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

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

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** (?) handles Multi-State Models for panel data;
- **mcmcR** (?) implements Monte Carlo Markov Chain approach;
- **hmm** (?) fits hidden Markov models with covariates;
- **mstate** fits Multi-State Models based on Markov chains for survival analysis (?).

Nevertheless, the R statistical environment (?) 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** (?) the first one. Notwithstanding, **markovchain** package (?) 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** (?) to perform efficient matrices powers; **igraph** (?) to perform pretty plotting of `markovchain` objects and **matlab** (?), 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 (?).
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. For continuous time Markov chains, refer to the CTMC vignette which is a part of the 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 $X_{n-1},\, X_{n-2}, \ldots,\, X_1$.

$$Pr\left(X_{n+1}=x_{n+1}\left|X_1=x_1,X_2=x_{2,...},X_n=x_n\right.\right)=Pr\left(X_{n+1}=x_{n+1}\left|X_n=x_n\right.\right). \tag{1}$$

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

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

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

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.

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

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

$$P=\left[\begin{array}{ccc} p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33} \end{array}\right]. \tag{4}$$

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

DTMC are explained in most theory books on stochastic processes, see **?** and **?** for example. Valuable references online available are: **?**, **?** and **?**.

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., $\exists n > 0 : \; p_{ij}^n > 0$. If both $s_i \rightarrow s_j$ and $s_j \rightarrow s_i$, then s_i and s_j are said to communicate.

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

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

The canonic 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 $k_i = gcd\{n : Pr\left(X_n = s_i \left| X_0 = s_i\right.\right) > 0\}$, 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 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

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

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 $\sum_j z_j = 1$.

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

$$\underset{n \to \infty}{lim} P^n = Z, \tag{7}$$

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

$$P = \left[\begin{array}{ccc} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{array} \right]. \tag{8}$$

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.

$$x^{(1)} = (0\,1\,0) \left[\begin{array}{ccc} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{array} \right] = (0.15\,0.45\,0.4)\,. \tag{9}$$

$$x^{(n)} = x^{(n-1)}P \to (0.15\,0.45\,0.4) \left[\begin{array}{ccc} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{array} \right] = (0.2425\,0.3725\,0.385)\,. \tag{10}$$

If, f.e., we are interested in the probability of reaching the state s_3 in two steps, then $Pr\left(X_2 = s_3 \left| X_0 = s_2 \right.\right) = 0.385$.

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 (?) are defined within the **markovchain** package as displayed:

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

Slots:

Name:            states            byrow transitionMatrix
Class:        character          logical           matrix

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

or in a shorter way, displayed below

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

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

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

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

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

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

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

3.2. Handling markovchain objects

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

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

Method	Purpose
`*`	Direct multiplication for transition matrices.
`[`	Direct access to the elements of the transition matrix.
`==`	Equality operator between two transition matrices.
`as`	Operator to convert `markovchain` objects into `data.frame` and `table` object.
`dim`	Dimension of the transition matrix.
`names`	Equal to `states`.
`plot`	`plot` method for `markovchain` objects.
`print`	`print` method for `markovchain` objects.
`show`	`show` method for `markovchain` objects.
`states`	Name of the transition states.
`t`	Transposition operator (which switches byrow slot value and modifies the transition matrix coherently).

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

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

        [,1]
sunny  0.390
cloudy 0.355
rain   0.255

R> round(after7Days, 3)
```

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

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

```
R> states(mcWeather)
```

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

```
R> names(mcWeather)
```

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

```
R> dim(mcWeather)
```

```
[1] 3
```

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

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

```
[1] 0.3
```

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

```
[1] 0.3
```

The transition matrix of a `markovchain` object can be displayed using `print` or `show` methods (the latter being less 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 (?). `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.

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

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

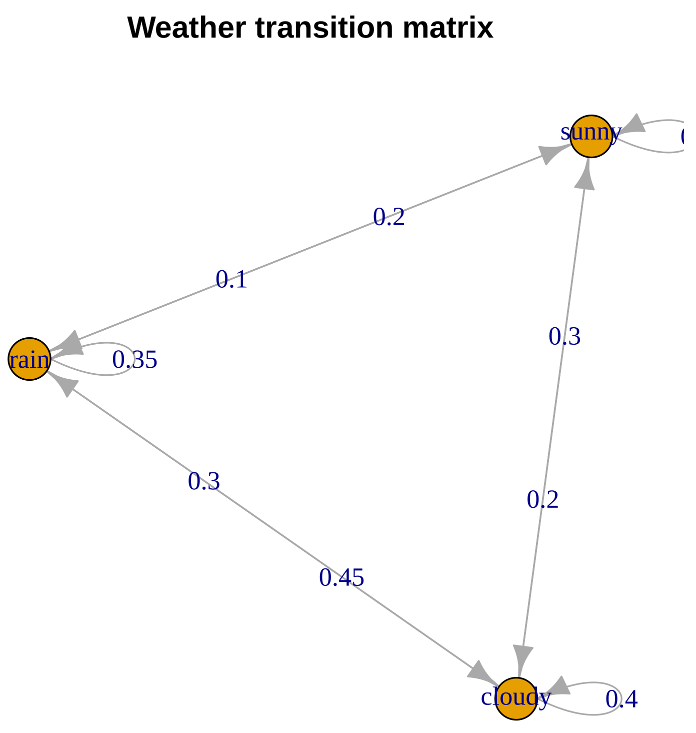

Figure 1: Weather example. Markov chain plot.

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

```
Weather
 A  3 - dimensional discrete Markov Chain with 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
```

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

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

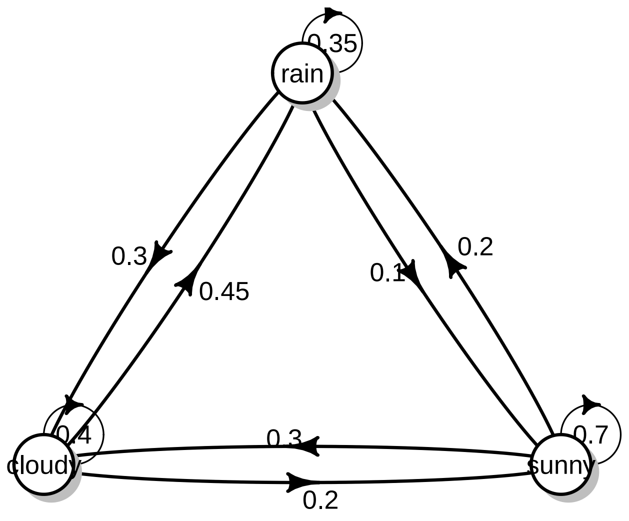

Figure 2: Weather example. Markov chain plot with diagram. plot(mcWeather, package="diagram", box.size = 0.04)

```
R> mcNew <- as(mcDf, "markovchain")
R> mcDf

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

R> mcIgraph <- as(mcWeather, "igraph")
R> library(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, 0) )
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 with 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]) {
+        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)
  0 1 2
0 0 0 1
1 0 0 1
2 0 0 1

Estimate of cov(P(1, 183))
    0 0 1 0 2 0 0 1 1 1 2 1           0 2           1 2 2 2
0 0   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
1 0   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
2 0   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
0 1   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
1 1   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
2 1   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0
0 2   0   0   0   0   0   0 -2.864030e-20 -1.126554e-19   0
1 2   0   0   0   0   0   0 -4.785736e-20  2.710505e-19   0
2 2   0   0   0   0   0   0  0.000000e+00  0.000000e+00   0

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

Unnamed Markov chain
 A  3 - dimensional discrete Markov Chain with following states
 0 1 2
 The transition matrix   (by rows)  is defined as follows
          0         1         2
0 0.0000000 0.5000000 0.5000000
1 0.5000000 0.0000000 0.5000000
2 0.3333333 0.3333333 0.3333333
```

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

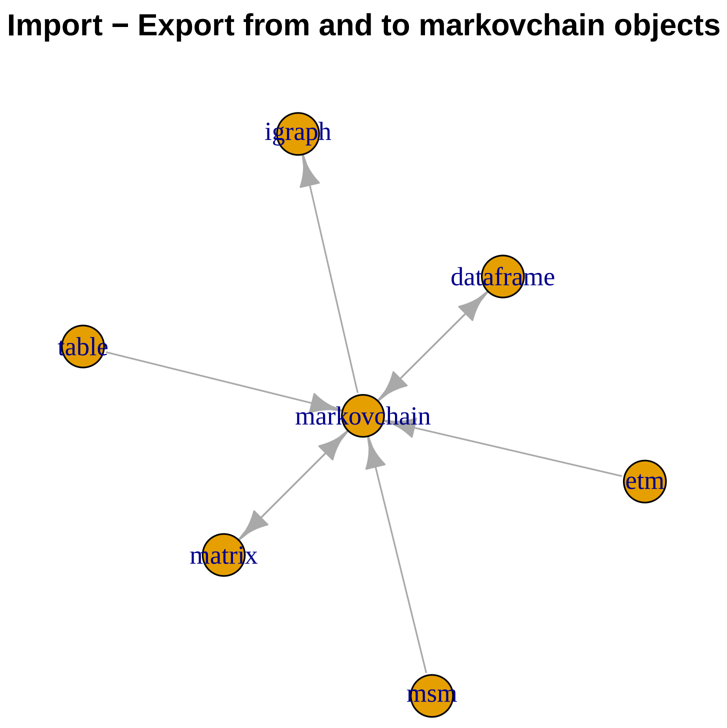

Figure 3: The **markovchain** methods for import and export.

```
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 with following states
 s1 s2 s3
 The transition matrix   (by rows)  is defined as follows
    s1  s2  s3
s1 0.1 0.8 0.1
s2 0.2 0.6 0.2
s3 0.3 0.4 0.3
```

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 `[`.

```
Continuous Care Health Community  list of Markov chain(s)
Markovchain  1
state t0
 A  3 - dimensional discrete Markov Chain with following states
 H I D
 The transition matrix   (by rows)  is defined as follows
    H   I   D
H 0.7 0.2 0.1
I 0.1 0.6 0.3
D 0.0 0.0 1.0

Markovchain  2
state t1
 A  3 - dimensional discrete Markov Chain with following states
 H I D
 The transition matrix   (by rows)  is defined as follows
    H   I   D
H 0.5 0.3 0.2
I 0.0 0.4 0.6
D 0.0 0.0 1.0

Markovchain  3
state t2
 A  3 - dimensional discrete Markov Chain with 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 with 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 with 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.

Dataset	Description
`blanden`	Mobility across income quartiles, ?.
`craigsendi`	CD4 cells, ?.
`preproglucacon`	Preproglucacon DNA basis, ?.
`rain`	Alofi Island rains, ?.
`holson`	Individual states trajectiories.

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

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

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

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, **?** and **?**.

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

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

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> conditionalDistribution(mcWeather, "sunny")

 sunny cloudy   rain
   0.7    0.2    0.1
```

The steady state(s), also known as stationary distribution(s), of the Markov chains are identified by the such described algorithm:

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.

The result is returned in matrix form.

```
R> steadyStates(mcWeather)

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

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

```
R> gamblerRuinMarkovChain <- function(moneyMax, prob = 0.5) {
+    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,] 1 0 0 0 0
[2,] 0 0 0 0 1
```

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

```
R> absorbingStates(mcGR4)

[1] "0" "4"

R> absorbingStates(mcWeather)

character(0)
```

The key function used within **?** (and **markovchain**'s derived functions) is `.commclassKernel`, that is called below.

```
R> .commclassesKernel <- function(P){
+    m <- ncol(P)
+          stateNames <- rownames(P)
+          T <- zeros(m)
+          i <- 1
+          while (i <= m) {
+                  a <- i
+                  b <- zeros(1,m)
+                  b[1,i] <- 1
+                   old <- 1
+                   new <- 0
+                   while (old != new) {
+                           old <- sum(find(b > 0))
+                           n <- size(a)[2]
+                           matr <- matrix(as.numeric(P[a,]), ncol = m,
+                        nrow = n)
+                           c <- colSums(matr)
+                           d <- find(c)
+                           n <- size(d)[2]
+                           b[1,d] <- ones(1,n)
+                           new <- sum(find(b>0))
+                           a <- d
+                   }
+                   T[i,] <- b
+                   i <- i+1 }
+           F <- t(T)
+           C <- (T > 0)&(F > 0)
+           v <- (apply(t(C) == t(T), 2, sum) == m)
+           colnames(C) <- stateNames
+           rownames(C) <- stateNames
+           names(v) <- stateNames
+           out <- list(C = C, v = v)
+           return(out)
+   }
```

The `.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 `summary` method for `markovchain` objects works.

The example matrix used in **?** well exemplifies the purpose of the function.

```
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[7, 7] <- 1/4; P[7,9] <- 3/4;
R> P[8, c(3, 4, 8, 10)] <- 1/4;
```

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

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

$v
    a     b     c     d     e     f     g     h     i     j
 TRUE  TRUE  TRUE FALSE FALSE  TRUE  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},{d,e},{f}
Transient classes:
{d,e},{h},{j}
The Markov chain is not irreducible
The absorbing states are: f
```

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

```
R> transientStates(probMc)

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

Listings from **?** have been adapted into `canonicForm` method that turns a Markov chain into canonic form.

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

Probability MC
 A  10 - dimensional discrete Markov Chain with following states
 a b c d e f g h i j
 The transition matrix   (by rows)  is defined as follows
    a         b    c         d         e f         g    h         i
a 0.5 0.0000000 0.50 0.0000000 0.0000000 0 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 0.3333333 0.3333333 0 0.0000000 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
          j
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> probMcCanonic

Probability MC
 A  10 - dimensional discrete Markov Chain with following states
 a c b g i f d e h j
 The transition matrix   (by rows)  is defined as follows
    a    c         b         g         i f         d         e    h
a 0.5 0.50 0.0000000 0.0000000 0.0000000 0 0.0000000 0.0000000 0.00
c 1.0 0.00 0.0000000 0.0000000 0.0000000 0 0.0000000 0.0000000 0.00
b 0.0 0.00 0.3333333 0.6666667 0.0000000 0 0.0000000 0.0000000 0.00
g 0.0 0.00 0.0000000 0.2500000 0.7500000 0 0.0000000 0.0000000 0.00
i 0.0 0.00 1.0000000 0.0000000 0.0000000 0 0.0000000 0.0000000 0.00
f 0.0 0.00 0.0000000 0.0000000 0.0000000 1 0.0000000 0.0000000 0.00
d 0.0 0.00 0.0000000 0.0000000 0.0000000 0 0.0000000 1.0000000 0.00
e 0.0 0.00 0.0000000 0.0000000 0.3333333 0 0.3333333 0.3333333 0.00
h 0.0 0.25 0.0000000 0.0000000 0.0000000 0 0.2500000 0.0000000 0.25
j 0.0 0.00 0.3333333 0.0000000 0.0000000 0 0.0000000 0.3333333 0.00
```

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

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

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

```
[1] TRUE
```

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

```
[1] FALSE
```

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

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

```
[1] TRUE
```

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

```
[1] 2
```

The example Markov chain found in Mathematica web site (?) has been used, and is plotted in Figure 4.

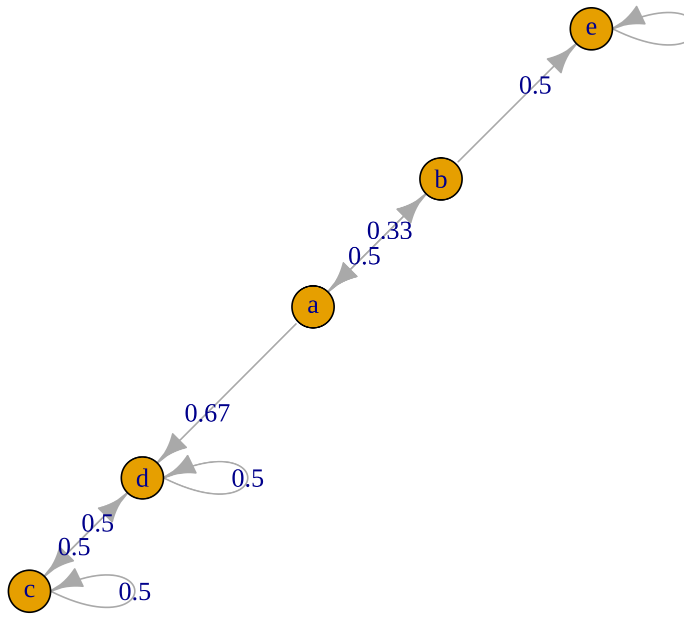

Figure 4: Mathematica 9 example. Markov chain plot.

```
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)
```

```
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
```

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

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

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

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

```
[1] 0.121
```

5. Statistical analysis

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

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

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

5.1. Simulation

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

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

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

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.

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

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

5.2. Estimation

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

Equation 11 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

$$\hat{p}_{ij}^{MLE} = \frac{n_{ij}}{\sum\limits_{u=1}^{k} n_{iu}}. \tag{11}$$

Equation 12 shows the `standardError` of the MLE (?).

$$SE_{ij} = \frac{\hat{p}_{ij}^{MLE}}{\sqrt{n_{ij}}} \tag{12}$$

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

```
Weather MLE
 A  3 - dimensional discrete Markov Chain with following states
 cloudy rain sunny
 The transition matrix   (by rows)  is defined as follows
          cloudy       rain     sunny
cloudy 0.3944954 0.32110092 0.2844037
rain   0.4050633 0.37974684 0.2151899
sunny  0.1931818 0.07954545 0.7272727
```

```
R> weatherFittedMLE$standardError
```

```
           cloudy       rain      sunny
cloudy 0.06015999 0.05427596 0.05108041
rain   0.07160575 0.06933197 0.05219121
sunny  0.03313041 0.02125942 0.06428243
```

The Laplace smoothing approach is a variation of the MLE, where the n_{ij} is substituted by $n_{ij} + \alpha$ (see Equation 13), being α an arbitrary positive stabilizing parameter.

$$\hat{p}_{ij}^{LS} = \frac{n_{ij} + \alpha}{\sum_{u=1}^{k} (n_{iu} + \alpha)} \tag{13}$$

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

```
Weather LAPLACE
 A  3 - dimensional discrete Markov Chain with following states
 cloudy rain sunny
 The transition matrix   (by rows)  is defined as follows
          cloudy       rain     sunny
cloudy 0.3944786 0.32110428 0.2844171
rain   0.4050361 0.37972922 0.2152347
sunny  0.1932057 0.07958871 0.7272056
```

Both MLE and Laplace approach are based on the `createSequenceMatrix` functions that converts a data (character) sequence into a contingency table, showing the $(X_t = i, X_{t+1} = j)$ distribution within the sample, as code below shows.

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

```
       cloudy rain sunny
cloudy     43   35    31
rain       32   30    17
sunny      34   14   128
```

An issue occurs when the sample contains only one realization of a state (say X_β) 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.

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 ${p^{\hat{MLE}}}_{ij}$ estimate is provided as well.

```
R> weatherFittedBOOT <- markovchainFit(data = weathersOfDays,
+                                       method = "bootstrap", nboot = 100)
R> weatherFittedBOOT$estimate

BootStrap Estimate
 A  3 - dimensional discrete Markov Chain with following states
 cloudy rain sunny
 The transition matrix   (by rows)  is defined as follows
          cloudy      rain     sunny
cloudy 0.3981566 0.3198228 0.2820206
rain   0.4141900 0.3717835 0.2140265
sunny  0.1919972 0.0802571 0.7277457

R> weatherFittedBOOT$standardError

            cloudy        rain       sunny
cloudy 0.004972001 0.004718384 0.004800766
rain   0.004972647 0.005072550 0.004558877
sunny  0.003539623 0.002329060 0.003683298
```

The bootstrapping process can be done in `parallel`.

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

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

```
R> RcppParallel::setNumThreads(4)
```

For more details, please refer to **RcppParallel** (http://rcppcore.github.io/RcppParallel/).

For all the fitting methods, the `logLikelihood` (?) denoted in Equation 14 is provided.

$$LLH = \sum_{i,j} n_{ij} * log(p_{ij}) \tag{14}$$

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

```
R> weatherFittedMLE$logLikelihood

[1] -334.9191

R> weatherFittedBOOT$logLikelihood

[1] -334.9384
```

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 15 and 16 (?) show the `confidenceInterval` of a fitting. Note that each entry of the matrices is bounded between 0 and 1.

$$LowerEndpoint_{ij} = p_{ij} - zscore(CL) * SE_{ij} \tag{15}$$

$$UpperEndpoint_{ij} = p_{ij} + zscore(CL) * SE_{ij} \tag{16}$$

```
R> weatherFittedMLE$confidenceInterval

$confidenceLevel
[1] 0.95

$lowerEndpointMatrix
          cloudy       rain     sunny
cloudy 0.2955410 0.23182491 0.2003839
rain   0.2872823 0.26570589 0.1293430
sunny  0.1386871 0.04457683 0.6215375
```

```
$upperEndpointMatrix
          cloudy      rain     sunny
cloudy 0.4934498 0.4103769 0.3684235
rain   0.5228443 0.4937878 0.3010368
sunny  0.2476765 0.1145141 0.8330079
```

```
R> weatherFittedBOOT$confidenceInterval
```

```
$confidenceLevel
[1] 0.95

$lowerEndpointMatrix
          cloudy       rain     sunny
cloudy 0.3899784 0.31206175 0.2741241
rain   0.4060107 0.36343986 0.2065279
sunny  0.1861750 0.07642614 0.7216873

$upperEndpointMatrix
          cloudy       rain     sunny
cloudy 0.4063348 0.32758386 0.2899172
rain   0.4223693 0.38012707 0.2215252
sunny  0.1978193 0.08408807 0.7338042
```

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 **?** that were themselves based on the **?** paper.

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

```
$confidenceLevel
[1] 0.95

$lowerEndpointMatrix
          cloudy       rain     sunny
cloudy 0.3027523 0.22935780 0.1926606
rain   0.2911392 0.26582278 0.1012658
sunny  0.1306818 0.01704545 0.6647727

$upperEndpointMatrix
          cloudy      rain     sunny
cloudy 0.5033721 0.4299776 0.3932804
rain   0.5230920 0.4977756 0.3332186
sunny  0.2581626 0.1445262 0.7922535
```

The functions for fitting DTMC have mostly been rewritten in C++ using **Rcpp ?** since version 0.2.

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

```
R> data(holson)
R> singleMc<-markovchainFit(data=holson[,2:12],name="holson")
R> mcListFit<-markovchainListFit(data=holson[,2:12],name="holson")
R> mcListFit$estimate[[1]]
```

```
time1
 A  3 - dimensional discrete Markov Chain with 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
```

The maximum a posteriori method (MAP) has been implemented using Bayesian inference (**?**). For details on usage, refer to the stand-alone vignette for MAP (**?**).

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

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

5.4. Statistical Tests

Divergence test for empirically estimated transition matrices

The ϕ-divergence test **?** between two empirically estimated transition matrices $m1$ and $m2$ is given by

$$T_n^{\phi}(m_1, m_2, mc) = \frac{2n}{\phi''(1)} \sum_{i=1}^{M} \frac{v_{i*}}{n} \sum_{j=1}^{M} p^0(i,j)\phi\left(\frac{m_1(i,j)}{m_2(i,j)}\right) \tag{17}$$

where mc is the markov chain sequence, n is the length of the sequence, M is the number of states, v_{i*} is the number of transitions starting from the state i, and ϕ is

$$\phi(x) = x \log x - x + 1. \tag{18}$$

6. Applications

This section shows applications of DTMC in various fields.

6.1. Weather forecasting

Markov chains provide a simple model to predict the next day's weather given the current meteorological condition. The first application herewith shown is the "Land of Oz example" from **?**, the second is the "Alofi Island Rainfall" from **?**.

Land of Oz

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

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

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

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

     rainy nice snowy
[1,]   0.5    0   0.5

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

     rainy nice snowy
[1,] 0.375 0.25 0.375

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

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

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

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

         rainy      nice     snowy
[1,] 0.4000244 0.1999512 0.4000244
```

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

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

     rainy nice snowy
[1,]   0.4  0.2   0.4
```

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

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

         rainy      nice     snowy
[1,] 0.4000244 0.2000122 0.3999634

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

```
          rainy      nice     snowy
[1,] 0.3999634 0.2000122 0.4000244
```

Alofi Island Rainfall

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

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

  0 1-5  6+
548 295 253
```

The underlying transition matrix is estimated as follows.

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

Alofi MC
 A  3 - dimensional discrete Markov Chain with following states
 0 1-5 6+
 The transition matrix   (by rows)  is defined as follows
            0       1-5        6+
0   0.6605839 0.2299270 0.1094891
1-5 0.4625850 0.3061224 0.2312925
6+  0.1976285 0.3122530 0.4901186
```

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

```
R> steadyStates(mcAlofi)

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

6.2. Finance and Economics

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

Finance

Credit ratings transitions have been successfully 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 (**?**), carrying Standard & Poor's published data.

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

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

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

[1] "D"
```

Economics

For a recent application of **markovchain** in Economic, see **?**.
A dynamic system generates two kinds of economic effects (**?**):

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

Let the monetary amount of being in a particular state be represented as a m-dimensional column vector c^{S}, while let the monetary amount of a transition be embodied in a C^R matrix in which each component specifies the monetary amount of going from state i to state j in a single step. Henceforth, Equation 19 represents the monetary of being in state i.

$$c_i = c_i^{\mathrm{S}} + \sum_{j=1}^{m} C_{ij}^{\mathrm{R}} p_{ij}. \tag{19}$$

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 20 holds:

$$E\left[f_n\left(X_n\right)|X_0 = i\right] = e_i P^n \bar{c}. \tag{20}$$

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

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

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

[1] 48.96009
```

6.3. Actuarial science

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

MPTL Bonus Malus

Bonus Malus (BM) contracts grant the policyholder a discount (enworsen) as a function of the number of claims in the experience period. The discount (enworsen) is applied on a premium that already allows for known (a priori) policyholder characteristics (**?**) and it usually depends on vehicle, territory, the demographic profile of the policyholder, and policy coverages deep (deductible and policy limits).
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 21:

$$bm_{t+1} = \max\left(1, bm_t - 1\right) * \left(\tilde{N} = 0\right) + \min\left(5, bm_t + 2 * \tilde{N}\right) * \left(\tilde{N} \geq 1\right). \qquad (21)$$

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

```
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)
+  }
R>
```

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> bmMc <- getBonusMalusMarkovChain(0.05)
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

```
R> 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). Key quantities of actuarial interest are: the expected present value of future benefits, $PVFB$, the (periodic) benefit premium, P, and the present value of future premium $PVFP$. A level benefit premium could be set equating at the beginning of the contract $PVFB = PVFP$. After the beginning of the contract the benefit reserve is the difference between $PVFB$ and $PVFP$. The example comes from ?. The interest rate is 5%, benefits are payable upon death (1000) and disability (500). Premiums are payable at the beginning of period only if the policyholder is active. The contract term is three years.

```
R> mcHI <- new("markovchain", states = c("active", "disable", "withdrawn",
+                                       "death"),
+           transitionMatrix = matrix(c(0.5, .25, .15, .1,
+                                     0.4, 0.4, 0.0, 0.2,
+                                     0, 0, 1, 0,
+                                     0, 0, 0, 1), byrow = TRUE, nrow = 4))
R> benefitVector <- as.matrix(c(0, 0, 500, 1000))
```

The policyholders is active at T_0. Therefore the expected states at $T_1, \ldots T_3$ are calculated in the following.

```
R> T0 <- t(as.matrix(c(1, 0, 0, 0)))
R> T1 <- T0 * mcHI
R> T2 <- T1 * mcHI
R> T3 <- T2 * mcHI
```

The present value of future benefit at T0 is given by

```
R> PVFB <- T0 %*% benefitVector * 1.05 ^ -0 +
+    T1 %*% benefitVector * 1.05 ^ -1+
+    T2 %*% benefitVector * 1.05 ^ -2 + T3 %*% benefitVector * 1.05 ^ -3
```

The yearly premium payable whether the insured is alive is as follows.

```
R> P <- PVFB / (T0[1] * 1.05 ^- 0 + T1[1] * 1.05 ^ -1 + T2[1] * 1.05 ^ -2)
```

The reserve at the beginning of the second year, in the case of the insured being alive, is as follows.

```
R> PVFB <- T2 %*% benefitVector * 1.05 ^ -1 + T3 %*% benefitVector * 1.05 ^ -2
R> PVFP <- P*(T1[1] * 1.05 ^ -0 + T2[1] * 1.05 ^ -1)
R> V <- PVFB - PVFP
R> V

         [,1]
[1,] 300.2528
```

6.4. Sociology

Markov chains have been actively used to model progressions and regressions between social classes. The first study was performed by ?, while a more recent application can be found in ?. 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.

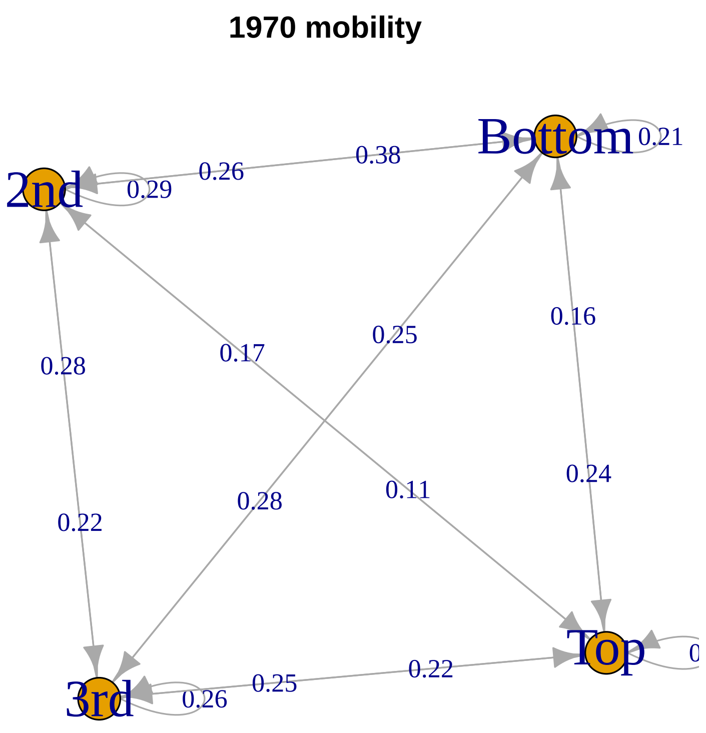

Figure 5: 1970 UK cohort mobility data.

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

Unnamed Markov chain
 A  4 - dimensional discrete Markov Chain with following states
 Bottom 2nd 3rd Top
 The transition matrix   (by rows)  is defined as follows
             2nd       3rd    Bottom       Top
Bottom 0.2900000 0.2200000 0.3800000 0.1100000
2nd    0.2772277 0.2574257 0.2475248 0.2178218
3rd    0.2626263 0.2828283 0.2121212 0.2424242
Top    0.1700000 0.2500000 0.1600000 0.4200000
```

The underlying transition graph is plotted in Figure 5.

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

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

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

6.5. Genetics and Medicine

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

Genetics

? discusses the use of Markov chains in model Preprogucacon gene protein bases sequence. The `preproglucacon` dataset in **markovchain** contains the dataset shown in the package.

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

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

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

Preproglucacon MC
 A  4 - dimensional discrete Markov Chain with following states
 A C G T
 The transition matrix   (by rows)  is defined as follows
          A         C          G         T
A 0.3585271 0.1434109 0.16666667 0.3313953
C 0.3840304 0.1558935 0.02281369 0.4372624
G 0.3053097 0.1991150 0.15044248 0.3451327
T 0.2844523 0.1819788 0.17667845 0.3568905
```

Medicine

Discrete-time Markov chains are also employed to study the progression of chronic diseases. The following example is taken from **?**. 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 with following states
 0-49 50-74 75-UP
 The transition matrix   (by rows)  is defined as follows
           0-49       50-74      75-UP
0-49  0.9216216 0.04459459 0.03378378
50-74 0.5811321 0.24150943 0.17735849
75-UP 0.2345679 0.23456790 0.53086420
```

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

```
R> eig <- eigen(mcM6@transitionMatrix)
R> D <- diag(eig$values)

R> V <- eig$vectors
R> V %*% D %*% solve(V)

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

R> d <- D ^ (1/6)
R> M <- V %*% d %*% solve(V)
R> mcM1 <- new("markovchain", transitionMatrix = M, states = hivStates)
```

7. Discussion, issues and future plans

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

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 intitial rewriting of internal function in C++ by means of **Rcpp** package (**?**) has been started.

Aknowledgments

The author wishes to thank Michael Cole, Tobi Gutman and Mildenberger Thoralf for their suggestions and bug checks. A very special thanks also to Tae Seung Kang (and to the other Google Summer of Code 2015 candidates) for having rewritten the fitting functions into **Rcpp**. A final thanks also to Dr. Simona C. Minotti and Dr. Mirko Signorelli for their support in drafting this version of the vignettes.

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