Discrete Time Markov Chains with R

by Giorgio Alfredo Spedicato

Abstract The **markovchain** package aims to provide S4 classes and methods to easily handle Discrete Time Markov Chains (DTMCs), filling the gap with what is currently available in the CRAN repository. In this work, I provide an exhaustive description of the main functions included in the package, as well as hands-on examples.

Introduction

DTMCs are a notable class of stochastic processes. Although their basic theory is not overly complex, they are extremely effective to model categorical data sequences (Ching et al., 2008). To illustrate, notable applications can be found in linguistic (see Markov's original paper Markov (1907)), information theory (Google original algorithm is based on Markov Chains theory, Lawrence Page et al. (1999)), medicine (transition across HIV severity states, Craig and Sendi (2002)), economics and sociology (Jones (1997) shows an application of Markov Chains to model social mobility).

The **markovchain** package (Spedicato, Giorgio Alfredo, 2016) provides an efficient tool to create, manage and analyse Markov Chains (MCs). Some of the main features include the possibility to: validate the input transition matrix, plot the transition matrix as a graph diagram, perform structural analysis of DTMCs (e.g. classification of transition matrices and states, analysis of the stationary distribution, etc ...), perform statistical inference (such as fitting transition matrices from various input data, simulating stochastic processes trajectories from a given DTMC, etc..). The author believes that no R package provides a unified infrastructure to easily manage DTMCs as **markovchain** does at the time this paper is being drafted.

The package targets data scientists using DTMC, Academia members, supporting faculty instructors, as well as students of undergraduate courses on Stochastic Processes.

The paper will be organized as follows: Section 2.2 gives a brief overview on R packages and alternative software that provide similar functionalities, Section 2.3 reviews DTMC basic theory, Section 2.4 discusses the package design and structure, Section 2.5 shows how to create and manipulate homogeneous DTMCs, Section 2.6 and Section 2.7 respectively present the functions created to perform structural analysis, and statistical inference on DTMCs. A brief overview of the functionalities written to deal with non - homogeneous discrete dime Markov chains (NHDTMCs) is provided in Section 2.8. A discussion on numerical reliability and computational performance is provided in Section 2.9. Finally, Section 2.10 draws final conclusions and briefly discusses future potential developments of the package.

Analysis of existing DTMC-related software

As reviewed later in more details, a DTMC is defined by a stochastic matrix known as transition matrix (TM), which is a square matrix satisfying Equation 1.

$$\begin{cases} P_{ij} \in [0,1] \, \forall i,j \\ \sum_i P_{ij} = 1 \end{cases} \tag{1}$$

Although defining a stochastic matrix is trivial in any mathematical or statistical software, a DTMC dedicated infrastructure can provide object oriented programmed methods to verify the validity of the input data (i.e. if the input matrix is a stochastic one) , as well as to perform structural analysis on DTMC objects.

Various packages mention MCs - related models in the CRAN repository, whereby a few of them will be now reviewed. The **clickstream** package (Scholz, 2016), on CRAN since 2014, aims to model websites click stream using higher order Markov Chains. It provides a `MarkovChain` S4 class that is similar to the `markovchain` class. Further, **DTMCPack** (Nicholson, William, 2013) and **MTCM** (Bessi, Alessandro, 2015) also work with DTMCs but provide even more limited functions: the first one focuses on creating simulations from a given DTMC, whilst the second contains only one function for estimating the underlying transition matrix for a given categorical sequence. Moreover, none of them appears to have been updated since 2015. The coverage of functionalities provided by **markovchain** package for analysing DTMCs appears to be more complete than the above mentioned packages, since none of them provides methods for importing or coercing transition matrices from other objects, such as R `matrices` or `data.frames`. Furthermore, **markovchain** is the only package providing a quick graph plotting facility for DTMC objects. The same applies when considering the functionalities used

to perform structural analysis of transition matrices and to fit DTMCs from various kind of input data. More interestingly, the **FuzzyStatProb** package (Pablo J. Villacorta and José L. Verdegay, 2016) gives an alternative approach for estimating the parameters of DTMCs using "fuzzy logic".

This review voluntarily omits discussing packages that are not specifically focused on DTMC. Nonetheless, the **depmixS4** (Visser and Speekenbrink, 2010) and the **HMM** (Himmelmann, 2010) packages deal with Hidden Markov Models (HMMs). In addition, the number of R packages focused on the estimation of statistical models using the Markov Chain Monte Carlo simulation approach is sensibly bigger. Finally, the **msm** (Jackson, 2011), **heemod** (Antoine Filipovi et al., 2017) and the **TPmsm** packages (Artur Araújo et al., 2014) focus on health applications of multi - state analysis using different kinds of models, including Markov-related ones among them.

Finally, among other well known software used in Mathematics and Statistics, only Mathematica (Wolfram Research, Inc., 2013) provides routines specifically written to deal with Markov processes at the author's knowledge. Nevertheless, the analysis of DTMCs could be easily handled within the Matlab programming language (MATLAB, 2017) due to its well known linear algebra capabilities.

Review of underlying theory

In this section a brief review of the theory of DTMCs is presented. Readers willing to dive deeper can inspect Cassandras (1993) and Grinstead and Snell (2012).

A *DTMC* is a stochastic process whose domain is a discrete set of states, $\{s_1, s_2, \ldots, s_k\}$. The chain starts in a generic state at time zero and moves from a state to another by steps. Let p_{ij} be the probability that a chain currently in state s_i moves to state s_j at the next step. The key characteristic of DTMC processes is that p_{ij} does not depend upon the previous state in the chain. The probability p_{ij} for a (finite) DTMC is defined by a transition matrix previously introduced (see Equation 1). It is also possible to define the TM by column, under the constraint that the sum of the elements in each column is 1.

To illustrate, a few toy - examples on transition matrices are now presented; the "Land of Oz" weather Matrix, Kemeny et al. (1974). Equation 2 shows the transition probability between (R)ainy, (N)ice and (S)now weathers.

$$\begin{bmatrix} & R & N & S \\ R & 0.5 & 0.25 & 0.25 \\ N & 0.5 & 0 & 0.5 \\ S & 0.25 & 0.25 & 0.5 \end{bmatrix} \tag{2}$$

Further, the Mathematica Matrix 3, taken from the Mathematica 9 Computer Algebra System manual (Wolfram Research, Inc., 2013), that will be used when discussing the analysis the structural proprieties of DTMCs, is as follows:

$$\begin{bmatrix} & A & B & C & D \\ A & 0.5 & 0.5 & 0 & 0 \\ B & 0.5 & 0.5 & 0 & 0 \\ C & 0.25 & 0.25 & 0.25 & 0.25 \\ D & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Simple operations on TMs allow to understand structural proprieties of DTMCs. For example, the $n-th$ power of P is a matrix whose entries represent the probabilities that a DTMC in state s_i at time t will be in state s_j at time $t+n$. In particular, if u_t is the probability vector for time t (that is, a vector whose $j-th$ entries represent the probability that the chain will be in the $j-th$ state at time t), then the distribution of the chain at time $t+n$ is given by $u_n = u * P^n$. Main properties of Markov chains are now presented.

A state s_i is *reachable* from state s_j if $\exists n \rightarrow p_{ij}^n > 0$. If the inverse is also true then s_i and s_j are said to *communicate*. For each MC, there always exists a unique decomposition of the state space into a sequence of disjoint subsets in which all the states within each subset communicate. Each subset is known as a *communicating class* of the MC. It is possible to link this decomposition to graph theory, since the communicating classes represent the strongly connected components of the graph underlying the transition matrix (Jarvis and Shier, 1999).

A state s_j of a DTMC is said to be *absorbing* if it is impossible to leave it, meaning $p_{jj} = 1$. An *absorbing Markov chain* is a chain that contains at least one absorbing state which can be reached, not necessarily in a single step. Non - absorbing states of an absorbing MC are defined as *transient states*. In addition, states that can be visited more than once by the MC are known as *recurrent states*.

If a DTMC contains $r \geq 1$ absorbing states, it is possible to re-arrange their order by separating

transient and absorbing states such that the t transient states come before the r absorbing ones. Such re-arranged matrix is said to be in *canonical form* (see Equation 4), where its composition can be represented by sub - matrices.

$$\begin{pmatrix} Q_{t,t} & R_{t,r} \\ 0_{r,t} & I_{r,r} \end{pmatrix} \tag{4}$$

Such matrices are: Q (a t-square sub - matrix containing the transition probabilities across transient states), R (a nonzero t-by-r matrix containing transition probabilities from non-absorbing to absorbing states), 0 (an r-by-t zero matrix), and I_r (an r-by-r identity matrix). It is possible to use these matrices to calculate various structural proprieties of the DTMC. Since $\lim_{n\to\infty} Q^n = 0$, it can be shown that in every absorbing matrix the probability to be eventually absorbed is 1, regardless of the state where the MC is initiated.

Further, in Equation 5 the *fundamental matrix* is presented, where the generic n_{ij} entry expresses the expected number of times the process will transit in state s_j, given that it started in state s_i. Also, the i-th entry of vector $t = N * \bar{1}$, being $\bar{1}$ a t-sized vector of ones, expresses the expected number of steps before an absorbing DTMC, started in state s_i, is absorbed. The b_{ij} entries of matrix $B = N * R$ are the probabilities that a DTMC started in state s_i will eventually be absorbed in state s_j. Finally, the probability of visiting the transient state j when starting from the transient state i is the h_{ij} entry of the matrix $H = (N - I_t) * N_{dg}^{-1}$, being dg the diagonal operator.

$$N = (I - Q)^{-} 1 = I + \sum_{i=0,1,\ldots,\infty} Q^i \tag{5}$$

A DTMC is said to be *ergodic* if there exist a number N such that it is possible to reach every state in at most N steps. If $P^n > 0$ for some n, then P is a *regular* DTMC.

Fixed row vectors $\bar{w}$, also known as *steady state vectors*, are vectors such that $\bar{w}P = \bar{w}$. Mathematically, they correspond to eigenvectors associated to unitary eigenvalues of the TM. It can be shown that $\lim_{n\to\infty} v * P^n = w$ and that $\lim_{n\to\infty} P^n = W$, where v is a generic stochastic vector and w is a matrix where all rows are $\bar{w}$.

The *mean first passage time* m_{ij} is the expected the number of steps needed to reach state s_j starting from state s_i, where $m_{ii} = 0$ by convention. For ergodic MCs, r_i is the mean recurrence time, that is the expected number of steps to return to s_i from s_i. It is possible to prove that $r_i = \frac{1}{w_i}$, where w_i is the i-th entry of $\bar{w}$. Further, let D be a diagonal matrix, in which the diagonal elements come from r_i, and let C be a matrix filled with ones. It is then possible to get the mean first passage matrix M from Equation 6.

$$(I - P) = C - M \tag{6}$$

Let $Z = (I - P + M)^{-1}$ be the fundamental matrix for an ergodic MC. It is possible to write m_{ij} as a function of Z and $\bar{w}$, as Equation 7 shows.

$$m_{ij} = \frac{z_{jj} - z_{ij}}{w_j} \tag{7}$$

A further topic in structural analysis of irreducible DTMCs is periodicity. The period of a state s_i, denoted as $d(i)$, is the greatest common divisor of n for which $p_{ii}^n > 0$. If the period is 1, the state is aperiodic, while if the period is greater than 2, the state is periodic; all states in the same class share the same period.

Given a generic DTMC, it is possible to simulate stochastic trajectories following the underlying MC from the TM. Given an initial state $s(t) = j$, the $s(t+1)$ state is sampled from the multinomial distribution whose probabilities are expressed by the j-th row. The sampled state indicates from which row the probabilities to sample $s(t+2)$ are taken from. Also, given a sample sequence, it is possible to estimate the TM of the underlying DTMC. Equation 8 shows the maximum likelihood estimator (MLE) of the TM p_{ij} entry, being the n_{ij} elements the number of sequences $\left(X_t = s_i, X_{t+1} = s_j\right)$ counted in the sample, that is:

$$\hat{p}_{ij}^{MLE} = \frac{n_{ij}}{\sum_{u=1}^{k} n_{iu}}. \tag{8}$$

Equation 10 shows asymptotic confidence intervals for p_{ij}. The bootstrap approach allows to define non - parametric ones.

$$LowerEndpoint_{ij} = p_{ij} - 1.96 * SE_{ij} \quad (9)$$

$$UpperEndpoint_{ij} = p_{ij} + 1.96 * SE_{ij} \quad (10)$$

The mode of the X_{t+1} conditional distribution given $X_t = s_j$ represents the prediction from a given DTMC and the current chain state $X_t = s_j$.

In conclusion, the **markovchain** package allows to perform statistical analysis on NHDTMCs, in the special case where they can be treated as sequential lists of DTMCs.

Implementation design and details

The **markovchain** package has been originally written in "native" R. Most functions have been therefore ported in **Rcpp** (Eddelbuettel, Dirk, 2013) since 2015, yielding sensible improvements in computational time. Other dependencies of **markovchain** are: **igraph** Csardi, Gabor and Nepusz, Tamas (2006), **matlab** Roebuck (2014), **Matrix** Bates and Maechler (2016) and **expm** Goulet et al. (2015) (for operation on matrices), and the **method** package for defining S4 classes.

Homogeneous DTMCs are defined by a dedicated S4 class, `"markovchain"`. Such class is defined by the following slots:

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

A `"markovchain"` S4 class has been designed based on Chambers, J.M. (2008) suggestions. For example, a S4 `setValidity` method checks the coherence of any newly created `markovchain` object, by verifying that either the rows or columns of the transition matrix sum to one, and that all elements are bounded between 0 and 1.
Another S4 class,`"markovchainList"`, has been created for handling non - homogeneous DTMCs. Finally, the package provides functions and S4 to analyse continuous MCs, as well as higher order MCs, although their discussion is beyond the scope of this paper.

Three vignettes documents the **markovchain** package. The first one broadly describes the functionalities of the package and it also presents real - world applications of DTMCs using the package. The second one, written using **knit** and **rmarkdown**, is a beamer presentation that quickly introduces the key functionalities of the package. The third one presents experimental functions for higher order and multivariate MCs. Finally, the `www.github.com/spedygiorgio/markovchain` GitHub page hosts the package's wiki as well as its development version.

Creating and manipulating markovchain objects

The package is loaded within R as follows:

```
library("markovchain")
```

Creating a `markovchain` object is easy, and can be done with provided code.

```
#using "long" approach for mcWeather

weatherStates <- c("rainy", "nice", "sunny")
weatherMatrix <- matrix(data = c(0.50, 0.25, 0.25,
0.5, 0.0, 0.5, 0.25, 0.25, 0.5), byrow = TRUE,
nrow = 3,dimnames = list(weatherStates, weatherStates))
mcWeather <- new("markovchain", states = weatherStates,
byrow = TRUE, transitionMatrix = weatherMatrix,
name = "Weather")

#using "quick" approach on Mathematica's DTMC

mathematicaMatr <- matrix(c(1/2, 1/2, 0, 0, 1/2, 1/2,
```

```
0, 0, 1/4, 1/4, 1/4, 1/4, 0, 0, 0, 1),byrow=TRUE, nrow=4)
mathematicaMc<-as(mathematicaMatr, "markovchain")

#both are markovchain objects
is(mcWeather,"markovchain")
[1] TRUE
is(mathematicaMc,"markovchain")
[1] TRUE
```

Commenting on the code snippet, the first part shows the "standard" approach to create a `markovchain`, by calling the `new` S4 method, while the second part shows the "quick" method, by coercing a `matrix` object into a `markovchain` one.

Specific methods allow to print and plot `markovchain` objects:

```
plot(mcWeather, main="Weather Markov Chain")
```

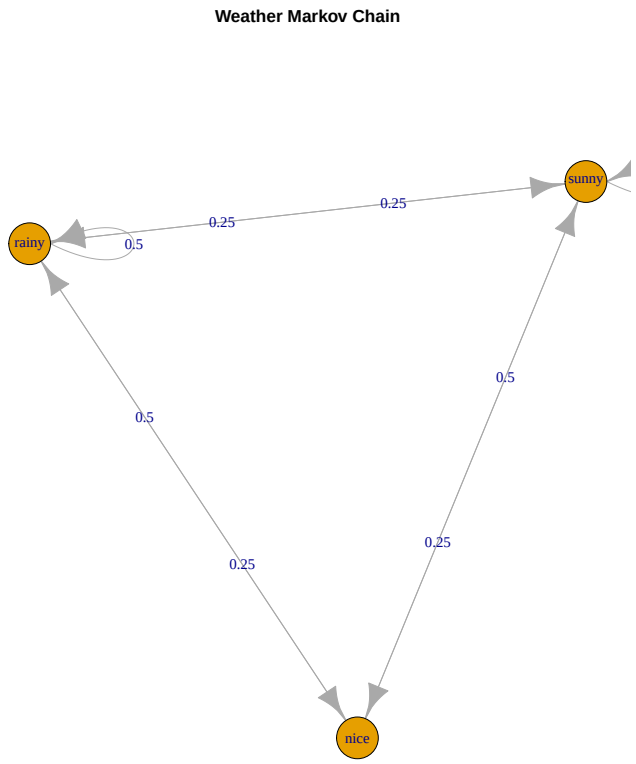

Figure 1: Plotting a `markovchain` object.

In particular, the `plot` method makes use of **igraph** package to draw the TM by default. It is possible to modify the plot either by passing further parameters via `...` or by choosing another plotting devices, as further specified in the package vignette.

Algebraic operations have been defined in `"markovchain"` classes, as of the following example:

```
initialState <- c(0, 1, 0)

#multiplication

after2Days <- initialState * (mcWeather * mcWeather)
after2Days
```

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

in which multiplications by vectors and exponentiation are intuitively performed, making easy to find the distribution of states at the n-th step.

A power operator also exists, ^, and it is based on the **expm** package (Goulet et al., 2015), providing efficient matrix exponentiation.

```
#after two days (by square power)

mcWeather^2

Weather^2
 A  3 - dimensional discrete MC defined by the following states:
 rainy, nice, sunny
 The transition matrix  (by rows)  is defined as follows:
       rainy   nice  sunny
rainy 0.4375 0.1875 0.3750
nice  0.3750 0.2500 0.3750
sunny 0.3750 0.1875 0.437
```

Finally, logical operators have been defined as well.

```
#logical equality and inequality
mcWeather==mcWeather
[1] TRUE
mcWeather!=mathematicaMc
[1] TRUE
```

Both the algebraic and logical operators have been defined by overriding standard R operators, providing a more concise and "natural" code, which can bring the advantage of being more appealing to a novice user, by executing certain operations on TM in an efficient way. Such approach has been stressed in both the class help file and the package vignette code to make the final user fully aware of any potential drawbacks of such choice.

Various convenience S4 methods have been defined to easily manipulate and manage `markovchain` objects. In the following examples, some of the implemented methods in the "`markovchain`" class are presented, allowing to: get and set names, return the MC dimension, transpose the transition matrix, and directly access the transition probabilities.

```
#some markovchain specific methods

#naming
name(mcWeather)
[1] "Weather"

name(mathematicaMc) <- "Mathematica Markov Chain"
#list of defined states
states(mcWeather)
[1] "rainy" "nice"  "sunny"

#the dimension
dim(mcWeather)
[1] 3

#transpose operator
t(mcWeather)

Unnamed Markov chain
A  3 - dimensional discrete Markov Chain defined by the following states:
rainy, nice, sunny
The transition matrix  (by cols)  is defined as follows:
      rainy nice sunny
rainy  0.50  0.5  0.25
nice   0.25  0.0  0.25
```

```
sunny  0.25  0.5  0.50

#two ways to get transition probabilities
transitionProbability(mcWeather, "nice", "sunny")
[1] 0.5
mcWeather[2,3]
 [1] 0.5
```

Finally, coerce methods allow to both import and export `markovchain` classes. Following, a brief example on how to transform a `markovchain` object into a `data.frame` one.

```
#exporting to data.frame and matrix

as(mcWeather, "data.frame")
      t0    t1 prob
 1 rainy rainy 0.50
 2 rainy  nice 0.25
 3 rainy sunny 0.25
 4  nice rainy 0.50
 5  nice  nice 0.00
 6  nice sunny 0.50
 7 sunny rainy 0.25
 8 sunny  nice 0.25
 9 sunny sunny 0.50
```

Structural properties of finite Markov chains

The **markovchain** package embeds functions to analyse the structural proprieties of DTMC. For example, it is possible to find the stationary distribution, as well as classify the states. Feres, Renaldo (2007) and Montgomery, James (2009) provide a full description of the algorithms underlying these functions, whilst a more theoretical perspective can be found in Brémaud, Pierre (1999). The Mathematica MC will be used to illustrate such features.

The `summary` method provides an overview of the structural properties of the DTMC process underlying the `markovchain` object.

```
#plotting and summarizing
plot(mathematicaMc)
```

```
summary(mathematicaMc)
 Mathematica Markov Chain  Markov chain that is composed by:
 Closed classes:
 s1 s2
 s4
 Recurrent classes:
 {s1,s2},{s4}
 Transient classes:
 {s3}
 The Markov chain is not irreducible
 The absorbing states are: s4
```

In the above example, closed and transient classes are identified, irreducibility checks are executed, and a list of absorbing states is returned. Further, it is known that a finite MC has at least one steady-state distribution, and the `steadyStates` method can be used to obtain it. To illustrate, for the `mcWeather` matrix there exist a one - dimensional solution, since the underlying TM is irreducible. A higher dimensional solution is given when the irreducibility property does not hold, as of the second example.

```
#probability with DTMC: stationary distribution
## when the TM is irreducibile
steadyStates(mcWeather)
      rainy nice sunny
```

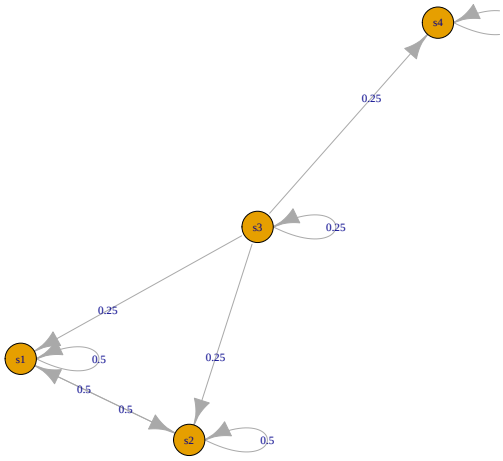

Figure 2: Plot of the Mathematica MC DTMC process.

```
 [1,]   0.4  0.2   0.4
## when reducibility applies
steadyStates(mathematicaMc)
       s1  s2 s3 s4
 [1,] 0.5 0.5  0  0
 [2,] 0.0 0.0  0  1
```

Specific methods and functions return transient and absorbing states, and check whether any state is accessible from another. Recurrent and communicating classes can be easily identified as well.

```
#probability with DTMC: classifying states

transientStates(mathematicaMc)
 [1] "s3"

absorbingStates(mathematicaMc)
 [1] "s4"

is.accessible(mathematicaMc, from = "s1",to="s4")
 [1] FALSE

#identifying recurrent and transient classes

recurrentClasses(mathematicaMc)
 [[1]]
 [1] "s1" "s2"

 [[2]]
 [1] "s4"

communicatingClasses(mathematicaMc)
 [[1]]
 [1] "s1" "s2"

 [[2]]
```

```
[1] "s3"

[[3]]
[1] "s4"
```

The communicating classes are the strongly connected components of the graph underlying the DTMC. It is possible to convert a `markovchain` object into an `igraph` one, in order to use **igraph**'s package clustering function to identify the strongly connected components as the following example displays:

```
library(igraph)
#converting to igraph

mathematica.igraph<-as(mathematicaMc,"igraph")

#finding and formatting the clusters
SCC <- clusters(mathematica.igraph, mode="strong")
V(mathematica.igraph)$color <- rainbow(SCC$no)[SCC$membership]

#plotting
plot(mathematica.igraph, mark.groups = split(1:vcount(mathematica.igraph), SCC$membership),
main="Communicating classes - strongly connected components")
```

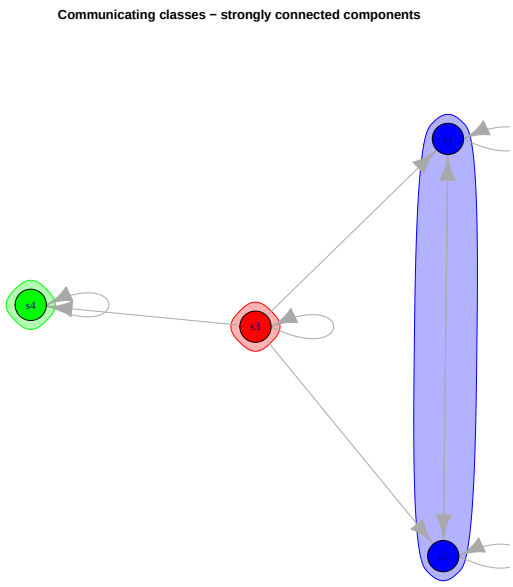

Figure 3: The communicating classes are the strongly connected components of the graph underlying the DTMC.

The three distinct clusters identified with different colors by the **igraph** package match with the partition of the transition matrix into communicating classes given by **markovchain** package's `communicatingClasses` function.

We now illustrate the Canonical Form and the Fundamental Matrix concepts using another example taken from classical theory: The Flipping Coin problem. Specifically, consider repeatedly flipping a fair coin until the sequence (heads, tails, heads) appears; it is possible to model such process using a DTMC with four states: "E" empty initial sequence, "H" head, "HT" head followed by tail, "HTH" head followed by tail and head.

```
# Flipping Coin Problem
```

```
## defining the matrix

flippingMatr <- matrix(0, nrow=4, ncol=4)
flippingMatr[1,1:2] <- 0.5
flippingMatr[2,2:3] <- 0.5
flippingMatr[3,c(1,4)] <- 0.5
flippingMatr[4,4] <- 1
rownames(flippingMatr) <-
colnames(flippingMatr) <- c("E","H","HT","HTH")

## creating the corresponding DTMC
flippingMc <- as(flippingMatr,"markovchain")
```

The following function returns the Q, R, and I matrices by properly combining functions and methods from the **markovchain** package.

```
#function to extract matrices

extractMatrices <- function(mcObj) {

  require(matlab)
  mcObj <- canonicForm(object = mcObj)

  #get the indices of transient and absorbing

  transIdx <- which(states(mcObj) %in% transientStates(mcObj))
  absIdx <- which(states(mcObj) %in% absorbingStates(mcObj))

  #get the Q, R and I matrices

  Q <- as.matrix(mcObj@transitionMatrix[transIdx,transIdx])
  R <- as.matrix(mcObj@transitionMatrix[transIdx,absIdx])
  I <- as.matrix(mcObj@transitionMatrix[absIdx, absIdx])

  #get the fundamental matrix

  N <- solve(eye(size(Q)) - Q)

  #computing final absorbion probabilities

  NR <- N %*% R

  #return
  out <- list(
    canonicalForm = mcObj,
    Q = Q,
    R = R,
    I = I,
    N=N,
    NR=NR
  )
  return(out)
}
```

The expected number of visits to transient state j starting from state i can be found in the corresponding entries of the *fundamental matrix* $N = (I_t - Q)^{-1}$. Therefore, the fundamental matrix for the above DTMC is:

```
#decompose the matrix

flipping.Dec <- extractMatrices(mcObj = flippingMc)
```

```
flipping.Fund <- flipping.Dec$N

#showing the fundamental matrix

flipping.Fund
```

```
   E H HT
E  4 4  2
H  2 4  2
HT 2 2  2
```

```
#expected number of steps before being absorbed

flipping.Fund%*%c(1,1,1)
```

```
   [,1]
E    10
H     8
HT    6
```

```
#calculating B matrix
#the probability to being absorbed in HTH state as a function of the starting transient state

flipping.B <- flipping.Fund%*%flipping.Dec$R
flipping.B
```

```
   [,1]
E     1
H     1
HT    1
```

```
#calculating H, probability of visiting transient state j starting in transient state i

flipping.H <- (flipping.Fund - matlab::eye(ncol(flipping.Fund))) *  solve(diag(diag(flipping.Fund)))
flipping.H
```

```
       E    H  HT
E  0.75 0.00 0.0
H  0.00 0.75 0.0
HT 0.00 0.00 0.5
```

The calculated fundamental matrix shows that the number of times the chain is in state HT, starting from state H is two. Also, the $N * \bar{1}$ vector indicates that if the chains starts in HT, the expected number of steps before being absorbed is eight. Since there is only one absorbing state, HTH, the probability to be absorbed in HTH is one, whichever the starting transient state is. Also, matrix H shows that the probability that a chain in state H will eventually visit again state H is 0.75.

It is possible to compute the distribution of first passage time, as the code that follows shows:

```
#first passage time

fptMc <- new("markovchain", transitionMatrix=matrix(c(0, 1/2, 1/2,1/2,0, 1/2,
1/2, 1/2, 0), byrow = TRUE,ncol=3), name="FistPassageTimeExample", states=c("a"  ,"b","c"))

firstPassage(fptMc,state = "a",5)
```

```
        a       b       c
 1 0.0000 0.50000 0.50000
 2 0.5000 0.25000 0.25000
 3 0.2500 0.12500 0.12500
 4 0.1250 0.06250 0.06250
 5 0.0625 0.03125 0.03125
```

The output of `firstPassage` function shows that the probability that the first hit of state "b" occurs at the second step is 0.25.

Periodicity analysis is shown in the following last example, in which the output shows that the DTMC has a period of 2.

```
#defining a toy - model matrix for periodicity

periodicMc<-as(matrix(c(0,1,1,0),nrow=2),"markovchain")
periodicMc
```

```
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
    s1 s2
 s1  0  1
 s2  1  0
```

```
#computing periodicity

period(periodicMc)
```

```
 [1] 2
```

Statistical inference using markovchain package

Statistical analysis functions allow to estimate a DTMC from data and to simulate a DTMC, and can be done through the `rmarkovchain` function:

```
weathersOfDays <- rmarkovchain(n = 30, object = mcWeather, t0 = "sunny")
weathersOfDays
```

```
  [1] "sunny" "sunny" "rainy" "rainy" "rainy" "nice"  "rainy" "rainy"
  [9] "rainy" "rainy" "nice"  "rainy" "rainy" "nice"  "sunny" "nice"
 [17] "rainy" "rainy" "sunny" "rainy" "rainy" "rainy" "sunny" "rainy"
 [25] "sunny" "sunny" "sunny" "sunny" "sunny" "rainy"
```

The code shown above simulates 30 observations from the weather DTMC previously introduced.

Next, the function `createSequenceMatrix` is used to obtain the *sequence matrix*, that is the empirical transition matrix between the preceding and subsequent state, for a given sequence, whilst the function `markovchainFit` fits DTMCs. We will exemplify the use of such functions on the `rain` data set (recorded daily rainfall volume in Alofi island) bundled within the package.

```
#loading the Alofi's rain data set

data(rain)
rain$rain[1:10]
```

```
  [1] "6+"  "1-5" "1-5" "1-5" "1-5" "1-5" "1-5" "6+"  "6+"  "6+"


#obtaining the empirical transition matrix

createSequenceMatrix(stringchar = rain$rain)


      0 1-5  6+
0   362 126  60
1-5 136  90  68
6+   50  79 124



#fitting the DTMC by MLE

alofiMcFitMle <- markovchainFit(data = rain$rain, method = "mle", name = "Alofi")
alofiMcFitMle


$estimate
Alofi
 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


$standardError
             0        1-5         6+
0   0.03471952 0.02048353 0.01413498
1-5 0.03966634 0.03226814 0.02804834
6+  0.02794888 0.03513120 0.04401395

$confidenceInterval
$confidenceInterval$confidenceLevel
[1] 0.95

$confidenceInterval$lowerEndpointMatrix
            0       1-5         6+
0   0.6034754 0.1962346 0.08623909
1-5 0.3973397 0.2530461 0.18515711
6+  0.1516566 0.2544673 0.41772208

$confidenceInterval$upperEndpointMatrix
            0       1-5        6+
0   0.7176925 0.2636194 0.1327390
1-5 0.5278304 0.3591988 0.2774279
6+  0.2436003 0.3700387 0.5625151


$logLikelihood
[1] -1040.419
```

Clearly, the `markovchainFit` function returns not only the pointwise estimate of the transition matrix, but also its standard error and confidence intervals. MLE estimates are provided by default, but a bootstrap one Efron, B. (1979) can also be obtained as the following code shows.

```
#estimating Alofi TM

alofiMcFitBoot <- markovchainFit(data = rain$rain, method = "bootstrap",
name = "Alofi",nboot=100)



#point estimate of the TM

alofiMcFitBoot$estimate


 Alofi
  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   0.6605457 0.2314278 0.1080264
 1-5 0.4646651 0.3071925 0.2281424
 6+  0.1976978 0.3115299 0.4907723



#95 CIs

alofiMcFitBoot$standardError


               0         1-5          6+
 0   0.001957644 0.001793261 0.001318923
 1-5 0.002733252 0.002712275 0.002273845
 6+  0.002647255 0.002949244 0.003075143
```

Subsequently, the three-days forward predictions from `alofiMcFitMle` object are generated, assuming that the last two days were "1-5" and "6+" respectively. Clearly only the last state matters for a MC stochastic process.

```
#obtain a prediction

predict(object = alofiMcFitMle$estimate, newdata = c("1-5", "6+"),n.ahead = 3)


 [1] "6+" "6+" "6+"


#obtain a prediction changing t-2 state

predict(object = alofiMcFitMle$estimate, newdata = c("0", "6+"),n.ahead = 3)


 [1] "6+" "6+" "6+"
```

Non homogeneous Markov chains

Non homogeneous DTMCs (NHDTMCs) can be handled using the `"markovchainList"` S4 class, which consists in a list of `markovchain` objects.

```
#define three DTMC
```

```
matr1<-matrix(c(0.2,.8,.4,.6),byrow=TRUE,ncol=2);mc1<-as(matr1, "markovchain")
matr2<-matrix(c(0.1,.9,.2,.8),byrow=TRUE,ncol=2);mc2<-as(matr2, "markovchain")
matr3<-matrix(c(0.5,.5,.2,.8),byrow=TRUE,ncol=2);mc3<-as(matr2, "markovchain")

#create the markovchainList to store NHDTMCs

mcList<-new("markovchainList", markovchains=list(mc1,mc2,mc3), name="My McList")
mcList
```

```
 My McList  list of Markov chain(s)
 Markovchain  1
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.2 0.8
 s2 0.4 0.6

 Markovchain  2
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.1 0.9
 s2 0.2 0.8

 Markovchain  3
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.1 0.9
 s2 0.2 0.8
```

The example above shows that creating a `markovchainList` S4 object is very simple. Moreover, the `rmarkovchain` function also works on objects from the `"markovchainList"` class.

```
#simulating a NHDTMC

mysim<-rmarkovchain(n=100, object=mcList,include.t0=TRUE,what="matrix")
head(mysim,n = 10)
```

```
       [,1] [,2] [,3] [,4]
  [1,] "s2" "s2" "s2" "s2"
  [2,] "s2" "s1" "s2" "s2"
  [3,] "s2" "s1" "s2" "s2"
  [4,] "s2" "s1" "s2" "s2"
  [5,] "s2" "s2" "s1" "s2"
  [6,] "s1" "s2" "s2" "s1"
  [7,] "s1" "s2" "s2" "s2"
  [8,] "s1" "s2" "s2" "s2"
  [9,] "s2" "s2" "s2" "s1"
 [10,] "s1" "s1" "s2" "s2"
```

Finally, it is possible to infer a non - homogeneous sequence of DTMC, that is a `markovchainList` object from a given matrix, where each row represents a single trajectory and each column stands for a different period.

```
#using holson data set

data(holson)
head(holson,n = 3)
```

```
   id time1 time2 time3 time4 time5 time6 time7 time8 time9 time10 time11
1  1     1     1     1     1     1     1     1     1     1      1      1
2  2     1     1     1     1     1     1     1     1     1      1      1
3  3     1     1     1     1     1     1     1     1     1      1      1
```

```
#fitting a NHDTMCs on holson data set

nhmcFit<-markovchainListFit(holson[,2:12])

#showing estimated DTMC for time 1 -> time 2 transitions

nhmcFit$estimate[[1]]
```

```
 time1
  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
```

```
#showing estimated DTMC for time 2 -> time 3 transitions

nhmcFit$estimate[[2]]
```

```
 time2
  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
```

Numerical reliability and computational performance

Numerical reliability

Finding the stationary distribution is a computational - intensive task that could raise numerical issues. The **markovchain** package relies on the R linear algebra facilities (built on LAPACK routines) when the `eigen` function is called to find the stationary distribution. An initial analysis of the numerical stability of the `markovchain` matrix computation has been performed estimating the error rate when calculating the stationary distribution on a large sample of simulated DTMC of a given size k (range set between 2 and 32). Initially, dense matrices were simulated. The following algorithm was used for a given k:

1. generate N random k-sized DTMCs, where each row $\bar{r}$ has been independently sampled from a Dirichlet distribution, $\bar{r} \sim Dir(\bar{\alpha})$. The Dirichlet parameters' vector, $\bar{\alpha}$ is itself assumed to follow an Uniform distribution (sampled independently for each row).
2. try to compute the steady - state distribution for the simulated DTMC.

3. calculate the success rate as the relative frequency of previous step non - failures at size k .

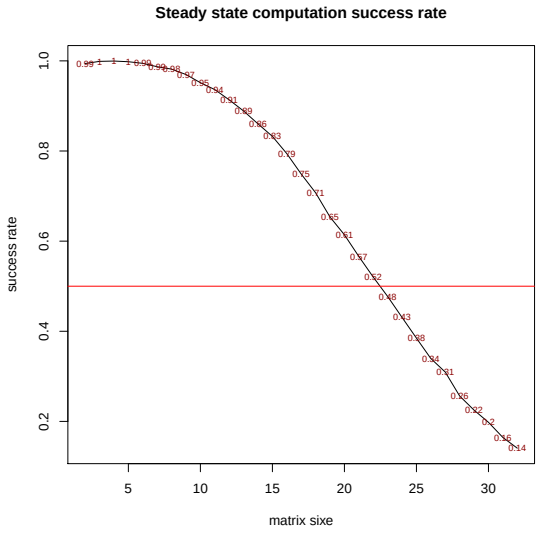

Figure 4: Steady state computation success rate.

The figure shown above displays the success rate observed by TM size. The success rate is higher than 95% for matrices no greater than 10 unit, then it decreases markedly and becomes lower than 50% for matrices bigger then 22. A deeper analysis allowed to identify that the failure reason was due to inaccuracy in the Dirichlet sampling function (row sums numerically different from zero). The TM simulation process was therefore revised normalizing the sum of each row to be numerically equal to one. The experiment was repeated at $2^3, 2^4, \ldots, 2^8$ TM sizes (wider matrices were not tested due to computational timing issues). The observed success rate was always 100% for the sampled TM sizes.

The first example deserves few more words, even if it does not demonstrate any shortcomings in the computational part of the package. Instead, it shows how easy it is to analyze numerically - incorrect TMs as the size of the problems dealt with increases. Various posts have been raised on this topic on the package Github address since the package was published on CRAN.

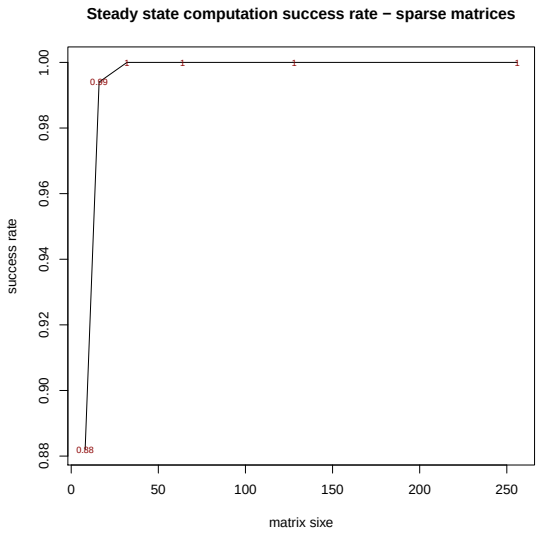

Figure 5: Steady state computation success rate, sparse matrices.

A final test has been performed using TMs with a sparsity factor of 75%. The observed success rate is 100% for matrices wider than 2^5, inexplicably lower (around 90%) for smaller matrices matrices.

The previous examples are clearly far to exhaustively assess the numerical reliability of the implemented algorithms that would require an much deeper analysis and beyond the scope of

the paper. In fact, the numerical reliability is likely to be significantly affected by particular TM structures. Nevertheless they can provide an initial insight about the dimension of the problems that the **markovchain** R package can "safely" handle. The R code used to generate the numerical reliability assessment herewith discussed is available in the "reliability.R" file within the demo folder of `markovchain` R package.

Computational performance

The computation time needed to estimate the TM from input data sequence depends by the size of input data, as the following example displays:

```
#using the rain data sequence
data(rain)
rainSequence<-rain$rain

#choosing different sample size
sizes<-c(10,50,100,250,500,1096)

#timing assessiment
microseconds<-numeric(length(sizes))

for(i in 1:length(sizes)) {

  mydim<-sizes[i]
  mysequence<-rainSequence[1:mydim]
  out<-microbenchmark(
    myFit<-markovchainFit(data=mysequence)
  )
  microseconds[i]<-mean(out$time)

}

plot(sizes, microseconds,type="o",col="steelblue",
xlab="character sequence size",ylab="microseconds",
main="Computational time vs size")
```

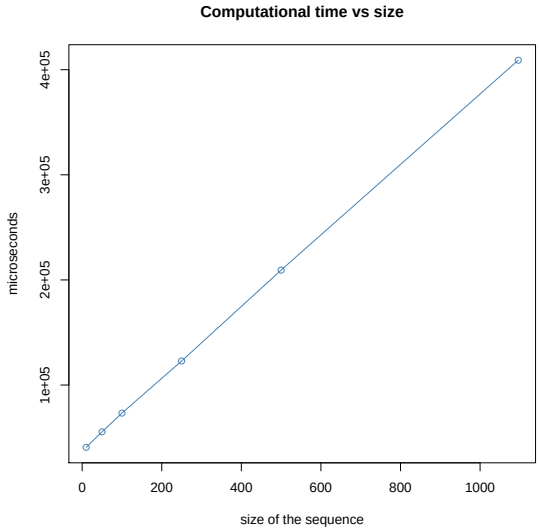

Figure 6: Computation time by size of input data sequence.

The plot shows that the computation time increases linearly with the size of input data sequence, as expected.

The last numeric example presented in the section discussing NHDTMCs shows the computational advantages of rewriting the kernel of core functions using **Rcpp** and **RcppParallel** snippets generated by (Allaire et al., 2016). The `rmarkovchain` function allows the final user to choose whether to use the C++ implementation and a parallel backend, by setting the boolean parameters `useRcpp` and `parallel` respectively.

```
microbenchmark(
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = F),
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = T,parallel = F),
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = T,parallel = T)
)
```

The omitted output of the code snippet shown above demonstrates that the joint use of **Rcpp** and **RcppParallel** fastens the simulations around 10x with respect to the pure R sequential implementation.

Conclusions, discussion and acknowledgements

The **markovchain** package has been designed in order to make common operations on DTMCs as easy as possible for statisticians. The package allows to create, manipulate, import and export DTMCs. Further, the author believes that the current version of the package fully satisfies standard needs such as inference of underlying TM from empirical data, and states classification of a given DTMC.

The author believes that no other R package provides a set of classes, methods, and functions as wide as the one provided in **markovchain**, as of May 2017.

The package's main vignette gives a complete descriptions of its capabilities, including bayesian estimation, statistical tests, classes and methods for continuous time MCs. Also, a separate vignette describes the functions designed to deal with higher order and multivariate MCs, and should still be considered experimental. In fact, such techniques are generally less used than standard DTMCs, and consequently much less literature, applied examples, and coded algorithms are available.

Clearly, an expanded version of the package's capabilities in that area is expected to be released in the future. Current development efforts target optimizing computation speed and reliability, and increasing the analysis capabilities regarding statistical inference. Rewriting core functions using **Rcpp** gave a major boost in terms of computing speed, as exemplified in previous sections. Moreover, the rewriting of the internal core parts of the code affected many functions, such as `markovchainFit` and `markovchainFitList`. Feedbacks provided by the users of the package at https://github.com/spedygiorgio/markovchain/issues have been extremely useful for improving the package. To illustrate, bugs due to numerical issues have been found when analyzing relatively big MCs and have led to revising the `steadyStates` function to be computationally more robust. A known limitation of the package is the lack of a deep assessment of the performance of the package's routines for a relatively large TM. In fact, improving the numerical reliability of the package for large DTMCs is an area on which efforts will be certainly allocated in the near future. At this regard, the implementation of numerical methods methods shown in Stewart (1994) will be explored.

Finally, the package has been available on CRAN since Summer 2013. Notably, it has been granted a funding slot in both 2015, 2016 and 2017 Google Summer of Code (GSOC) editions. In particular, during 2015 GSOC a material part of R code has been ported in **Rcpp** coding, yielding considerable fastening in computational time. The author is extremely grateful to Tae Seung Kang, Sai Bhargav Yalamanchi and Deepak Yadav for their contribution in improving the package. A special thank should be given to the RJournal referees for their constructive comments.

Giorgio Alfredo Spedicato
UnipolSai Assicurazioni
Piazza della Costituzione 2
Bologna 40128, Italy
spedicato_giorgio@yahoo.it

Bibliography

J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for Rcpp*, 2016. URL https://CRAN.R-project.org/package=RcppParallel. R package version 4.3.20. [p102]

Antoine Filipovi, C., Pierucci, Kevin, Zarca, and Isabelle Durand-Zaleski. Markov models for health economic evaluation: The r package heemod. *ArXiv e-prints*, 2017. URL https://pierucci.org/heemod. R package version 0.9.0. [p85]

Artur Araújo, Luís Meira-Machado, and Javier Roca-Pardiñas. **TPmsm**: Estimation of the transition probabilities in 3-state models. *Journal of Statistical Software*, 62(4):1–29, 2014. URL http://www.jstatsoft.org/v62/i04/. [p85]

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2016. URL http://CRAN.R-project.org/package=Matrix. R package version 1.2-6. [p87]

Bessi, Alessandro. *MCTM: Markov Chains Transition Matrices*, 2015. URL http://CRAN.R-project.org/package=MCTM. R package version 1.0. [p84]

Brémaud, Pierre. *Discrete-Time Markov Models*. Springer-Verlag, 1999. [p90]

C. G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. CRC, 1993. [p85]

Chambers, J.M. *Software for Data Analysis: Programming with R*. Statistics and computing. Springer-Verlag, 2008. ISBN 9780387759357. [p87]

W.-K. Ching, M. K. Ng, and E. S. Fung. Higher-order multivariate markov chains and their applications. *Linear Algebra and its Applications*, 428(2):492–507, 2008. [p84]

B. A. Craig and P. P. Sendi. Estimation of the transition matrix of a discrete-time markov chain. *Health Economics*, 11(1), 2002. [p84]

Csardi, Gabor and Nepusz, Tamas. The **igraph** software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net. [p87]

Eddelbuettel, Dirk. *Seamless R and C++ Integration with* **Rcpp**. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p87]

Efron, B. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 1979. URL http://dx.doi.org/10.1214/aos/1176344552. [p96]

Feres, Renaldo. Notes for math 450 matlab listings for markov chains, 2007. URL http://www.math.wustl.edu/~feres/Math450Lect04.pdf. [p90]

V. Goulet, C. Dutang, M. Maechler, D. Firth, M. Shapira, M. Stadelmann, and expm-developers@lists.R-forge.R-project.org. *Expm: Matrix Exponential*, 2015. URL http://CRAN.R-project.org/package=expm. R package version 0.999-0. [p87, 89]

C. M. Grinstead and J. L. Snell. *Introduction to Probability*. American Mathematical Soc., 2012. [p85]

L. Himmelmann. *HMM: HMM - Hidden Markov Models*, 2010. URL https://CRAN.R-project.org/package=HMM. R package version 1.0. [p85]

C. H. Jackson. Multi-state models for panel data: The **msm** package for r. *Journal of Statistical Software*, 38(8):1–29, 2011. URL http://www.jstatsoft.org/v38/i08/. [p85]

J. Jarvis and D. R. Shier. Graph-theoretic analysis of finite markov chains. *Applied mathematical modeling: a multidisciplinary approach*, 1999. [p85]

C. I. Jones. On the evolution of the world income distribution. *Available at SSRN 59412*, 1997. [p84]

J. G. Kemeny, J. L. Snell, and G. L. Thompson. Finite mathematics. *DC Murdoch, Linear Algebra for Undergraduates*, 1974. [p85]

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999. URL http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120. [p84]

A. A. Markov. Issledovanie zamechatel nogo sluchaya zavisimyh ispytanij. *Izvestiya Akademii Nauk, SPb, VI seriya*, 1(93):61–80, 1907. [p84]

MATLAB. *Version 9.2.0 (R2017a)*. The MathWorks Inc., Natick, Massachusetts, 2017. [p85]

Montgomery, James. Communication classes, 2009. URL http://www.ssc.wisc.edu/~jmontgom/commclasses.pdf. [p90]

Nicholson, William. **DTMCPack***: Suite of Functions Related to Discrete-Time Discrete-State Markov Chains*, 2013. URL http://CRAN.R-project.org/package=DTMCPack. R package version 0.1-2. [p84]

Pablo J. Villacorta and José L. Verdegay. **FuzzyStatProb**: An r package for the estimation of fuzzy stationary probabilities from a sequence of observations of an unknown Markov chain. *Journal of Statistical Software*, 71(8):1–27, 2016. URL https://doi.org/10.18637/jss.v071.i08. [p85]

P. Roebuck. *Matlab: MATLAB Emulation Package*, 2014. URL http://CRAN.R-project.org/package=matlab. R package version 1.0.2. [p87]

M. Scholz. The R package **clickstream**: Analyzing clickstream data with markov chains. *Journal of Statistical Software*, 74(4):1–17, 2016. URL https://doi.org/10.18637/jss.v074.i04. [p84]

Spedicato, Giorgio Alfredo. **markovchain***: An R Package to Easily Handle Discrete Markov Chains*, 2016. R package version 0.6.5. [p84]

W. J. Stewart. *Introduction to the Numerical Solutions of Markov Chains*. Princeton Univ. Press, 1994. [p102]

I. Visser and M. Speekenbrink. **depmixS4**: An r package for hidden markov models. *Journal of Statistical Software*, 36(7):1–21, 2010. URL http://www.jstatsoft.org/v36/i07/. [p85]

Wolfram Research, Inc. *Mathematica*. Wolfram Research, Inc., ninth edition, 2013. [p85]