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

Giorgio Alfredo Spedicato, Ph.D C.Stat ACAS Mirko Signorelli, M.D. Statistics and economics

Abstract

markovchain aims to fill a gap within R packages providing S4 classes and methods to easily handling discrete markov chains. The S4 class structure will be presented as well implemented classes and methods. Applied examples will follow

Keywords: markov chain, transition probabilities.

1. Introduction

Markov chains represent a class of stochastic processes of great interest for the wide spectrum of practical applications. In particular, discrete markov chains permit to model the transition probabilities between possible discrete states by the aid of matrices. Various R packages deals with Markov chains processes and their applications: **msm** (**?**) works with Multi-State Models for Panel Data, **mcmcR** (**?**) is only one of the many package that implements Monte Carlo Markov Chain approach for estimating models' parameters, **hmm** fits hidden markov models taking into account covariates. R statistical environments seems to lack a simple R package that coherently defines S4 classes for discrete Markov chains and that allows the statistical analyst to perform probabilistic analysis and statistical infrence. **markovchain** (**?**) aims to offer greater flexibility in handling discrete time Markov chains. The paper is structured as it follows: Section 2 briefly revies mathematic and definitions on discrete Markov chains, Section 4 shows applied example of discrete Markov chains in various fields.

2. Markov chains mathematic revies

Definitions

A discrete-time Markow chain is a sequence of random variables $X_1, X_2, X_3, ...$ with the property of memorylessness (or Markov property), so that the next state of X_{n+1} depends on the current state of X_n only and doesn't depend from the events that preceded it:

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

The set $S = \{s_1, s_2, ..., s_r\}$ of possible states of X_j is called state space of the chain. In discrete-time Markov chain, S is finite or countable.

A Markow chain is stationary (or time-homogeneous) if $Pr\left(X_{n+1} = x \left| X_n = y \right.\right) = Pr\left(X_n = x \left| X_{n-1} = y \right.\right)$, in other words, if the underlying transition probabilities do not change as time moves on.

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

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

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

If the Markov chain is stationary $p_{ij} = Pr\left(X_{k+1} = s_j \left| X_k = s_i \right.\right)$ and $p_{ij}^{(n)} = Pr\left(X_{n+k} = s_j \left| X_k = s_i \right.\right)$, where $k > 0$.

The probability distributions of transitions from one state to another can be represented into a transition matrix P, in which the element of position (i, j) is the probability p_{ij}; for instance, if $r = 3$ the transition matrix P is

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

The distribution over the states can be written as a stocastic row vector x: if the current state of x is s_2, $x = (0\,1\,0)$. As a consequence, the relation between $x^{(n+1)}$ and $x^{(n)}$ is $x^{(n+1)} = x^{(n)}P$ and, recursively, $x^{(n+2)} = x^{(n)}P^2$, $x^{(n+k)} = x^{(n)}P^k$, $k > 0$.

Example

Consider the following numerical example. Suppose we have a Markov chain with a set of 3 possible states s_1, s_2 and s_3. Let the transition matrix be

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

$p_{11} = 0.5$ means that the probability that $X_{n+1} = s_1$ given that we observed $X_n = s_1$ is 0.5, and so on. If in the current state we have $X_n = s_2$, then

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

$$x^{(n+2)} = x^{(n+1)}P = (0.15\,0.45\,0.4)\left[\begin{array}{ccc} 0.5 & 0.2 & 0.3 \\ 0.15 & 0.45 & 0.4 \\ 0.25 & 0.35 & 0.4 \end{array}\right] = (0.2425\,0.3725\,0.385)$$

and so on. The last result means that $Pr\left(X_{n+2} = s_1 \left| X_n = s_2 \right.\right) = 0.2425$, $Pr\left(X_{n+2} = s_2 \left| X_n = s_2 \right.\right) = 0.3725$ and $Pr\left(X_{n+2} = s_3 \left| X_n = s_2 \right.\right) = 0.385$.

Properties

A state s_j is said to be accessible from a state s_i (written $s_i \rightarrow s_j$) if a system started in state s_i has a positive probability of transitioning into state s_j at a certain point. If both $s_i \rightarrow s_j$ and $s_j \rightarrow s_i$ the states s_i and s_j are said to communicate.

A state s_i has a period k if any return to state s_i must occur in multiples of k steps, that is $k = gcd\left\{ n : Pr\left(X_n = s_i \left| X_0 = s_i \right. \right) > 0 \right\}$,where 'gcd' is the greatest common divisor. If $k = 1$ the state is said to be aperiodic, if $k > 1$ the state is periodic with period k.

A state s_i is said to be transient if, given that we start in state s_i, there is a positive probability that we will never return to s_i; otherwise, s_i is recurrent (or persistent or absorbing). A Markov chain is absorbing if there is at least one recurrent state; otherwise, the chain is said to be ergodic (or irreducible) if it is possible to get to any state from any state.

A Markov chain is said to be regular if some power of the transition matrix has positive elements only; note that regular chains form a subset of ergodic chains.

An interesting property of regular Markov chains is that, if P is the $k \times k$ transition matrix and $z = (z_1, ..., z_k)$ is the eigenvector of P having $\sum_{i=1}^{k} z_i = 1$,

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

where Z is the matrix having all rows equal to z.

3. The structure of the package

3.1. Creating markovchain objects

The package **markovchain** contains classes and methods that handle markov chain in a convenient manner.

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

```
R> #library("markovchain") #quando viene pubblicato
R> #per ora fare il source
R> #workDirGiorgio='D:/Universita/Ricerca/markovchain/'
R> workDirGiorgio2='F:\\giorgio lavoro\\universita\\markovChain'
R> setwd(workDirGiorgio2)
R> #workDirMirko='C:/Users/Mirko/Desktop/markovchain/'
R> #workDirGiorgioDropBox='D:\\Dropbox\\Dropbox\\markovchain'
R> #setwd(workDirMirko)
R>
R> library(expm)
R> library(igraph)
R> library(matlab)
R> source('./R Code/classesAndMethods.R')
R> source('./R Code/functions4Fitting.R')
```

The `markovchain` and `markovchainList` S4 classes (**?**)chambers) is defined within the **markovchain** package as displayed:

```
Class "markovchain" [in ".GlobalEnv"]
```

```
Slots:

Name:            states           byrow transitionMatrix
Class:        character         logical           matrix

Name:              name
Class:        character

Class "markovchainList" [in ".GlobalEnv"]

Slots:

Name:  markovchains         name
Class:         list    character
```

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 transition matrix.
4. name: optional character element to name the Markov chain

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=mat
+                          0.3,0.4, 0.3,
+                          0.2,0.45,0.35),byrow=byRow, nrow=3), name="Weather")
R>
```

When new("markovchain") is called alone a defaut Markov chain is created.

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

The quicker form of object creation is made possible thanks to the implemented `initialize` S4 method that assures:

- 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, according to the value of `byrow` slot.
- the columns and rows nams of `transitionMatrix` to be defined and to coincide with `states` vector slot.

`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
```

3.2. Handling markovchain objects

markovchain contains two classes, `markovchain` and `markovchainList`. `markovchain` objects handle discrete Markov chains, whilst `markovchainList` objects consists in list of `markovchain` that can be useful to model non - homogeneous Markov chain processess.

Following methods have been implemented within the package for `markovchain` and `markovchainLists` respectively:

```
Function: * (package base)
e1="markovchain", e2="markovchain"
e1="markovchain", e2="matrix"
e1="markovchain", e2="numeric"
e1="matrix", e2="markovchain"
e1="numeric", e2="markovchain"

Function: [ (package base)
x="markovchain", i="ANY", j="ANY", drop="ANY"

Function: ^ (package base)
e1="markovchain", e2="numeric"

Function: == (package base)
e1="markovchain", e2="markovchain"

Function: absorbingStates (package .GlobalEnv)
object="markovchain"

Function: coerce (package methods)
from="data.frame", to="markovchain"
from="markovchain", to="data.frame"
```

```
Function: dim (package base)
x="markovchain"

Function: initialize (package methods)
.Object="markovchain"

Function: length (package base)

Function: plotMc (package .GlobalEnv)
object="markovchain"

Function: print (package base)
x="markovchain"

Function: show (package methods)
object="markovchain"

Function: states (package .GlobalEnv)
object="markovchain"

Function: steadyStates (package .GlobalEnv)
object="markovchain"

Function: t (package base)
x="markovchain"

Function: transitionProbability (package .GlobalEnv)
object="markovchain"


Function: [[ (package base)
x="markovchainList"

Function: dim (package base)
x="markovchainList"

Function: initialize (package methods)
.Object="markovchainList"
    (inherited from: .Object="ANY")

Function: length (package base)

Function: print (package base)
x="markovchainList"

Function: show (package methods)
object="markovchainList"
```

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

Method	Purpose
`*`	Algebraic operators on the transition matrix.
`[`	Direct access to transition matrix elements.
`==`	Equality operator on the transition matrix.
`dim`	Dimenion of the transition matrix.
`states`	Defined transition states.
`t`	Transposition operator (it switches byrow slot value and modifies the transition matrix coherent
`as`	Operator con switch from `markovchain` objects to `data.frame` objects and vice - versa.

Table 1: **markovchain** methods: matrix handling.

Operations on the markovchains objects can be easily performed. Using the previously defined matrix we can find what is the probability distribution of expected weather states two and seven days after, given actual state to be cloudy.

```
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> after7Days

         sunny    cloudy      rain
[1,] 0.4622776 0.3188612 0.2188612
```

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

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

        [,1]
sunny  0.390
cloudy 0.355
rain   0.255

R> after7Days
```

```
            [,1]
sunny  0.3172005
cloudy 0.3188612
rain   0.3192764
```

Basing informational methods have been defined for `markovchain` objects to quickly get states and dimension.

```
R> states(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
```

A transition matrix can be displayed using `print`, `show` methods (the latter being less laconic). Similarly, the underlying transition probability diagram can be plot by the use of `plotMc` method that was based on **igraph** package (?) as Figure 1 displays.

```
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

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
```

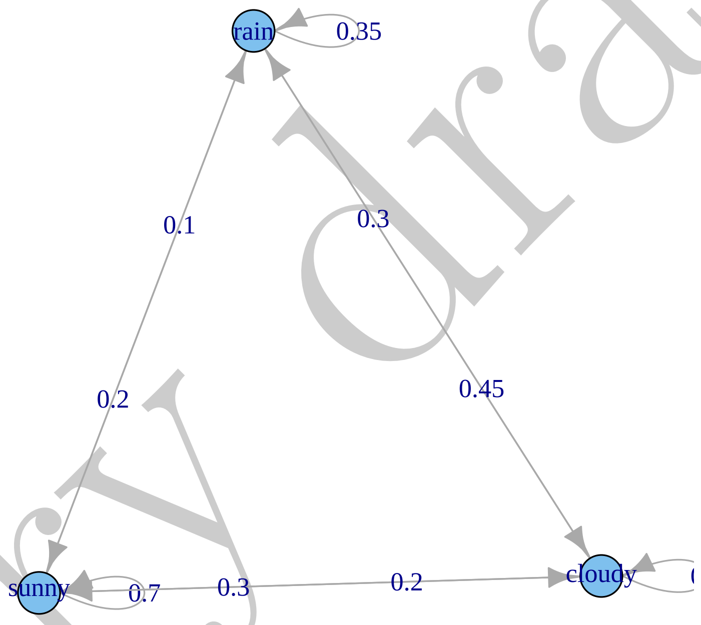

Figure 1: Weather example Markov chain plot

The **igraph** package (**?**) is used for plotting. `...` additional parameters are passed to `graph.adjacency` function to control the graph layout.

Exporting to `data.frame` is possible and similarly it is possible to import.

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

Similarly it is possible to export a `markovchain` class toward an adjacency matrix.

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 can change by year.

It is possible to perform direct access to `markovchainList` elements as well as determining the number of underlying `markovchain` objects contained therin in advance.

```
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
```

3.3. Statistics with markovchain objects

Probabilistic analysis

Table 2 shows methods appliable on `markovchain` objects to perform probabilistic analysis.

Method	Purpose
`absorbingStates`	it returns the absorbing states of the transition matrix, if any.
`steadyStates`	it returns the vector(s) of steady state(s) in matricial form.

Table 2: **markovchain** methods: statistical operations.

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

1. decompose the Markov Chain in eigenvalues and eigenvectors.
2. consider only eigenvectors corresponding to eigenvalues equal to one.
3. normalize such eigenvalues so the sum of their components to total one.

The result is returned in matricial form.

```
R> steadyStates(mcWeather)

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

It is possible a Markov chain to have more than one stationary distribuition, 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
```

Any absorbing state is determined by the inspection of results returned by `steadyStates` method.

```
R> absorbingStates(mcGR4)

[1] "0" "4"

R> absorbingStates(mcWeather)
```

```
character(0)
```

Statistical analysis

Table 3 lists functions (and their purpose) as implemented within the package that helps to fit and simulate discrete time Markov chains.

Function	Purpose
`markovchainFit`	function to return fitten markov chain for a given sequence.
`rmarkovchain`	function to sample from `markovchain` or `markovchainList` objects.

Table 3: **markovchain** statistical functions.

Simulating a random sequence from an underlying Markov chain is quite easy thanks to the function `rmarkovchain`. The following code generates a "year" of weather states according to **?** underlying markovian stochastic process.

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

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

Similarly, it is possible to simulate one o more sequence from a non-homogeneous markov chain, as the following code (applied on CCHC example) displays.

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

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

Similarly, a `markovchain` object can be fit from given data. The most straightforward approach is maximum likelihood.

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

MLE Fit
 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 0.4396552 0.27586207 0.2844828
rain   0.4050633 0.41772152 0.1772152
sunny  0.2011834 0.08284024 0.7159763
```

Nevertheless a bootstrap version of maximum likehihood has been developed in order to assess the variability of estimate.

```
R> weatherFittedBOOT<-markovchainFit(data=weathersOfDays, method="bootstrap",nboot=50)
R> weatherFittedBOOT$estimate

BootStrap Estimate
 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 0.4467616 0.26769280 0.2855456
2 0.4119685 0.40353663 0.1844949
3 0.2019854 0.08065979 0.7173548

R> weatherFittedBOOT$standardError

           [,1]       [,2]       [,3]
[1,] 0.04194215 0.04194890 0.04514097
[2,] 0.05777718 0.06246215 0.04566470
[3,] 0.03060798 0.02189225 0.03191420
```

4. Applied examples

4.1. Actuarial examples

Markov chains are widely applied in the fields of actuarial science. 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 reseves 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 differenbe between $PVFB$ and $PVFP$. The first example shows the pricing and reserving of a (simple) health insurance contract. The second example analyze the evolution of a MTPL portfolio characterized by Bonus Malus experience rating feature.

Health insurance example

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 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,.2,
+                                      0,0,1,0,
+                                      0,0,0,1), byrow=TRUE, nrow=4))
R> benefitVector=as.matrix(c(0,0,500,1000))
R>
```

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

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

Therefore the present value of future benefit at T0 is

```
R> PVFB=T0%*%benefitVector*1.05^-0+T1%*%benefitVector*1.05^-1+T2%*%benefitVector*1.05^-2+T
```

and the yearly premium payable whether the insured is alive is

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

The reserve at the beginning of year two, in case of the insured being alive, is

```
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
```

5. Aknowledgments

References

Affiliation:
Mirko Signorelli
signorellimirko@hotmail.it