{summary} method for \texttt{markovchain} objects works.

The example matrix used in Feres (2007) well exemplifies the purpose of the function.
Giorgio Alfredo Spedicato, Tae Seung Kang, Sai Bhargav Yalamanchi, Deepak Yadav  21

R> P <- matlab::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[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> .commclassesKernel(P)

$C
      a  b  c  d  e  f  g  h  i  j
a  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
b  FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE TRUE FALSE FALSE
c  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
d  FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
e  FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
f  FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
g  FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE TRUE FALSE FALSE
h  FALSE FALSE FALSE  TRUE FALSE FALSE FALSE TRUE FALSE FALSE
i  FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE TRUE FALSE
j  FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE TRUE

$v
      a  b  c  d  e  f  g  h  i  j
TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE TRUE FALSE

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
R> transientStates(probMc)
[1] "d" "e" "h" "j"
```

Listings from Feres (2007) have been adapted into `canonicForm` method that turns a Markov chain into canonic form.

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

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

```
a 0.0000000
b 0.0000000
c 0.0000000
d 0.0000000
e 0.0000000
f 0.0000000
g 0.0000000
h 0.2500000
i 0.0000000
j 0.3333333
```

```r
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:
The function \texttt{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.

\[
\begin{array}{cccccccc}
a & c & b & g & i & f & d & e & h \\
\hline
a & 0.5 & 0.50 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
c & 1.0 & 0.00 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
b & 0.0 & 0.00 & 0.333333 & 0.666667 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
g & 0.0 & 0.00 & 0.000000 & 0.750000 & 0.250000 & 0.000000 & 0.000000 & 0.00 \\
i & 0.0 & 0.00 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
f & 0.0 & 0.00 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
d & 0.0 & 0.00 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.00 \\
e & 0.0 & 0.00 & 0.000000 & 0.333333 & 0.666667 & 0.000000 & 0.000000 & 0.00 \\
h & 0.0 & 0.25 & 0.000000 & 0.000000 & 0.250000 & 0.000000 & 0.000000 & 0.25 \\
j & 0.0 & 0.00 & 0.333333 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.333333 \\
\end{array}
\]

In Section 2.2 we observed that, if a DTMC is irreducible, all its states share the same periodicity. Then, the \texttt{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 \texttt{is.irreducible} and, in the latter case, the method \texttt{period} is used to compute the periodicity of the chain.

\[
\begin{array}{cccccccc}
E & a & b & c & d & e & f & g & h \\
\hline
E & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
E & 0 & 1/3 & 0 & 1/3 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 1/4 & 0 & 1/4 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
R> E <- matrix(0, nrow = 4, ncol = 4)
R> E[1, 2] <- 1
R> E[2, 1] <- 1/3; E[2, 3] <- 2/3
R> E[4, 3] <- 1
R> mcE <- new("markovchain", states = c("a", "b", "c", "d"),
\]

\texttt{is.irreducible} and \texttt{period} methods are used to check if a DTMC is irreducible and to compute its periodicity, if applicable.
The example Markov chain found in Mathematica web site (Wolfram Research 2013a) has been used, and is plotted in Figure 4.

\begin{Verbatim}
R> require(matlab)
R> mathematicaMatr <- zeros(5)
R> mathematicaMatr[1,] <- c(0, 1/3, 0, 2/3, 0)
R> mathematicaMatr[2,] <- c(1/2, 0, 0, 0, 1/2)
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)
\end{Verbatim}

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

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

\begin{Verbatim}
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)
+ }
\end{Verbatim}
Figure 4: Mathematica 9 example. Markov chain plot.
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
R> firstPassagePdF <- firstPassage(object = mcWeather, state = "sunny", n = 10)
R> firstPassagePdF[3, 3]
[1] 0.121
```

5. Statistical analysis

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>markovchainFit</td>
<td>Function to return fitted Markov chain for a given sequence.</td>
</tr>
<tr>
<td>predict</td>
<td>Method to calculate predictions from markovchain or markovchainList objects.</td>
</tr>
<tr>
<td>rmarkovchain</td>
<td>Function to sample from markovchain or markovchainList objects.</td>
</tr>
</tbody>
</table>

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
R> weathersOfDays <- rmarkovchain(n = 365, object = mcWeather, t0 = "sunny")
R> weathersOfDays[1:30]
```

```
[1] "sunny" "cloudy" "sunny" "sunny" "sunny" "rain" "cloudy"
[8] "cloudy" "rain" "rain" "rain" "cloudy" "rain"
[15] "rain" "rain" "rain" "cloudy" "cloudy" "sunny" "sunny"
[22] "sunny" "sunny" "cloudy" "sunny" "sunny" "sunny" "sunny"
[29] "cloudy" "sunny"
```

Similarly, it is possible to simulate one or more sequences from a non-homogeneous Markov chain, as the following code (applied on CCHC example) exemplifies.
Two advance parameters are available to the \texttt{rmarkovchain} method which helps you decide which implementation to use. There are four options available: \texttt{R}, \texttt{R} in parallel, \texttt{C++} and \texttt{C++} in parallel. Two boolean parameters \texttt{useRcpp} and \texttt{parallel} will decide which implementation will be used. Default is \texttt{useRcpp = TRUE} and \texttt{parallel = FALSE} i.e. \texttt{C++} implementation. The \texttt{C++} implementation is generally faster than the \texttt{R} implementation. If you have multicore processors then you can take advantage of \texttt{parallel} parameter by setting it to \texttt{TRUE}. When both \texttt{Rcpp=TRUE} and \texttt{parallel=TRUE} the parallelization has been carried out using \texttt{RcppParallel} package (Allaire, Francois, Ushey, Vandenbrouck, Geelnard, and Intel 2016).

5.2. Estimation
A time homogeneous Markov chain can be fit from given data. Four methods have been implemented within current version of \texttt{markovchain} package: maximum likelihood, maximum likelihood with Laplace smoothing, Bootstrap approach, maximum a posteriori.

Equation 12 shows the maximum likelihood estimator (MLE) of the $p_{ij}$ entry, where the $n_{ij}$ element consists in the number sequences $(X_t = s_i, X_{t+1} = s_j)$ found in the sample, that is

$$\hat{p}_{ij}^{MLE} = \frac{n_{ij}}{\sum_{u=1}^{k} n_{iu}}. \quad (12)$$

Equation 13 shows the standard error of the MLE (Skuriat-Olechnowska 2005).

$$SE_{ij} = \frac{\hat{p}_{ij}^{MLE}}{\sqrt{n_{ij}}} \quad (13)$$

\texttt{R} weatherFittedMLE <- \texttt{markovchainFit(data = weathersOfDays, method = "mle", name = "Weather MLE")}

\texttt{R} weatherFittedMLE$estimate

\texttt{Weather MLE}
A 3-dimensional discrete Markov Chain defined by the following states:
The markovchain package

cloudy, rain, sunny
The transition matrix (by rows) is defined as follows:

\[
\begin{array}{ccc}
\text{cloudy} & \text{rain} & \text{sunny} \\
\text{cloudy} & 0.4067797 & 0.33898305 & 0.2542373 \\
\text{rain} & 0.4315789 & 0.42105263 & 0.1473684 \\
\text{sunny} & 0.1920530 & 0.09933775 & 0.7086093
\end{array}
\]

R> weatherFittedMLE$standardError

\[
\begin{array}{ccc}
\text{cloudy} & \text{rain} & \text{sunny} \\
\text{cloudy} & 0.05871359 & 0.05359793 & 0.04641717 \\
\text{rain} & 0.06740131 & 0.06657427 & 0.03938587 \\
\text{sunny} & 0.03566334 & 0.02564890 & 0.06850384
\end{array}
\]

The Laplace smoothing approach is a variation of the MLE, where the \(n_{ij}\) is substituted by \(n_{ij} + \alpha\) (see Equation 14), being \(\alpha\) an arbitrary positive stabilizing parameter.

\[
\hat{p}_{ij}^{LS} = \frac{n_{ij} + \alpha}{\sum_{u=1}^{k} (n_{iu} + \alpha)} \tag{14}
\]

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:

\[
\begin{array}{ccc}
\text{cloudy} & \text{rain} & \text{sunny} \\
\text{cloudy} & 0.4067610 & 0.33898161 & 0.2542574 \\
\text{rain} & 0.4315479 & 0.42102494 & 0.1474271 \\
\text{sunny} & 0.1920810 & 0.09938423 & 0.7085347
\end{array}
\]

(NOTE: The Confidence Interval option is enabled by default. Remove this option to fasten computations.) Both MLE and Laplace approach are based on the `createSequenceMatrix` functions that returns the raw counts transition matrix.

R> createSequenceMatrix(stringchar = weathersOfDays)

\[
\begin{array}{ccc}
\text{cloudy} & \text{rain} & \text{sunny} \\
\text{cloudy} & 48 & 40 & 30 \\
\text{rain} & 41 & 40 & 14 \\
\text{sunny} & 29 & 15 & 107
\end{array}
\]
stringchar could contain NA values, and the transitions containing NA would be ignored.

An issue occurs when the sample contains only one realization of a state (say $X_{\beta}$) which is located at the end of the data sequence, since it yields to a row of zero (no sample to estimate the conditional distribution of the transition). In this case the estimated transition matrix is corrected assuming $p_{\beta,j} = 1/k$, being $k$ the possible states.

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

```r
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 has been developed within the package in order to provide an indication of the variability of $\hat{p}_{ij}$ estimates. The bootstrap approach implemented within the markovchain package follows these steps:

1. bootstrap the data sequences following the conditional distributions of states estimated from the original one. The default bootstrap samples is 10, as specified in nboot parameter of markovchainFit function.
2. apply MLE estimation on bootstrapped data sequences that are saved in bootStrapSamples slot of the returned list.
3. the $p^{BOOTSTRAP}_{ij}$ is the average of all $p^{MLE}_{ij}$ across the bootStrapSamples list, normalized by row. A standardError of $p^{MLE}_{ij}$ estimate is provided as well.

```r
R> weatherFittedBOOT <- markovchainFit(data = weathersOfDays, 
+ method = "bootstrap", nboot = 20)
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 0.4122969 0.32484731 0.2628558
rain 0.4336146 0.43261227 0.1337732
sunny 0.2096854 0.09843513 0.6918795
```

```r
R> weatherFittedBOOT$standardError
```

```
cloudy rain sunny
cloudy 0.007877334 0.007990708 0.008554771
rain 0.009276170 0.008899492 0.009473045
sunny 0.007924977 0.006227795 0.009041946
```
The bootstrapping process can be done in parallel thanks to the RcppParallel package (Allaire et al. 2016). Parallelized implementation is definitively suggested when the data sample size or the required number of bootstrap runs is high.

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

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

For all the fitting methods, the logLikelihood (Skuriat-Olechnowska 2005) denoted in Equation 15 is provided.

\[
LLH = \sum_{i,j} n_{ij} \cdot \log(p_{ij})
\]

(15)

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

```r
R> weatherFittedMLE$logLikelihood

[1] -342.7361
```

```r
R> weatherFittedBOOT$logLikelihood

[1] -343.0181
```

Confidence matrices of estimated parameters (parametric for MLE, non-parametric for BootStrap) are available as well. The confidenceInterval is provided with the two matrices: `lowerEndpointMatrix` and `upperEndpointMatrix`. The confidence level (CL) is 0.95 by default and can be given as an argument of the function `markovchainFit`. This is used to obtain the standard score (z-score). Equations 16 and 17 (Skuriat-Olechnowska 2005) show the confidenceInterval of a fitting. Note that each entry of the matrices is bounded between 0 and 1.

\[
LowerEndpoint_{ij} = p_{ij} - zscore(CL) \cdot SE_{ij}
\]

(16)

\[
UpperEndpoint_{ij} = p_{ij} + zscore(CL) \cdot SE_{ij}
\]

(17)
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.

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

The same applies for markovchainList.

```r
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 0.0000000
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 0.0000000
2 0.2551724 0.5103448 0.2344828
3 0.0000000 0.0862069 0.9137931
```

Markovchain 3
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 0.94765840 0.04820937 0.004132231
2 0.26119403 0.66417910 0.074626866
3 0.01428571 0.13571429 0.850000000
```

Markovchain 4
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:
Finally, given a list object, it is possible to fit a markovchain object or to obtain the raw transition matrix.

```r
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 0.2000000 0.1414214 0.2000000
b 0.3464102 0.0000000 0.2828427
c 0.2886751 0.2357023 0.1666667

$confidenceLevel
[1] 0.95

$lowerEndpointMatrix
  a  b    c
a 0.07102927 0 0.07102927
b 0.03020599 0 0.00000000
c 0.02517166 0 0.00000000

$upperEndpointMatrix
  a  b    c
a 0.7289707 0.4326174 0.7289707
```
The markovchain package

b 1.0000000 0.0000000 0.8652349
c 0.9748283 0.7210291 0.4408089

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:
    a, b, c
  The transition matrix (by rows) is defined as follows:
    a b c
  a 0.3333333 0.3333333 0.3333333
  b 1.0000000 0.0000000 0.0000000
  c 0.0000000 1.0000000 0.0000000

Markovchain 3
  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 0.5000000 0.0000000 0.5000000
  b 0.5000000 0.0000000 0.5000000
  c 0.3333333 0.3333333 0.3333333

Markovchain 4
  Unnamed Markov chain
  A 2 - dimensional discrete Markov Chain defined by the following states:
    a, c
  The transition matrix (by rows) is defined as follows:
    a c
  a 0.5 0.5
  c 1.0 0.0
Markovchain 5
Unnamed Markov chain
A 2-dimensional discrete Markov Chain defined by the following states: 
\( a, c \)
The transition matrix (by rows) is defined as follows:
\[
\begin{array}{ccc}
a & c \\
a & 0 & 1 \\
c & 0 & 1 \\
\end{array}
\]

Markovchain 6
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:
\[
\begin{array}{ccc}
a & b & c \\
a & 0.3333333 & 0.3333333 & 0.3333333 \\
b & 0.3333333 & 0.3333333 & 0.3333333 \\
c & 0.5000000 & 0.5000000 & 0.0000000 \\
\end{array}
\]
If any transition contains \( \text{NA} \), it will be ignored in the results as the above example showed.

5.3. 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 non-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 year Healty 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 timex sequence since the underlying non-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
R> predict(mcCCRC, newdata = c("H", "H"), n.ahead = 5, continue = TRUE)
[1] "H" "D" "D" "D" "D"
```

### 5.4. Statistical Tests

In this section, we describe the statistical tests: assessing the Markov property (`verifyMarkovProperty`), the order (`assessOrder`), the statinarity (`assessStationarity`) of a Markov chain sequence, and the divergence test for empirically estimated transition matrices (`divergenceTest`). Most of such tests are based on the $\chi^2$ statistics. Relevant references are Kullback et al. (1962) and Anderson and Goodman (1957).

#### 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 $x_1, x_2, \ldots, x_N$ 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 $n_{ijk}$ follows a Binomial distribution with parameters $n_{ij}$ and $p_{jk}$. A classical $\chi^2$ test can check this distributional assumption, since $\sum_i \sum_j \sum_k n_{ijk} - n_{ij}hat p_{jk} \sim \chi^2(|S|^3)$ where $|S|$ is the cardinality of the state space.

```r
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)
```

Testing markovianity property on given data sequence 
Chi - square statistic is: 0.28 degrees of freedom are: 8 and corresponding p-value is:

#### 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.

<table>
<thead>
<tr>
<th>past</th>
<th>present</th>
<th>future</th>
<th>future</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6: Contingency table to assess the order for the present state a.
Using the table, the function performs the $\chi^2$ test by calling the chisq.test function. This test returns a list of the chi-squared value and the p-value. If the p-value is greater than the given significance level, we cannot reject the hypothesis that the sequence is of first order.

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

The `assessOrder` test statistic is: 26.09575 the Chi-Square d.f. are: 12 the p-value is 0.01040395

Assessing the stationarity of a Markov chain sequence

The `assessStationarity` function assesses if the transition probabilities of the given chain change over time. To be more specific, the chain is stationary if the following condition meets.

\[ p_{ij}(t) = p_{ij} \] for all \( t \) \hspace{1cm} (18)

For each possible state, we construct a contingency table of the estimated transition probabilities over time as shown in Table 7.

<table>
<thead>
<tr>
<th>time (t)</th>
<th>probability of transition to a</th>
<th>probability of transition to b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>16</td>
<td>0.44</td>
<td>0.56</td>
</tr>
</tbody>
</table>

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

Using the table, the function performs the $\chi^2$ test by calling the chisq.test function. This test returns a list of the chi-squared value and the p-value. If the p-value is greater than the given significance level, we cannot reject the hypothesis that the sequence is stationary.

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

The `assessStationarity` test statistic is: 4.181815 the Chi-Square d.f. are: 54 the p-value is 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. Bein \( f_{ij} \) the raw transition count, Kullback et al. (1962) shows that \( 2 \sum_{i=1}^{r} \sum_{j=1}^{r} f_{ij} \ln \frac{f_{ij}}{p_{ij}} \sim \chi^2 (r * (r - 1)) \). The following example is taken from Kullback et al. (1962):
The markovchain package

```r
sequence<-c(0,1,2,2,1,0,0,0,0,0,1,2,2,1,0,0,1,0,0,0,0,0,0,1,1,1,0,0,0,0,2,1,0,
+ 0,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,2,1,0,
+ 0,1,1,0,0,0,1,2,2,0,0,0,0,0,2,2,2,1,1,1,1,0,1,1,1,1,0,2,1,1,
+ 0,0,0,0,0,2,2,1,1,1,1,1,2,1,2,0,0,0,1,2,2,2,0,0,0,0,1,1)
mc=matrix(c(5/8,1/4,1/8,1/4,1/2,1/4,1/4,3/8,3/8),byrow=TRUE, nrow=3)
rownames(mc)<-colnames(mc)<-0:2; theoreticalMc<-as(mc, "markovchain")
```

Testing whether the transition matrix is compatible with

```
  0 1 2
0 51 11 8
1 12 31 9
2 6 11 10
```

ChiSq statistic is 6.551795 d.o.f are 6 corresponding p-value is 0.3642899

```
$statistic
  0
  6.551795

$dof
[1] 6

$pvalue
  0
  0.3642899
```

The second one is implemented by the `verifyHomogeneity` function, inspired by (Kullback et al. 1962, section 9). Assuming that \( i = 1, 2, \ldots, s \) DTMC samples are available and that the cardinality of the state space is \( r \) it verifies whether the \( s \) chains belongs to the same unknown one. Kullback et al. (1962) shows that its test statistics follows a chi-square law, 

\[
2 \sum_{i=1}^{s} \sum_{j=1}^{r} \sum_{k=1}^{r} f_{ijk} \ln \frac{n_{ijk}}{f_{i.} f_{.j.} f_{..}} \sim \chi^2 (r \times (r - 1)).
\]

Also the following example is taken from Kullback et al. (1962):

```r
data(kullback)
```

Testing homogeneity of DTMC underlying input list

ChiSq statistic is 275.9963 d.o.f are 35 corresponding p-value is 0

```
$statistic
  1
  275.9963
```
5.5. Continuous Times Markov Chains

Intro

The markovchain package provides functionality for continuous time Markov chains (CTMCs). CTMCs are a generalisation 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 $X$ upto 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 homogenity - $P(X(t) = j|X(s) = k) = P(X(t-s) = j|X(0) = k)$.

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 memory less property that only the exponential random variable exhibits. Therefore, this is the sought distribution, and each state $s \in S$ has an exponential holding parameter $\lambda(s)$. Since $E(T_x) = \frac{1}{\lambda(x)}$, higher the rate $\lambda(x)$, 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 $y \neq x$) - $p_{xy} = P(X(T_x) = y|X(0) = x)$ 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) $P_{ij}(t) = \delta_{ij}e^{-\lambda(t)t} + \int_0^t \lambda(i)e^{-\lambda(t)s} \sum_{k \neq i} Q_{ik} P_{kj}(t-s) ds$. 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 $\eta$, which is unique up to multiplicative factors. Moreover, for each $k \in S$, we have

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

where $\pi$ is the unique invariant measure of the embedded discrete time Markov chain $X_n$. Finally, $\eta$ satisfies

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

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

**Estimation**

Let the data set be $D = \{(s_0,t_0),(s_1,t_1),...,(s_{N-1},t_{N-1})\}$ 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 $\lambda$ 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 $Pr(D|\theta) \propto \lambda(s_0)e^{-\lambda(s_0)(t_1-t_0)}Pr(s_1|s_0) \cdot \lambda(s_1)e^{-\lambda(s_1)(t_2-t_1)}Pr(s_2|s_1) \cdots \lambda(s_{N-1})e^{-\lambda(s_{N-1})(t_N-t_{N-1})}Pr(s_N|s_{N-1})$.

Let $n(j|i)$ denote the number of $i\to j$ 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

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

**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 $i \in I$, where $I$ 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:

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

(19)

**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, byrow, 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 $\sigma$ state to the $\sigma^*$ 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
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
R> statesDist <- c(0.8, 0.2)
R> rctmc(n = 3, ctmc = molecularCTMC, initDist = statesDist, out.type = "df", include.T0 = FALSE)
```
To obtain the stationary distribution simply invoke the `steadyStates` function

```r
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
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 0 0
b 0 0 0
c 0 0 0

$errors$dtmcConfidenceInterval$upperEndpointMatrix
   a     b     c
a 0.0000000 1.0000000 0.8816179
b 0.8816179 0.0000000 0.0000000
One approach to obtain the generator matrix is to apply the \texttt{logm} function from the \texttt{expm} package on a transition matrix. Numeric issues arise, see Israel, Rosenthal, and Wei (2001). For example, applying the standard \texttt{method} ('Higham08') on \texttt{mcWeather} raises an error, whilst the alternative method (eigenvalue decomposition) is ok. The following code estimates the generator matrix of the \texttt{mcWeather} transition matrix.

\begin{verbatim}
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
\end{verbatim}

Therefore, the "half-day" transition probability for \texttt{mcWeather DTMC} is

\begin{verbatim}
R> mcWeatherHalfDayTM <- expm::expm(mcWeatherQ*.5)
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
\end{verbatim}

The \texttt{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 \texttt{markovchain} and \texttt{ctmcd} computations.

\begin{verbatim}
R> require(ctmcd)
R> require(expm)
R> #defines a function to transform a GM into a TM
\end{verbatim}
The `markovchain` package

```r
R> 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<-as.matrix(object["par"])
+   #must add importClassesFrom("markovchain",markovchain) in the NAMESPACE
+   #must add importFrom(expm, "expm")
+   transitionMatrix<-expm(generator_matrix*t)
+   out<-as(transitionMatrix,"markovchain")
+   return(out)
+ }
R> #loading ctmcd dataset
R> data(tm_abs)
R> gm0=matrix(1,8,8) #initializing
R> diag(gm0)=0
R> diag(gm0)=-rowSums(gm0)
R> gm0[8,]=0
R> gmem=gm(tm_abs,te=1,method="EM",gmguess=gm0) #estimating GM
R> mc_at_2=gm_to_markovchain(object=gmem, t=2) #converting to TM at time 2

5.6. Pseudo - Bayesian Estimation

Hu, Kiesel, and Perraudin (2002) shows 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 \( \alpha \times Q + (1 - \alpha) \times P \), where \( \alpha \) is defined for each row as

\[
\alpha_i = \frac{K_i}{v(i) + K_i} \\
K_i = \frac{v(i) - \sum_j(Y_{ij} - v(i)q_{ij})^2}{\sum_j(Y_{ij} - v(i)q_{ij})^2}
\] (20)

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

```r
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)
+   \[
+   \alpha_i = \frac{K_i}{v(i) + K_i} \\
+   K_i = \frac{v(i) - \sum_j(Y_{ij} - v(i)q_{ij})^2}{\sum_j(Y_{ij} - v(i)q_{ij})^2}
+   \] (20)
+   return(VQ/K_i_den)
+ }
```
\[ K_i \leftarrow K_i_{\text{num}}/K_i_{\text{den}} \]

\[ \# \text{get the alpha vector} \]
\[ \alpha \leftarrow K_i / (v_i+K_i) \]

\[ \# \text{empirical transition matrix} \]
\[ \text{Emp} \leftarrow \text{raw/rowSums(raw)} \]

\[ \# \text{get the estimate} \]
\[ \text{out} \leftarrow \text{matrix(0, nrow= nrow(raw), ncol=ncol(raw))} \]
\[ \text{for} (i \text{ in 1:nrow(out)}) \{ \]
\[ \text{out}[i,] \leftarrow \text{alpha}[i] * \text{apriori}[i,] + (1-\text{alpha}[i]) * \text{Emp}[i,] \]
\[ \} \]
\[ \text{return(out)} \]

We then apply it to the weather example:

```r
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> smallSample <- rmarkovchain(n=20, object = trueMc)
R> smallSampleRawTransitions <- createSequenceMatrix(stringchar = smallSample)
R> pseudoBayesEstimator(raw = smallSampleRawTransitions, apriori = aprioriMc@transitionMatrix) - trueMc@transitionMatrix
```

```
s1  s2
s1 -0.10000000  0.10000000
s2 -0.03153639  0.03153639
```

```r
R> biggerSample <- rmarkovchain(n=100, object = trueMc)
R> biggerSampleRawTransitions <- createSequenceMatrix(stringchar = biggerSample)
R> pseudoBayesEstimator(raw = biggerSampleRawTransitions, apriori = aprioriMc@transitionMatrix) - trueMc@transitionMatrix
```

```
s1  s2
s1  0.02240738 -0.02240738
s2 -0.02107810  0.02107810
```

```r
R> bigSample <- rmarkovchain(n=1000, object = trueMc)
R> bigSampleRawTransitions <- createSequenceMatrix(stringchar = bigSample)
R> pseudoBayesEstimator(raw = bigSampleRawTransitions, apriori = aprioriMc@transitionMatrix)
```

```
s1  s2
s1  0.011686151 -0.011686151
s2  0.002490415 -0.002490415
```

R>
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 \( \theta \). The object of interest is \( P(\theta | D) \) (posterior density). \( A \) represents an alphabet class, each of whose members represent a state of the chain. Therefore

\[
D = s_0 s_1 ... s_{N-1}, s_t \in A
\]

where \( N \) is the length of the data set. Also,

\[
\theta = \{ p(s|u), s \in A, u \in A \}
\]

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

Our objective is to find \( \theta \) which maximises the posterior. That is, if our solution is denoted by \( \hat{\theta} \), then

\[
\hat{\theta} = \arg\max_{\theta} P(\theta | D)
\]

where the search space is the set of right stochastic matrices of dimension \( |A| \times |A| \).

\( n(u,s) \) denotes the number of times the word \( us \) occurs in \( D \) and \( n(u) = \sum_{s \in A} n(u,s) \). The hyperparameters 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

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

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 hyperparameters determine the form of the prior distribution, which is a product of Dirichlet distributions.
\[
P(\theta) = \prod_{u \in A} \left\{ \frac{\Gamma(\alpha(u))}{\prod_{s \in A} \Gamma(\alpha(u,s))} \prod_{s \in A} p(s|u)^{\alpha(u,s)-1} \right\}
\]

where \(\Gamma(.)\) is the Gamma function. The hyperparameters 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 hyperparameter 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

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

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

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

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

\[
P(s|u) \sim Beta(n(u,s) + \alpha(u,s), n(u) + \alpha(u) - n(u,s) - \alpha(u,s))
\]

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

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

The function also returns the expected value, given by

\[
E_{\text{post}}p(s|u) = \left\{ \frac{n(u,s) + \alpha(u,s)}{n(u) + \alpha(u)}, s \in A, u \in A \right\}
\]

The variance is given by

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

The square root of this quantity is the standard error, which is returned by the function.

The confidence intervals are constructed 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

\[
P(D'|D) = \prod_{u \in A} \frac{\prod_{s \in A} \Gamma(\alpha(u)) \prod_{s \in A} \Gamma(n(u, s) + m(u, s) + \alpha(u, s))}{\prod_{s \in A} \Gamma(\alpha(u) + n(u) + m(u))}
\]

**Choosing the hyperparameters**

The hyperparameters 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 hyperparameter matrix. It is assumed that this belief matrix corresponds to the mean value of the parameters. Since the relation

\[
E_{\text{prior}} p(s|u) = \alpha(u, s)
\]

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

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

\[
\alpha(u, s) - 1 = m(u, s)
\]

where \(m(u, s)\) is the \(u > s\) 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
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 hyperparameters 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
\begin{verbatim}
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 0.3709677 0.2741935 0.3548387
rain 0.3809524 0.3809524 0.2380952
sunny 0.2135922 0.0776699 0.7087379

$expectedValue
cloudy rain sunny
cloudy 0.3692308 0.27692308 0.3538462
rain 0.3777778 0.37777778 0.2444444
sunny 0.2169811 0.08490566 0.7087379

$standardError
[,1]  [,2]  [,3]
[1,] 0.05940353 0.05508075 0.05885769
[2,] 0.07148453 0.07148453 0.06336426
[3,] 0.03984788 0.02694693 0.04438059

$confidenceInterval
$confidenceInterval$confidenceLevel
[1] 0.92

$confidenceInterval$lowerEndpointMatrix
[,1]  [,2]  [,3]
[1,] 0.2788643 0.1837403 0.2628715
[2,] 0.2728115 0.2728115 0.1312487
[3,] 0.1436350 0.0000000 0.6344429

$confidenceInterval$upperEndpointMatrix
[,1]  [,2]  [,3]
[1,] 0.4955338 0.3757588 0.4749592
[2,] 0.5431506 0.5431506 0.3509441
[3,] 0.2829567 0.1249744 1.0000000
\end{verbatim}
$\logLikelihood$

[1] -182.4175

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

[1] -151.6332

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  | 0.3709677 0.2741935 0.3548387
rain    | 0.3809524 0.3809524 0.2380952
sunny   | 0.2135922 0.0776699 0.7087379

$expectedValue$

cloudy  | rain | sunny
--------|------|------
cloudy  | 0.3692308 0.27692308 0.3538462
rain    | 0.3777778 0.37777778 0.2444444
sunny   | 0.2169811 0.08490566 0.6981132

$standardError$

[,1]       [,2]       [,3]
[1,] 0.05940353 0.05508075 0.05885769
[2,] 0.07148453 0.07148453 0.06336426
[3,] 0.03984788 0.02694693 0.04438059

$confidenceInterval$

$confidenceInterval$confidenceLevel
[1] 0.92

$confidenceInterval$lowerEndpointMatrix

[,1]       [,2]       [,3]
[1,] 0.2788643 0.1837403 0.2628715
[2,] 0.2728115 0.2728115 0.1312487
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 hyperparameters from this known transition matrix itself, and use this to obtain our MAP estimates.

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

$scaledInference

<table>
<thead>
<tr>
<th></th>
<th>cloudy</th>
<th>rain</th>
<th>sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudy</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>rain</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>sunny</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Alternatively, we can use a data sample to infer the hyperparameters.

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

$dataInference

<table>
<thead>
<tr>
<th></th>
<th>cloudy</th>
<th>rain</th>
<th>sunny</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloudy</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>rain</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>sunny</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

In order to use the inferred hyperparameter 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 hyperparameters, then the prior is uniform. This is the same as maximum likelihood.

R> data(preproglucacon)
R> preproglucacon <- preproglucacon[2]
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 0.3585271 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

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 0.3585271 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

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"

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)))
W1 <- W0 * mcWP; W1
W2 <- W0 * (mcWP ^ 2); W2
W3 <- W0 * (mcWP ^ 3); W3
```

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)
W7
```

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

```r
q <- steadyStates(mcWP)
q
```
The markovchain package

```
> mcWP <- matrix(c(0.4, 0.2, 0.4), nrow = 1)
```

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

**Finance**

Credit ratings transitions have been successfully modelled 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
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, 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
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 $CR$ matrix...
in which each component specifies the monetary amount of going from state i to state j in a single step. Henceforth, Equation 21 represents the monetary of being in state $i$.

$$c_i = c_i^S + \sum_{j=1}^{m} C_{ij}^R p_j.$$  \hspace{1cm} (21)

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 22 holds:

$$E[f_n(X_n)|X_0 = i] = e_i P^n \bar{c}.$$  \hspace{1cm} (22)

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 <- zeros(2); P[1, 1] <- .9; P[1, 2] <- .1; P[2, 2] <- .95; P[2, 1] <- .05;
R> rownames(P) <- statesNames; colnames(P) <- statesNames
R> mp <- new("markovchain", transitionMatrix = P, name = "Telephone company")
R> M <- zeros(2); M[1, 1] <- -20; M[1, 2] <- -30; M[2, 1] <- -40; M[2, 2] <- 0

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

R> c1 <- 100 + conditionalDistribution(mp, state = "customer")  \%\% M[1,]
R> c2 <- 0 + conditionalDistribution(mp, 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) * mp ^ 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
Since the proposed BM level depends on the claim on the previous period, it can be modelled 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 23:

\[
bm_{t+1} = \max (1, bm_t - 1) \cdot (\tilde N = 0) + \min (5, bm_t + 2 \cdot \tilde N) \cdot (\tilde N \geq 1). 
\] (23)

The number of claim \(\tilde N\) is a random variable that is assumed to be Poisson distributed.

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.

```r
R> getBonusMalusMarkovChain <- function(lambda) + {
+   bmMatr <- 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.

```r
R> bmMc <- getBonusMalusMarkovChain(0.05)
```

```r
R> as.numeric(steadyStates(bmMc))
```

```
[1] 0.895836079 0.045930498 0.048285405 0.005969247 0.003978772
```

If the underlying BM coefficients of the class are 0.5, 0.7, 0.9, 1.0, 1.25, this means that the average BM coefficient applied on the long run to the class is given by
The markovchain package

```r
t> sum(as.numeric(steadyStates(bmMc)) * c(0.5, 0.7, 0.9, 1, 1.25))
```

```
[1] 0.534469
```

This means that the average premium paid by policyholders in the portfolio almost halves in the long run.

Health insurance example

Actuaries quantify the risk inherent in insurance contracts evaluating the premium of insurance contract to be sold (therefore covering future risk) and evaluating the actuarial reserves of existing portfolios (the liabilities in terms of benefits or claims payments due to policyholder arising from previously sold contracts), see Deshmukh (2012) for details.

An applied example can be performed using the data from De Angelis, Paolo and Di Falco, L. (2016) that has been saved in the `exdata` folder.

```r
t> ltcDemoPath<-system.file("exdata", "ltdItaData.txt", package = "markovchain")
t> ltcDemo<-read.table(file = ltcDemoPath, header=TRUE, sep=";",dec = ".")
t> head(ltcDemo)
```

```
   age pAD pID pAI pAA
1   20 0.0004616002 0.01083364 0.0001762467 0.9993622
2   21 0.0004824888 0.01079719 0.0001710577 0.9993465
3   22 0.0004949938 0.01177076 0.0001592333 0.9993458
4   23 0.0005042935 0.01260574 0.0001605731 0.9993351
5   24 0.0005074193 0.01260574 0.0001606504 0.9993319
6   25 0.0005154267 0.01526364 0.0001643603 0.9993202
```

The data shows the probability of transition between the state of (A)ctive, to (I)ll and Dead. It is easy to complete the transition matrix.

```r
t> ltcDemo<-transform(ltcDemo,
+   pIA=0,
+   pII=1-pID,
+   pDD=1,
+   pDA=0,
+   pDI=0)
```

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

```r
t> possibleStates<-c("A","I","D")
t> 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,] "A" "A" "A" "D" "D" "D" "D" "D" "D" "D" "D" "D"
[3,] "A" "A" "A" "D" "D" "D" "D" "D" "D" "D" "D" "D"
[4,] "A" "A" "A" "I" "I" "D" "D" "D" "D" "D" "D" "D"
[5,] "A" "I" "D" "D" "D" "D" "D" "D" "D" "D" "D" "D"
[7,] "A" "A" "A" "A" "A" "D" "D" "D" "D" "D" "D" "D"
[8,] "A" "A" "A" "A" "A" "A" "D" "D" "D" "D" "D" "D"
[9,] "A" "A" "A" "A" "A" "D" "D" "D" "D" "D" "D" "D"
[10,] "A" "A" "A" "A" "A" "A" "I" "D" "D" "D" "D" "D"
```
Let's consider 1000 simulated live trajectories, for a healthy guy aged 80. We can compute the expected time a guy will be disabled starting active at age 100.

```r
R> transitionsSince80 <- getFullTransitionTable(age=80)
R> lifeTrajectories <- rmarkovchain(n=1e3, 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] 1.361
```

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

[1] 13621.67
```

### 6.4. Sociology

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
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:

<table>
<thead>
<tr>
<th></th>
<th>2nd</th>
<th>3rd</th>
<th>Bottom</th>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom</td>
<td>0.2900000</td>
<td>0.2200000</td>
<td>0.3800000</td>
<td>0.1100000</td>
</tr>
<tr>
<td>2nd</td>
<td>0.2772277</td>
<td>0.2574257</td>
<td>0.2475248</td>
<td>0.2178218</td>
</tr>
<tr>
<td>3rd</td>
<td>0.2626263</td>
<td>0.2828283</td>
<td>0.2121212</td>
<td>0.2424242</td>
</tr>
<tr>
<td>Top</td>
<td>0.1700000</td>
<td>0.2500000</td>
<td>0.1600000</td>
<td>0.4200000</td>
</tr>
</tbody>
</table>

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


R> data("preproglucagon", package = "markovchain")

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

R> mcProtein <- markovchainFit(preproglucagon$preproglucagon, + name = "Preproglucagon MC")$estimate
R> mcProtein

Preproglucagon 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:
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.3585271</td>
<td>0.1434109</td>
<td>0.16666667</td>
<td>0.3313953</td>
</tr>
<tr>
<td>C</td>
<td>0.3840304</td>
<td>0.1558935</td>
<td>0.02281369</td>
<td>0.4372624</td>
</tr>
<tr>
<td>G</td>
<td>0.3053097</td>
<td>0.1991150</td>
<td>0.15044248</td>
<td>0.3451327</td>
</tr>
<tr>
<td>T</td>
<td>0.2844523</td>
<td>0.1819788</td>
<td>0.17667845</td>
<td>0.3568905</td>
</tr>
</tbody>
</table>

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:

\[
\begin{array}{ccc}
0-49 & 50-74 & 75-UP \\
0-49 & 0.9216216 & 0.04459459 & 0.03378378 \\
50-74 & 0.5811321 & 0.24150943 & 0.17735849 \\
75-UP & 0.2345679 & 0.23456790 & 0.53086420 \\
\end{array}
\]

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
R> eig <- eigen(mcM6@transitionMatrix)
R> D <- diag(eig$values)
R> V <- eig$vectors
R> V %*% D %*% solve(V)
```

\[
\begin{array}{ccc}
[1,] & 0.9216216 & 0.04459459 & 0.03378378 \\
[2,] & 0.5811321 & 0.24150943 & 0.17735849 \\
[3,] & 0.2345679 & 0.23456790 & 0.53086420 \\
\end{array}
\]

```r
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.

Some numerical issues have been found when working with matrix algebra using R internal linear algebra kernel (the same calculations performed with MATLAB gave a more accurate result). Some temporary workarounds have been implemented. For example, the condition for row/column sums to be equal to one is valid up to fifth decimal. Similarly, when extracting the eigenvectors only the real part is taken.

Such limitations are expected to be overcome in future releases. Similarly, future versions of the package are expected to improve the code in terms of numerical accuracy and rapidity. An initial rewriting of internal function in C++ by means of Repp package (Eddelbuettel 2013) has been started.

### Acknowledgments

The package was selected for Google Summer of Code 2015 support. The authors wish to thank Michael Cole, Tobi Gutman and Mildenberger 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


**Affiliation:**
Giorgio Alfredo Spedicato
Ph.D C.Stat ACAS
UnipolSai R&D
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