MARKOV MODELS

1. Markov Models

1.1. OMM – Observable Markov Model

1.2. HMM – Hidden Markov Model
1.2.1. Problem 1
1.2.1.1. Brute force calculation
1.2.1.2. Backward calculation
1.2.1.3. Forward calculation
1.2.1.4. Proving validity of forward and backward equations
1.2.1.5. Fast full table calculation
1.2.2. Problem 2
1.2.2.1. Viterbi algorithm
1.2.3. Problem 3
1.2.3.1. Baum-Welch method

1.3. Implementation of HMM
main
learn
Display HMM outputs
createSS
createOS
displayTableWithSymbols
displayTableWithNumberComponents
Problem 1
bruteForceCalculation
backwardCalculation
forwardCalculation
fastFullMethod
Problem 2
viterbiAlgorithm
Problem 3
newPi
newS
MATRIX OPERATIONS
displayMatrix
transponMatrix
multiplieMatrices
multiplieCells
addMatrices
combineMatrices
maxInstedSum
increaseCounter
randomCellFromVector

1.4. Program output

1. Three Problems of Hidden Markov Models

This paper covers theory which is related with solving three basic problems of Hidden Markov Models. Goal was to produce an approach which is different from most of the other literature that can be found on this subject. Main idea was to approach the problems as if they were not already solved and then to try to build up some kind of logic that will help us create programming procedures an mathematical formulas which can then be used to solve the problems. Goals of the paper can be sumorized in folowing three points.

- <u>Simple explanation</u>
  One of the atentions of this paper was to give detailed but yet as simple as possible explanation of HMM and its problems withouth using complex mathematics. This paper is attended for wide odience that might not have enough technical knowledge to understand the problems as they are presented in most of the other literture covering this topic where a high experience in symbolic language of mathematics and probability is requierd.

- <u>Logic behind formulas</u>
  Other goal of this paper was to present the logic that lies behind HMM and its problems. Goal was not to teach the reader which formulas can be used to solve HMM problems, but rather how to get to that formulas using plain logic. By aquireng full understanding of HMM problems new algorithms and solutions might be developed for better performance.

- <u>Simple results</u>
  At last, by introduction of matrices and some special matrix operations, existing formulas were made much simpler. This resulted in higher resembles between solutions of different problems and such formulas are easier to remember then complex procedures.

- There are two types of Markov Models:

  1. OMM – Observable Markov Model
  2. HMM – Hidden Markov Model

- Observable Markov Model is defined as folows:

$$\lambda=(\pi,S) \qquad (1.1)$$

$$\pi=\begin{bmatrix}\pi_1 & \cdots & \pi_N\end{bmatrix} \quad , \quad S=\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & & s_{2N} \\ \vdots & & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{NN}\end{bmatrix} \qquad (1.2)$$

$$\pi_c = P(s_1 = c \mid \lambda) \quad , \quad s_{rc} = P(s_{t+1} = c \mid s_t = r, \lambda) \qquad (1.3)$$

where:
λ symbol for OMM,
S state-transition probability distribution matrix,
π initial-state probability distribution matrix,
N number of possible states in which model can be,
s_{rc} probability that model λ will switch to state 'c' if it is in state 'r', $1\le r,c \le N$,
π_c probability that first state of the model λ will be 'c', $1\le c\le N$,
s_t state of the model on step 't', these states are labeled as $s_t \in \{1,2,...,N\}$.

From (1.3) we can conclude that sum of all elements in the same row, in matrices π and S, is always 1, which can be mathematicly expressed as folows:

$$\sum_{c=1}^{N} \pi_c = 1 \quad , \quad \sum_{c=1}^{N} s_{rc} = 1$$

- Hidden Markov Model is defined as folows:

$$\lambda=(\pi,S,O) \qquad (1.4)$$

$$\pi=\begin{bmatrix}\pi_1 & \cdots & \pi_N\end{bmatrix} \quad , \quad S=\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & & s_{2N} \\ \vdots & & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{NN}\end{bmatrix} \quad , \quad O=\begin{bmatrix} o_{11} & o_{12} & \cdots & o_{1V} \\ o_{21} & o_{22} & & o_{2V} \\ \vdots & & \ddots & \vdots \\ o_{N1} & o_{N2} & \cdots & o_{NV}\end{bmatrix} \qquad (1.5)$$

$$\pi_c = P(s_1 = c \mid \lambda) \quad , \quad s_{rc} = P(s_{t+1} = c \mid s_t = r, \lambda) \quad , \quad o_{rc} = P(o_t = c \mid s_t = r, \lambda) \qquad (1.6)$$

where: λ symbol for OMM,
π initial-state probability distribution matrix,
S state-transition probability distribution matrix,
O observation-symbol probability distribution matrix,
N number of possible states in which model can be,
V number of possible variables that model can output,
π_c probability that first state of the model λ will be 'c', $1 \leq c \leq N$,
s_{rc} probability that model λ will switch to state 'c' if it is in state 'r', $1 \leq r,c \leq N$,
o_{rc} probability that model λ will ouput variable 'c' if it is in state 'r', $1 \leq r \leq N$, $1 \leq c \leq V$,
s_t state of the model on step 't', these states are labeled as $s_t \in \{1,2,...,N\}$,
o_t output variable of the model on step 't', these output varaibles are labeled as $o_t \in \{1,2,...,N\}$.

From (1.3) we can conclude that sum of all elements in the same row, in matrices π and S, is always 1, which can be mathematicly expressed as folows:

$$\sum_{c=1}^{N} \pi_c = 1 \quad , \quad \sum_{c=1}^{N} s_{rc} = 1 \quad , \quad \sum_{c=1}^{N} o_{rc} = 1$$

- Observable Markov Model is called observable because at each step we have information in which state the model is, meaning that we are able to observe the states of the model as he switches from one state to another.

- Hidden Markov Model is based on Observable Markov Model but with the upgrade that at each state, model outputs a single variable with some probability. It is called hidden because we are not aware of models states, like we are in OMM, but we only see these outputs. These probabilities are defined by introducing O matrix.

- Folowing chapters will focus on solving three problems of HMM which occur when HMM is used in pattern recognition.

1. Find probability that model will give defined output-sequence, for example V=(a,a,b,a,c).
2. Find state-sequence which is most probable to give defined output-sequence, for example V=(a,a,b,a,c).
3. Find model which is most probable to give defined output-sequence, for example V=(a,a,b,a,c).

To better understand the above problems here is a short interpretation of them when HMM is used in speech recognition.

- Problem 1:
When a new word is given to such system, system will transform this word into output-sequence and then it will solve problem 1 for each of the HMMs in systems memory. By taking the bigest result, it will find HMM which is most probable to give that output-sequence, and will recognize given word as word which was connected to that HMM.

- Problem 2:
Problem two occurs when we have to find initial state fot HMM while solving problem three.

- Problem 3:
Spoken word is transformed into output-sequence. For this word we want to build HMM. At first HMM parameters are set by some rule which is independed of the given word. Then problem one is solved. After that we change HMM parameters, solve problem one again after which the resulted probability should be greater. By iterating this process we finaly get an HMM which has high probability of producing output-sequence of the given word. This means that solving problem three means finding a rule a rule for getting new parameters from existing ones.
At the end of the learning process each word is connected to exactly one HMM. Since system transforms each word into output-sequence, which is how HMMs see words, probabiliy that resulted HMM will give output sequence equal to the output-sequence of the connected word must be greater then probability that HMM will give ouput-sequence of any other word in memory.

1.1.1. Problem 1

- Problem 1:
  Solution to problem 1 is to find probability that HMM will produce given output-sequence, for instance V=(a,a,b,a,c). This can be mathematicly expressed as:

$$P(V|\lambda) \tag{1.7}$$

  This is equal to finding the sum of all the rows of the table 1.2 which will be explained later. In folowing chapters, three methods for solving this problem are given.

- Brute force calculation:
  Brute force calculation method is the simplest because it calculates the result exactly as it is defined. Major drawback of this method is that is extremly time consuming to the point that it becomes unusable for larger models or ouput-sequences. Luckely, faster algorithms were developed which are based on the fact that brute force calculation is not saving some of the intermediate results but calculates the same expressions more then once.

- Backward and Forward calculation:
  Fastest possible way to solve problem one is to use backward or forward calculation. These two methods are basicly the same and differ only in the direction in which calculations are made, which is the reason why they have such names.
  For instance if we have output sequence O=(a,b,c) first step of forward calculation will give us probability that HMM will produce output-sequence O=(a), second step will give us probability that HMM will produce output-sequence O=(a,b) and so on. This means that steps are going in the forward way, in the way the output-sequence was created. Each step has o phisical menaning, it is a solution to problem one for each of the shorter output-sequences.
  On the other hand, in backward calculation only the final result has phisical meaning as solution of problem one for given output-sequence.

- Fast full table calculation:
  Forth way of solving problem one is a method which is by its time consuption between brute force calculation on one side and forward and backward calculations on the other side. However, beside solving problem one, intermidate results of this method gives us some extra informations which can not be optained through forward and backward calculations.

- These four methods will be explained in folowing chapters. Explanations start by using concrete model as defined below:

$$\lambda=(\pi,S,O) \tag{1.8}$$

$$\pi = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \quad , \quad S = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \quad , \quad O = \begin{bmatrix} o_{1a} & o_{1b} & o_{1c} \\ o_{2a} & o_{2b} & o_{2c} \end{bmatrix} \tag{1.9}$$

  Problem one is then solved with the assumption of output-sequence:

$$(a,b,c) \tag{1.10}$$

  Observations made by analysing solutions for this example will then be used for creation of formulas which can be used for general HMM as defined with (1.4) till (1.6).

  As the last step this formulas will be proved to be generaly valid.

1.1.1.1. Brute force calculation

To calculate probability that model will output exact sequence of variables O=(a,b,c), we can simply take N copyes of such model and from them take those that produced O=(a,b,c). All of the models which produced O=(a,b,c) will not go through the same state-sequences. Idea is to choose one state seuqnence, count all the models which produced O=(a,b,c), do the same with all other state-sequences and summ the results which is then the solution to our problem. This procedure is now explained in detail.

From all models which produced O=(a,b,c) we will now take all models which went through state sequence S=(1,2,2).

From any amount of models, only π_1 percent of them will start with state 1.
That means that from all N models, $\pi_1 N$ models will start with state 1.

From any amount of models that are in state 1, only o_{a1} percent of them will output variable a.
That means that from $(\pi_1 N)$ previous models (they are all in state 1), $o_{1a}(\pi_1 N)$ models will output variable a.

From any amount of models that are in state 1, only s_{12} percent of them will switch to state 2.
That means that from $(o_{a1}\pi_1 N)$ previous models (they are all in state 1), $s_{12}(o_{1a}\pi_1 N)$ models will switch to state 2.

That means that number of models that went through states S=(1,2,2) and produced outputs O=(a,b,c) is:

$$o_{2c}s_{22}o_{2b}s_{12}o_{1a}\pi_1 N$$

If we switch the order we can write this as:

$$N\pi_1 o_{1a}s_{12}o_{2b}s_{22}o_{2c}$$

If we do the same calculation for all posible state sequences we would get folowing table where we have highlited in red the above result:

111	$N\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{11}o_{1c}$
112	$N\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{12}o_{2c}$
121	$N\pi_1 o_{1a}\cdot s_{12}o_{2b}\cdot s_{21}o_{1c}$
122	$N\pi_1 o_{1a}\cdot s_{12}o_{2b}\cdot s_{22}o_{2c}$
211	$N\pi_2 o_{2a}\cdot s_{21}o_{1b}\cdot s_{11}o_{1c}$
212	$N\pi_2 o_{2a}\cdot s_{21}o_{1b}\cdot s_{12}o_{2c}$
221	$N\pi_2 o_{2a}\cdot s_{22}o_{2b}\cdot s_{21}o_{1c}$
222	$N\pi_2 o_{2a}\cdot s_{22}o_{2b}\cdot s_{22}o_{2c}$

Tab.1 .1 Number of models with output-sequence O=(a,b,c) per state-sequence.

First line means that from all the models that went through states 111 only $N\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{11}o_{1c}$ produced outputs O=(a,b,c).
Second line means that from all the models that went through states 112 only $N\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{12}o_{2c}$ produced outputs O=(a,b,c).
If we summ all of these results together we will get the number of models which produced outputs O=(a,b,c).
If we then divide this number with the total number of models N, we get probability that the model will produce output sequence O=(a,b,c) which is the solution to our problem.

For further analysys it is usefull to rewrite the above table by ommiting varaible N:

111	$\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{11}o_{1c}$
112	$\pi_1 o_{1a}\cdot s_{11}o_{1b}\cdot s_{12}o_{2c}$
121	$\pi_1 o_{1a}\cdot s_{12}o_{2b}\cdot s_{21}o_{1c}$
122	$\pi_1 o_{1a}\cdot s_{12}o_{2b}\cdot s_{22}o_{2c}$
211	$\pi_2 o_{2a}\cdot s_{21}o_{1b}\cdot s_{11}o_{1c}$
212	$\pi_2 o_{2a}\cdot s_{21}o_{1b}\cdot s_{12}o_{2c}$
221	$\pi_2 o_{2a}\cdot s_{22}o_{2b}\cdot s_{21}o_{1c}$
222	$\pi_2 o_{2a}\cdot s_{22}o_{2b}\cdot s_{22}o_{2c}$

Tab.1.2 Probability for each state-sequence that model will produce output-sequence (a,b,c).

Now each row presents probability that a model will give output-sequence (a,b,c) while siultaniously going through given state-sequence.

We will now calculate number of needed multiplications and summands.
For each state sequence we have 5 multiplications (because in expression $\pi_1\cdot o_{1a}\cdot s_{11}\cdot o_{1b}\cdot s_{11}\cdot o_{1c}$ there are 5 dots between variables). Using induction we can see that if output sequence has T outputs we would have T+T-1=2T-1 multiplications.
Total number of such test sequences is 8. Using induction we can se that if number of possible states was N, then number of possible test sequences would be N^T. This is the total number of needed summands.
That means that total number of multiplications is $8\cdot 5=40$, or in general case it is $N^T(2T-1)$.

Such a great number of calculations can be greatly reduced. With the closer inspection of the above table it can be seen that a lot of mulitpilications are made more then once. For instance, lines with indexes 111 and 112 both start with multiplications $\pi_1 o_{1a.}$ Idea is to make this multipication only ones, then to save this result into memory and use it next time we have to calculate $\pi_1 o_{1a}$. This way, four multiplications needed to calculate $\pi_1 o_{1a}$ in 111,112,121 and 122 are transformed into one multiplication and three memory operations which is much faster for computer to calculate.

Sum of the all rows of the above table can be mathematicly defined as folows:

$$\sum_{i_3=1}^{2}\sum_{i_2=1}^{2}\sum_{i_1=1}^{2}\pi_{i_1} o_{i_1 a} s_{i_1 i_2} o_{i_2 b} s_{i_2 i_3} o_{i_3 c} \tag{1.11}$$

Generaly this formula can be written as folows:

$$\sum_{i_T=1}^{N}\cdots\sum_{i_2=1}^{N}\sum_{i_1=1}^{N}\pi_{i_1} o_{i_1 o_1} s_{i_1 i_2} o_{i_2 o_2} \ldots s_{i_{T-1} i_T} o_{i_T o_T} \tag{1.12}$$

The above formula doesn't need to be proven because it comes directly from the definition of the problem.

1.1.1.2. Backward calculation

- With backward procedure we will try to use minimal number of operations to calculate the sum of all rows in the table 1.2. We will start our explanation of Backward-calculation by deriving this procedure for example defined by (1.3).
  To easier ilustrate the procedue here we show table 1.2 once again:

		$\beta_1(1)$	$\beta_2(1)$	$\beta_3(1)=1$
111	$\pi_1 o_{1a}$	$\cdot s_{11}o_{1b}$	$\cdot s_{11}o_{1c}$	
112	$\pi_1 o_{1a}$	$\cdot s_{11}o_{1b}$	$\cdot s_{12}o_{2c}$	
121	$\pi_1 o_{1a}$	$\cdot s_{12}o_{2b}$	$\cdot s_{21}o_{1c}$	
122	$\pi_1 o_{1a}$	$\cdot s_{12}o_{2b}$	$\cdot s_{22}o_{2c}$	
211	$\pi_2 o_{2a}$	$\cdot s_{21}o_{1b}$	$\cdot s_{11}o_{1c}$	
212	$\pi_2 o_{2a}$	$\cdot s_{21}o_{1b}$	$\cdot s_{12}o_{2c}$	
221	$\pi_2 o_{2a}$	$\cdot s_{22}o_{2b}$	$\cdot s_{21}o_{1c}$	
222	$\pi_2 o_{2a}$	$\cdot s_{22}o_{2b}$	$\cdot s_{22}o_{2c}$	

Tab.1.3 Probability for each state-sequence that model will produce output-sequence O=(a,b,c).

First we will define β variables which will help us to present the procedure in compact way.

Lets use value 1 to define symbol $\beta_3(1)$: $\beta_3(1) = 1$ (1.13)
Lets use value 1 to define symbol $\beta_3(2)$: $\beta_3(2) = 1$ (1.14)

Lets give the sum of all parts inside upper full square symbol $\beta_2(1)$: $\beta_2(1) = s_{11}o_{1c}+s_{12}o_{2c}$ (1.15)
Lets give the sum of all parts inside lower full square symbol $\beta_2(2)$: $\beta_2(2) = s_{21}o_{1c}+s_{22}o_{2c}$ (1.16)

Lets give the sum of all parts inside upper doted square symbol $\beta_1(1)$: $\beta_1(1) = s_{11}o_{1b}\beta_2(1)+s_{12}o_{2b}\beta_2(2)$ (1.17)
Lets give the sum of all parts inside lower doted square symbol $\beta_1(2)$: $\beta_1(2) = s_{21}o_{1b}\beta_2(1)+s_{22}o_{2b}\beta_2(2)$ (1.18)

Total sum of all rows can be calculated as: $P(V|\lambda) = \pi_1 o_{1a}\beta_1(1)+\pi_2 o_{2a}\beta_1(2)$ (1.19)

In order to simplifie formulas (1.13) till (1.19) we will present them in matrix form as folows:

$$\begin{bmatrix}\beta_3(1)\\ \beta_3(2)\end{bmatrix}=\begin{bmatrix}1\\ 1\end{bmatrix} \tag{1.20}$$

$$\begin{bmatrix}\beta_2(1)\\ \beta_2(2)\end{bmatrix}=\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1c} & 0\\ 0 & o_{2c}\end{bmatrix}\begin{bmatrix}\beta_3(1)\\ \beta_3(2)\end{bmatrix} \tag{1.21}$$

$$\begin{bmatrix}\beta_1(1)\\ \beta_1(2)\end{bmatrix}=\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1b} & 0\\ 0 & o_{2b}\end{bmatrix}\begin{bmatrix}\beta_2(1)\\ \beta_2(2)\end{bmatrix} \tag{1.22}$$

$$P(V\,|\,\lambda)=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix}\begin{bmatrix}o_{1a} & 0\\ 0 & o_{2a}\end{bmatrix}\begin{bmatrix}\beta_1(1)\\ \beta_1(2)\end{bmatrix} \tag{1.23}$$

By using folowing substitutions:

$$\beta_3=\begin{bmatrix}\beta_3(1)\\ \beta_3(2)\end{bmatrix}\quad,\quad \beta_2=\begin{bmatrix}\beta_2(1)\\ \beta_2(2)\end{bmatrix}\quad,\quad \beta_1=\begin{bmatrix}\beta_1(1)\\ \beta_1(2)\end{bmatrix} \tag{1.24}$$

$$O_3=\begin{bmatrix}o_{1c} & 0\\ 0 & o_{2c}\end{bmatrix}\quad,\quad O_2=\begin{bmatrix}o_{1b} & 0\\ 0 & o_{2b}\end{bmatrix}\quad,\quad O_1=\begin{bmatrix}o_{1a} & 0\\ 0 & o_{2a}\end{bmatrix}$$

$$\pi=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix}\quad,\quad \vec{I}_2=\begin{bmatrix}1\\ 1\end{bmatrix} \tag{1.25}$$

equations (1.20) to (1.23) can be shown like this:

$$\beta_3=\vec{I}_2 \tag{1.26}$$

$$\beta_2=SO_3\beta_3 \tag{1.27}$$

$$\beta_1=SO_2\beta_2 \tag{1.28}$$

$$P(V\,|\,\lambda)=\pi O_1\beta_1 \tag{1.29}$$

By inserting equations (1.26) till (1.28) into (1.29) we get:

$$P(V\,|\,\lambda)=\pi O_1\beta_1=\pi O_1 SO_2\beta_2=\pi O_1 SO_2 SO_3\beta_3$$

$$P(V\,|\,\lambda)=\pi O_1 \underbrace{SO_2 \underbrace{SO_3 \underbrace{\vec{I}_2}_{\beta_3}}_{\beta_2}}_{\beta_1} \tag{1.30}$$

Procedure is called backward becuase equations (1.26) till (1.28) define that equation (1.30) is calculated from right to left that is in the oposite directions from how state-sequence is created

- Now that we have solved problem 1 for example defined with (1.3) we will try to use this result to try to find procedure for solving problem 1 for general HMM defined with (1.5). Formulas (1.26) till (1.29) show a pattern which can be used to rewrite these formulas in folowing way:

$$\beta_t=\begin{bmatrix}\beta_t(1)\\ \vdots\\ \beta_t(T)\end{bmatrix} \tag{1.31}$$

$$O_t=\begin{bmatrix}o_{1o_t} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & o_{No_t}\end{bmatrix} \quad , \quad \beta_T=\vec{I}_N \quad , \quad \beta_{t-1}=SO_t\beta_t \quad , \quad 1\le t\le T \quad , \quad \vec{I}_N=\begin{bmatrix}1\\ \vdots\\ 1\end{bmatrix}_N \tag{1.32}$$

$$P(V\,|\,\lambda)=\pi O_1\beta_1 \tag{1.33}$$

Formula (1.30) also clearly shows a pattern which can be used to solve problem 1 for any HMM with general number of states, output variables and number of variables in output-sequence, and for such case formula (1.30) is transformed into:

$$P(V\,|\,\lambda)=\pi O_1 SO_2\cdots SO_T\vec{I}_N \tag{1.34}$$

It is very important to keep in mind that at this point, general procedures for solving problem 1, defined with (1.32), (1.33) and (1.34) are just an assumption which still has to be mathematicly proven.

From (1.32) it is seen that each β can be defined as folows:

$$\beta_T=\vec{I}_N \tag{1.35}$$

$$\beta_t=SO_{t+1}\ldots SO_T\,\vec{I}_N \quad , \quad T\text{-}1\ge t\ge 2 \tag{1.36}$$

$$\beta_1=SO_2\ldots SO_T\,\vec{I}_N \tag{1.37}$$

- As defined in (1.32) β has folowing meaning:

$$\beta_t(s)=P(o_{t+1},o_{t+2},\ldots,o_T|s_t=s,\lambda) \tag{1.38}$$

where: $\beta_t(s)$ – probability of partial observation sequence from t to T, with given state 's' at time 't' and HMM model λ.

In other words $\beta_t(s)$ is probability that at time 't' model will be in state 's', and that from step 't' till step 'T' model produced output varaibles which are the same as the ones from desired output sequence.

We shall now show that (1.38) really is valid for $\beta_2(s)$ defined with (1.15) and (1.16). To do so we will use variables defined with (1.73). Equations (1.15) and (1.16) can be presented as singal formula like folows:

$$\beta_2(s)=s_{s1}o_{1o_3}+s_{s2}o_{2o_3} \tag{1.39}$$

Inserting (1.6) into (1.39) we get:

$$\beta_2(s)=P(s_3=1\,|\,s_2=s,\lambda)\cdot P(o_3\,|\,s_3=2,\lambda)+P(s_3=2\,|\,s_2=s,\lambda)\cdot P(o_3\,|\,s_3=2,\lambda) \tag{1.40}$$

$$\beta_2(s)=P(s_3=1\,|\,s_2=s,\lambda)\cdot P(o_3\,|\,s_3=2,s_2=s\lambda)+P(s_3=2\,|\,s_2=s,\lambda)\cdot P(o_3\,|\,s_3=2,s_2=s,\lambda) \tag{1.41}$$

Using (1.73), (1.41) can be shown like folows:

$$\beta_2(s)=\frac{N(s_3=1,s_2=s,\lambda)}{N(s_2=s,\lambda)}\frac{N(o_3,s_3=2,s_2=s,\lambda)}{N(s_3=1,s_2=s,\lambda)}+\frac{N(s_3=2,s_2=s,\lambda)}{N(s_2=s,\lambda)}\frac{(o_3,s_3=2,s_2=s,\lambda)}{N(o_3,s_3=2,s_2=s,\lambda)} \tag{1.42}$$

$$\beta_2(s) = \frac{N(o_3, s_3 = 1, s_2 = s, \lambda)}{N(s_2 = s, \lambda)} + \frac{N(o_3, s_3 = 2, s_2 = s, \lambda)}{N(s_2 = s, \lambda)}$$

$$\beta_2(s) = \frac{N(o_3, s_3 = 1, s_2 = s, \lambda) + N(o_3, s_3 = 2, s_2 = s, \lambda)}{N(s_2 = s, \lambda)} \tag{1.43}$$

Numerator of (1.43) can be rewritten like folows:

$$\beta_2(s) = \frac{N(o_3 = c, s_2 = s, \lambda)}{N(s_2 = 1, \lambda)} \tag{1.44}$$

Using (1.43), (1.44) can be presented as folows:

$$\beta_2(s) = P(o_3 \mid s_2 = s, \lambda) \tag{1.45}$$

Since (1.45) is consistent with (1.38) this proves that (1.45) is valid for $\beta_2(s)$. Using the same procedure it can be proven that (1.38) is also valid for all other β parameters.

1.1.1.3. Forward calculation

- With Forward procedure we will try to use minimal number of operations to calculate the sum of all rows in the table 1.2. We will start our explanation of Forward-calculation by deriving this procedure for example defined by (1.3). First we reorange table 1.2 in the way that we start rolling state values from right to left, after which we get:

	$\alpha_1(1)$	$\alpha_2(1)$	$\alpha_3(1)$
111	$\boxed{\pi_1 o_{1a}}\cdot$	$s_{11}o_{1b}\cdot$	$s_{11}o_{1c}$
211	$\boxed{\pi_2 o_{2a}}\cdot$	$s_{21}o_{1b}\cdot$	$s_{11}o_{1c}$
121	$\pi_1 o_{1a}\cdot$	$s_{12}o_{2b}\cdot$	$s_{21}o_{1c}$
221	$\pi_2 o_{2a}\cdot$	$s_{22}o_{2b}\cdot$	$s_{21}o_{1c}$
112	$\pi_1 o_{1a}\cdot$	$s_{11}o_{1b}\cdot$	$s_{12}o_{2c}$
212	$\pi_2 o_{2a}\cdot$	$s_{21}o_{1b}\cdot$	$s_{12}o_{2c}$
122	$\pi_1 o_{1a}\cdot$	$s_{12}o_{2b}\cdot$	$s_{22}o_{2c}$
222	$\pi_2 o_{2a}\cdot$	$s_{22}o_{2b}\cdot$	$s_{22}o_{2c}$

Tab.1.4 Probability for each state-sequence that model will produce output-sequence O=(a,b,c).

First we will define α variables which will help us to present the procedure in compact way.

Lets give the sum of all parts inside upper full square symbol $\alpha_1(1)$: $\alpha_1(1)= \pi_1 o_{1a}$ (1.46)
Lets give the sum of all parts inside lower full square symbol $\alpha_1(2)$: $\alpha_1(2)= \pi_2 o_{2a}$ (1.47)

Lets give the sum of all parts inside upper doted square symbol $\alpha_2(1)$: $\alpha_2(1)=\alpha_1(1)s_{11}o_{1b}+\alpha_1(2)s_{21}o_{1b}$ (1.48)
Lets give the sum of all parts inside lower doted square symbol $\alpha_2(2)$: $\alpha_2(2)=\alpha_1(1)s_{12}o_{2b}+\alpha_1(2)s_{22}o_{2b}$ (1.49)

Lets give the sum of all parts inside upper dashed symbol $\alpha_3(1)$: $\alpha_3(1)=\alpha_2(1)s_{11}o_{1c}+\alpha_2(2)s_{21}o_{1c}$ (1.50)
Lets give the sum of all parts inside lower dashed symbol $\alpha_3(2)$: $\alpha_3(2)=\alpha_2(1)s_{12}o_{2c}+\alpha_2(2)s_{22}o_{2c}$ (1.51)

Total sum of all rows in table 1.4 can be calculated as: $P(V|\lambda)=\alpha_3(1)+\alpha_3(2)$ (1.52)

In order to simplifie formulas (1.46) till (1.52) we will present them in matrix form as folows:

$$\begin{bmatrix}\alpha_1(1) & \alpha_1(2)\end{bmatrix}=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix}\begin{bmatrix}o_{1a} & 0\\ 0 & o_{2a}\end{bmatrix} \qquad (1.53)$$

$$\begin{bmatrix}\alpha_2(1) & \alpha_2(2)\end{bmatrix}=\begin{bmatrix}\alpha_1(1) & \alpha_1(2)\end{bmatrix}\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1b} & 0\\ 0 & o_{2b}\end{bmatrix} \qquad (1.54)$$

$$\begin{bmatrix}\alpha_3(1) & \alpha_3(2)\end{bmatrix}=\begin{bmatrix}\alpha_2(1) & \alpha_2(2)\end{bmatrix}\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1c} & 0\\ 0 & o_{2c}\end{bmatrix} \qquad (1.55)$$

$$P(V\,|\,\lambda)=\begin{bmatrix}\alpha_3(1) & \alpha_3(2)\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix} \qquad (1.56)$$

By using folowing substitutions:

$$\alpha_1=\begin{bmatrix}\alpha_1(1) & \alpha_1(2)\end{bmatrix} \quad , \quad \alpha_2=\begin{bmatrix}\alpha_2(1) & \alpha_2(2)\end{bmatrix} \quad , \quad \alpha_3=\begin{bmatrix}\alpha_3(1) & \alpha_3(2)\end{bmatrix} \qquad (1.57)$$

$$O_1=\begin{bmatrix}o_{1a} & 0\\ 0 & o_{2a}\end{bmatrix} \quad , \quad O_2=\begin{bmatrix}o_{1b} & 0\\ 0 & o_{2b}\end{bmatrix} \quad , \quad O_3=\begin{bmatrix}o_{1c} & 0\\ 0 & o_{2c}\end{bmatrix} \qquad (1.58)$$

$$\pi=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix} \quad , \quad \vec{I}_2=\begin{bmatrix}1\\ 1\end{bmatrix} \qquad (1.59)$$

equations (1.53) till (1.56) can be shown like this:

$$\alpha_1=\pi O_1 \qquad (1.60)$$

$$\alpha_2=\alpha_1 S O_2 \qquad (1.61)$$

$$\alpha_3=\alpha_2 S O_3 \qquad (1.62)$$

$$P(V|\lambda)=\alpha_3\vec{I}_2 \tag{1.63}$$

By inserting equations (1.60) till (1.62) into (1.63) we can write the result in the folowing way:

$$P(V|\lambda)=\alpha_3\vec{I}_2=\alpha_2 SO_3\vec{I}_2=\alpha_1 SO_2 SO_3\vec{I}_2$$

$$P(V|\lambda)=\underbrace{\underbrace{\underbrace{\pi O_1}_{\alpha_1} SO_2}_{\alpha_2} SO_3}_{\alpha_3}\vec{I}_2 \tag{1.64}$$

Procedure is called forward becuase equations (1.26) till (1.28) define that equation (1.30) is calculated from left to right that is in the same directions from how state-sequence is created.

Equation (1.64) is the same as equation (1.30) aquired by backward procedure. This shows that forward and backward procedures are generaly the same. The only difference is in the direction in which formulas (1.30) and (1.64) are calculated.

- Now that we have solved problem 1 for example defined with (1.3) we will try to use this result to try to find procedure for solving problem 1 for general HMM defined with (1.5). Formulas (1.60) till (1.63) show a pattern which can be used to rewrite these formulas in folowing way:

$$\alpha_t=\left[\alpha_t(1)\ \cdots\ \alpha_t(T)\right] \tag{1.65}$$

$$O_t=\begin{bmatrix} o_{1o_t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & o_{No_t}\end{bmatrix} \quad , \quad \alpha_1=\pi O_1 \quad , \quad \alpha_t=\alpha_{t-1}^T SO_t \quad , \quad 2\le t\le T \quad , \quad \vec{I}_N=\begin{bmatrix}1\\ \vdots \\ 1\end{bmatrix}_N \tag{1.66}$$

$$P(V|\lambda)=\alpha_T\vec{I}_N \tag{1.67}$$

Formula (1.64) also clearly shows a pattern which can be used to solve problem 1 for any HMM with general number of states, output variables and number of variables in output-sequence, and for such case formula (1.64) is transformed into:

$$P(V|\lambda)=\pi O_1 SO_2\cdots SO_T\vec{I}_N \tag{1.68}$$

It is very important to keep in mind that at this point, general procedures for solving problem 1, defined with (1.66), (1.67) and (1.69) are just an assumption which still has to be mathematicly proven.

From (1.66) it is seen that each α can be defined as folows:

$$\alpha_1=\pi O_1 \tag{1.70}$$

$$\alpha_t=\pi O_1 SO_2 ... SO_T \quad , \quad 2\le t\le T \tag{1.71}$$

- As defined in (1.66) α has folowing meaning:

$$\alpha_t(s)=P(o_1,o_2,...,o_t,s_t=s|\lambda) \tag{1.72}$$

where: $\alpha_t(s)$ – probability of partial observation sequence from 1 to t, with given state 's' at time 't' and HMM model λ.

In other words $\alpha_t(s)$ is probability that at time 't' model will be in state 's', and that till including that time step model produced output varaibles which are the same as the one from desired output sequence.

We shall now show that (1.66) really is valid for $\alpha_1(s)$. To do so we will first introduce folowing variables:

$$P(A_1,...,A_p|B_1,...,B_q)=\frac{N(A_1,...,A_p,B_1,...,B_q)}{N(B_1,...,B_q)} \tag{1.73}$$

where: $P(A_1,...,A_p|B_1,...,B_q)=$ probability that object will satisfy conditions $A_1,...,A_p$ if conditions $B_1,...,B_q$ are already satisfied

$N(A_1,...,A_p,B_1,...,B_q)=$ probability that object will saisfy all of the conditions $A_1,...,A_p,B_1,...,B_q$

Inserting (1.6) into (1.46) or (1.47) we get:

$$\alpha_1(s) = \pi_s o_{so_1} = P(s_1 = s \mid \lambda) \cdot P(o_1 \mid s_1 = s, \lambda) \tag{1.74}$$

Using (1.73) we can transform (1.74) into:

$$\alpha_1(s) = \frac{N(s_1 = s, \lambda)}{N(\lambda)} \frac{N(o_1, s_1 = s, \lambda)}{N(s_1 = s, \lambda)} = \frac{N(o_1, s_1 = 1, \lambda)}{N(\lambda)} \tag{1.75}$$

Using (1.73) once again, but in the oposite direction, we can transform (1.75) into:

$$\alpha_1(s) = P(o_1, s_1 = 1 \mid \lambda) \tag{1.76}$$

Since (1.76) is consistent with (1.72) this proves that (1.72) is valid for $\alpha_1(s)$. Using the same procedure it can be proven that (1.72) is also valid for all other α parameters.

1.1.1.4. Proving validity of forward and backward equations

In this chapter we will prove that by formula (1.30), which is the same as formula (1.68), we can solve problem one for any HMM. We write this formula once again:

$$\text{result} = \pi O_1 S O_2 \cdots S O_T \vec{I}_N \tag{1.77}$$

By proving formula (1.77) at the same time we are also proving Forward procedure defined with (1.66) and (1.67) and Backward procedure defined with (1.32), (1.33) because (1.77) is derived from each of them.

- First we will solve problem one for the case where output-sequence has only one element, which means that also state-sequence has only one element which, is the starting state. Equation (1.77) for such case looks like this:

$$\text{result} = \pi O_1 \vec{I}_N \tag{1.78}$$

Using O_1 from (1.66), expression πO_1 from (1.78) can be calculated as folows:

$$\pi O_1 = \begin{bmatrix} \pi_1 & \cdots & \pi_N \end{bmatrix} \begin{bmatrix} o_{1o_1} & 0 & \cdots & 0 \\ 0 & o_{2o_1} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & o_{No_1} \end{bmatrix} \tag{1.79}$$

After matrix multiplication, equation (1.79) results in:

$$\pi O_1 = \begin{bmatrix} \pi_1 o_{1o_1} & \cdots & \pi_N o_{No_1} \end{bmatrix} \tag{1.80}$$

By introducing folowing variables:

$$e_{i_1}^{(1)} = \pi_{i_1} o_{i_1 o_1} \quad , \quad i_1=1,2,...,N \tag{1.81}$$

equation (1.80) can be rewritten as:

$$\pi O_1 = \begin{bmatrix} e_1^{(1)} & \cdots & e_N^{(1)} \end{bmatrix} \tag{1.82}$$

Inserting (1.82) and $\vec{I}$ from (1.66) into (1.78) we get:

$$\text{result} = \begin{bmatrix} e_1^{(1)} & \cdots & e_N^{(1)} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{Nx1} \tag{1.83}$$

This equation will be crutial in proving (1.77) because this structure will repeat itself as the state sequence continues to grow. In order to later show this repetition we will calculate (1.83) which gices us:

$$\text{result} = e_1^{(1)} + ... + e_N^{(1)} \tag{1.84}$$

$$\text{result} = \sum_{i_1=1}^{N} e_{i_1}^{(1)} \tag{1.85}$$

- As a second step we will solve problem one for the case where output-sequence has only two elements, which means that also state-sequence has only two elements. Equation (1.77) for such case looks like this:

$$\text{result} = \pi O_1 S O_2 \vec{I} \tag{1.86}$$

Inserting O_t from (1.66) and S from (1.2) into (1.86) we get:

$$S O_2 = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & & s_{2N} \\ \vdots & & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{NN} \end{bmatrix} \begin{bmatrix} o_{1o_2} & 0 & \cdots & 0 \\ 0 & o_{2o_2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & o_{No_2} \end{bmatrix} = \begin{bmatrix} s_{11}o_{1o_2} & s_{12}o_{1o_2} & \cdots & s_{1N}o_{1o_2} \\ s_{21}o_{2o_2} & s_{22}o_{2o_2} & & s_{2N}o_{2o_2} \\ \vdots & & \ddots & \vdots \\ s_{N1}o_{No_2} & s_{N2}o_{No_2} & \cdots & s_{NN}o_{No_2} \end{bmatrix} \tag{1.87}$$

Using (1.87) and (1.82) first part of equation (1.86) can be written as folows:

$$\pi O_1 SO_2 = \begin{bmatrix} e_1^{(1)} & \cdots & e_N^{(1)} \end{bmatrix} \begin{bmatrix} s_{11}o_{1o_2} & s_{12}o_{1o_2} & \cdots & s_{1N}o_{1o_2} \\ s_{21}o_{2o_2} & s_{22}o_{2o_2} & & s_{2N}o_{2o_2} \\ \vdots & & \ddots & \vdots \\ s_{N1}o_{No_2} & s_{N2}o_{No_2} & \cdots & s_{NN}o_{No_2} \end{bmatrix} \tag{1.88}$$

By introducing folowing variables:

$$E_{i_2 i_1}^{(2)} = s_{i_1 i_2} o_{o_2 i_2}\ , \qquad i_2=1,2,...,N \tag{1.89}$$

equation (1.88) can be written as:

$$\pi O_1 SO_2 = \begin{bmatrix} e_1^{(1)} & \cdots & e_N^{(1)} \end{bmatrix} \begin{bmatrix} E_{11}^{(2)} & E_{12}^{(2)} & \cdots & E_{1N}^{(2)} \\ E_{21}^{(2)} & E_{22}^{(2)} & & E_{2N}^{(2)} \\ \vdots & & \ddots & \vdots \\ E_{N1}^{(2)} & E_{N2}^{(2)} & \cdots & E_{NN}^{(2)} \end{bmatrix}$$

$$\pi O_1 SO_2 = \begin{bmatrix} \sum_{i_1=1}^{N} e_{i_1}^{(1)} E_{i_1 1}^{(2)} & \cdots & \sum_{i_1=1}^{N} e_{i_1}^{(1)} E_{i_1 N}^{(2)} \end{bmatrix} \tag{1.90}$$

Introducing new set of vairables:

$$e_{i_2}^{(2)} = \sum_{i_1=1}^{N} e_{i_1}^{(1)} E_{i_1 i_2}^{(2)}\ , \qquad i_2=1,2,...,N \tag{1.91}$$

equation (1.90) can be written as folows:

$$\pi O_1 SO_2 = \begin{bmatrix} e_1^{(2)} & \cdots & e_N^{(2)} \end{bmatrix} \tag{1.92}$$

Inserting (1.92) and $\vec{I}$ from (1.66) into (1.86) we get:

$$\text{result} = \begin{bmatrix} e_1^{(2)} & \cdots & e_N^{(2)} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{Nx1} \tag{1.93}$$

Here it should be noted that equation (1.93) has the same structure as equation (1.83). In a while we will use this information to end this proof. By calculating (1.93) we get:

$$\text{result} = e_1^{(2)} + ... + e_N^{(2)} \tag{1.94}$$

$$\text{result} = \sum_{i_2=1}^{N} e_{i_2}^{(2)} \tag{1.95}$$

Inserting (1.91) into (1.95) we get:

$$\text{result} = \sum_{i_2=1}^{N} \sum_{i_1=1}^{N} e_{i_1}^{(1)} E_{i_1 i_2}^{(2)} \tag{1.96}$$

Inserting (1.81) and i2=1,2,...,N (1.89) into (1.96) we get :

$$\text{result} = \sum_{i_2=1}^{N} \sum_{i_1=1}^{N} \pi_{i_1} o_{i_1 o_1} s_{i_1 i_2} o_{i_2 o_2} \tag{1.97}$$

At this point it is be helpfull to observe similareties between (1.97) and (1.11). If we were to use above procedure to take into account next obeservable ouptut we would then get formula which is exactlu the same as (1.11) which proofs that Backward and Forward procedures really to give the same result as brute force calculation which definition for solving problem 1. The key point of the proof was to show that at each new iteration, which is equal to expanding (1.77) with next SO_t value, we are able to transorm the equation so that it has structure as in equations (1.83) and (1.93). The only thing that changes with each iteration are newly introduced indexes which for geral structure results in formula (1.12).

1.1.2. Using α and β together

If we take N copies of HMM and execute these copies, from (1.72) and (1.38) we can conclude folowing:

$N\alpha_t(s)$ = number of HMMs which on step 't' were in state 's' and till including this step produced valid output-variables. (1.98)

$N\beta_t(s)$ = number of HMMs which on step 't' were in state 's' and after that step produced valid output-variables. (1.99)

Since (1.99) is valid for any amount of HMMs, if we use it on (1.98) we get:

$[N\alpha_t(s)]\beta_t(s)$ = number of HMMs which produced valid output-sequence and on step 't' were in state 's'. (1.100)

Equation (1.98) could also be extenden in the folowing way with the folowing meaning:

$N\alpha_t(s_1)s_{s_1s_2}o_{s_2o_{t+1}}$ = Number of HMMs which on step 't' were in state 's_1', on step 't+1' were in state 's_2' and till including step 't+1' produced valid output-variables. (1.101)

If we now use (1.99) on (1.101) we would get:

$[N\alpha_t(s_1)s_{s_1s_2}o_{s_2o_{t+1}}]\beta_{t+1}(s_2) =$ Number of HMMs which produced valid ouput-sequence and from step 't' to step 't+1' switched from state 's_1' to state 's_2'. (1.102)

Equations **Error! Not a valid link.** till (1.102) show how α and β could be used in defining various combinations of state and output sequences which will be used later in couping with problem 3.

If we use '?' character for replacing single output-variable or state, then for the example of four states in state-sequence, $\alpha_2(1)\beta_1(1)$ state-sequence combinations could be symbolicly presented in folowing way:

$$\alpha_2(1) \rightarrow \begin{array}{cccc} o_1 & o_2 & ? & ? \\ ? & 1 & ? & ? \end{array}$$

$$\beta_2(1) \rightarrow \begin{array}{cccc} ? & ? & o_3 & o_4 \\ ? & 1 & ? & ? \end{array}$$

$$\alpha_2(1)\beta_1(1) \rightarrow \begin{array}{cccc} o_1 & o_2 & o_3 & o_4 \\ ? & 1 & ? & ? \end{array}$$

Using the same method, $\alpha_2(1)s_{15}o_{5o_3}\beta_3(5)$ could be presented as folows:

$$\alpha_2(1) \rightarrow \begin{array}{cccc} o_1 & o_2 & ? & ? \\ ? & 1 & ? & ? \end{array}$$

$$\beta_3(5) \rightarrow \begin{array}{cccc} ? & ? & ? & o_4 \\ ? & ? & 5 & ? \end{array}$$

$$\alpha_2(1)s_{15}o_{5o_3}\beta_3(5) \rightarrow \begin{array}{cccc} o_1 & o_2 & o_3 & o_4 \\ ? & 1 & 5 & ? \end{array}$$

1.1.2.1. Fast full table calculation

- Goal of this method is to calculate the value of each row from table 1.2 in the fastest possible way by executing minimum number of calculation. That menas that this method does the same job as brute-force method, but faster. This method can also be used to calculate total sum of all rows, but it will do this slower then Backward and Forward calculation. On the other hand Backward and Forward calculation do not give us information about the value of each row.

- Definition of black dot operator

Method will use a specail matrix operation called "black dot operator" which is defined as folows:

$$A \bullet B = R$$

Take any element A_{rc} from matrix A. Multiply it with all elements which are in row "c" in B matrix going from left to right. Place the results in any of the remaining rows of the resulting matrix R going again from left to right. Repeat the procedure until all elements from matrix A are used.

Position yourself on the first row of the left matrix and take all elements going from left to right. Then go to the next row and so on. Fill rows of the resulting matrix as you go from top to the bottom. This procedure is ilustrated by the below example:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \bullet \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{12}B_{21} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \\ A_{22}B_{21} & A_{22}B_{22} \\ A_{31}B_{11} & A_{31}B_{12} \\ A_{32}B_{21} & A_{32}B_{22} \\ A_{41}B_{11} & A_{41}B_{12} \\ A_{42}B_{21} & A_{42}B_{22} \end{bmatrix}$$

- Method will be explained on the HMM example defined with (1.1) where we will gradualy build possible resulting test-sequnces.

- After first output-variable is observed table 1.2 . would look like this:

1	$\pi_1 o_{1a}$
2	$\pi_2 o_{2a}$

Tab.1.5 Probabilities after one output variable.

For calculating table 1.5 no special procedure is requiered because table is to simple. The values of the rows can simply be presented in the folowing matrix:

$$\begin{bmatrix} \pi_1 o_{a1} & \pi_2 o_{a2} \end{bmatrix} \tag{1.103}$$

Each of the elements in the matrix (1.103) present one of the possbible states which can be shown like this:

$$\begin{bmatrix} 1 & 2 \end{bmatrix}$$

- After second output-variable is observed table 1.2 . would look like this:

11	$\pi_1 o_{1a} \cdot s_{11} o_{1b}$
11	$\pi_1 o_{1a} \cdot s_{11} o_{1b}$
12	$\pi_1 o_{1a} \cdot s_{12} o_{2b}$
12	$\pi_1 o_{1a} \cdot s_{12} o_{2b}$

Tab.1.6 Probabilities after two output variables.

In this case calculating values of each row in table 1.6 can be made optimal using folowing logic. After first observed output-variable, model is either in state 1 or in state 2. In the next step each of these two states can change to two different states resulting in four different combinations presented in table 1.6. Value of each row in the table 1.6 can be calculated using black-dot matrix operation as folows:

$$\begin{bmatrix} \pi_1 o_{a1} & \pi_2 o_{a2} \end{bmatrix} \bullet \begin{bmatrix} s_{11} o_{1b} & s_{12} o_{2b} \\ s_{21} o_{1b} & s_{22} o_{2b} \end{bmatrix} = \begin{bmatrix} \pi_1 o_{1a} s_{11} o_{1b} & \pi_1 o_{1a} s_{12} o_{2b} \\ \pi_2 o_{2a} s_{21} o_{1b} & \pi_2 o_{2a} s_{22} o_{2b} \end{bmatrix} \tag{1.104}$$

Equation (1.104) caculates values of the rows in table 1.6 optimaly because none of the operations is being repeated. For instance, to calculate values of first two rows we would have to multiplie π_1 and o_{1a} twice, because this expression is presented in both rows. What equation (1.104) does is that it multiplies π_1 and o_{1a} only once and then stores that result in left matrix to be used as many times as needed, cutting down on needed numerical operations. It should also be noted that the left matrix was already calculated in the previous step in equation (1.103). This is very important feature of this procedure. As the state-sequence grows calculactions made in previous step is used to calculate results of the next step calculatin only new necesseary steps.

Each of the elements in the resulting matrix of equation (1.104) are connected to some state-sequence which can be presented as foows:

$$\begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$$

- After third output-variable is observed table 1.2 . would look like this:

111	$\pi_1 o_{1a} \cdot s_{11} o_{1b} \cdot s_{11} o_{1c}$
112	$\pi_1 o_{1a} \cdot s_{11} o_{1b} \cdot s_{12} o_{2c}$
121	$\pi_1 o_{1a} \cdot s_{12} o_{2b} \cdot s_{21} o_{1c}$
122	$\pi_1 o_{1a} \cdot s_{12} o_{2b} \cdot s_{22} o_{2c}$
211	$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{11} o_{1c}$
212	$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{12} o_{2c}$
221	$\pi_2 o_{2a} \cdot s_{22} o_{2b} \cdot s_{21} o_{1c}$
222	$\pi_2 o_{2a} \cdot s_{22} o_{2b} \cdot s_{22} o_{2c}$

Tab.1.7 Probabilities after three output variables.

In previous step we had four different combinations. In the next step each of them can go in two different states resulting in eight new different combination presented in table 1.7. To calculate values of the rows in table 1.7 folowing calculation can be made:

$$\begin{bmatrix} \pi_1 o_{1a} s_{11} o_{1b} & \pi_1 o_{1a} s_{12} o_{2b} \\ \pi_2 o_{2a} s_{21} o_{1b} & \pi_2 o_{2a} s_{22} o_{2b} \end{bmatrix} \bullet \begin{bmatrix} s_{11} o_{1c} & s_{12} o_{2c} \\ s_{21} o_{1c} & s_{22} o_{2c} \end{bmatrix} = \begin{bmatrix} \pi_1 o_{1a} s_{11} o_{1b} s_{11} o_{1c} & \pi_1 o_{1a} s_{11} o_{1b} s_{12} o_{2c} \\ \pi_1 o_{1a} s_{12} o_{2b} s_{21} o_{1c} & \pi_1 o_{1a} s_{12} o_{2b} s_{22} o_{2c} \\ \pi_2 o_{2a} s_{21} o_{1b} s_{11} o_{1c} & \pi_2 o_{2a} s_{21} o_{1b} s_{12} o_{2c} \\ \pi_2 o_{2a} s_{22} o_{2b} s_{21} o_{1c} & \pi_2 o_{2a} s_{22} o_{2b} s_{22} o_{2c} \end{bmatrix} \quad (1.105)$$

In equation (1.105) left matrix was already calculated in equation (1.104). Left matrix contains state sequences from the previous
step. First column contains state-sequences ending in state 1, second column contains state-sequences ending in state 2, and so on if we would have HMM with more then two possible states. Since elements of the left matrix which belong to first column are state sequences ending in state 1, they can only be combined with state transitions presented in the first row of the right matrix, because they present transitions from state 1. This is why the black-dot transformation was defined in the way defined at the begining of the chapter.

Each of the elements in the resulting matrix of equation (1.105) are connected to some state-sequence which can be presented as folows:

$$\begin{bmatrix} 111 & 112 \\ 121 & 122 \\ 211 & 212 \\ 221 & 222 \end{bmatrix}$$

If we give symbols γ_1,γ_2 and γ_3 to the matrices which are results of the equations (1.103) till (1.105), we could then write these equations as folows:

$$\gamma_1 = \pi O_1 \quad (1.106)$$

$$\gamma_2 = \gamma_1 \bullet (SO_2) \quad (1.107)$$

$$\gamma_3 = \gamma_2 \bullet (SO_3) \quad (1.108)$$

By inserting (1.106) into (1.107), and then inserting this result into (1.108) we get:

$$\gamma_3 = \gamma_2 \bullet (SO_3) = [\gamma_1 \bullet (SO_2)](SO_3)$$

$$\gamma_3 = [(\pi O_1) \bullet (SO_2)] \bullet (SO_3) \quad (1.109)$$

1.1.3. Problem 2

Solution to problem 2 is to find state-sequence which is most probable to give defined output-sequence, for instance (a,a,b,a,c). This means finding the row of table 1.2 which has the biggest value. This can be done using brute-force method or using fast full table method. Another way to do it is Viterbi algorithm which does this task with the least amount of calculations and it will be explained in the next chapter.

1.1.3.1. Viterbi algorithm

This algorithm is very similar with forward-algorithm explained while solving problem 1, because it also calculates and uses variables previously defined as α. To ilustrate this procedure folowing tabel will be used.

1111	$\pi_1 o_{1a}\cdot S_{11} o_{1b}\cdot S_{11} o_{1c}\cdot S_{11} o_{1d}$	1111		1111	
2111	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{11} o_{1d}$	2111	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{11} o_{1d}$	2111	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{11} o_{1d}$
1211	$\pi_1 o_{1a}\cdot S_{12} o_{2b}\cdot S_{21} o_{1c}\cdot S_{11} o_{1d}$	1211		1211	
2211	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{21} o_{1c}\cdot S_{11} o_{1d}$	2211	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{21} o_{1c}\cdot S_{11} o_{1d}$	2211	
1121	$\pi_1 o_{1a}\cdot S_{11} o_{1b}\cdot S_{12} o_{2c}\cdot S_{21} o_{1d}$	1121		1121	
2121	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{21} o_{1d}$	2121	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{21} o_{1d}$	2121	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{21} o_{1d}$
1221	$\pi_1 o_{1a}\cdot S_{12} o_{2b}\cdot S_{22} o_{2c}\cdot S_{21} o_{1d}$	1221		1221	
2221	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{22} o_{2c}\cdot S_{21} o_{1d}$	2221	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{22} o_{2c}\cdot S_{21} o_{1d}$	2221	
1112	$\pi_1 o_{1a}\cdot S_{11} o_{1b}\cdot S_{11} o_{1c}\cdot S_{12} o_{2d}$	1112		1112	
2112	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{12} o_{2d}$	2112	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{12} o_{2d}$	2112	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{11} o_{1c}\cdot S_{12} o_{2d}$
1212	$\pi_1 o_{1a}\cdot S_{12} o_{2b}\cdot S_{21} o_{1c}\cdot S_{12} o_{2d}$	1212		1212	
2212	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{21} o_{1c}\cdot S_{12} o_{2d}$	2212	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{21} o_{1c}\cdot S_{12} o_{2d}$	2212	
1122	$\pi_1 o_{1a}\cdot S_{11} o_{1b}\cdot S_{12} o_{2c}\cdot S_{22} o_{2d}$	1122		1122	
2122	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{22} o_{2d}$	2122	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{22} o_{2d}$	2122	$\pi_2 o_{2a}\cdot S_{21} o_{1b}\cdot S_{12} o_{2c}\cdot S_{22} o_{2d}$
1222	$\pi_1 o_{1a}\cdot S_{12} o_{2b}\cdot S_{22} o_{2c}\cdot S_{22} o_{2d}$	1222		1222	
2222	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{22} o_{2c}\cdot S_{22} o_{2d}$	2222	$\pi_2 o_{2a}\cdot S_{22} o_{2b}\cdot S_{22} o_{2c}\cdot S_{22} o_{2d}$	2222	
a)		b)		c)	

Tab.1.8 Probability for each state-sequence that model will produce output-sequence O=(a,b,c).

First we shall take a closer look at table 1.8 a). To find out which row between 1111 & 2111 has bigger value we only have to look at yellow parts since both of these sequences end with $o_{1b}\cdot s_{11}o_{1c}\cdot s_{11}o_{1d}$. This is the key point of Viterbi algorithm enableing ust to make our conclusions only with fraction of operations nedeed in brute force calculation. Going through all the rows in table 1.8a) we have to find pairs that have the same ending, like in the example with rows 1111 & 2111. It can be seen that first two rows have the same ending, then second two rows also and so on. So these pairs would be 1111&2111, 1211&2211, 1121&2121 and so on. All this pairs are shown in 1.8a) separated with the solid line.Now lets say that folowing two relations are valid:

$$\pi_1 o_{1a}\cdot S_{11} < \pi_2 o_{2a}\cdot S_{21} \tag{1.110}$$

$$\pi_1 o_{1a}\cdot S_{12} < \pi_2 o_{2a}\cdot S_{22} \tag{1.111}$$

Relation (1.110) gives us information that from all state-sequences which on the second place have state 1 only the ones starting with state 2 are possible winners.
Relation (1.111) gives us information that from all state-sequences which on the second place have state 2 only the ones starting with state 2 are possible winners.
This information needs to be saved so that we couls reconstruct state-sequence. For that reason folowing matrix is introduced:

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} & w_{14} \\ 0 & w_{22} & w_{23} & w_{24} \end{bmatrix} \tag{1.112}$$

where: W – matrix of winners
w_{sp} – from all test-sequences which on the 'p' place have state 's' only the ones which in place p-1 had state W_{sp} are possible winners.

Results from (1.110) and (1.111) can now be saved in matrix W, defined with (1.112), like this:

$$W = \begin{bmatrix} 0 & 2 & w_{13} & w_{14} \\ 0 & 2 & w_{23} & w_{24} \end{bmatrix} \tag{1.113}$$

If value of each row is symbolicly presented with its index, then from (1.110) folowing relations folow:

1111 < 2111 , 1121 < 2121 , 1112 < 2112 , 1122 < 2122

From (1.111) folowing relations folow:

1211 < 2211 , 1221 < 2221 , 1212 < 2212 , 1222 < 2222

In table 1.8 a) winner of each pair is given a square and these rows are then rewritten in table 1.8 b). Table 1.8 b) presents second step in which we again have to find pairs that end with the same variables. These pairs are 2111&2211, 2121&221, 2112&2212 and 2122&222 and are separated with solid line. To find rows with bigger value in this pairs, again only two relations have to be calculated, and we shall assume that they are valid as folows:

$$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{11} > \pi_2 o_{2a} \cdot s_{22} o_{2b} \cdot s_{21} \qquad (1.114)$$

$$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{12} > \pi_2 o_{2a} \cdot s_{22} o_{2b} \cdot s_{22} \qquad (1.115)$$

Relation (1.114) gives us information that from all state-sequences
which on the 3rd place have state 1 only the ones which on the 2nd place have state 1 are possible winners.

Relation (1.115) gives us information that from all state-sequences
which on the 3rd place have state 2 only the ones which on the 2nd place have state 1 are possible winners.

Results from (1.114) and (1.115) can now be saved in matrix W, defined with (1.112), like this:

$$W = \begin{bmatrix} 0 & 2 & 1 & w_{14} \\ 0 & 2 & 1 & w_{24} \end{bmatrix} \qquad (1.116)$$

From (1.114) folowing relations folow:

2111 > 2211 , 2112 > 2212

From (1.115) folowing relations folow:

2121 < 2221 , 2122 < 222

In table 1.8 b) winner of each pair is given a square and these rows are then rewritten in table 1.8 c). Table 1.8 c) presents third step in which we again have to find pairs that end with the same variables. These pairs are 2111&2121 and 2112&2122. To find rows with bigger value in this pairs, again only two relations have to be calculated, and we shall assume that they are valid as folows:

$$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{11} o_{1c} \cdot s_{11} < \pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{12} o_{2c} \cdot s_{21} \qquad (1.117)$$

$$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{11} o_{1c} \cdot s_{12} > \pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{12} o_{2c} \cdot s_{22} \qquad (1.118)$$

Relation (1.117) gives us information that from all state-sequences
which on the last 4th place have state 1 only the ones which on the 3rd place have state 2 are possible winners.

Relation (1.118) gives us information that from all state-sequences
which on the last 4th place have state 2 only the ones which on the 3nd place have state 1 are possible winners.

Results from (1.117) and (1.118) can now be saved in matrix W, defined with (1.112), like this:

$$W = \begin{bmatrix} 0 & 2 & 1 & 2 \\ 0 & 2 & 1 & 1 \end{bmatrix} \qquad (1.119)$$

From (1.117) folowing relations folow:

2111 < 2121

From (1.118) folowing relations folow:

2111 < 2122

In table 1.8 c) winner of each pair is given a square. This means that after this third step we are left with only two which values might be solutions to Viterbi algorithm. These rows are:

2121	$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{12} o_{2c} \cdot s_{21} o_{1d}$
2112	$\pi_2 o_{2a} \cdot s_{21} o_{1b} \cdot s_{11} o_{1c} \cdot s_{12} o_{2d}$

Since this two expressions are complitly different they all parts of them have to computed to see which one has greater value. We shall assume that folowing relation is valid:

$$\pi_2 o_{2a}\cdot s_{21} o_{1b}\cdot s_{12} o_{2c}\cdot s_{21} o_{1d} > \pi_2 o_{2a}\cdot s_{21} o_{1b}\cdot s_{11} o_{1c}\cdot s_{12} o_{2d} \tag{1.120}$$

Relation (1.120) gives us information that from all state-sequences only the ones which end in state 1 are possible winners. This information is not entered into matrix W, instead this information is used to reconstruct winner state-sequence from the data entered in matrix W. Since Viterbi algorithm concluded that the last state of winner state-sequence is 1, we only have to take the first row from matrix W to get the full state-sequce, which in this case would be 2121.

- The procedure can also be described with folowing equations:

$\delta_1(1)=\pi_1 o_{a1}$
$\delta_1(2)=\pi_2 o_{a2}$

$\delta_2(1)=\max[\delta_1(1)\cdot s_{11} o_{1b},\ \ \delta_1(2)\cdot s_{21} o_{1b}]=\max[\pi_1 o_{1a}\cdot s_{11} o_{1b},\ \pi_2 o_{2a}\cdot s_{21} o_{1b}]$
$\delta_2(2)=\max[\delta_1(1)\cdot s_{12} o_{b2},\ \ \delta_1(2)\cdot s_{22} o_{2b}]=\max[\pi_1 o_{1a}\cdot s_{12} o_{2b},\ \pi_2 o_{a2}\cdot s_{22} o_{2b}]$

$\delta_3(1)=\max[\delta_2(1)\cdot s_{11} o_{1c},\ \ \delta_2(2)\cdot s_{21} o_{1c}]=\max[\pi_2 o_{2a}\cdot s_{21} o_{1b}\cdot s_{11} o_{1c},\ \pi_2 o_{2a} s_{22} o_{2b}\cdot s_{21} o_{1c}]$
$\delta_3(2)=\max[\delta_2(1)\cdot s_{12} o_{2c},\ \ \delta_2(2)\cdot s_{22} o_{2c}]=\max[\pi_2 o_{2a}\cdot s_{21} o_{1b}\cdot s_{12} o_{2c},\ \pi_2 o_{2a} s_{22} o_{2b}\cdot s_{22} o_{2c}]$

$\text{result} = \max[\delta_3(1),\ \delta_3(2)]=\max[\pi_2 o_{2a}\cdot s_{21} o_{1b}\cdot s_{11} o_{1c},\ \pi_2 o_{2a} s_{22} o_{2b}\cdot s_{22} o_{2c}]$

- The above formulas can be presented with folowing matrix equations where max_instead_sum means that after you multiplie cells you don't add them but you take result which was maximum.

$$\begin{bmatrix}\delta_1(1) & \delta_1(2)\end{bmatrix}=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix}\begin{bmatrix}o_{1a} & 0\\ 0 & o_{2a}\end{bmatrix} \tag{1.121}$$

$$\begin{bmatrix}\delta_2(1) & \delta_2(2)\end{bmatrix}=\begin{bmatrix}\delta_1(1) & \delta_1(2)\end{bmatrix}\oplus\left(\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1b} & 0\\ 0 & o_{2b}\end{bmatrix}\right) \tag{1.122}$$

$$\begin{bmatrix}\delta_3(1) & \delta_3(2)\end{bmatrix}=\begin{bmatrix}\delta_2(1) & \delta_2(2)\end{bmatrix}\oplus\left(\begin{bmatrix}s_{11} & s_{12}\\ s_{21} & s_{22}\end{bmatrix}\begin{bmatrix}o_{1c} & 0\\ 0 & o_{2c}\end{bmatrix}\right) \tag{1.123}$$

$$\text{result}=\begin{bmatrix}\delta_3(1) & \delta_3(2)\end{bmatrix}\oplus\begin{bmatrix}1\\ 1\\ 1\end{bmatrix} \tag{1.124}$$

- By using folowing substitutions:

$$\delta_1=\begin{bmatrix}\delta_1(1) & \delta_1(2)\end{bmatrix}\quad ;\quad \delta_2=\begin{bmatrix}\delta_2(1) & \delta_2(2)\end{bmatrix}\quad ;\quad \delta_3=\begin{bmatrix}\delta_3(1) & \delta_3(2)\end{bmatrix} \tag{1.125}$$

$$O_1=\begin{bmatrix}o_{a1} & 0\\ 0 & o_{a2}\end{bmatrix}\quad ,\quad O_2=\begin{bmatrix}o_{b1} & 0\\ 0 & o_{b2}\end{bmatrix}\quad ,\quad O_3=\begin{bmatrix}o_{c1} & 0\\ 0 & o_{c2}\end{bmatrix} \tag{1.126}$$

$$\pi=\begin{bmatrix}\pi_1 & \pi_2\end{bmatrix} \tag{1.127}$$

euqations (1.121) till (1.124) can be shown like this:

$$\delta_1=\pi O_1 \tag{1.128}$$

$$\delta_2=\delta_1\oplus SO_2 \tag{1.129}$$

$$\delta_3=\delta_2\oplus SO_3 \tag{1.130}$$

$$\text{result}=\delta_3\oplus\vec{I} \tag{1.131}$$

Inserting (1.128) till (1.130) into (1.131) we get:

$$\text{result}=\delta_2\oplus SO_3\oplus\vec{I}=\delta_1\oplus SO_2\oplus SO_3\oplus\vec{I} \tag{1.132}$$

$$\text{result}=[\{\underbrace{\underbrace{\underbrace{(\pi O_1)}_{\delta_1}\oplus(SO_2)]}_{\delta_1}\oplus(SO_3)\}}_{\delta_3}\oplus\vec{I} \tag{1.133}$$

- General formula for above matrices can be defined as folows:

$$\delta_1 = \pi \cdot \left(O_t \vec{I} \right)$$

$$\delta_t = \delta_t \; \max\_op \left\{ S\left(O_t \vec{I} \right) \right\} \qquad ; \qquad 2 \le t \le T \quad ; \qquad O_t = \begin{bmatrix} o_{1o_t} \\ \vdots \\ o_{No_t} \end{bmatrix}_N$$

1.1.4. Problem 3

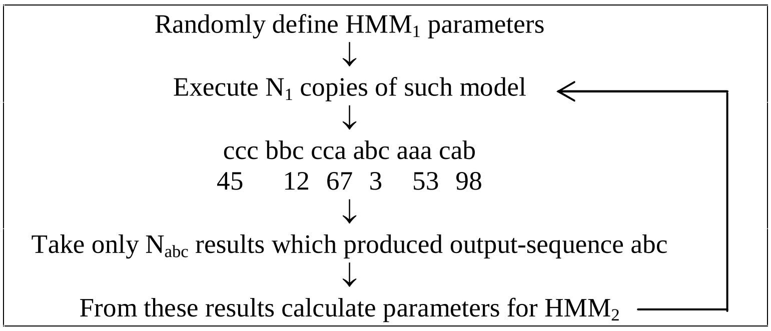
Solution to problem 3 is to alter HMM parameters in the way to find paremeters for which HMM is most probable to give defined output-sequence, for instance (a,a,b,a,c).
First step of this method would be to choose starting values for matrices π, S and O which can be chosen randomly.
Second step is solving problem 1 which gives us a mesure of performance of this starting model.
Third step would then be to calculate new parameters which then gives us matrices $\hat{\pi}$, $\hat{S}$ and $\hat{O}$.
From this point on, procedures described in second and third steps are being repeated until wanted mesure of performance is reached. Extra informations about this method can be found at http://labrosa.ee.columbia.edu/doc/HTKBook21/node7.html.
One way to do this is to use Baum-Welch methos explained in folowing chapter.

1.1.4.1. Baum-Welch method

Main idea behind this method goes like this. Define some HMM_1 with random parameters. Create N_1 copies of such model and execute each of them. Since the behaviour of each copie is random, defined by matrices π_1, S_1 and O_1, we would get a lot of different combinations of state-sequences and output-sequences. To solve problem 3 for output-sequence (a,b,c), from all of these copies take only those which gave this output-sequence. Lets say that number of those was N_{abc}.

We can now pretend that there exists some HMM_2 for which we can repeat above procedure of creating N_{abc} copies, and for which the result of above procedure is exactly those N_{abc} taken from HMM_1. That means that all the copies of such HMM_2 gave output-sequence (a,b,c) which means that this model has near 100% probability to create such sequence and such model would be excellent solution to our problem 3, which is to find HMM which is most probable to give output-sequence (a,b,c).

After finding parameters for such HMM_2 procedure is repeated and for the next step this HMM_2 becomes HMM_1 with the exception that this time parameters were not randomply set. Such a procedure is graphicly presented on folowing figure:



Randomly define HMM1 parameters
Execute N1 copies of such model
ccc bbc cca abc aaa cab
45 12 67 3 53 98
Take only Nabc results which produced output-sequence abc
From these results calculate parameters for HMM2

Fig.1.134 Principal of Baum-Welch method.

Only thing that reamins now is to show how HMM parameters can be calculated. To easier ilustrate this, tables 1.3 and 1.4, from chapters about Backward and Forward calculations, are rewritten here in table 1.9 a) and b) with the difference that each row is multiplied with *N* representing total number of models that produced given state-sequence:

	$\alpha_1(1)$	$\alpha_2(1)$	$\alpha_3(1)$	
111	$N\cdot\pi_1 o_{1a}$	$\cdot s_{11} o_{1b}$	$\cdot s_{11} o_{1c}$	$=N_{111}$
211	$N\cdot\pi_2 o_{2a}$	$\cdot s_{21} o_{1b}$	$\cdot s_{11} o_{1c}$	$=N_{211}$
121	$N\cdot\pi_1 o_{1a}$	$\cdot s_{12} o_{2b}$	$\cdot s_{21} o_{1c}$	$=N_{121}$
221	$N\cdot\pi_2 o_{2a}$	$\cdot s_{22} o_{2b}$	$\cdot s_{21} o_{1c}$	$=N_{221}$
112	$N\cdot\pi_1 o_{1a}$	$\cdot s_{11} o_{1b}$	$\cdot s_{12} o_{2c}$	$=N_{112}$
212	$N\cdot\pi_2 o_{2a}$	$\cdot s_{21} o_{1b}$	$\cdot s_{12} o_{2c}$	$=N_{212}$
122	$N\cdot\pi_1 o_{1a}$	$\cdot s_{12} o_{2b}$	$\cdot s_{22} o_{2c}$	$=N_{122}$
222	$N\cdot\pi_2 o_{2a}$	$\cdot s_{22} o_{2b}$	$\cdot s_{22} o_{2c}$	$=N_{222}$

a)

		$\beta_1(1)$	$\beta_2(1)$	$\beta_3(1)=1$
111	$N\cdot\pi_1 o_{1a}$	$s_{11} o_{1b}$	$\cdot s_{11} o_{1c}$	$=N_{111}$
112	$N\cdot\pi_1 o_{1a}$	$s_{11} o_{1b}$	$\cdot s_{12} o_{2c}$	$=N_{112}$
121	$N\cdot\pi_1 o_{1a}$	$s_{12} o_{2b}$	$\cdot s_{21} o_{1c}$	$=N_{121}$
122	$N\cdot\pi_1 o_{1a}$	$s_{12} o_{2b}$	$\cdot s_{22} o_{2c}$	$=N_{122}$
211	$N\cdot\pi_2 o_{2a}$	$s_{21} o_{1b}$	$\cdot s_{11} o_{1c}$	$=N_{211}$
212	$N\cdot\pi_2 o_{2a}$	$s_{21} o_{1b}$	$\cdot s_{12} o_{2c}$	$=N_{212}$
221	$N\cdot\pi_2 o_{2a}$	$s_{22} o_{2b}$	$\cdot s_{21} o_{1c}$	$=N_{221}$
222	$N\cdot\pi_2 o_{2a}$	$s_{22} o_{2b}$	$\cdot s_{22} o_{2c}$	$=N_{222}$

b)

Tab.1.9 State seqeuce probabilities for given output sequence (a,b,c)

We shall define this new estimated HMM as folows:

$$\lambda=(\pi,S,O) \tag{1.135}$$

$$\hat{\pi} = \begin{bmatrix} \hat{\pi}_1 & \cdots & \hat{\pi}_N \end{bmatrix} \quad , \quad \hat{S} = \begin{bmatrix} \hat{s}_{11} & \hat{s}_{12} & \cdots & \hat{s}_{1N} \\ \hat{s}_{21} & \hat{s}_{22} & & \hat{s}_{2N} \\ \vdots & & \ddots & \vdots \\ \hat{s}_{N1} & \hat{s}_{N2} & \cdots & \hat{s}_{NN} \end{bmatrix} \quad , \quad \hat{O} = \begin{bmatrix} \hat{o}_{11} & \hat{o}_{12} & \cdots & \hat{o}_{1V} \\ \hat{o}_{21} & \hat{o}_{22} & & \hat{o}_{2V} \\ \vdots & & \ddots & \vdots \\ \hat{o}_{N1} & \hat{o}_{N2} & \cdots & \hat{o}_{NV} \end{bmatrix} \qquad (1.136)$$

1.1.4.2. Calculating matrix $\hat{\pi}$

We shall first introduce folowing variables:

$$N_s, \hat{N} \tag{1.137}$$

where: N_s = number of λ models which started with state 's' and produced desired output sequence.
$\hat{N}$ = all λ models which produced desired output sequence.

Relation between variables (1.137) is as folows:

$$N=N_1+...+N_N \tag{1.138}$$

Using (1.137), $\hat{\pi}$ from (1.136) can be defined as folows:

$$\hat{\pi} = \begin{bmatrix} \frac{N_1}{\hat{N}} & ... & \frac{N_N}{\hat{N}} \end{bmatrix} \tag{1.139}$$

If we introduce folowing matrix:

$$\underline{N} = \begin{bmatrix} N_1 & ... & N_N \end{bmatrix} \tag{1.140}$$

then because of (1.138), matrix (1.139) can be calculated by calculating (1.140) and then dividing each element with the sum of its row. To be able to mathematicly express this we will introduce folowing matrix operation on the single matrix:

$$M_{rc}^{DIV} = \frac{M_{rc}}{\sum_{i=1}^{C} M_{ri}} \quad , \quad M_{rc} \in M_{(RxC)} \tag{1.141}$$

Using (1.140) and(1.141), we can express (1.139) as folows:

$$\hat{\pi} = \underline{N}^{DIV} \tag{1.142}$$

Using (1.100) we can calculate any N_s as folows:

$$N_s = N\alpha_1(s)\beta_1(s) \tag{1.143}$$

Inserting (1.143) into (1.140) we get:

$$\underline{N} = N\begin{bmatrix} \alpha_1(1)\beta_1(1) & \cdots & \alpha_1(N)\beta_1(N) \end{bmatrix} \tag{1.144}$$

Using matrix operation defined with (1.159), (1.144) can be presented as folows:

$$\underline{N} = N\begin{bmatrix} \alpha_1(1) & \cdots & \alpha_1(N) \end{bmatrix} \otimes \begin{bmatrix} \beta_1(1) & \cdots & \beta_1(N) \end{bmatrix} \tag{1.145}$$

Using (1.65) for α_t and (1.31) for β_t, equation (1.145) can be rewritten as:

$$\underline{N} = N\alpha_1 \otimes \beta_1^T \tag{1.146}$$

Inserting (1.146) into (1.142) we get:

$$\hat{\pi} = \left[N\alpha_1 \otimes \beta_1^T \right]^{DIV} \tag{1.147}$$

Equation (1.147) can be further simplified by excluding variable N which will appear in nominator and denominator of each element of $\hat{\pi}$:

$$\hat{\pi} = \left[\alpha_1 \otimes \beta_1^T \right]^{DIV} \tag{1.148}$$

To solve (1.148) one needs to calculate only first element of Forward procedure and it also has to calculate the whole Backward procedure in order to get β_1. Therefore Backward procedure can also be used to calculat

1.1.4.3. Example for calculating $\hat{\pi}$

In this chapter we shall show how equation (1.148) is used for calculating $\hat{\pi}$ for a model defined with (1.8) and (1.9). For such model (1.140) becomes:

$$\underline{N} = \left[N_1 \quad N_2 \right] \tag{1.149}$$

Using (1.100), (1.140) becomes:

$$\underline{N} = N\left[\alpha_1(1)\beta_1(1) \quad \alpha_1(2)\beta_1(2) \right] \tag{1.150}$$

Using matrix operation defined with (1.159), (1.150) can be presented as folows:

$$\underline{N} = N\left[\alpha_1(1)\beta \quad \alpha_1(2) \right] \otimes \left[\beta_1(1) \quad \beta_1(2) \right] \tag{1.151}$$

Using (1.65) for α_t and (1.31) for β_t, equation (1.151) can be rewritten as:

$$\underline{N} = N\alpha_1 \otimes \beta_1^T \tag{1.152}$$

1.1.4.4. Calculating $\hat{S}$

We shall first introduce folowing variables:

$$N_{s1s2},\ N_s \tag{1.153}$$

where: N_{s1s2} = counting in all λ models which produced desired output sequence, how many times did models switch from state s_1 to state s_2.

N_s = counting in all λ models which produced desired output sequence, how many times were models in state s.

Relation between variables (1.153) is as folows:

$$N_i = N_{i1} + \ldots + N_{iN} \tag{1.154}$$

Using (1.153), $\hat{S}$ from (1.136) can be defined as folows:

$$\hat{S} = \begin{bmatrix} \frac{N_{11}}{N_1} & \frac{N_{12}}{N_1} & \cdots & \frac{N_{1N}}{N_1} \\ \frac{N_{21}}{N_2} & \frac{N_{22}}{N_2} & & \frac{N_{2N}}{N_2} \\ \vdots & & \ddots & \vdots \\ \frac{N_{N1}}{N_N} & \frac{N_{N2}}{N_N} & \cdots & \frac{N_{NN}}{N_N} \end{bmatrix} \tag{1.155}$$

If we introduce folowing matrix:

$$\underline{N} = \begin{bmatrix} N_{11} & N_{12} & \cdots & N_{1N} \\ N_{21} & N_{22} & & N_{2N} \\ \vdots & & \ddots & \vdots \\ N_{2N1} & N_{N2} & \cdots & N_{NN} \end{bmatrix} \tag{1.156}$$

then because of (1.154), matrix (1.188) can be calculated by calculating (1.148) and then dividing each element with the sum of its row. Using (1.141) and (1.156), we can express (1.155) as folows:

$$\hat{S} = \underline{N}^{DIV} \tag{1.157}$$

To calculate any N_{rc} element first we can count all state-sequences that start with states 'rc' then add those that have 'rc' combination on second place and so on. Using (1.102) this can be presented as:

$$N_{rc} = N \sum_{t=1}^{T-1} \alpha_t(r) s_{rc} o_{co_{t+1}} \beta_{t+1}(c) \tag{1.158}$$

Equation (1.158) can be made simpler by expressing it in a matrix form. For this reason we need to define matrix operation which simply multiplies elements of matrices which are on the same position:

$$(A \otimes B)_{rc} = A_{rc} B_{rc} \tag{1.159}$$

We shall now assume that matrix $\underline{N}$ could also be generated by folowing equation;

$$\underline{N} = N \sum_{t=1}^{T-1} \left(\alpha_t \beta_{t+1}\right) \otimes \left(SO_{t+1}\right) \tag{1.160}$$

- We shall now prove that equation (1.160) holds. Using rules for matrix sumation, we can transform (1.160) into:

$$\underline{N}_{rc} = N \sum_{t=1}^{T-1} \left[\left(\alpha_t \beta_{t+1}\right) \otimes \left(SO_{t+1}\right)\right]_{rc} \tag{1.161}$$

Using (1.159) this can be further trasformed into:

$$\underline{N}_{rc} = N\sum_{t=1}^{T-1}\left(\alpha_t\beta_{t+1}\right)_{rc}\left(SO_{t+1}\right)_{rc} \tag{1.162}$$

Using rules for matrix multiplication, (1.65) for α_t, (1.31) for β_t, and (1.66) for O_T , (1.162) can be trasformed into:

$$\underline{N}_{rc} = N\sum_{t=1}^{T-1}\alpha_t(r)\beta_{t+1}(c)s_{rc}o_{co_{t+1}} \tag{1.163}$$

Since (1.163) is the same as (1.158), we have proved that (1.160) is valid.

Inserting (1.160) into (1.157) we get final equation for calculating $\hat{S}$:

$$\hat{S} = \left[N\sum_{t=1}^{T-1}\left(\alpha_t\beta_{t+1}\right)\otimes\left(SO_{t+1}\right)\right]^{DIV} \tag{1.164}$$

Equation (1.157) can be further simplified by excluding variable N which will appear in nominator and denominator of each element of $\hat{S}$:

$$\hat{S} = \left[\sum_{t=1}^{T-1}\left(\alpha_t\beta_{t+1}\right)\otimes\left(SO_{t+1}\right)\right]^{DIV} \tag{1.165}$$

Using (1.70) and (1.71) for α_t and (1.35) till (1.37) for β_t, for output sequence of four variables (1.165) becomes:

$$\begin{aligned}\hat{S} = \{&[(\pi O_1)(SO_3)(SO_4)\otimes(SO_2)]+\\ &[(\pi O_1)(SO_2)(SO_4)\otimes(SO_3)]+\\ &[(\pi O_1)(SO_2)(SO_3)\otimes(SO_4)]\}^{DIV}\end{aligned} \tag{1.166}$$

1.1.4.5. Example for calculating $\hat{S}$

In this chapter we shall show how equation (1.165) is used for calculating $\hat{S}$ for a model defined with (1.8) and (1.9). For such model (1.156) becomes:

$$\underline{N} = \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} \tag{1.167}$$

Elements of (1.167) can be calculated as folows:

$$N_{11}= (N_{111}+N_{112})+(N_{111}+N_{211}) \qquad N_{12}= (N_{121}+N_{122})+(N_{112}+N_{212}) \tag{1.168}$$

$$N_{21}= (N_{211}+N_{212})+(N_{121}+N_{221}) \qquad N_{22}= (N_{221}+N_{222})+(N_{122}+N_{222}) \tag{1.169}$$

The way in which summands are grouped in equations (1.168) and (1.169) is not random.
To calculate N_{21} first we looked for all state-sequences which on the first place have state 2 and on the second state 1.
We see that this is valid only for N_{211} and N_{212}.
Then we looked for all state sequences which on the second place have state 2 and on the third state 1.
We see that this is valid only for N_{121} and N_{221}.
The same procedure was used to calculate all other N_{rc} elements.

Using (1.169) we can now calculate equations (1.168) and (1.169):

$$N_{11}= N\alpha_1(1)s_{11}o_{1b}\beta_2(1)+N\alpha_2(1)s_{11}o_{1c} \qquad N_{12}= N\alpha_1(1)s_{12}o_{2b}\beta_2(2)+N\alpha_2(1)s_{12}o_{2c} \tag{1.170}$$

$$N_{21}= N\alpha_1(2)s_{21}o_{1b}\beta_2(1)+N\alpha_2(2)s_{21}o_{1c} \qquad N_{22}= N\alpha_1(2)s_{22}o_{2b}\beta_2(1)+N\alpha_2(2)s_{22}o_{2c} \tag{1.171}$$

Inserting equations (1.46) till (1.49) for $\alpha_1(1)$, $\alpha_1(2)$, $\alpha_2(1)$ and $\alpha_2(2)$ and equations (1.15) and (1.16) for $\beta_2(1)$ and $\beta_2(2)$, where all these variables are graphicly shown on table 1.9, into equations N12= Na1(1)s12o2bb2(2)+Na2(1)s12o2c (1.192) till (1.155) it can be shown that the result is correct.

Inserting (1.170) into (1.171) we get:

$$\underline{N} = \begin{bmatrix} N\alpha_1(1)s_{11}o_{1b}\beta_2(1) + N\alpha_2(1)s_{11}o_{1c} \ulcorner & N\alpha_1(1)s_{12}o_{2b}\beta_2(2) + N\alpha_2(1)s_{12}o_{2c} \\ N\alpha_1(2)s_{21}o_{1b}\beta_2(1) + N\alpha_2(2)s_{21}o_{1c} & N\alpha_1(2)s_{22}o_{2b}\beta_2(2) + N\alpha_2(2)s_{22}o_{2c} \end{bmatrix} \tag{1.172}$$

Matrix (1.172) be split into sum of two matrices like this:

$$\underline{N}^{(1)} = N\begin{bmatrix} \alpha_1(1)s_{11}o_{1b}\beta_2(1) \ulcorner & \alpha_1(1)s_{12}o_{2b}\beta_2(2) \\ \alpha_1(2)s_{21}o_{1b}\beta_2(1) & \alpha_1(2)s_{22}o_{2b}\beta_2(2) \end{bmatrix} \tag{1.173}$$

$$\underline{N}^{(2)} = N\begin{bmatrix} \alpha_2(1)s_{11}o_{1c} \ulcorner & \alpha_2(1)s_{12}o_{2c} \\ \alpha_2(2)s_{21}o_{1c} & \alpha_2(2)s_{22}o_{2c} \end{bmatrix} \tag{1.174}$$

$$\underline{N} = \underline{N}^{(1)} + \underline{N}^{(2)} \tag{1.175}$$

From (1.195) and (1.196) we see that these matrices could be calculated using folowing formula, which we will imidiatly prove:

$$N^{(1)} = (\alpha_1\beta_2) \otimes (SO_2) \tag{1.176}$$

$$N^{(2)} = (\alpha_2\beta_3) \otimes (SO_3) \tag{1.177}$$

We will proove (1.169) and (1.170) simply by calculating the expressions. Inserting α_1 from (1.57), β_2 from (1.24), S from (1.9) and O_2 from (1.58) into (1.169) we get:

$$N^{(1)} = \left(\begin{bmatrix} \alpha_1(1) \\ \alpha_1(2) \end{bmatrix} \begin{bmatrix} \beta_2(1) & \beta_2(2) \end{bmatrix}\right) \otimes \left(\begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{bmatrix} o_{1b} & 0 \\ 0 & o_{2b} \end{bmatrix}\right) \tag{1.178}$$

$$N^{(1)} = \begin{bmatrix} \alpha_1(1)\beta_2(1) & \alpha_1(1)\beta_2(2) \\ \alpha_1(2)\beta_2(1) & \alpha_1(2)\beta_2(2) \end{bmatrix} \otimes \begin{bmatrix} s_{11}o_{1b} & s_{12}o_{2b} \\ s_{21}o_{1b} & s_{22}o_{2b} \end{bmatrix}$$

$$N^{(1)} = \begin{bmatrix} \alpha_1(1)\beta_2(1)s_{11}o_{1b} & \alpha_1(1)\beta_2(2)s_{12}o_{2b} \\ \alpha_1(2)\beta_2(1)s_{21}o_{1b} & \alpha_1(2)\beta_2(2)s_{22}o_{2b} \end{bmatrix} \tag{1.179}$$

Matrix (1.172) is the same as (1.166) which proves that expression (1.169) really can be used to calculate (1.166). Inserting α_2 from (1.57), β_3 from (1.24), S from (1.9) and O_3 from (1.58) into (1.170) we get:

$$N^{(2)} = \left(\begin{bmatrix} \alpha_2(1) \\ \alpha_2(2) \end{bmatrix} \begin{bmatrix} \beta_3(1) & \beta_3(2) \end{bmatrix} \right) \otimes \left(\begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{bmatrix} o_{1c} & 0 \\ 0 & o_{2c} \end{bmatrix} \right)$$

$$N^{(2)} = \begin{bmatrix} \alpha_2(1)\beta_3(1) & \alpha_2(1)\beta_3(2) \\ \alpha_2(2)\beta_3(1) & \alpha_2(2)\beta_3(2) \end{bmatrix} \otimes \begin{bmatrix} s_{11}o_{1c} & s_{12}o_{2c} \\ s_{21}o_{1c} & s_{22}o_{2c} \end{bmatrix}$$

$$N^{(2)} = \begin{bmatrix} \alpha_2(1)\beta_3(1)s_{11}o_{1c} & \alpha_2(1)\beta_3(2)s_{12}o_{2c} \\ \alpha_2(2)\beta_3(1)s_{21}o_{1c} & \alpha_2(2)\beta_3(2)s_{22}o_{2c} \end{bmatrix} \tag{1.180}$$

Matrix (1.173) is the same as (1.167) which proves that expression (1.170) really can be used to calculate (1.167).

From (1.170) we can presume that for longer output sequences that equation becomes:

$$\underline{N} = \sum_{t=1}^{T-1} \underline{N}^{(t)} \tag{1.181}$$

And from (1.169) and (1.169) we can persume that folowing formula is valid:

$$N^{(t)} = \left(\alpha_t \beta_{t+1}\right) \otimes \left(SO_{t+1}\right) \tag{1.182}$$

Inserting (1.175) into (1.174) we get:

$$\underline{N} = \sum_{t=1}^{T-1} \left(\alpha_t \beta_{t+1}\right) \otimes \left(SO_{t+1}\right) \tag{1.183}$$

Inserting (1.176) into (1.157) we get:

$$\hat{S} = \left[\sum_{t=1}^{T-1} \left(\alpha_t \beta_{t+1}\right) \otimes \left(SO_{t+1}\right) \right]^{DIV} \tag{1.184}$$

Since equation (1.177) is the same as (1.165) this second procedure gave us the same result but withouth proving it that is is actualy valid. On the other hand benefith of this aproach is that is much simpler since it doesn't use equation (1.158) which is a bit complex to prove and was not done in this paper.

1.1.4.6. Calculating matrix $\hat{O}$

We shall first introduce folowing variables:

$$N_{so}, N_s \tag{1.185}$$

where: N_{so}= how many times model λ was in state 's' and produced output 'o', considering only state-sequences which produced desired output sequence,

N_s = how many times model λ was in state 's', considering only state-sequences which produced desired output sequence.

Using (1.185), $\hat{O}$ from (1.136) can be defined as folows:

$$\hat{O} = \begin{bmatrix} \frac{N_{11}}{N_1} & \frac{N_{12}}{N_1} & \cdots & \frac{N_{1N}}{N_1} \\ \frac{N_{21}}{N_2} & \frac{N_{22}}{N_2} & & \frac{N_{2N}}{N_2} \\ \vdots & & \ddots & \vdots \\ \frac{N_{N1}}{N_N} & \frac{N_{N2}}{N_N} & \cdots & \frac{N_{NN}}{N_N} \end{bmatrix} \tag{1.186}$$

For parameters defined with (1.185) folowing relation is valid:

$$N_i = N_{i1} + ... + N_{iN} \tag{1.187}$$

If we introduce folowing matrix:

$$\underline{N} = \begin{bmatrix} N_{11} & N_{12} & \cdots & N_{1N} \\ N_{21} & N_{22} & & N_{2N} \\ \vdots & & \ddots & \vdots \\ N_{2N1} & N_{N2} & \cdots & N_{NN} \end{bmatrix} \tag{1.188}$$

then because of (1.187) matrix (1.186) can be calculated by calculating (1.188) and then dividing each element with the sum of its row. Using matrix operation (1.141) this can be mathematiclty expressed as folows:

$$\hat{O} = \underline{N}^{DIV} \tag{1.189}$$

Since $N\alpha_t(s)\beta_t(s)$ is number of state sequences which at time 't' were at state 's' and produced desired output sequence. If at that time variable 'o' was outputed that menas that $N\alpha_t(s)\beta_t(s)$ will contribute to N_{so}. So to calculate N_{so} we have to add $N\alpha_t(s)\beta_t(s)$ for steps 't' when output 'o' was released.

$$N_{so} = \sum_{\substack{t=1 \\ v_t=o}}^{T} \alpha_t(s)\beta_t(s) \tag{1.190}$$

To be able to express (1.187) in the matrix form we shall define a matrix which has only one row and all elements are zero except the element at the position equal to the index of the matrix:

$$I_t = [0 \ ... \ 0 \ e_t{=}1 \ 0 \ ... \ 0] \tag{1.191}$$

where: e_t – matrix element at column 't'.

Using (1.65) for α_t, (1.31) for β_t and using (1.191), we can transform (1.190) into:

$$\underline{N} = \sum_{t=1}^{T} (\alpha_t \otimes \beta_t) I_{o_t} \tag{1.192}$$

1.1.4.7. Example for calculating $\hat{O}$

In this chapter we shall show how equation (1.192) is used for calculating $\hat{O}$ for a model defined with (1.8) and (1.9), and for output sequence given as:

$$V = (v_1,v_3,v_3) = (a,c,c) \tag{1.193}$$

For such model (1.188) becomes:

$$\underline{N} = \begin{bmatrix} N_{1a} & N_{1b} & N_{1c} \\ N_{2a} & N_{2b} & N_{2c} \end{bmatrix} \tag{1.194}$$

We will first calculate N_{rc} elements by using (1.100):

$$N_{1a}=N\alpha_1(1)\beta_1(1) \qquad N_{1b}=0 \qquad N_{1c}=N\alpha_2(1)\beta_2(1)+\alpha_3(1)\beta_3(1) \tag{1.195}$$

$$N_{2a}=N\alpha_1(2)\beta_1(2) \qquad N_{2b}=0 \qquad N_{1c}=N\alpha_2(2)\beta_2(2)+\alpha_3(2)\beta_3(2) \tag{1.196}$$

Now we will insert (1.195) and (1.196) into (1.194) which gives:

$$\underline{N} = N\begin{bmatrix} \alpha_1(1)\beta_1(1) & 0 & \alpha_2(1)\beta_2(1)+\alpha_3(1)\beta_3(1) \\ \alpha_1(2)\beta_1(2) & 0 & \alpha_2(2)\beta_2(2)+\alpha_3(2)\beta_3(2) \end{bmatrix} \tag{1.197}$$

$$\frac{\underline{N}}{N} = \begin{bmatrix} \alpha_1(1)\beta_1(1) & 0 & 0 \\ \alpha_1(2)\beta_1(2) & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \alpha_2(1)\beta_2(1) \\ 0 & 0 & \alpha_2(2)\beta_2(2) \end{bmatrix} + \begin{bmatrix} 0 & 0 & \alpha_3(1)\beta_3(1) \\ 0 & 0 & \alpha_3(2)\beta_3(2) \end{bmatrix} \tag{1.198}$$

$$\frac{\underline{N}}{N} = \begin{bmatrix} \alpha_1(1)\beta_1(1) \\ \alpha_1(2)\beta_1(2) \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \alpha_2(1)\beta_2(1) \\ \alpha_2(2)\beta_2(2) \end{bmatrix}\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \alpha_3(1)\beta_3(1) \\ \alpha_3(2)\beta_3(2) \end{bmatrix}\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{1.199}$$

Using matrix operator (1.159), (1.199) is transformed into:

$$\frac{\underline{N}}{N} = \left(\begin{bmatrix} \alpha_1(1) \\ \alpha_1(2) \end{bmatrix} \otimes \begin{bmatrix} \beta_1(1) \\ \beta_1(2) \end{bmatrix}\right)\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \left(\begin{bmatrix} \alpha_2(1) \\ \alpha_2(2) \end{bmatrix} \otimes \begin{bmatrix} \beta_2(1) \\ \beta_2(2) \end{bmatrix}\right)\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + \left(\begin{bmatrix} \alpha_3(1) \\ \alpha_3(2) \end{bmatrix} \otimes \begin{bmatrix} \beta_3(1) \\ \beta_3(2) \end{bmatrix}\right)\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{1.200}$$

Using (1.65) for α_t, (1.31) for β_t, (1.200) becomes

$$\frac{\underline{N}}{N} = \left(\alpha_1^T \otimes \beta_1\right)\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \left(\alpha_2^T \otimes \beta_2\right)\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + \left(\alpha_3^T \otimes \beta_3\right)\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{1.201}$$

Using (1.191), (1.201) can be transformed into:

$$\frac{\underline{N}}{N} = \sum_{t=1}^{3}\left(\alpha_t^T \otimes \beta_t\right) I_{o_t} \tag{1.202}$$

LEFT OVERS

number of state-sequence which in step 1 were in state 1 and which produced requierd output-sequence

Principals for calculating matrix O will be explained for the case of model with two states and output-sequence (a,c,c). For such example we can make a table like the table for Backward and Forward calculations:

111	$N\cdot\pi_1 o_{1a}\cdot s_{11} o_{1c}\cdot s_{11} o_{1c} = N_{111}$
211	$N\cdot\pi_2 o_{2a}\cdot s_{21} o_{1c}\cdot s_{11} o_{1c} = N_{211}$
121	$N\cdot\pi_1 o_{1a}\cdot s_{12} o_{2c}\cdot s_{21} o_{1c} = N_{121}$
221	$N\cdot\pi_2 o_{2a}\cdot s_{22} o_{2c}\cdot s_{21} o_{1c} = N_{221}$
112	$N\cdot\pi_1 o_{1a}\cdot s_{11} o_{1c}\cdot s_{12} o_{2c} = N_{112}$
212	$N\cdot\pi_2 o_{2a}\cdot s_{21} o_{1c}\cdot s_{12} o_{2c} = N_{212}$
122	$N\cdot\pi_1 o_{1a}\cdot s_{12} o_{2c}\cdot s_{22} o_{2c} = N_{122}$
222	$N\cdot\pi_2 o_{2a}\cdot s_{22} o_{2c}\cdot s_{22} o_{2c} = N_{222}$

Tab.1.10 Table of state-sequences.

Using table 1.10 we will construct another table in which we shall state how many times in each state-sequence we have occurences of elements of matrix O. This table would look like this:

acc	o_{1a}	o_{1b}	o_{1c}	o_{2a}	o_{2b}	o_{2c}
111	1	0	2	0	0	0
112	1	0	1	0	0	1
121	1	0	1	0	0	1
122	1	0	0	0	0	2
211	0	0	2	1	0	0
212	0	0	1	1	0	1
221	0	0	1	1	0	1
222	0	0	0	1	0	2

Tab.1.11Number of ocuurences of elements of matrix O in each state sequence.

In table 1.10 in row 111 we have 1 occurence of o_{1a} and that is why we in table 1.11 put number 1 in the row 111 under o_{1a}. In table 1.10 in row 111 we have 2 occurences of o_{1c} and that is why we in table 1.11 put number 2 in the row 111 under o_{1c}. In table 1.10 in row 111 there are no occurences of o_{2a} and o_{2c} and therefore the remaining elements in table 1.11 in row 111 are left to be zero.
It is very important to notice that values in table 1.11 are constant during the procedure

From all the models N, we had N_{111} models that went through state-sequence 111. This information is taken from table 1.10. In each model that went through state-sequence 111 model was twice in state 1 producin 'c' at the same time. This information is taken from table 1.11. That menas that if we only take models which went through state-sequence 111, combination o_{1c} would appear $2{*}N_{111}$ times. If we take all the models then total number of combinations is:

$$N_{o_{1c}} = 2*N_{111} + N_{112} + N_{121} + N_{122} + 2*N_{211} + N_{212} + N_{221} \tag{1.203}$$

$$N_{o_{1a}} = N_{111} + N_{112} + N_{121} + N_{122} \tag{1.204}$$

These values can be stored in folowing matrix:

$$\begin{bmatrix} N_{o_{1a}} & N_{o_{1b}} & N_{o_{1c}} \\ N_{o_{2a}} & N_{o_{2b}} & N_{o_{2c}} \end{bmatrix} \tag{1.205}$$

Matrix O can now be directly calculated from (1.205) by dividing each element by the sum of its row:

$$O = \begin{bmatrix} \dfrac{N_{o_{1a}}}{N_{o_{1a}} + N_{o_{1b}} + N_{o_{1c}}} & \dfrac{N_{o_{1b}}}{N_{o_{1a}} + N_{o_{1b}} + N_{o_{1c}}} & \dfrac{N_{o_{1c}}}{N_{o_{1a}} + N_{o_{1b}} + N_{o_{1c}}} \\ \dfrac{N_{o_{2a}}}{N_{o_{2a}} + N_{o_{2b}} + N_{o_{2c}}} & \dfrac{N_{o_{2b}}}{N_{o_{2a}} + N_{o_{2b}} + N_{o_{2c}}} & \dfrac{N_{o_{2c}}}{N_{o_{2a}} + N_{o_{2b}} + N_{o_{2c}}} \end{bmatrix} \tag{1.206}$$

But since we don't have informations about exact number of models but we only have information about probabilities, to calculate (1.206) first we transform formula (1.203) and (1.204) by dividing each of them with N:

$$P_{o_{1c}} = \frac{N_{o_{1c}}}{N} = 2 * P_{111} + P_{112} + P_{121} + P_{122} + 2 * P_{211} + P_{212} + P_{221} \quad (1.207)$$

$$P_{o_{1a}} = \frac{N_{o_{1a}}}{N} = P_{111} + P_{112} + P_{121} + P_{122} \quad (1.208)$$

Multipling and dividing each element of (1.208) by N we get:

$$O = \begin{bmatrix} \frac{P_{o_{1a}}}{P_{o_{1a}} + P_{o_{1b}} + P_{o_{1c}}} & \frac{P_{o_{1b}}}{P_{o_{1a}} + P_{o_{1b}} + P_{o_{1c}}} & \frac{P_{o_{1c}}}{P_{o_{1a}} + P_{o_{1b}} + P_{o_{1c}}} \\ \frac{P_{o_{2a}}}{P_{o_{2a}} + P_{o_{2b}} + P_{o_{2c}}} & \frac{P_{o_{2b}}}{P_{o_{2a}} + P_{o_{2b}} + P_{o_{2c}}} & \frac{P_{o_{2c}}}{P_{o_{2a}} + P_{o_{2b}} + P_{o_{2c}}} \end{bmatrix} \quad (1.209)$$

P_{111} and others can be most efectivly calculated using Fast-full table calculation method.

Sum of all rows in table table 1.9 a) would be the number of all the models that produced output-sequence (a,b,c). In ideal case we would like to get a model that would only output output-sequence (a,b,c). Idea is to get as close as possible to that ideal by assuming that all the models did only output output-sequences (a,b,c). In that case

- $\gamma_t(s)=P(s_t=s|O,\lambda)$=probability that model is at time t in state s.

$$\gamma_1(1)=\frac{\alpha_1(1)\beta_1(1)}{\alpha_1(1)\beta_1(1)+\alpha_1(2)\beta_1(2)}=\frac{\text{top 4 in B}}{\text{all}}$$
$$\gamma_1(2)=\frac{\alpha_1(2)\beta_1(2)}{\alpha_1(1)\beta_1(1)+\alpha_1(2)\beta_1(2)}=\frac{\text{bottom 4 in B}}{\text{all}}$$

$$\gamma_2(1)=\frac{\alpha_2(1)\beta_2(1)}{\alpha_2(1)\beta_2(1)+\alpha_2(2)\beta_2(2)}=\frac{\text{top 2 in upper4 and lower4 B}}{\text{all}}$$
$$\gamma_2(2)=\frac{\alpha_2(2)\beta_2(2)}{\alpha_2(1)\beta_2(1)+\alpha_2(2)\beta_2(2)}=\frac{\text{bottom 2 in upper4 and lower4 B}}{\text{all}}$$

$$\gamma_3(1)=\frac{\alpha_3(1)\beta_3(1)}{\alpha_3(1)\beta_3(1)+\alpha_3(2)\beta_3(2)}=\frac{\text{top 4 in A}}{\text{all}}$$
$$\gamma_3(2)=\frac{\alpha_3(2)\beta_3(2)}{\alpha_3(1)\beta_3(1)+\alpha_3(2)\beta_3(2)}=\frac{\text{bottom 4 in A}}{\text{all}}$$

- $\xi_t(i,j)=P(s_t=i,s_{t+1}=j|O,\lambda)$=probability that at moment t model is in state i, and at the next moment it is in state j

$$s_{11}^{(1)}=\xi_1(1,1)=\frac{\alpha_1(1)s_{11}o_{b1}\beta_2(1)}{\text{all}}\qquad s_{12}^{(1)}=\xi_1(1,2)=\frac{\alpha_1(1)s_{12}o_{b2}\beta_2(2)}{\text{all}}$$
$$s_{21}^{(1)}=\xi_1(2,1)=\frac{\alpha_1(2)s_{21}o_{b1}\beta_2(1)}{\text{all}}\qquad s_{22}^{(1)}=\xi_1(2,2)=\frac{\alpha_1(2)s_{22}o_{b2}\beta_2(2)}{\text{all}}$$

$$s_{11}^{(2)}=\xi_2(1,1)=\frac{\alpha_2(1)s_{11}o_{c1}\beta_3(1)}{\text{all}}\qquad s_{12}^{(2)}=\xi_2(1,2)=\frac{\alpha_2(1)s_{12}o_{c2}\beta_3(2)}{\text{all}}$$
$$s_{21}^{(2)}=\xi_2(2,1)=\frac{\alpha_2(2)s_{21}o_{c1}\beta_3(1)}{\text{all}}\qquad s_{22}^{(2)}=\xi_2(2,2)=\frac{\alpha_2(2)s_{22}o_{c2}\beta_3(2)}{\text{all}}$$

-

$$\begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}\begin{bmatrix} o_{1a} & o_{1b} \\ o_{2a} & o_{2b} \end{bmatrix}$$

Lets discuss some step in which we want output variable to be a.

If o_{1a} is very big that means that if model gets in state 1 there will be high probability that it will output variable 'a'. If that is what we want then it make sence to increas probability that model switches to state 1. This means increasing s_{11} and s_{21}.

On the other hand if o_{1a} is very small that menas that if model gets in state 1 there will be small probability that it will output variable 'a'. If that is what we want then it make sence to decreas probability that model switches to state 1. This means decreasing s_{11} and s_{21}.

First point says "big $o_{1a}\Rightarrow$increas s_{11} and s_{21}.".
Second point says "small $o_{1a}\Rightarrow$decreas s_{11} and s_{21}.".

One way to do both points is to multiplie s_{11} and s_{21} with o_{1a}.
The same observation can be made for s_{12}, s_{22} and o_{2a}.

After multipling matrix S like it was discusses we get folowing matrix.

$$\begin{bmatrix} s_{11}o_{1a} & s_{12}o_{2a} \\ s_{21}o_{1a} & s_{22}o_{2a} \end{bmatrix}$$

This matrix is not new state transitions matrix because the above procedure does not gurante that the sum of all rows is one. To achive that each element of the above matrix needs to be divided with the sum of its row. Which gives us new state transitions matrix:

$$\hat{S} = \begin{bmatrix} \dfrac{s_{11}o_{1a}}{s_{11}o_{1a}+s_{12}o_{2a}} & \dfrac{s_{12}o_{2a}}{s_{11}o_{1a}+s_{12}o_{2a}} \\ \dfrac{s_{21}o_{1a}}{s_{21}o_{1a}+s_{22}o_{2a}} & \dfrac{s_{22}o_{2a}}{s_{21}o_{1a}+s_{22}o_{2a}} \end{bmatrix}$$

We shall now compare s_{11} and $\hat{s}_{11}$.

$$s_{11} ? \hat{s}_{11}$$

$$s_{11} ? \frac{s_{11}o_{1a}}{s_{11}o_{1a}+s_{12}o_{2a}}$$

$$1 ? \frac{o_{1a}}{s_{11}o_{1a}+s_{12}o_{2a}}$$

$$s_{11}o_{1a}+s_{12}o_{2a} ? o_{1a}$$

$$s_{12}o_{2a} ? o_{1a}(1-s_{11})$$

$$s_{12}o_{2a} ? o_{1a}s_{12}$$

$$o_{2a} ? o_{1a}$$

Folowing conclusion can be made from the result. If o_{1a} was bigger then o_{2a}, that mens that we want to increas probability of model to go to state 1 because then we have higher probability to get variable a. This means that we want to increas all state transitions that go to state 1. One of those transitions is s_{11} and from the above we see that the procedure increases s_{11}.

$$s_{rc} ? \hat{s}_{rc}$$

$$s_{rc} ? \frac{s_{rc}o_{ca}}{\sum_{i=1}^{N} s_{ri}o_{ia}}$$

$$1 ? \frac{o_{ca}}{\sum_{i=1}^{N} s_{ri}o_{ia}}$$

$$\sum_{i=1}^{N} s_{ri}o_{ia} ? o_{ca}$$

$$\sum_{\substack{i=1\\ i\neq c}}^{N} s_{ri}o_{ia} ? o_{ca}(1-s_{rc})$$

$$\sum_{\substack{i=1\\ i\neq c}}^{N} s_{ri}o_{ia} ? \sum_{\substack{i=1\\ i\neq c}}^{N} s_{ri}o_{ca}$$

$$\sum_{\substack{i=1\\ i\neq c}}^{N} s_{ri}(o_{ia}-o_{ca}) ? 0$$

In the case when i=c expression inside round brecket is zero and doesn't add up to expression we can reduce complexitiy of above equation by writing:

$$\sum_{i=1}^{N} s_{ri}(o_{ia}-o_{ca}) ? 0$$

$$s_{11} = s_{11}(o_{1a}-o_{1a})+s_{12}(o_{2a}-o_{1a})+s_{13}(o_{3a}-o_{1a})$$
$$s_{21} = s_{21}(o_{1a}-o_{1a})+s_{22}(o_{2a}-o_{1a})+s_{23}(o_{3a}-o_{1a})$$
$$s_{31} = s_{31}(o_{1a}-o_{1a})+s_{32}(o_{2a}-o_{1a})+s_{33}(o_{3a}-o_{1a})$$

$$s_{21} \quad \sum_{\substack{i=1 \\ i\neq 1}}^{N} s_{2i}(o_{ia} - o_{1a})?0$$

1.2. Conclusion

Goals of this paper can be presented in folowing few points.

- <u>Simple explanation</u>
  One of the atentions of this paper was to give detailed but yet as simple as possible exaplanation of HMM and its problems withouth using complex mathematics. This paper is attended for wide odience that might not have enough technical knowledge to understand the problems as they are presented in most of the other literture covering this topic where a high experience in symbolic language of mathematics and probability is requierd.

- <u>Logic behind formulas</u>
  Other goal of this paper was to present the logic that lies behind HMM and its problems. Goal was not to teach the reader which formulas can be used to solve HMM problems, but rather how to get to that formulas using plain logic. By aquireng full understanding of HMM problems new algorithms and solutions might be developed for better performance.

- <u>Simple results</u>
  At last, by introduction of matrixes and some special matrix operations, existing formulas were made much simpler. This resulted in higher resembles between solutions of different problems and such formulas are easier to remember then complex procedures before them. This can be ilustrated by presenting the formulas described in previous chapters:

Forward procedure:	$\text{result} = \{[(\pi O_1)(SO_2)](SO_3)\}\vec{I}$
Backward procedure:	$\text{result} = (\pi O_1)\{(SO_2)[(SO_3)\vec{I}]\}$
Fast Full Table Calculation:	$\text{result} = [(\pi O_1) \bullet (SO_2)] \bullet (SO_3)$
Viterbi Algorithm:	$\text{result} = \{[(\pi O_1) \oplus (SO_2)] \oplus (SO_3)\} \oplus \vec{I}$
Baum_Welch method:	$\hat{S} = S_1 + S_2 = \{[(\pi O_1)SO_3]\vec{I}\} \otimes (SO_2) + \{[(\pi O_1)(SO_2)]\vec{I}\} \otimes (SO_3)$

1.2.1.1. Calculating matrix $\hat{\pi}$

We shall first introduce folowing variables:

$$N_s, \hat{N} \tag{2.1}$$

where: N_s = number of λ models which started with state 's' and produced desired output sequence.
$\hat{N}$ = all λ models which produced desired output sequence.

Relation between variables (1.137) is as folows:

$$N = N_1 + ... + N_N$$

Using (1.137), elements of $\hat{\pi}$ from (1.136) can be defined as folows:

$$\hat{\pi}_s = \frac{N_s}{\hat{N}} \tag{2.2}$$

Inserting (1.100) into (1.139) we get:

$$\hat{\pi}_s = \frac{N\alpha_1(s)\beta_1(s)}{\hat{N}} \tag{2.3}$$

$$\hat{\pi}_s = \frac{\alpha_1(s)\beta_1(s)}{\frac{\hat{N}}{N}} \tag{2.4}$$

Since denumerator of (1.141) is solution to problem 1 as defined with (1.7), equation can be written as:

$$\hat{\pi}_s = \frac{\alpha_1(s)\beta_1(s)}{P(V \mid \lambda)} \tag{2.5}$$

To transform solution **Error! Not a valid link.** into matrix form we have to do folowing. Inserting (1.142) into $\hat{\pi}$ from (1.136) we get:

$$\pi = \frac{1}{P(V \mid \lambda)} \begin{bmatrix} \alpha_1(1)\beta_1(1) & \cdots & \alpha_1(N)\beta_1(N) \end{bmatrix} \tag{2.6}$$

Using natrix operation defined with (1.159), (1.144) can be presented as folows:

$$\pi = \frac{1}{P(V\,|\,\lambda)}\left[\alpha_1(1) \quad \cdots \quad \alpha_1(N)\right] \otimes \left[\beta_1(1) \quad \cdots \quad \beta_1(N)\right] \tag{2.7}$$

Using (1.65) for α_t and (1.31) for β_t, equation (1.145) can be rewritten as:

$$\hat{\pi} = \frac{\alpha_1 \otimes \beta_1^T}{P(V\,|\,\lambda)} \tag{2.8}$$

To solve (1.146) one needs to calculate only first element of Forward procedure and it also has to calculate the whole Backward procedure in order to get β_1. Therefore Backward procedure can also be used to calculate $P(V|\lambda)$ using (1.33)

1.3. Implementation of HMM

This shapter contains code listing used to implement theory described in previous chapters.

```
//ALL vectors are presented as 2D arrays with one column.There are no 1D arrays and also no arrays with
only one row.
//S=[s11 s12] O=[oa1 ob1 oc1]
//  [s21 s22]   [oa2 ob2 oc2]

import java.util.Random;
import java.lang.Math;
import java.text.DecimalFormat;

public class Speech {

  //DEFINE HMM.
  static  double[][]    Pi  = { {0.3},            //[Pi1]           sum of the column is 100%
                                {0.7}};           //[Pi2]

  static  double[][]    S   = { {0.1,0.9},        //[s11 s12]       sum of each row is 100%
                                {0.2,0.8}};       //[s21 s22]

  static  double[][]    O   = { {0.2,0.3,0.5},    //[o1a o1b o1c]   sum of each row is 100%
                                {0.1,0.3,0.6}};   //[o2a o2b o2c]

  //SS & SO
  static  double[][]    OS  = { {1},              //Oberved output symbols.
                                {1},
                                {0}};
  static  double[][]    SS  = new double[OS.length][0];

  //SAVE ALFA & BETA.
  static  double    saveAlfa[][] = new double[S.length][OS.length];
  static  double    saveBeta[][] = new double[S.length][OS.length];

  //DISPLAY FORMATS.
  static DecimalFormat intFormat = new DecimalFormat("#");
  static DecimalFormat decFormat = new DecimalFormat("0.0000000");

  //OTHERS.
  static  Random    random    = new Random();
  static  double    totalSum  = 0;

  //============================================================================
  //FUNCTION: main
  //============================================================================
  public static void main(String[] args)   {
    System.out.println("SPEECH");

    SS = createSS(Pi,S,5);      displayMatrix(transponMatrix(SS),"SS",intFormat);
    OS = createOS(SS,O);        displayMatrix(transponMatrix(OS),"OS",intFormat);

    displayMatrix(Pi,"Pi" ,decFormat);
    displayMatrix(S ,"S"  ,decFormat);
    displayMatrix(O ,"O"  ,decFormat);
    displayMatrix(transponMatrix(OS),"OS" ,intFormat);
    displayTableWithSymbols(S,OS);
    displayTableWithNumberComponents(Pi,S,O,OS);
    bruteForceCalculation(Pi,S,O,OS,decFormat);
    saveBeta = backwardCalculation(Pi,S,O,OS,decFormat);
    saveAlfa = forwardCalculation(Pi,S,O,OS,decFormat);
    fastFullMethod(Pi,S,O,OS,decFormat);
    viterbiAlgorithm(Pi,S,O,OS,decFormat);

    learn();
  }

  //============================================================================
  //FUNCTION: learn
  //============================================================================
  static public void learn(){
    for(int i=0;i<100;i++){
      Pi = newPi(saveAlfa,saveBeta);
      S  = newS(S,O,OS,saveAlfa,saveBeta);
      bruteForceCalculation(Pi,S,O,OS,decFormat);
    }
  }

  //============================================================================
  //FUNCTION: createSS
  //============================================================================
  /**Create State Sequence. This function creates one test sequence for given MM defined with T,Pi and S.
   * static  int       T       = 10;           //Number of observations.
   * static  double    Pi[]    = {0.5,0.5};
   *
   * static  double    S[][]   = { {0.1,0.9},
   *                               {0.1,0.9}};
   * double[]  SS      = new double[T];
   * SS = createSS(Pi,S,T);   displaySS(SS);
   */
  static public double[][] createSS(double[][] Pi, double[][] S, int T){

    double[][]  SS          = new double[T][1];
    double[][]  stateProbs  = new double[S.length][1];

    SS[0][0] = randomCellFromVector(Pi);
    for(int i=1;i<T;i++) {
      for(int k=0;k<S.length;k++) stateProbs[k][0] = S[ (int)SS[i-1][0] ][k];
      SS[i][0] = randomCellFromVector(stateProbs);
    }
    return SS;
  }

  //=============================================================================
  //FUNCTION: createOS
  //=============================================================================
  /**
   * Create Output-sequence
   * static  double    O[][]   = { {0.1,0.8,0.1},     O=[oa1 ob1 oc1]
   *                               {0.1,0.8,0.1}};      [oa2 ob2 oc2]
   * double[]  SS      = new double[T];
   * SS = createSS(Pi,S,T);      displaySS(SS);
   * OS = createOS(SS,O);        displayOS(OS);
   */
  static public double[][] createOS(double[][] SS, double[][] O){

    int         T   = SS.length;
    double[][]  OS  = new double[T][1];
    double[][]  outProbs  = new double[O.length][1];

    for(int i=0;i<T;i++) {
      for(int k=0;k<S.length;k++) outProbs[k][0] = O[(int)SS[i][0]][k];         //Take current state.Take
row from O corresponding to that state.
      OS[i][0] = randomCellFromVector(outProbs)+1;
    }
    return OS;
  }

  //============================================================================
  //FUNCTION: displayTableWithSymbols
  //============================================================================
  /**Displays table where each row represents one possible state-sequence combination, represented with
symbols.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                         {0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    * displayTableWithSymbols(Pi,S,O,OS);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * 111 pi1o1a s11o1b s11o1c
    * 112 pi1o1a s11o1b s12o2c
    * 121 pi1o1a s12o2b s21o1c
    * 122 pi1o1a s12o2b s22o2c
    * 211 pi2o2a s21o1b s11o1c
    * 212 pi2o2a s21o1b s12o2c
    * 221 pi2o2a s22o2b s21o1c
    * 222 pi2o2a s22o2b s22o2c
    * </pre>
    *
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param OS Output matrix.  Observed sequence of output symbols.
    **/
  static public void displayTableWithSymbols(double[][] S, double[][] OS){

    //VARIABLES.
    int   T       = OS.length;
    int[] SSCode  = new int[T];                             //It will go: 111,112,121,122
    int   NofS    = S.length;                               //Number of states.

    //DISPLAY SS CODES.
    for(int i=0;i<T;i++) {SSCode[i]=1;  SSCode[T-1]=0;}     //SET SS CODE TO 110.
    for(int i=0;i<Math.pow(NofS,T);i++){                    //INCREASE SS CODE ALL THE WAY.

      //INCREASE AND DISPLAY SS CODE.
      SSCode=increaseCounter(SSCode,T-1,NofS);
      System.out.print("\n");
      for(int ci=0;ci<T;ci++) System.out.print(SSCode[ci]);

      //DISPLAY pi RIGHT TO SS CODE.
      System.out.print(" "+"pi"+SSCode[0]);
      System.out.print("o"+SSCode[0]+(char)(OS[0][0]+97));

      //DISPLAY S11 RIGHT TO SS CODE.
      for(int k=1;k<T;k++) {
        System.out.print(" s"+SSCode[k-1]+SSCode[k]);
        System.out.print("o"+SSCode[k]+(char)(OS[k][0]+97));
      }
    }
  }

  //===========================================================================
  //FUNCTION: displayTableWithNumberComponents
  //===========================================================================
  /**Displays table where each row represents one possible state-sequence combination, represented with
basic probabilities.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    * displayTableWithSymbols(Pi,S,O,OS);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * 111 0.3*0.2 * 0.1*0.3 * 0.1*0.5
    * 112 0.3*0.2 * 0.1*0.3 * 0.9*0.6
    * 121 0.3*0.2 * 0.9*0.3 * 0.2*0.5
    * 122 0.3*0.2 * 0.9*0.3 * 0.8*0.6
    * 211 0.7*0.1 * 0.2*0.3 * 0.1*0.5
    * 212 0.7*0.1 * 0.2*0.3 * 0.9*0.6
    * 221 0.7*0.1 * 0.8*0.3 * 0.2*0.5
    * 222 0.7*0.1 * 0.8*0.3 * 0.8*0.6
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    **/
  static public void displayTableWithNumberComponents(double[][] Pi, double[][] S, double[][] O,
double[][] OS){

    System.out.println();

    //VARIABLES.
    int     T        = OS.length;
    int[]   SSCode   = new int[T];                                              //It will go:
111,112,121,122
    int     NofS     = S.length;                                                //Number of states.
    double  result   = 0;
    int     state    = 0;

    //DISPLAY SS CODES.
    for(int i=0;i<T;i++) {SSCode[i]=1;  SSCode[T-1]=0;}                         //SET SS CODE TO 110.
    for(int i=0;i<Math.pow(NofS,T);i++){                                        //INCREASE SS CODE ALL THE
WAY.

      //INCREASE AND DISPLAY SS CODE.
      SSCode=increaseCounter(SSCode,T-1,NofS);
      System.out.print("\n");
      for(int ci=0;ci<T;ci++) System.out.print(SSCode[ci]);
      //DISPLAY pi RIGHT TO SS CODE.
      state = SSCode[0]-1;
      System.out.print(" "+Pi[state][0]);
      System.out.print("*"+O[state][(int)OS[0][0]]);

      //DISPLAY S11 RIGHT TO SS CODE.
      for(int k=1;k<T;k++) {
        System.out.print(" * "+S[SSCode[k-1]-1][SSCode[k]-1]);
        System.out.print("*"+O[SSCode[k]-1][(int)OS[k][0]]);
      }
    }
    System.out.println();
  }

  //============================================================================
  //FUNCTION: bruteForceCalculation
  //============================================================================
  /**Displays table where each row represents one possible state-sequence combination.Total number for
each row is displayed.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    * bruteForceCalculation(Pi,S,O,OS);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * Brute force calculation
    * 111 0.0000900
    * 112 0.0009720
    * 121 0.0016200
    * 122 0.0077760
    * 211 0.0002100
    * 212 0.0022680
    * 221 0.0016800
    * 222 0.0080640
    * Sum of all rows is:0.0226800
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    */
  static public void bruteForceCalculation(double[][] Pi, double[][] S, double[][] O, double[][] OS,
DecimalFormat dispFormat){

    //System.out.println("\nBrute force calculation");

    //VARIABLES.
    int       T           = OS.length;
    int[]     SSCode      = new int[T];     //It will go: 111,112,121,122
    int       NofS        = S.length;       //Number of states.
    double[]  rowsResults = new double[(int)Math.pow(NofS,T)];
    double    result      = 0;
    int       state       = 0;

    //SET SS CODE TO 110.
    for(int i=0;i<T;i++) {SSCode[i]=1;  SSCode[T-1]=0;}

    //INCREASE SS CODE ALL THE WAY.
    for(int i=0;i<Math.pow(NofS,T);i++){

      //INCREASE AND DISPLAY SS CODE.
      SSCode=increaseCounter(SSCode,T-1,NofS);
      for(int ci=0;ci<T;ci++) //System.out.print(SSCode[ci]);

      //DISPLAY pi RIGHT TO SS CODE.
      state   = SSCode[0]-1;
      result  = Pi[state][0] * O[state][(int)OS[0][0]];

      //DISPLAY S11 RIGHT TO SS CODE.
      for(int k=1;k<T;k++) {
        result = result * S[SSCode[k-1]-1][SSCode[k]-1] * O[SSCode[k]-1][(int)OS[k][0]];
      }

      //System.out.println(" "+dispFormat.format(result));
      rowsResults[i] = result;
    }

    //DISPLAY TOTAL SUM OF ALL ROWS.
    double sumOfAllRows = 0;
    for(int i=0;i<Math.pow(NofS,T);i++)  sumOfAllRows = sumOfAllRows + rowsResults[i];
    System.out.println("Sum of all rows is:"+dispFormat.format(sumOfAllRows));
    totalSum = sumOfAllRows;
  }

  //===========================================================================
  //FUNCTION: backwardCalculation
  //===========================================================================
  /**Implements Backward calculation.It returns all beta.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    *
    * double      saveBeta[][] = new double[OS.length][S.length];
    *
    * saveBeta = backwardCalculation(Pi,S,O,OS);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * Backward result is:0.0226800
    *
    * <U>EXAMPLE RETURN</U>:
    * saveBeta = [beta1(1) beta2(1) beta3(1)]
    *            [beta1(2) beta2(2) beta3(2)]
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    */
  //All beta will be saved in the folowing way: saveBeta=[beta1(1) beta1(2)]
  //                                                     [beta2(1) beta2(2)]
  //                                                     [beta3(1) beta3(2)]
  static public double[][] backwardCalculation(double[][] Pi, double[][] S, double[][] O, double[][] OS,
DecimalFormat dispFormat){

    //VARIABLES.
    int         NofS    = S.length;                                             //Number of states.
    int         T       = OS.length;
    double[][]  result  = new double[NofS][1];
    double[][]  beta    = new double[NofS][1];
    double[][]  saveBeta= new double[NofS][T];
    double[][]  o       = new double[NofS][NofS];
    double      backwardResult = 0;

    //SET betaT to 1111.
    for(int r=0;r<NofS;r++) beta[r][0]=1;

    //SAVE LAST BETA.
    for(int r=0;r<NofS;r++)  saveBeta[r][T-1] = beta[r][0];

    //LOOP CALCULATION.                                                         //Bt-1=S o I Bt; BT=1
    for(int i=T-1;i>0;i--){

      //CALCULATE o MATRIX.
      for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[i][0]];                      //SET o.

      //CALCULATE BETA.
      result=multiplieMatrices(S,o);
      result=multiplieMatrices(result,beta);
      beta=result;

      //SAVE BETA.
      for(int r=0;r<NofS;r++)  saveBeta[r][i-1] = beta[r][0];
    }

    //FINAL STEP.
    for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[0][0]];                        //CALCULATE LAST o MATRIX.
    result=multiplieMatrices(o,beta);
    result=multiplieMatrices(transponMatrix(Pi),result);
    backwardResult = result[0][0];

    //DISPLAY RESULT.
    System.out.println("\nBackward result is:"+dispFormat.format(backwardResult));

    return saveBeta;
  }

  //============================================================================
  //FUNCTION: forwardCalculation
  //============================================================================
  /**Implements Forward calculation.It returns all alfa.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    *
    * double      saveBeta[][] = new double[OS.length][S.length];
    *
    * saveBeta = backwardCalculation(Pi,S,O,OS);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * Forward result is:0.0226800
    *
    * <U>EXAMPLE RETURN</U>:
    * saveAlfa = [alfa1(1) alfa2(1) alfa3(1)]
    *            [alfa1(2) alfa2(2) alfa3(2)]
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    */
  static public double[][] forwardCalculation(double[][] Pi, double[][] S, double[][] O, double[][] OS,
DecimalFormat dispFormat){

    //VARIABLES.
    int         NofS    = S.length;                                            //Number of states.
    int         T       = OS.length;
    double[][]  o       = new double[NofS][NofS];
    double[][]  alfa    = new double[NofS][1];
    double[][]  saveAlfa= new double[NofS][T];
    double[][]  result  = new double[NofS][1];
    double      forwardResult = 0;

    //CALCULATE FIRST ALFA.
    for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[0][0]];                        //CALCULATE FIRST o
MATRIX.
    alfa=transponMatrix( multiplieMatrices(transponMatrix(Pi),o) );

    //SAVE FIRST ALFA.
    for(int r=0;r<NofS;r++)  saveAlfa[r][0] = alfa[r][0];

    //CALCULATE AND SAVE OTHER ALFA.
    for(int i=1;i<T;i++) {

      //CALCULATE o MATRIX.
      for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[i][0]];                      //CALCULATE o MATRIX.

      //CALCULATE ALFA.
      result=multiplieMatrices(transponMatrix(alfa),S);
      result=multiplieMatrices(result,o);
      alfa=transponMatrix(result);

      //SAVE ALFA.
      for(int r=0;r<NofS;r++)  saveAlfa[r][i] = alfa[r][0];
    }

    //ADD ALL ALFA.
    for(int i=0;i<alfa.length;i++)
      forwardResult = forwardResult + alfa[i][0];

    //DISPLAY RESULT.
    System.out.println("\nForward result is: "+dispFormat.format(forwardResult));

    return saveAlfa;
  }

  //============================================================================
  //FUNCTION: fastFullMethod
  //============================================================================
  /**Implements fastFull calculation.It returns all alfa.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]          sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    *
    * fastFullMethod(Pi,S,O,OS,decFormat);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * fastFullMethod
    * 0.0000900 0.0009720
    * 0.0016200 0.0077760
    * 0.0002100 0.0022680
    * 0.0016800 0.0080640
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    */
  static public void fastFullMethod(double[][] Pi, double[][] S, double[][] O, double[][] OS,
DecimalFormat dispFormat){

    //VARIABLES.
    int         NofS    = S.length;                                            //Number of states.
    int         T       = OS.length;
    double[][]  o       = new double[NofS][NofS];
    double[][]  fast    = new double[NofS][1];
    double[][]  result  = new double[NofS][1];

    //CALCULATE FIRST STEP.
    for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[0][0]];                                   //SET o WITH
a.
    fast=multiplieMatrices(transponMatrix(Pi),o);

    //CALCULATE OTHER FAST
    for(int i=1;i<T;i++) {
      for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[i][0]];                                 //SET o.
      result=multiplieMatrices(S,o);
      result=combineMatrices(fast,result);
      fast=result;
    }

    //DISPLAY RESULT.
    displayMatrix(result,"fastFullMethod",decFormat);

    //DISPLAY SUM OF ALL.
    double sum = 0;
    for(int r=0;r<result.length;r++)
      for(int c=0;c<result[0].length;c++)
        sum = sum + result[r][c];
    System.out.println("Sum of all cells is:"+dispFormat.format(sum));
  }

  //============================================================================
  //FUNCTION: viterbiAlgorithm
  //===========================================================================
  /**Implements Viterbi algorithm which is equivalent to finding table row with biggest number which is
equivalent to finding state sequence which is most probable to produse given output sequence OS.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double[][]  Pi      = { {0.3},            //[Pi1]           sum of the column is 100%
    *                         {0.7}};           //[Pi2]
    *
    * double      S[][]   = { {0.1,0.9},
    *                         {0.1,0.9}};
    *
    * double      O[][]   = { {0.1,0.8,0.1},
    *                       { 0.1,0.8,0.1}};
    *
    * double[][]  OS      = { {0},              //Oberved output symbols.
    *                         {1},
    *                         {2}};
    *
    * viterbiAlgorithm(Pi,S,O,OS,decFormat);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * Viterbi result(row with biggest number): 0.0080640
    *
    * </pre>
    *
    * @param Pi Pi vector.      Defines probabilities of model starting in each state.
    * @param S  State matrix.   Defines probabilities of model going from one state to another.It is
needed here only to calculate number of possible states.
    * @param O  Pi vector.      Defines probabilities of model outputing symbol at each state.
    * @param OS Output matrix.  Observed sequence of output symbols.
    */
  static public void viterbiAlgorithm(double[][] Pi, double[][] S, double[][] O, double[][] OS,
DecimalFormat dispFormat){

    //VARIABLES.
    int         NofS    = S.length;                                             //Number of states.
    int         T       = OS.length;
    double[][]  delta   = new double[NofS][1];
    double[][]  o       = new double[NofS][NofS];
    double[][]  result  = new double[NofS][1];
    double      viterbiResult = 0;

    //CALCULATE FIRST STEP.
    for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[0][0]];                        //SET o WITH a.
    delta=transponMatrix( multiplieMatrices(transponMatrix(Pi),o) );

    //CALCULATE OTHER FAST
    for(int i=1;i<T;i++) {
      for(int r=0;r<NofS;r++) o[r][r]=O[r][(int)OS[i][0]];                      //SET o.
      result=multiplieMatrices(S,o);
      result=maxInstedSum(transponMatrix(delta),result);
      delta=transponMatrix(result);
    }

    //FIND MAX DELTA
    for(int i=0;i<delta.length;i++)
      if(delta[i][0] > viterbiResult)
        viterbiResult = delta[i][0];
    System.out.println("\nViterbi result(row with biggest number): "+dispFormat.format(viterbiResult));

  }

  //============================================================================
  //FUNCTION: newPi
  //============================================================================
  static public double[][] newPi(double[][] saveAlfa, double[][] saveBeta){

    int NofS  = saveAlfa.length;
    double[][] newPi = new double[NofS][1];

    for(int r=0;r<NofS;r++)
      newPi[r][0] = saveAlfa[r][0] * saveBeta[r][0] / totalSum;

    return newPi;
  }

  //============================================================================
  //FUNCTION: newS
  //============================================================================
  static public double[][] newS(double[][] S, double[][] O, double[][] OS, double[][] saveAlfa, double[][]
saveBeta){

    int NofS  = saveAlfa.length;
    int T     = saveAlfa[0].length;                                             //Number of observations.
    double[][]    o     = new double[NofS][NofS];
    double[][]    alfa  = new double[NofS][1];
    double[][]    beta  = new double[NofS][1];
    double[][]    left  = new double[NofS][NofS];
    double[][]    right = new double[NofS][NofS];
    double[][][]  saveS = new double[T-1][NofS][NofS];
    double[][]    rowSums = new double[NofS][1];
    double[][]    result= new double[NofS][NofS];

    for(int i=0;i<T-1;i++) {                                                     //T=3,T-1=2,i=0,1 which is
alfa1,alfa2

      //TAKE NEXT ALFA & BETA.
      for(int r=0;r<NofS;r++){
        alfa[r][0] = saveAlfa[r][i];
        beta[r][0] = saveBeta[r][i+1];
      }

      //CALCULATE NEXT o MATRIX.
      for(int k=0;k<NofS;k++)  o[k][k]=O[k][(int)OS[i+1][0]];

      //CALCULATE FORMULA.
      right     = multiplieMatrices(S,o);
      left      = multiplieMatrices(alfa,transponMatrix(beta));
      saveS[i]  = multiplieCells(left,right);
    }

    //SET RESULT MATRIX TO 0.
    for(int r=0;r<NofS;r++)
      for(int c=0;c<NofS;c++)
        result[r][c] = 0;

    //CALCULATE SUM.
    for(int i=0;i<T-1;i++) result = addMatrices(result,saveS[i]);

    //CALCULATE SUM OF EACH ROW.
    for(int r=0;r<NofS;r++) {
      rowSums[r][0] = 0;
      for(int c=0;c<NofS;c++)
        rowSums[r][0] = rowSums[r][0] + result[r][c];
    }

    //DIVIDE ELEMENTS WITH ROW SUMS.
    for(int r=0;r<NofS;r++) {
      for(int c=0;c<NofS;c++)
        result[r][c] = result[r][c]/rowSums[r][0];
    }

    return result;
  }


//**********************************************************************************************************
//**********************************************************************************************************
//                                                            MATRIX OPERATIONS
//**********************************************************************************************************
//**********************************************************************************************************

  //=============================================================================
  //FUNCTION: displayMatrix
  //=============================================================================
  /**Displays all cells of the given matrix.<p>
   *
   * <pre>
   * <U>EXAMPLE CODE</U>:
   * DecimalFormat decFormat = new DecimalFormat("0.0000000");
   * double    S[][]   = { {0.1,0.9},
   *                       {0.2,0.8}};
   * displayMatrix(S,"Matrix S",decFormat);
   *
   * <U>EXAMPLE OUTPUT</U>:
   * S
   * 0.1000000 0.9000000
   * 0.2000000 0.8000000
   * </pre>
   *
   * @param S           Matrix which cells are to be displayed.Matrix is presented as 2D array of type
double.
   * @param title       Text to display as title before the matris is diplayed. This is usually the name
of the matrix.
   * @param dispFormat  Defines format in which matrix elements will be displayed.
   */
  static public void displayMatrix(double S[][], String title, DecimalFormat dispFormat){
    System.out.print("\n"+title);
    for(int r=0;r<S.length;r++){
      System.out.println();
      for(int c=0;c<S[0].length;c++){
        System.out.print(dispFormat.format(S[r][c])+" ");
      }
   }
   System.out.println("");
  }

  //============================================================================
  //FUNCTION: transponMatrix
  //============================================================================
  /**Reoranges elements by switching row and column index of each element.<p>
   *
   * <pre>
   * double    S[][]   = { {11,12},
   *                       {21,22}};
   * S = transponMatrix(S);
   *
   * <U>EXAMPLE RETURN</U>:
   * S =  [11,21]
   *      [12,22]
   * </pre>
   *
   * @param   S           Matrix which cells are to be reoranged.
   * @return  double[][]  New matrix, which is the result of the operation.
   */
  static public double[][] transponMatrix(double[][] A){
    double[][] R = new double[A[0].length][A.length];
    for(int r=0;r<A.length;r++)
      for(int c=0;c<A[0].length;c++)
        R[c][r]=A[r][c];
    return R;
  }

  //============================================================================
  //FUNCTION: multiplieMatrices
  //============================================================================
  /**Mulitplies two matrices.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double    A[][]  = { {1,9}};
    *
    * double    B[][]  = { {2,3},
    *                      {1,5}};
    * A = multiplieMatrices(A,B);
    *
    * <U>EXAMPLE RETURN</U>:
    * A=[11 30]
    *   [12 30]
    *
    * @param   A           Matrix on the left to be multiplied
    * @param   B           Matrix on the right to be multiplied
    * @return  double[][]  New matrix, which is the result of the operation.
    */
  static public double[][] multiplieMatrices(double[][] A, double[][] B){

    //CHECK IF DIEMENSIONS ARE VALID.
    if(A[0].length!=B.length) {
      System.out.println("ERROR from multiplieMatrices: invalid dimensions!");
      return null;
    }

    double[][]  R   = new double[A.length][B[0].length];
    double      sum = 0;

    for(int r=0;r<A.length;r++){
      for(int c=0;c<B[0].length;c++){
        sum = 0;
        for(int k=0;k<A[0].length;k++){
          sum = sum + A[r][k]*B[k][c];
        }
        R[r][c]=sum;
      }
    }

    return R;
  }

  //============================================================================
  //FUNCTION: multiplieCells
  //============================================================================
  /**Mulitplies two matrices.<p>
```

```
  *
  * <pre>
  * <U>EXAMPLE CODE</U>:
  *
  * @param   A           Matrix on the left to be added.
  * @param   B           Matrix on the right to be added.
  * @return  double[][]  New matrix, which is the result of the operation.
  */
static public double[][] multiplieCells(double[][] A, double[][] B){

  //CHECK IF DIEMENSIONS ARE VALID.
  if(A.length!=B.length || A[0].length!=B[0].length) {
    System.out.println("ERROR from addMatrices: Matrices don't have the same dimension!");
    return null;
  }

  double[][]  R   = new double[A.length][B[0].length];

  for(int r=0;r<A.length;r++){
    for(int c=0;c<B[0].length;c++){
      R[r][c]=A[r][c]*B[r][c];
    }
  }

  return R;
}

//============================================================================
//FUNCTION: addMatrices
//============================================================================
/**Mulitplies two matrices.<p>
  *
  * <pre>
  * <U>EXAMPLE CODE</U>:
  *
  * @param   A           Matrix on the left to be added.
  * @param   B           Matrix on the right to be added.
  * @return  double[][]  New matrix, which is the result of the operation.
  */
static public double[][] addMatrices(double[][] A, double[][] B){

  //CHECK IF DIEMENSIONS ARE VALID.
  if(A.length!=B.length || A[0].length!=B[0].length) {
    System.out.println("ERROR from addMatrices: Matrices don't have the same dimension!");
    return null;
  }

  double[][]  R   = new double[A.length][B[0].length];

  for(int r=0;r<A.length;r++){
    for(int c=0;c<B[0].length;c++){
      R[r][c]=A[r][c]+B[r][c];
    }
  }

  return R;
}

//============================================================================
//FUNCTION: combineMatrices
//============================================================================
/**Comines two matrices.<p>
  *
  * <pre>
  * <U>EXAMPLE CODE</U>:
  * double    A[][]  = { {1,9}};
  *
  * double    B[][]  = { {2,3},
  *                      {1,5}};
  * A = combineMatrices(A,B);
  *
  * <U>EXAMPLE RETURN</U>:
  *
  * @param   A           Matrix on the left to be combined
  * @param   B           Matrix on the right to be combined
  * @return  double[][]  New matrix, which is the result of the operation.
  */
static public double[][] combineMatrices(double[][] A, double[][] B){

  int rR = -1;
  double[][] R = new double[A.length*A[0].length][B[0].length];

  for(int rA=0;rA<A.length;rA++){
    for(int cA=0;cA<A[0].length;cA++){
      rR++;
```

```
        for(int cB=0;cB<B[0].length;cB++) {
            R[rR][cB]=A[rA][cA]*B[cA][cB];
        }
      }
    }
    return R;
  }

  //============================================================================
  //FUNCTION: maxInstedSum
  //============================================================================
  /**This operator works like normal multiplier but instead os summing results it takes the maximum value.
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * double    A[][]   = { {1,4}};
    *
    * double    B[][]   = { {2,3},
    *                       {1,5}};
    * maxInstedSum(A,B);
    *
    * <U>EXAMPLE RETURN</U>:
    * R = [4 20]
    *
    * @param   A           Matrix on the left to be combined
    * @param   B           Matrix on the right to be combined
    * @return  double[][]  New matrix, which is the result of the operation.
    * </pre>
   **/
  static public double[][] maxInstedSum(double[][] A, double[][] B){

    double[][] R = new double[A.length][B[0].length];
    double max   = 0;
    double multi = 0;

    for(int r=0;r<A.length;r++){
      for(int c=0;c<B[0].length;c++){
        max = 0;
        for(int k=0;k<A[0].length;k++){
          multi = A[r][k]*B[k][c];
          if(multi>max) max = multi;
        }
        R[r][c]=max;
      }
    }

    return R;
  }

  //=============================================================================
  //FUNCTION: increaseCounter
  //=============================================================================
  /**Increases counter by one at the given position.<p>
    *
    * <pre>
    * <U>EXAMPLE CODE</U>:
    * int[] counter   = new int[3];     //Example: 122
    * int   pos       = 0;              //Equivalent to adding 1 to counter.
    * int   NofDigits = 2;              //Two possible digits: 1 and 2.
    * counter  = increaseCounter(counter,pos,NofDigits);
    *
    * <U>EXAMPLE OUTPUT</U>:
    * counter = 221;
    * </pre>
    *
    * @param counter    Counter
    * @param pos        At which position should the counter be increased.pos=0 is equivalent to adding 1
to counter.
    * @param NofDigits  Number of possible digits.If it is 2, and counter is in state 122, adding 1 will
change counter to 211.
   */
  //Counter goes from 1.
  static public int[] increaseCounter(int counter[], int pos, int NofDigits){
      if(counter[pos]<NofDigits) {
        counter[pos]++;
        return counter;
      }
      else{
        counter[pos]=1;
        return increaseCounter(counter,--pos,NofDigits);
      }
  }

  //=============================================================================
  //FUNCTION: randomCellFromVector
  //============================================================================
  /**Input to function is vector.Sum of all cells in that vector should be one because they present 
probabilitie of cell activation.
   *static  double    Pi[]    = {0.5,0.5};
    */
  static public double randomCellFromVector(double[][] v){

    double    current = 0;
    double    rand    = random.nextDouble();

    for(int s=0;s<v.length;s++){
      current = current + v[s][0];
      if (rand<=current) return s;
    }
    return -1;
  }

}
```

1.4. Program output

This chapter presents starting and ending output of learning algorithm.

```
SPEECH

Pi
0.2000000
0.8000000

S
0.3000000 0.7000000
0.2000000 0.8000000

O
0.3000000 0.3000000 0.4000000
0.1000000 0.6000000 0.3000000

OS
2 1 0

111 pi1o1c s11o1b s11o1a
112 pi1o1c s11o1b s12o2a
121 pi1o1c s12o2b s21o1a
122 pi1o1c s12o2b s22o2a
211 pi2o2c s21o1b s11o1a
212 pi2o2c s21o1b s12o2a
221 pi2o2c s22o2b s21o1a
222 pi2o2c s22o2b s22o2a

111 0.2*0.4 * 0.3*0.3 * 0.3*0.3
112 0.2*0.4 * 0.3*0.3 * 0.7*0.1
121 0.2*0.4 * 0.7*0.6 * 0.2*0.3
122 0.2*0.4 * 0.7*0.6 * 0.8*0.1
211 0.8*0.3 * 0.2*0.3 * 0.3*0.3
212 0.8*0.3 * 0.2*0.3 * 0.7*0.1
221 0.8*0.3 * 0.8*0.6 * 0.2*0.3
222 0.8*0.3 * 0.8*0.6 * 0.8*0.1

Sum of all rows is:0.0242880

Backward result is:0.0242880

Forward result is: 0.0242880

fastFullMethod
0.0006480
0.0005040
0.0020160
0.0026880
0.0012960
```

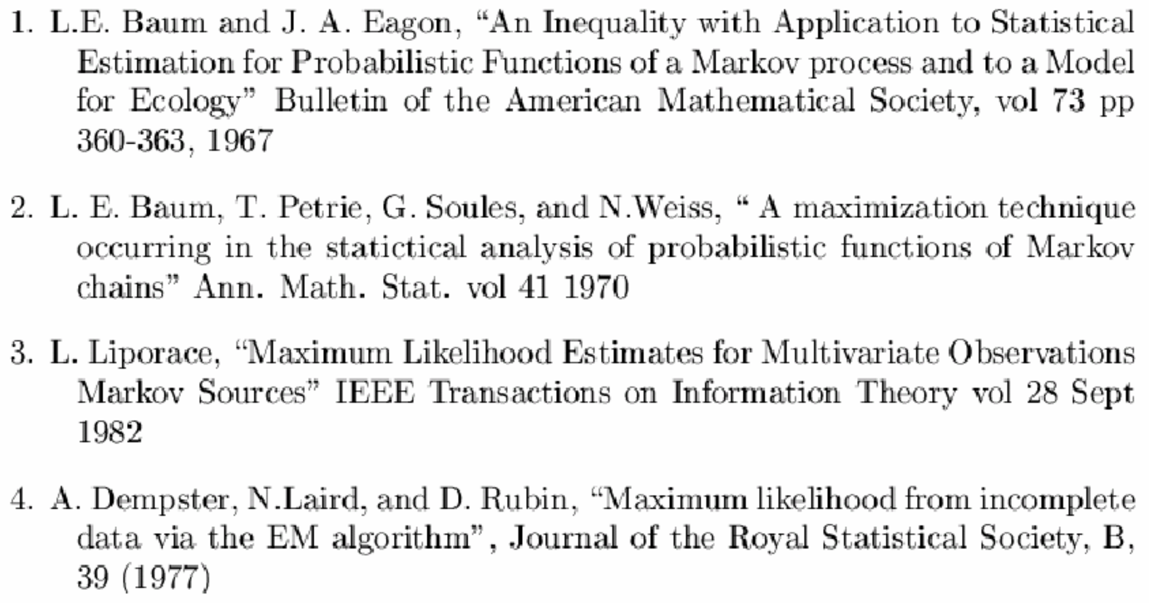
```
0.0010080
0.0069120
0.0092160
Sum of all cells is:0.0242880
Ending result looks like this:

Pi
0.0000000
1.0000000

S
1.0000000 0.0000000
1.0000000 0.0000000

O
0.5000000 0.5000000 0.0000000
0.0000000 0.0000000 1.0000000

Sum of all rows is:0.2500000

Backward result is:0.2500000

Forward result is: 0.2500000
```

Conclusion is that starting HMM model had probability 0.024 of generating given sequence.
At the end of learning HMM model had probability 0.25 of generating given sequence.

http://www.itl.nist.gov/div898/handbook/eda/section3/eda361.htm probability

Baum and Eagon prvi nacin na koji su dokazali

Baum, Petrie, Soules and Weis jednostavniji nacin
Kullback-Leibler diveregence of two distributions



1. L.E. Baum and J. A. Eagon, "An Inequality with Application to Statistical Estimation for Probabilistic Functions of a Markov process and to a Model for Ecology" Bulletin of the American Mathematical Society, vol 73 pp 360-363, 1967

2. L. E. Baum, T. Petrie, G. Soules, and N.Weiss, " A maximization technique occurring in the statictical analysis of probabilistic functions of Markov chains" Ann. Math. Stat. vol 41 1970

3. L. Liporace, "Maximum Likelihood Estimates for Multivariate Observations Markov Sources" IEEE Transactions on Information Theory vol 28 Sept 1982

4. A. Dempster, N.Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", Journal of the Royal Statistical Society, B, 39 (1977)

- **Problem 3:**

- Brute force calculation
  - $P(V|\lambda)$=tablica
  - $P(V|\lambda)$=opca formula

– Forward calculation – $P(V\mid\lambda)$=tablica – uvodjenje α-varijabli – $P(V\mid\lambda)$=α-varijable – uvodjenje α-matrica – $P(V\mid\lambda)$=α-matrice – $P(V\mid\lambda)$=SOS-matrice – probability meaning of α-varijabli – dokazan probability meaning za $\alpha_1(1)$-varijablu	– Backward calculation – $P(V\mid\lambda)$=tablica – uvodjenje β-varijabli – $P(V\mid\lambda)$=β-varijable – uvodjenje β-matrica – $P(V\mid\lambda)$=β-matrice – $P(V\mid\lambda)$=SOS-matrice – probability meaning of β-varijabli – dokazan probability meaning za $\alpha_2(1)$-varijablu
– Estimate π – idea – variable solution – matrix solution – Estimate S – General proof – S=matrix-formula – Estimate O – General proof – O=matrix-formula – example	