Manual for Package MARCH

André Berchtold and Oliver Buschor

May 26, 2016

1 Introduction

This manual should explain how the MARCH package can be used. Therefore it is a precondition that you have a compiler which can deal with R code, R-Studio is recommended.
Together with manual_usage.R this file will get you a basic introduction to MARCH. It is a program to compute Double Chain Markov Models, including special cases like the Hidden Markov Models and homogeneous Markov chains. The different models and their application will be discussed later.
To install the package MARCH you can type the following Code into the console of R-Studio.

```
install.packages("march", repos="http://R-Forge.R-project.org")
```

With the following command the namespace of MARCH loaded and attached to the search list.

```
library(march)
```

Further the package TraMineR, is provided and available on CRAN. With the following code it can be downloaded and installed in R-Studio.

```
install.packages("TraMineR")
```

To attach TraMineR to the search list the following command is necessary.

```
library(TraMineR)
```

It is possible to download MARCH directly from the CRAN Homepage here. When MARCH is installed a new R Project in R Studio can be created. Therefore you choose:

File ▷ New Project ▷ Existing Directory ▷ Directory to your MARCHfile

Make sure the data which is used for the Markovian models is located in the data folder on the toplevel environment of your package MARCH. After a reload MARCH, just close and open the project again, the data should be available. With the following command it is possible to test if the data is available, respectively all the data in your R environment are shown.

```
data()
```

Scroll down to:

```
Data sets in package 'NameOfProject'
```

If everything worked well the data should appear here and can now be used by the MARCH package.

2 Building Models

2.1 MARCH Dataset

First the data must be load and turned into a MARCH dataset with the following commands. Here this is shown by the example of the pewee_df data.

```
data(pewee_df)
PEWEE <- march.dataset.loadFromDataFrame(pewee_df,
                        MARGIN = 1, weights = NA, missingDataRep = NA)
```

If everything worked well it should be able to see the Data and the Values which are now a callable option for the different Markovian models as shown in Figure 1 for the example of the pewee_df. For a better explanation and several examples concerning load

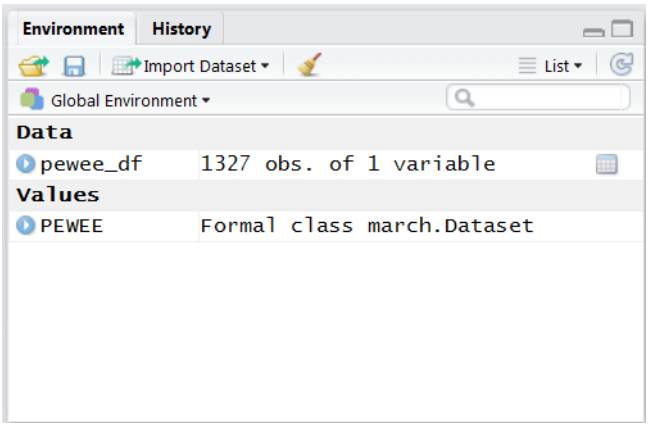

Figure 1: Data and Values of the pewee_df example

data from a Dataframe see the following help command.

```
help(march.dataset.loadFromDataFrame)
```

2.2 Building Homogenous Markov Chains Models

Now all the preparations to start a markovian models are done. To start an example of the Independence Model is chosen. It calculates the loglikelihood, the total number or

variables (dsl) aswell as the Aikake (AIC) and the Bayesian (BIC) Information Criterion. The outcome indP is the percentage for each of the variables. The model is built with the following command.

```
Indep <- march.indep.construct(PEWEE)
```

The Indep class should now appear in the environment window of your R-Studio, Figure 1. Notice that the model is built with the MARCH dataset! Further the results and a summary should be printed.

```
print(Indep)
march.summary(Indep)
```

Now the results should appear in the console of the R-Studio as it is shown for the pewee_df example in Figure 2. Now the maxOrder parameter is used. With the output

```
Independence
             1         2         3
indP 0.5207234 0.2690279 0.2102487

      1   2   3
indC 691 357 279

ll :  -1354.713
dsL:  1327
> march.summary(Indep)
                    ll param      BIC      AIC
Independence -1354.713     2 2723.807 2713.425
```

Figure 2: Results of the pewee_df example

of the following commands it is obvious that the total amount of used variables (dsl) is reduced by the the value of maxOrder. The Independence Model needs the first 5 variables to create the model starting from the 6th variable.

```
Indep.2 <- march.indep.construct(PEWEE,maxOrder=5)
print(Indep.2)
```

To build Markov chains with the loaded data the following command is used. The first two examples are both first order and only differs in their value of the maximal order.

```
MC1 <- march.mc.construct(y=PEWEE,order=1,maxOrder=1)
print(MC1)
MC1.2 <- march.mc.construct(PEWEE,order=1,maxOrder=5)
print(MC1.2)
```

With:
y: MARCH Dataset
order: order of the constructed Markov ChainYY
maxOrder: maximum visible order among the set of Markovian models to compare

In the third example a Markov chain second order is created by MARCH.

```
MC2 <- march.mc.construct(PEWEE,order=2,maxOrder=5)
print(MC2)
```

Now the differences between 1st and 2nd order are visible. For more information about the construction of the independence model, the Markov chains (mc) or the summary of the results see:

```
help(march.indep.construct)
help(march.mc.construct)
help(march.summary)
```

as well as their description of the class:

```
help("march.Indep-class")
help("march.Mc-class")
```

For further informations and a mathematical background explanation of Markov Chains see Berchtold and Raftery (2002).

2.3 Comparison of the Mixture Transition Distributions, High Order Markov Chains and the Independent Model

In the next example a comparison between three different models with sleep data is built. Therefore the Indepence model is built first and then Markov chains of 1st to 3rd order. The maximal order of 3 is the same for every model. How the models are built and the results are saved in a list is shwon by the following commands.

```
models <- list()
models[[length(models)+1]] <- march.indep.construct(sleep,maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(sleep,order=1,maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(sleep,order=2,maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(sleep,order=3,maxOrder=3)
```

To show the performance of the indicators like loglikelihood (ll), number of independent parameters (param), as well as the Bayesian (BIC) and the Aikake Information Criterion (AIC) it is necessary to bind all these informations together in a variable, here called r which then can be printed.

```
r <- do.call(rbind,lapply(models,march.summary))
print(r)
```

The result should appear like shown in Figure 3 Further a Mixture Transition Distributions (MTD) models with different orders can be created with the following command.

```
mtd <- march.mtd.construct(y=march.Dataset, order, maxOrder )
```

	ll	param	BIC	AIC
Independence	-4590.459	5	9221.431	9190.918
MC(1)	-3723.445	30	7689.968	7506.890
MC(2)	-3459.642	130	7972.619	7179.283
MC(3)	-3122.209	295	8634.681	6834.418

Figure 3: Summary of the informations about Independent Model and MC 1 to 3

For more information about the construction of the MTD models and a general description of MTD see the help pages.

```
help("march.mtd.construct")
help("march.Mtd-class")
```

These models are explained in Berchtold and Raftery (2002). Now a first example of the MTD models will be shown and the output shortly discussed.

```
mtd2 <- march.mtd.construct(sleep,order=2,maxOrder=3)
print(mtd2)
march.summary(mtd2)
```

The output of the summary command are the transition matrix Q, the vector of lag parameters phi as well as the loglikelihood ll and the number of parameters dsL.
Now some more MTD models with different orders are created which later will be compared to the models calculated before.

```
mtd3 <- march.mtd.construct(sleep,order=3,maxOrder=3)
print(mtd3)
march.summary(mtd3)

mtdg2 <- march.mtd.construct(sleep,order=2,maxOrder=3,mtdg=TRUE)
print(mtdg2)
march.summary(mtdg2)

mtdg3 <- march.mtd.construct(sleep,order=3,maxOrder=3,mtdg=TRUE)
print(mtdg3)
march.summary(mtdg3)
```

Now to compare all the different models calculated above a list which contains all the informations is created. Then the output of the command is the summary of all the models which now can be compared easily.

```
models_all <- list(8)
models_all[1] <- models[1]
models_all[2] <- models[2]
models_all[3] <- models[3]
models_all[4] <- mtd2
models_all[5] <- mtdg2
```

```
models_all[6] <- models[4]
models_all[7] <- mtd3
models_all[8] <- mtdg3

r <- do.call(rbind,lapply(models_all,march.summary))
print(r)
```

In Figure 4 the summary of the results of the different models are shown. With this

	ll	param	BIC	AIC
Independence	-1350.7647	2	2715.9061	2705.5293
MC(1)	-698.4928	5	1432.9276	1406.9856
MC(2)	-368.8243	9	802.3443	755.6485
MTD(2)	-3540.9461	28	7308.7646	7137.8922
MTDg(2)	-3532.8112	47	7446.4439	7159.6224
MC(3)	-354.1780	14	808.9938	736.3560
MTD(3)	-3499.9546	29	7234.8841	7057.9091
MTDg(3)	-3476.2332	59	7430.5190	7070.4664

Figure 4: Summary of the different models

summary it is detectable that in relation to the BIC the Markov Chain model 2nd order is the best. According to the AIC the Markov Chain model 3rd order is the best. To print detailled informations about a model the following command is necessary.

```
print(models[i])
```

Here i is the dummie variable of the location of the model inside the array list. For example the Independence model is in the first place of the list so the dummie variable is replaced by 1.

2.4 Building Hidden Markov Chain Model

Now the hidden models are introduce and compared with the homogenous models already built. Here a introduction into two different Hidden models, the Hidden Markov Model and the Double Chain Markov model is given. First the data is loaded and a march.Dataset is created therefore several different homogenous models are built and compared, so the the best model can be found. The maximal order is set to 3. This is done with the following commands.

```
data(pewee_df)
PEWEE <- march.dataset.loadFromDataFrame(pewee_df,
                    MARGIN = 1, weights = NA, missingDataRep = NA)

models <- list()
models[[length(models)+1]] <- march.indep.construct(PEWEE,maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(PEWEE,order=1,
maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(PEWEE,order=2,
```

```
maxOrder=3)
models[[length(models)+1]] <- march.mc.construct(PEWEE,order=3,
maxOrder=3)
models[[length(models)+1]] <- march.mtd.construct(PEWEE,order=2,
maxOrder=3,llStop=0.0001)
models[[length(models)+1]] <- march.mtd.construct(PEWEE,mtdg=TRUE,order=2,
maxOrder=3,llStop=0.0001)
models[[length(models)+1]] <- march.mtd.construct(PEWEE,order=3,
maxOrder=3,llStop=0.0001)
models[[length(models)+1]] <- march.mtd.construct(PEWEE,mtdg=TRUE,order=3,
maxOrder=3,llStop=0.0001)

r <- do.call(rbind,lapply(models,march.summary))
print(r)
```

Now a comparison between the different homogenous models can be made. According to the BIC, the Markov Chain Model second order fits the data best, because this model has the lowest BIC value as seen in Figure 5. Now to show more detailed informations

	ll	param	BIC	AIC
Independence	-1350.7647	2	2715.9061	2705.5293
MC(1)	-698.4928	5	1432.9276	1406.9856
MC(2)	-368.8243	9	802.3443	755.6485
MC(3)	-354.1780	14	808.9938	736.3560
MTD(2)	-570.3554	6	1183.8413	1152.7108
MTDg(2)	-499.6280	9	1063.9517	1017.2560
MTD(3)	-570.3549	6	1183.8402	1152.7097
MTDg(3)	-488.8700	11	1056.8126	999.7401

Figure 5: Summary of the different models

about the best model we can print them with the following command.

```
print(models[3])
```

May with another march.Dataset another model will be the best. To print it, the right position of the models[] array needs to be written into the comman. To compare the best homogenous model with the Hidden Markov Models they need to be created first. First the Hidden Markov model is created. For more general informations and informations about the construction of Hidden Markov models see these help commands.

```
help(march.dcmm.construct)
help("march.Dcmm-class")
```

Now we want to create a Hidden Markovian Model therefore the following constructor is used.

```
march.dcmm.construct(y, orderHC, orderVC, M, gen = 5, popSize = 4,
         maxOrder = orderVC, seedModel = NULL, iterBw = 2, stopBw = 0.1)
```

With

y:	is the march.Dataset
orderHC:	of the hidden chain
orderVC:	the order of the visible chain
M:	the number of hidden state
gen:	the number of generation performed by an evolutionary alglorithm
popSize:	the number of individuals stored into the population
maxOrder:	the maximum visible order among the set of Markovian models to compare
seedModel:	a model to optimize using Baum-Welch algorithm
iterBw:	the number of iteration performed by the Baum-Welch algorithm
stopBw:	is the minimum increase in quality (log-likelihood) authorized in the Baum-Welch algorithm

To build Hidden Markovian Models (HMM) with the command described above it is necessary to set the order of the visible chain to zero (orderVC = 0). With the following commands two HMM are built and one is printed.

```
HMM.1 <- march.dcmm.construct(PEWEE,orderHC=1,M=2,orderVC=0,maxOrder=3,
                              popSize=10,gen=5)
march.summary(HMM.1)

HMM.2 <- march.dcmm.construct(PEWEE,orderHC=1,M=2,orderVC=0,maxOrder=3,
                              popSize=1,gen=1,iterBw=50,stopBw=0.0001)
march.summary(HMM.2)
```

This should get the output similar to what is visible in Figure 6 where the value in brackets in the first column is the order of the hidden chain (orderHC). To print further

```
> march.summary(HMM.2)
             ll param      BIC      AIC
Hmm(1) -690.8603     7 1432.039 1395.721
```

Figure 6: the summary of a Hidden Markovian Model first order

details about the model the print command is used. The help page to the Dcmm.class where the output is described is also useful.

```
print(HMM.2)
help("march.Dcmm-class")
```

2.5 Building Double Chain Markov Model

To create Double Chain Markov models (DCMM) the same constructor is used with which the HMM have been built already. But now the order of visible chains and hidden chain is set to greater than 0 so there is at least one hidden chain and one visible chain.

```
march.dcmm.construct(y, orderHC > 0, orderVC > 0, M, gen = 5, popSize = 4,
        maxOrder = orderVC, seedModel = NULL, iterBw = 2, stopBw = 0.1)
```

For more informations and a detailled explanation of the DCMM as well as a detailed mathematical description read Berchtold (2002). To create the first example of a DCMM the following code with one visible and one hidden chain is used.

```
DCMM.1A <- march.dcmm.construct(PEWEE,orderHC=1,M=2,orderVC=1,maxOrder=3,
                                popSize=4,gen=5,iterBw=2,stopBw=0.0001)
print(DCMM.1)
```

Now additional iterations should be calculated. Fot this the Baum Welch (BW) Algortihm is used and the number of iterations is set to 50 (iterBw = 50). The previous model is used as a seed Model.

```
DCMM.1 <- march.dcmm.construct(PEWEE,orderHC=1,M=2,orderVC=1,maxOrder=3,
                               seedModel=DCMM.1A,iterBw=50,stopBw=0.0001)
print(DCMM.1)

march.summary(DCMM.1)
march.summary(DCMM.2)
```

In Figure 7 the comparison between these two different computation of a DCMM is shown. The name of the models is followed by the orders of the HC and VC in brackets, here for both models and both orders 1. It is obvious that with additional iterations the BIC as well as the AIC have improved. DCMM.1A is first calculated and DCMM.1 is calculated with additional BW iterations and the first model as a seed model.

```
> march.summary(DCMM.1A)
                ll param      BIC      AIC
Dcmm(1,1) -694.3972    15 1496.621 1418.794
> march.summary(DCMM.1)
                ll param      BIC      AIC
Dcmm(1,1) -688.2029    14 1477.044 1404.406
```

Figure 7: the summary of the two DCMM.

In the next example the hidden states are raised, which is represented by the variable M.

```
DCMM.2 <- march.dcmm.construct(PEWEE,orderHC=1,M=3,orderVC=1,maxOrder=3,
                               popSize=1,gen=1,iterBw=50,stopBw=0.0001)
march.summary(DCMM.2)
```

A comparison between Figure 8 and Figure 7 shows the improvement of the AIC and the BIC. But with the higher number of parametres the computation time increases.

```
> march.summary(DCMM.2)
                ll param      BIC      AIC
Dcmm(1,1) -494.5805    25 1168.871 1039.161
```

Figure 8: the summary of the DCMM with more hidden states.

For the next example the hidden order is raised to 2 and the number of hidden states is reduced to the former value.

```
DCMM.3 <- march.dcmm.construct(PEWEE,orderHC=2,M=2,orderVC=1,maxOrder=3,
                               popSize=1,gen=1,iterBw=50,stopBw=0.0001)
march.summary(DCMM.3)
```

In Figure 9 the improvement of BIC and AIC is visible as well as the reduction of the parametres. An improvement of the loglikelihood is done in comparison with the previous models.

```
> march.summary(DCMM.3)
                ll param      BIC      AIC
Dcmm(2,1) -445.7023    15 999.2307 921.4045
```

Figure 9: the summary of the DCMM with more hidden chains.

In the last example of DCMM the order of VC is raised to 2 and the order of the HC is reset to 1.

```
DCMM.4 <- march.dcmm.construct(PEWEE,orderHC=1,M=2,orderVC=2,maxOrder=3,
                               popSize=1,gen=1,iterBw=50,stopBw=0.0001)
march.summary(DCMM.4)
```

Figure 10 shows the summary. Of all examples this model has the best results with respect to the BIC, AIC and loglikelihood.

```
> march.summary(DCMM.4)
                ll param      BIC      AIC
Dcmm(1,2) -343.2579    39 966.8639 764.5158
```

Figure 10: the summary of the DCMM with more visible chains.

For the best fitting model (DCMM.4) the optimal sequence of hidden states should be extracted. This is occured with the usage of the viterbi algorithm.

```
help(march.dcmm.viterbi)
HS <- march.dcmm.viterbi(DCMM.4,PEWEE)

print(HS)
print(PEWEE@y)
```

For further informations and a mathematical explanation of the viterbi algorithm read Forney Jr (1973) and Berchtold (2002)

2.6 Analysis of the hidden states

To analyse the hidden states we use the example provided in the help page of the march.dcmm.construct() function. Therefore the HMM is built.

```
help(march.dcmm.construct)
data(sleep)
HMM <- march.dcmm.construct(sleep,orderHC=1,orderVC=0,M=3,gen=1,
                            popSize=1,iterBw=10,stopBw=0.0001)
print(HMM)
```

For the computation of the most likely sequences of the hidden states the Viterbi Algorithm is used.

```
HS <- march.dcmm.viterbi(HMM,sleep)
```

Now the most likely sequence of hidden states is converted into a data frame using the following code with the for loop.

```
HS2 <- matrix(NA,nrow=1000,ncol=7)
HS2 <- as.data.frame(HS2)
for (i in 1:1000){
  temp <- HS[[i]]
  ltemp <- length(temp)
  HS2[i,1:ltemp] <- temp
}
```

To visualize the hidden states the package TraMineR is required which already should be loaded. If not see the first chapter where it is described how to load a package. With the following code two plots are created.

```
hs.labels <- c("1", "2", "3")
hs.seq <- seqdef(HS2, alphabet=hs.labels, states = hs.labels,
labels = hs.labels)

seqfplot(hs.seq, border = NA, title = "10 most frequent sequences")
seqdplot(hs.seq, border = NA,
                title = "Distribution of hidden states by period")
help(seqplot)
```

If the following error message appears, the plot window is not big enough. In R-Studio it is easy to size this window up.

```
Error in plot.new() : figure margins too large
```

2.7 Hierarchical Models

In another example a strictly hierarchical model is created. Therefore we need seed model which is created with the code below. In fact it is an HMM first order.

```
Seed.Model <- march.dcmm.construct(sleep,orderHC=1,orderVC=0,M=3,gen=1,
                                   popSize=1,iterBw=1,stopBw=0.0001)
```

```
print(Seed.Model)

HMM.h1 <- Seed.Model
```

Now a matrix for the high order transition probabilities between hidden states is used. Here called A which is mathematical explained in Berchtold (2002)

```
A <-  matrix(nrow=3,ncol=3,c(0.5,0.5,0,
                             0,0.5,0.5,
                             0,0,1),byrow=TRUE)
A
```

This matrix A is now used to calculate the next HMM with the usage of the seed model calculated before.

```
HMM.h1@A <- A
HMM.h1 <- march.dcmm.construct(sleep,orderHC=1,orderVC=0,M=3,gen=1,
                               popSize=1,iterBw=5,stopBw=0.0001,
                               seedModel=HMM.h1)
print(HMM.h1)
```

To extract the hidden state sequences the Viterbi Algorithm is taken.

```
HS.h1 <- march.dcmm.viterbi(HMM.h1,sleep)
```

The triangular model calculated in the next step only differs from the model before only in the transition matrix between hidden states A. The seed model is the same model as calculated above so please make sure it has been built.

```
HMM.h2 <- Seed.Model
A <-  matrix(nrow=3,ncol=3,c(0.5,0.25,0.25,
                             0,0.5,0.5,
                             0,0,1),byrow=TRUE)
A
HMM.h2@A <- A
HMM.h2 <- march.dcmm.construct(sleep,orderHC=1,orderVC=0,M=3,gen=1,
                               popSize=1,iterBw=5,stopBw=0.0001,
                               seedModel=HMM.h2)
print(HMM.h2)
```

All this three different model calculations only differs in their transition matrix A which is shown in Figure 11. For the classification the hidden state sequences are extracted.

```
HS.c <- march.dcmm.viterbi(HMM.c,sleep)
```

This is convertet into a data frame with the following code.

```
> A                    > A                    > A
     [,1] [,2] [,3]         [,1] [,2] [,3]         [,1] [,2] [,3]
[1,]  0.5  0.5  0.0    [1,]  0.5 0.25 0.25    [1,]    1    0    0
[2,]  0.0  0.5  0.5    [2,]  0.0 0.50 0.50    [2,]    0    1    0
[3,]  0.0  0.0  1.0    [3,]  0.0 0.00 1.00    [3,]    0    0    1
```

Figure 11: The differences between the matrices A from left to right: first model, triangular model and classification model

```
HSc <- matrix(NA,nrow=1000,ncol=7)
HSc <- as.data.frame(HSc)
for (i in 1:1000){
  temp <- HS.c[[i]]
  ltemp <- length(temp)
  HSc[i,1:ltemp] <- temp
}
```

For this data frame a seqdef object is created with the help of the TraMineR package. Therefore the following code is necessary.

```
Creation of a TraMineR seqdef object for the observed data
hs.labels.c <- c("0","1","2","3","4","5")
hs.c <- seqdef(sleep_df, alphabet=hs.labels.c, states = hs.labels.c,labels = hs.labels.c)

# Presentation of the data by group
seqfplot(hs.c, border = NA, group=HSc[,1],title = "25 most frequent sequences",tlim=1:25)
seqdplot(hs.c, border = NA, group=HSc[,1],title = "Distribution of observed data")
```

If the following error message appears, may the plot window is not big enough. In R-Studio it is easy to size this window up.

```
Error in plot.new() : figure margins too large
```

Bibliography

Berchtold, A. (2002). High-order extensions of the double chain markov model. *Stochastic Models*, 18(2):193–227.

Berchtold, A. and Raftery, A. E. (2002). The mixture transition distribution model for high-order markov chains and non-gaussian time series. *Statistical Science*, pages 328–356.

Forney Jr, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.