Seasonal Hidden Markov Models for Stochastic Time Series with Periodically Varying Characteristics

Arthur M. Lewis

Portland State University

Let us know how access to this document benefits you.

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Electronics Commons

Recommended Citation


10.15760/etd.6932

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.
THESIS APPROVAL

The abstract and thesis of Arthur M. Lewis for the Master of Science in Electrical Engineering was presented July 5, 1995 and accepted by the thesis committee and the department.

COMMITTEE APPROVALS:

Andrew M Fraser, Chair

Lee Casperson

Roy Koch
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:

Rolf Schaumann, Chair
Department of Electrical Engineering

*********************************************************************************************

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by on 26 Sep 1995
ABSTRACT


Title: Seasonal Hidden Markov Models for Stochastic Time Series with Periodically Varying Characteristics

Novel seasonal hidden Markov models (SHMMs) for stochastic time series with periodically varying characteristics are developed.

Nonlinear interactions among SHMM parameters prevent the use of the forward-backward algorithms which are usually used to fit hidden Markov models to a data sequence. Instead, Powell’s direction set method for optimizing a function is repeatedly applied to adjust SHMM parameters to fit a data sequence.

SHMMs are applied to a set of meteorological data consisting of 9 years of daily rain gauge readings from four sites. The fitted models capture both the annual patterns and the short term persistence of rainfall patterns across the four sites.
SEASONAL HIDDEN MARKOV MODELS FOR STOCHASTIC TIME SERIES WITH PERIODICALLY VARYING CHARACTERISTICS

by

ARTHUR M. LEWIS

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING

Portland State University

1995
Title: Seasonal Hidden Markov Models for Stochastic Time Series with Periodically Varying Characteristics

Novel seasonal hidden Markov models (SHMMs) for stochastic time series with periodically varying characteristics are developed.

Nonlinear interactions among SHMM parameters prevent the use of the forward-backward algorithms which are usually used to fit hidden Markov models to a data sequence. Instead, Powell’s direction set method for optimizing a function is repeatedly applied to adjust SHMM parameters to fit a data sequence.

SHMMs are applied to a set of meteorological data consisting of 9 years of daily rain gauge readings from four sites. The fitted models capture both the annual patterns and the short term persistence of rainfall patterns across the four sites.
ACKNOWLEDGMENTS

I thank Dr. Andrew Fraser for suggesting the thesis topic and patiently supporting my slow progress as I attempted to apply and then abandoned a long sequence of optimizing schemes for fitting SHMMs to a data sequence.

I also thank Dr. Roy Koch for his patience and support during the completion of this project.
# Table of Contents

1995 ................................................................................................................................. i
ABSTRACT ........................................................................................................................... ii
ACKNOWLEDGMENTS ......................................................................................................... iii

1 INTRODUCTION ............................................................................................................... 1
1.1 A Brief Review of Hidden Markov Models ...................................................................... 1
1.2 Seasonal Hidden Markov Models .................................................................................. 4

2 MATHEMATICS OF HMMs AND SHMMs .................................................................... 5
2.1 HMM Definition .............................................................................................................. 5
2.2 SHMM Definition .......................................................................................................... 6
2.3 Likelihood ...................................................................................................................... 7
2.4 Sequence Probability ................................................................................................... 8
2.5 Log Likelihood ............................................................................................................. 9

3 FITTING A SHMM TO A DATA SEQUENCE ............................................................... 10
3.1 Difficulties Fitting SHMMs ......................................................................................... 10
3.2 Adaptive Simulated Annealing .................................................................................... 12
3.3 Powell's Direction Set Method .................................................................................... 12
3.4 Transformations Between Bounded and Unbounded Parameters ................................. 13

4 PROGRAMS .................................................................................................................. 15
4.1 Program shmmrpow ................................................................................................. 15
4.2 Program shmmbrpo .................................................................................................... 18
4.3 Programs chkshmm, chshmmb ................................................................................. 19
4.4 Programs mod2prob, mod2bpro ............................................................................. 19
4.5 Program seqpairs .................................................................................................. 20

5 THE RAIN GAUGE DATA ........................................................................................... 21
5.1 Missing Data ............................................................................................................. 21
5.2 Conversion to Discrete Data Sequences .................................................................. 23

6 SHMM RESULTS ........................................................................................................ 29
6.1 Optimization Experiments ....................................................................................... 29
6.2 Optimized SHMMs and HMMs - 16 Level Models .................................................. 31
6.3 Increasing the Number of States .............................................................................. 36

7 CONCLUSIONS ............................................................................................................ 43
7.1 Some Observations on the Nature of the Optimized SHMMs ..................................... 43
7.2 Extending the Application of Non-Stationary HMMs ................................................. 45
7.3 Conclusions ............................................................................................................ 46

APPENDIX A PROGRAM FILES ..................................................................................... 47
A1 shmmrpow Makefile ................................................................................................. 47
A2 shmmrpow shhm.h include file ............................................................................ 49
A3 shmmrpow source files ...................................................................................... 52
A4 shmmbrpo.c Makefile ............................................................................................ 64
A5 shmmbrpo shmmb.h include file ........................................................................... 67
A6 shmmbrpo source files ....................................................................................... 70
A7 seqcalc Makefile, source & include files .............................................................. 74
A8 utilimps.c utilimps.h source & include files ...................................................... 87

APPENDIX B PARAMETERS OF OPTIMIZED SHMM ...................................................... 92

APPENDIX C MEAN PROBABILITIES, ALL 9 RUNS, 16 PATTERN SHMMs ......................... 97
List of Figures

Fig. 1.1 A simple Markov process .................................................................2
Fig. 1.2 A hidden Markov model, discrete outputs .......................................3
Fig. 3.1 Bounded/Unbounded Transformation ................................................14
Fig. 5.2 Binary Sequences, Four Sites, Annual Plots .....................................24
Fig. 5.3 16 Pattern Sequence, Annual Plots ..................................................26
Fig. 5.4 Number of "Wet" Site Sequence, Annual Plots ..................................27
Fig. 6.1 A Typical Sequence Generated by 2 State SHMM y5s2g1i8.1 ...............34
Fig. 6.2 A Typical Sequence Generated by 6 State HMM y5s6g0i3.2 ...............35
Fig. 6.3 Best 2 State SHMM Probabilities ....................................................37
Fig. 6.4 Best 3 State SHMM Probabilities ....................................................38
Fig. 6.5 Best 4 State SHMM Probabilities ....................................................39
Fig. 6.6 Best 5 State SHMM Probabilities ....................................................40
Fig. 6.7 Best 6 State SHMM Probabilities ....................................................41
Fig. 6.8 61 Day Moving Averages, 16 Patterns ..........................................42

List of Tables

Table 2.1 HMM and SHMM parameters. Parameter counts for S states and K outputs
Table 5.1 Rain Pattern Codes .................................................................28
Table 6.1 Best of 9 Optimized SHMMs and HMMs - 5 Year Training ...........32
Table 6.2 Best of 9 Optimized SHMMs and HMMs - 4 Year Training ...........33
1 Introduction

The behavior of stochastic data sequences may be modeled by hidden Markov models (HMMs). A HMM which has been fitted to a stochastic sequence may be used to estimate and predict the behavior of the sequence beyond the available data. HMMs have been widely applied in speech recognition for the conversion of speech waveforms to phoneme sequences and the recognition of phoneme sequences as words[1].

This thesis discusses seasonal hidden Markov models (SHMMs) for stochastic time series with periodically varying characteristics. The SHMMs are applied to a sequence of rain gauge readings which extend over a period of 9 years. These SHMMs can be used to generate long artificial sequences which emulate the behavior of shorter real data sequences without simply repeating the original data.

1.1 A Brief Review of Hidden Markov Models

A first order Markov process generates a stochastic sequence of states in which the probability of each successor state depends only the state preceding it. Figure 1.1
illustrates a 3 state Markov process. The arrows in the figure represent possible state-to-state transitions. The process is characterized by the state transition probabilities. The probability that predecessor state \( i \) is followed by successor state \( j \) is represented by \( a_{ji} \).

Fig. 1.1 A simple Markov process

\[ a_{ji} = \text{transition probability, predecessor state } i \text{ to successor state } j \]
In a hidden Markov model (HMM)\cite{2,3,4} the sequence of states is not directly observable. Instead, each state stochastically produces observable outputs some probability distribution. In Fig. 1.2 arrows to the discrete outputs produced by each state are added to the previous sketch. Each state-to-output probability is represented by $b_{kj}$.

Fig. 1.2 A hidden Markov model, discrete outputs

$a_{ij} = $ Transition probability, state $i$ to state $j$

$b_{kj} = $ probability that state $j$ generates output $k$
The actual states are usually unknown in a HMM. Indeed, HMM parameters (probabilities) which can generate a given data sequence are normally inferred by calculation from the data sequence itself. The forward-backward method is well established as the preferred method for generating a maximum likelihood estimate of HMM parameters with a given number of states.

Hidden Markov models have previously been applied by Zucchini & Gutorp (Z&G)[5] to describe sequences of meteorological precipitation patterns. Z&G calculated separate HMMs for different portions of the year in order to cope with the differences in rainfall through the year.

1.2 Seasonal Hidden Markov Models

The present work adopts a different approach using seasonal hidden Markov models. A SHMM incorporates seasonal variation of the observable sequence behavior through the yearly cycle via seasonal variation parameters. Unfortunately the seasonal parameters prevent the application of well-established algorithms for fitting a SHMM to a data sequence. Other optimizing techniques must be used to fit a SHMM.
2 Mathematics of HMMs and SHMMs

2.1 HMM Definition

A HMM of a discrete data sequence\(^1\) can be described by two matrices of probabilities:

\[
a[j][i] = P( \text{state } j \text{ at time } t \mid \text{state } i \text{ at time } t-1), \quad (2.1)
\]

\[
b[k][j] = P( \text{observable output } k \mid \text{state } j ). \quad (2.2)
\]

The state transition matrix \( a \) has dimensions \( S \times S \), where \( S = \text{number of states in the model} \). The state output matrix \( b \) has dimensions \( K \times S \), where \( K = \text{number of discrete observable values} \). As each column of \( a \) and of \( b \) represents a complete set of probabilities, each column must sum to 1.

The model may also include a vector of \( S \) initial state probabilities:

\[
a0[j] = P( \text{state } j \mid t = 1 ). \quad (2.3)
\]

---

\(^1\) HMMs can also model sequences of continuous observables. These will not be treated here.
2.2 SHMM Definition

In our SHMMs, the probability elements of a vary periodically. Their variation is handled as follows: Each element of a matrix of relative frequencies is calculated as a product $c[j][i]h[j]$, where

$$c[j][i] = \text{a numerical scale factor relating predecessor and successor state } i \text{ and } j,$$  
(2.4)

$$h[j] = \text{a seasonal "shape" factor depending on time } t \text{ and on successor state } j.$$  
(2.5)

A circular normal function has been chosen for the seasonal factor:

$$h(t) = e^{g \left( \cos \left( 2\pi \frac{t - tp}{per} \right) - 1 \right)}$$  
(2.6)

where

$$per = \text{the period length},$$  
(2.7)

$$tp = \text{the phase at which } h(t) \text{ reaches its peak value of } 1$$  
(2.8)

$$g = \text{a shape parameter which determines the width of the peak.}$$  
(2.9)

For $g$ values less than 0, $h(t)$ has a shape near its peak very much like a linear normal curve. The minimum value of $h(t)$ of $e^{-2g}$ occurs $180^\circ$ from the peak. As $g$ approaches 0, $h(t)$ becomes a constant 1. This peak-value-normalized circular normal function can be calculated faster than the area-normalized version described by Johnson & Kotz[5].
The time-dependent matrix $a$ is calculated by normalizing each column of $c[j][i]h[j]$ relative frequencies to a sum of 1 to yield probabilities.

$$a[j][i] = \frac{c[j][i]h[j]}{\sum_{j=1}^{I} c[j][i]h[j]} \quad (2.10)$$

Table 2.1 summarizes the parameters which specify HMMs and SHMMs. A SHMM with $S$ states has only $2S$ more parameters than a HMM with $S$ states. These are the parameters which must be determined in fitting a HMM or SHMM to a data sequence. A small number of parameters is desirable to minimize overfitting the model to a given data sequence.

<table>
<thead>
<tr>
<th>parameters</th>
<th>HMM</th>
<th>SHMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[j][i]$</td>
<td>$S \times S$</td>
<td>$S \times S$</td>
</tr>
<tr>
<td>$c[j][i]$</td>
<td>$S \times S$</td>
<td>$S \times S$</td>
</tr>
<tr>
<td>$b[k][j]$</td>
<td>$S \times K$</td>
<td>$S \times K$</td>
</tr>
<tr>
<td>$g[i]$</td>
<td>$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>$t[p][j]$</td>
<td>$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>$a0[j]$</td>
<td>$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>total parameters</td>
<td>$S \times (S + K + 1)$</td>
<td>$S \times (S + K + 3)$</td>
</tr>
</tbody>
</table>

Table 2.1 HMM and SHMM parameters. Parameter counts for $S$ states and $K$ outputs

### 2.3 Likelihood

There are some problems in calculating the likelihood (or probability) that a given HMM or SHMM produces an observed data sequence.
2.4 Sequence Probability

The direct method of going through all possible state sequences and adding up their probabilities directly is impractical because, for \( S \) states and \( T \) steps, the number of state sequences \( S^T \) grows exponentially with \( T \). It is more practical to step through the data sequence while calculating the distribution of state probabilities \( p\text{st}[j] \) at each time \( t \). This may be done by cycling through the following steps.

Let

\[
p\text{st}[j] = P(\text{state } j \mid \text{ step } t) \text{ before observable output } y[t] \text{ is specified.} \quad (2.11)
\]

Then

\[
p\text{sy}[j] = P(\text{state } j, \text{ observable output } y[t] \mid \text{ step } t) = p\text{st}[j] b[y[t]][j], \quad (2.12)
\]

and

\[
p\text{yt} = P(\text{output } y[t] \mid \text{ time } t) = \sum_{j=1}^{S} p\text{st}[j] b[y[t]], \quad (2.13)
\]

After \( y[t] \) is specified, the state probabilities at time \( t \) are (by Bayes' relationship)

\[
P(\text{state } j \mid \text{ step } t, y[t]) = p\text{sy}[j] / p\text{yt}. \quad (2.14)
\]

Then for the successor state at \( t+1 \), before observable output \( y[t] \) is specified,

\[
p\text{st}[j] = P(\text{state } j \mid \text{ step } t) = \frac{\sum_{i=1}^{S} a[j|i] p\text{sy}[i]}{p\text{yt}}. \quad (2.15)
\]
2.5 Log Likelihood

The likelihood of the entire sequence is the product of the probabilities for each step. The product of many such probabilities will underflow a computer's floating point representation. Underflow can be avoided by accumulating

\[ \ln p = \sum_{t=1}^{T} \log(p(y_t)) \]  

(2.16)

The sequence log likelihood will scale linearly with sequence length. If we divide log likelihood \( \ln p \) by \( T \), we get a measure of information per sequence step. This value can provide for useful comparisons between sequences of different lengths. The geometric mean probability per step is

\[ mps = \exp(\ln p / T). \]  

(2.17)

The perplexity, \( 1/mps \), may be more is readily interpreted. Smaller perplexity corresponds to greater likelihood.
3 Fitting a SHMM to a Data Sequence

3.1 Difficulties Fitting SHMMs

A HMM of an observed data sequence is usually obtained via a Forward-Backward algorithm [1]. This procedure starts with an initial set of HMM parameters and iteratively calculates model parameters with improved likelihood. The best result from several initial parameter sets is often used since the procedure can only find a local optimum. Unfortunately, interactions of the SHMM parameters prevent application of a Forward-Backward algorithm. Another method must be found for adjusting SHMM model parameters.

Several optimization methods were explored in attempts to maximize the likelihood of SHMMs. Various problems were encountered with most methods because of constraints on the parameters and the logarithm function necessary in the likelihood evaluation.
All SHMM model parameters except $tp[j]$'s must be non-negative. Several methods (including some proprietary packages) attempt evaluations with out-of-bounds trial parameters. Such evaluations may abort optimization.

Each row of a model's state-to-state transition probability matrix $a$ and its state output probability matrix $b$ must add up to 1. Many optimization methods, which cope well with simple parameter bounds, cannot tolerate such equality constraints. The equality constraints cannot be eliminated algebraically since we do not know in advance which components may become 0 during the optimization calculations.

The equality constraints confine acceptable parameter sets to linear subspaces of the full unconstrained parameter ranges.

Furthermore, the $h[j](t)$ factors contribute to the $a$ matrix element values in a nonlinear manner. Only the ratios of $h[j](t)$'s really contribute. Experimental optimizations have produced models with very similar output sequence probabilities from very dissimilar sets of $h[j](t)$ functions. The corresponding acceptable parameter subspaces are very nonlinear. The mps gradient along such subspaces is zero. The zero-gradient
directions probably caused the failure of optimization by conjugate gradient methods.

3.2 Adaptive Simulated Annealing

Adaptive Simulated Annealing (ASA)[7], a method which required bounds on the parameters seemed promising. This method should be able to find the global optimum. ASA worked quite satisfactorily on 2 state models. It did not work well on models with more states. As the number of states increased, ASA produced progressively worse "optimized" models. Although the ASA package has a large number of adjustable parameters, I was unable to find a satisfactory combination of parameters.

3.3 Powell's Direction Set Method

Finally Powell's direction set method[8] for finding a local optimum was tried. Powell's direction set method successively improves a function of \( n \) variables along \( n \) initially orthogonal directions. The initial directions are simply the \( n \) unit vectors in each parameter direction. After improvement in all \( n \) directions, the direction vector that produced the largest improvement is (usually) replaced by a vector in the direction resulting from all \( n \) improvements. The sweeps through \( n \) directions repeat until the improvement becomes small. The method does not maintain the conjugate relationship among the \( n \) directions and they may become oriented in similar non-productive
directions. The method may then terminate prematurely. Fortunately the Numerical Recipes implementation counts the number of internal iterations. Our program saves the parameter set which powell() returns. If powell() has performed more than one internal iteration, our program restarts powell() up to 8 more times with the newest parameter set and a fresh set of initial unit direction vectors.

Powell's method with restarts finds only a local optimum, so each model type was run with 9 different random initial parameter sets. The best models improved with more states, but we should expect still further improvement with additional initial conditions.

Transformations Between Bounded and Unbounded Parameters

Powell's method requires unbounded parameters while most of the SHMM parameters are bounded. These conflicting requirements were accommodated by transformations between a set of bounded SHMM parameters and a set of unbounded optimizer parameters. The transformation from unbounded to bounded parameters initially yields relative frequencies. The relative frequencies are subsequently normalized to probability vectors summing to 1.

The following bounded/unbounded transformations can be calculated quickly in both directions:
The relationship is illustrated as the solid portions of the curve in Fig 3.1. It consists of two hyperbolic segments with the same slope at their common point.

Fig. 3.1 Bounded/Unbounded Transformation
4 Programs

Several programs were written to optimize SHMMs as well as to simply evaluate the geometric mean probability of a data sequence for a specified model. These programs, written in C, were compiled under gcc and run on several Sun Sparc workstations.

The program Makefiles, include files and source files are listed in the Appendix A.

4.1 Program shmmrpow

Program shmmrpow initializes a SHMM configuration with a randomly generated set of values and then calls subroutine powell to optimize the model. It saves the results and restarts powell with those results up to 8 times or until powell reports only one iteration. shmmrpow can reinitialize and repeat this procedure an arbitrary number of times. The major steps and subroutine of shmmrpow are outlined below:
shmmrpow() {
  readpow()
    read command line parameters
    data sequence, file name, T sequence length, P period,
    K number of output levels, S number of states,
    gmax upper limit for g parameters,
    results file name leading characters,
    number of fresh initializations of powell to run
  mpsinit()
    create static arrays for mps evaluations
    and for optimizing parameters
  read y data sequence from file

  for( start = 1; start <= nstarts; start++) {
    randomxmat()
      fill mps model arrays with random model

    for(repeat = 1; repeat <= 9; repeat++){
      open answer file
      xv2opt()
        convert bounded model parameters to
        unbounded optimizing parameters
        initialize direction matrix xi to identity matrix
        powell(powv, xi, ... , &iter, &value, func)
      writeans()
      if( iter <= 1) break;
    }
  }
}

func()
{
  optv2xv( ovec, xv );
    convert unbounded parameters to bounded model parameters
  return -mps(xv);     call mps() to evaluate likelihood per step, return
  value
}
Routine powell() and its subroutines linmin(), brent(), fldim(), mnbrak() from Numerical Recipes[9] are written to execute Powell's direction set procedure in float precision. The routines were compiled with double precision operations by writing a source code file powcomb.c which defines float as double, defines a few Numerical Recipes routines as double routines, and then includes the subroutines' source code as well as powell.c. The object code produced from powcomb.c replaces the original float variables and operations with doubles.

powell() is passed a set of unbounded optimizing parameters powv, a matrix xi specifying the optimization directions, and the function func(). powell() returns the number of iterations iter, and the optimized function value.

func() calculates the bounded mps() parameters corresponding to powell()'s unbounded parameter and then calls mps() to obtain the new function value.

The subroutine writans() converts the result of each run through powell() to SHMM parameters and displays or saves them in a compact ASCII format which can be read by other programs. The first line of the compact format includes the letters "T", "P", "K" and "S". Each of these is followed by an integer which gives the sequence length, model period, number of outputs, and number of states respectively.
The following rows list the model parameters. The parameters associated with each predecessor state are printed in one column. A one or two character identifier indicates the start of each set of SHMM parameters:

The “aO” row shows the initial state probabilities.

The “c” (or sometimes “a”) indicates the start of the state transition matrix.

A “b” indicates the start of the state output matrix.

An “f” (or sometimes “tp”) starts the row of seasonal factor peak position.

The fractional part indicates the fraction of the year of the peak.

The integer part is simply an artifact of the optimization details.

A “g” starts the row of seasonal shape parameters.

The next line gives the mps, 1/mps, mps bits, and the number of iterations from the last call to powell(). The last line reproduces the command line that called \texttt{shmmrpow} followed by the name of the file it recorded.

4.2 Program \texttt{shmmbrpo}

A similar program \texttt{shmmbrpo} optimizes SHMM models in which the data is treated as 4 independent binary sequences instead of a single sequence of 16 distinct values. Binary sequence models are not expected to capture the rainfall patterns as well as the 16-valued models.
4.3 Programs chkshmm, chshmmb

In addition, much simpler programs, chkshmm and chshmmb were written to read the ASCII SHMM files produced by shmmrpow and shmmbrpo and then evaluate \( mps \) for a given data sequence. Since the given data sequence and its starting point need bear no relationship to the model's sequence and its starting point, the model's initial state probabilities vector \( a_0 \) is ignored in this evaluation. Instead the initial state probabilities can be calculated as the largest eigenvector of the state-transition-matrix for the initial time. Since the largest eigenvalue should be 1, this eigenvector can be evaluated by extracting any column vector of values from a high power of \( a \) and normalizing the vector to sum to 1.

4.4 Programs mod2prob, mod2bpro

Another pair of simpler programs, mod2prob and mod2bpro, read ASCII SHMM files and calculate the values of the seasonal factors and the state and observable output probabilities at weekly intervals through the year. These probability vectors are simply eigenvectors of the state-to-state-transformation matrix corresponding to the eigenvalue 1. The probability vectors are then used to calculate the (quasistatic) probabilities of the observable outputs vs. time. The results can be plotted and compared with moving averages of the actual data sequences.
4.5 Program seqpairs

The program seqpairs counts the different values in a data sequence. It also constructs a table of the predecessor/successor value pairs in a data sequence. The counts generated by the program are useful in comparing sequences.
5 The Rain Gauge Data

The SHMM techniques are applied to a set of rain gauge data. The raw data consists of daily rain gauge readings at four weather stations in the St. Joseph River watershed above Cour d'Alene, Idaho. Nine years (3285 days) of daily readings are available. Figure 5.1 is a graph of the raw data for the four sites. The data begins with October 1982. The largest value, 5.1 inches, occurs on day 146 of year 5.

5.1 Missing Data

The breaks in the curves of Fig. 5.1 represent missing rain gauge readings. There are 187 missing readings. These make up 1.42% of all the readings. The missing rain gauge readings were filled in by copying data sequences from other years in which the available data looked most similar. The missing data could be more properly allowed for only with more complicated likelihood calculations. The complications would add nothing to the demonstration of SHMM methods.
Fig. 5.1 Raw Data, Annual Plots
5.2 Conversion to Discrete Data Sequences

Replacing each non-zero rain gauge reading with a 1 yields a daily binary sequence of 1's and 0's for precipitation and no precipitation for each site. Figure 5.2 graphs all four sites' "rainy" day 1's as black dots.
Fig. 5.2 Binary Sequences, Four Sites, Annual Plots
Figure 5.3 shows data sequence $y_{16}$, a discrete 16-level encoding of the 4 binary site sequences. Each of the 16 discrete levels corresponds to a different pattern of precipitation across the 4 sites. The code, shown in table I below, has been chosen so that a completely dry day is represented by $y[t] = 1$ and the wettest day is represented by $y[t] = 16$. Wetter conditions have higher 16 level code numbers.

Figure 5.3 shows that the all dry pattern $y = 1$ is most common. The next most common patterns are the all $y = 16$ (all wet) and $y = 12$ (a 3 wet site pattern).

The data can also be reduced to a sequence of daily counts of “wet” sites as shown in Fig. 5.4.
Fig. 5.3 16 Pattern Sequence, Annual Plots
Fig. 5.4 Number of "Wet" Sites Sequence, Annual Plots
A model of the data should be expected to show the following properties which can be seen by examining these graphs:

The conditions at the 4 sites tend to be correlated.

Dry and wet conditions tend to alternate over a period of a few to several days.

Summers are drier than winters.

The rain patterns vary substantially from year to year.

The daily precipitation patterns are encoded as a sequence of numbers from 1 to 16 according to the code of Table 5.1. This code is monotonic in the number of wet sites, i.e., an increase in $y$ never decreases the number of wet sites. This sequence aids in displaying "raininess" graphically.

Table 5.1 Rain Pattern Codes

<table>
<thead>
<tr>
<th>site</th>
<th>site</th>
<th>site</th>
<th>site</th>
<th># wet sites</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>
6 SHMM Results

6.1 Optimization Experiments

A variety of optimizations were carried out for both seasonal and non-seasonal hidden Markov models.

Optimization of non-seasonal HMMs was accomplished simply by setting \( g_{\text{max}} = 0 \). The SHMM runs were made with \( g_{\text{max}} = 1 \). Preliminary experimental runs showed the best optimizations had \( g \) values smaller than 1.0. Larger \( g_{\text{max}} \) values seemed to give poorer results.

SHMM and HMM optimizations were carried out for 2, 3, 4, 5, and 6 states. A 1 state HMM optimization was also carried out as a check. As expected the 1 state model yields probabilities which are simply proportional to the relative frequencies of the observed outputs. Optimizations were done for both the encoded 16 level sequence representation and for the 4 binary sequence representation.
For each model configuration, the results of starting with 9 different random sets of parameter values were accumulated. The variation in mean probability per step within each set of 9 runs indicates that the \( mps \) function probably has many local maxima. As the number of states increases, the calculation effort that must be expended to find a near-global optimum increases rapidly. Nevertheless, the number of fresh starts was not increased for larger numbers of states.

The model parameters of the best SHMMs are displayed in Appendix B in the compact ASCII format produced by the programs.

The model parameter files recorded by \texttt{shmmrpow} for the best of each set of the 9 SHMMs are listed in Appendix B. Careful examination and interpretation of these numerical results show that each includes a dry state which tends to persist for a few days. Each also includes a wet state that tends to persist for a few days. The remaining states of the larger models tend not to persist. These transient states may be auxiliary wet states, characteristic of the wet/dry transitions, or start-up states associated with the \( a_0 \) initial state probabilities. These optimized SHMMs tend to reflect the irregular cyclic alteration between wet and dry states which we observe in weather patterns of the area.
6.2 Optimized SHMMs and HMMs - 16 Level Models

Optimized models were obtained using the first 5 years of the data sequence for both the 16 level models using restarted Powell program \texttt{shmmrpos}. The models were checked with \texttt{chkshmm} against both the original 5 year training sequence and tested with the last 4 years of the data sequence.

The results are tabulated in table 6.1 below. The table lists the values of $1/\text{mps}$ simply because these seem to be easiest to compare and interpret. Smaller $1/\text{mps}$ values are better.

The table shows “training $1/\text{mps}$” values and “chk5yr $1/\text{mps}$”. Both values are calculated from the same data sequence. The training calculation uses $a_0[j]$ initial state probabilities while the “chk5yr” values calculate initial state probabilities from the $a$ matrix. The latter $1/\text{mps}$’s are slightly smaller but are more appropriate for comparison with the “chk4yr $1/\text{mps}$” values from the 4 year test data sequence. The $a_0[j]$ values are incorrect for a new data sequence.

Several interesting regularities can be seen in the results:

1. The 2 state models are better than the “1 state” model.
2. The best training $1/\text{mps}$ improves as more states are added.
3 The SHMM with S states is better than the HMM model with S+1 states but not as good as the HMM model with S+2 states.

4 Each model’s 4 year test values are consistently better than the 5 year values.

<table>
<thead>
<tr>
<th>states</th>
<th>model file</th>
<th>training mps</th>
<th>training bits</th>
<th>training 1/mps</th>
<th>chk5yr 1/mps</th>
<th>chk4yr 1/mps</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>1</td>
<td>0.0996</td>
<td>3.3272</td>
<td>10.0369</td>
<td>10.0379</td>
<td>9.3543</td>
</tr>
<tr>
<td>HMM</td>
<td>2</td>
<td>0.1210</td>
<td>3.0473</td>
<td>8.2664</td>
<td>8.2696</td>
<td>7.6207</td>
</tr>
<tr>
<td>SHMM</td>
<td>2</td>
<td>0.1221</td>
<td>3.0340</td>
<td>8.1910</td>
<td>8.1930</td>
<td>7.5344</td>
</tr>
<tr>
<td>HMM</td>
<td>3</td>
<td>0.1210</td>
<td>3.0468</td>
<td>8.2635</td>
<td>8.2677</td>
<td>7.4027</td>
</tr>
<tr>
<td>SHMM</td>
<td>3</td>
<td>0.1247</td>
<td>3.0036</td>
<td>8.0199</td>
<td>8.0219</td>
<td>7.4203</td>
</tr>
<tr>
<td>HMM</td>
<td>4</td>
<td>0.1244</td>
<td>3.0068</td>
<td>8.0377</td>
<td>8.0407</td>
<td>7.5069</td>
</tr>
<tr>
<td>SHMM</td>
<td>4</td>
<td>0.1269</td>
<td>2.9777</td>
<td>7.8775</td>
<td>7.8896</td>
<td>7.2032</td>
</tr>
<tr>
<td>HMM</td>
<td>5</td>
<td>0.1261</td>
<td>2.9869</td>
<td>7.9274</td>
<td>7.9302</td>
<td>7.5015</td>
</tr>
<tr>
<td>SHMM</td>
<td>5</td>
<td>0.1294</td>
<td>2.9500</td>
<td>7.7276</td>
<td>7.7300</td>
<td>7.2857</td>
</tr>
<tr>
<td>HMM</td>
<td>6</td>
<td>0.1275</td>
<td>2.9716</td>
<td>7.8438</td>
<td>7.8486</td>
<td>7.2065</td>
</tr>
<tr>
<td>SHMM</td>
<td>6</td>
<td>0.1300</td>
<td>2.9437</td>
<td>7.6940</td>
<td>7.6986</td>
<td>7.3353</td>
</tr>
<tr>
<td>HMM</td>
<td>9</td>
<td>0.1282</td>
<td>2.9632</td>
<td>7.7983</td>
<td>7.8036</td>
<td>7.6897</td>
</tr>
<tr>
<td>SHMM</td>
<td>9</td>
<td>0.1315</td>
<td>2.9269</td>
<td>7.6046</td>
<td>7.6086</td>
<td>7.7806</td>
</tr>
</tbody>
</table>

Table 6.1 Best of 9 Optimized SHMMs and HMMs - 5 Year Training

Another set of optimized models was created by training on the last 4 years of the data sequence. The best of these models are listed in Table 6.2 below. Five year results are again consistently poorer than the 4 year training and 4 year check values.

We conclude that the last 4 years of the data sequence is somehow “more typical” than the first 5 years. The first 5 years somehow exhibits more variation.
Although adding 2 states to a HMM appears to yield a better 1/\text{mps} value than a SHMM with the original number of states, it is misleading to assume the better value implies a more satisfactory model however, since the HMM cannot have consistent seasonal variation.

<table>
<thead>
<tr>
<th></th>
<th>states</th>
<th>model file</th>
<th>training mps</th>
<th>training bits</th>
<th>training 1/\text{mps}</th>
<th>chk4yr 1/\text{mps}</th>
<th>chk5yr 1/\text{mps}</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>1</td>
<td>y4s1g0i11.2</td>
<td>0.1098</td>
<td>3.1875</td>
<td>9.1100</td>
<td>9.1091</td>
<td>10.3111</td>
</tr>
<tr>
<td>HMM</td>
<td>2</td>
<td>y4s2g0i18.2</td>
<td>0.1373</td>
<td>2.8651</td>
<td>7.2858</td>
<td>7.2844</td>
<td>8.7199</td>
</tr>
<tr>
<td>SHMM</td>
<td>2</td>
<td>y4s2g1i11.2</td>
<td>0.1392</td>
<td>2.8452</td>
<td>7.1858</td>
<td>7.1923</td>
<td>8.6373</td>
</tr>
<tr>
<td>HMM</td>
<td>3</td>
<td>y4s3g0i18.3</td>
<td>0.1438</td>
<td>2.7977</td>
<td>6.9535</td>
<td>6.9581</td>
<td>8.5625</td>
</tr>
<tr>
<td>SHMM</td>
<td>3</td>
<td>y4s3g1i11.4</td>
<td>0.1460</td>
<td>2.7762</td>
<td>6.8504</td>
<td>6.8588</td>
<td>8.3629</td>
</tr>
<tr>
<td>HMM</td>
<td>4</td>
<td>y4s4g0i18.3</td>
<td>0.1459</td>
<td>2.7771</td>
<td>6.8549</td>
<td>6.8575</td>
<td>8.5108</td>
</tr>
<tr>
<td>SHMM</td>
<td>4</td>
<td>y4s4g1i18.3</td>
<td>0.1468</td>
<td>2.7681</td>
<td>6.8121</td>
<td>6.8092</td>
<td>8.4430</td>
</tr>
<tr>
<td>HMM</td>
<td>5</td>
<td>y4s5g0i11.3</td>
<td>0.1475</td>
<td>2.7614</td>
<td>6.7804</td>
<td>6.8121</td>
<td>8.6350</td>
</tr>
<tr>
<td>SHMM</td>
<td>5</td>
<td>y4s5g1i12.3</td>
<td>0.1513</td>
<td>2.7246</td>
<td>6.6098</td>
<td>6.6162</td>
<td>8.4127</td>
</tr>
<tr>
<td>HMM</td>
<td>6</td>
<td>y4s6g0i13.4</td>
<td>0.1489</td>
<td>2.7476</td>
<td>6.7161</td>
<td>6.7195</td>
<td>8.6619</td>
</tr>
<tr>
<td>SHMM</td>
<td>6</td>
<td>y4s6g1i13.9</td>
<td>0.1532</td>
<td>2.7062</td>
<td>6.5259</td>
<td>6.5263</td>
<td>8.6531</td>
</tr>
</tbody>
</table>

Table 6.2 Best of 9 Optimized SHMMs and HMMs - 4 Year Training

Program seqcalc was used to generate typical sequences from the models. A typical sequence generated by an optimized 2 state SHMM is displayed in Fig. 6.1.

Figure 6.2 displays a typical sequence from a 6 state HMM. Both graphs show irregular variations between wet and dry conditions over a few days but only the SHMM sequence shows wet winters and dry summers.
Fig. 6.2  A Typical Sequence Generated by 6 State HMM y5s6g0i3.2
6.3 Increasing the Number of States

Optimized SHMM models with more states fit the data sequence better. It is difficult to discern the improvement in the typical sequences generated from the models however. Figures 6.3 through 6.7 show how the state and output patterns of the best 5 year SHMMs vary through the year. The data for these curves were calculated with program mod2prob.

The curves of the 2 state SHMM all have the same symmetrical contours with the peak of one curve lining up with either the maximum or minimum of all the others. The curves in a 3 state SHMM show some asymmetries and can have different shapes. With more states, the pattern probability curves can exhibit still more variation. Although the moving average graph shows many short term jiggles, the overall match improves with more states.

The SHMM output probability curves may be compared with curves of the 61 day moving averages shown in Fig. 6.8. The seasonal variation of wet and dry conditions is clearly visible in all these curves. The all dry 0000 y1 pattern dominates in the summer. In the winter the all wet 1111 y16 and “3 wet” 0111 y12 pattern probabilities together become comparable to or larger than the y1 probability.
Fig. 6.3 Best 2 State SHMM Probabilities
Fig. 6.4 Best 3 State SHMM Probabilities
Fig. 6.5 Best 4 State SHMM Probabilities
Fig. 6.6 Best 5 State SHMM Probabilities
Fig. 6.7 Best 6 State SHMM Probabilities
Fig. 6.8  61 Day Moving Averages,  16 Patterns
7 Conclusions

7.1 Some Observations on the Nature of the Optimized SHMMs

Examination of the optimized SHMMs prompts some observations on potential improvements of the method.

In the 9 state SHMM of Appendix B, only 8 states actually contribute to the model. The state transition matrix row of all 0's cannot contribute to the model. We expect an active ninth state to make at least some small contribution. This observation supports the notion that searching the parameter space of larger models should be more thorough.

Appendix C lists the values obtained from the repeated fresh starting points. The last columns show the ranking of each of the runs as determined by the shmmrpow and by chkshmm for both the first 5 years and the last 4 years. Note that the 4 year
rankings differ from the 5 year rankings even though the 4 year values are consistently better than the 5 year values. The discrepancy can be regarded as the result of over-training. The optimizations are sometimes too good and the model fits the special details of the training data too well. Improved generalization could be achieved by periodically checking the model against the test data set as is commonly done in artificial neural network training.

A partial explanation for the ranking discrepancy showed up with an earlier version `shmmrpow` which rounded parameter values smaller than 0.00005 to 0. When `chkshmm` was run some the larger earlier version optimized SHMMs trained with the 4 year data, `chkshmm` failed by returning `NaN` (not a number) instead of a valid `mps` value. This occurred when `chkshmm` was required to take the logarithm of a zero probability. The current version of the program saves the small values in scientific format. This modification permits rare test sequences to have small non-zero probabilities.

The result of running program `seqpairs` on the first 5 years, the last 4 years, and the complete 9 years is shown in Appendix D. The program counts the number of occurrences of predecessor/successor output pairs in each data sequence and displays the results by columns of predecessors and rows of successors. All three `seqpair`
displays have counts of zero for some pairs. The zeros of the 5 year and 4 year sequences do not coincide. The zeros represent rare sub-sequences which may produce small elements in the state transition matrix.

The training set simply is not large enough to characterize and model rare events. This problem is probably inherent in the size of the available data sets.

7.2 Extending the Application of Non-Stationary HMMs

New approaches to fitting non-stationary SHMMs can be used to enlarge the application of non-stationary HMMs in the following ways:

Non-stationary HMMs may include dependence on predecessor data sequence values. Non-stationary HMMs may include dependence on other time series. For example, the HMM relationships might be exploited to construct a state model which in which the state and current values are used to predict whether a time series is in a generally increasing or decreasing mode.

Non-stationary HMMs may be trained to criteria other than the maximum likelihood of the entire sequence. The present maximum likelihood criterion puts equal
emphasis on each time step. A training function which tends to equalize the accuracy of each type of prediction may also prove useful.

### 7.3 Conclusions

Seasonal hidden Markov models have been described and fitted to given data sequences. The results show that hidden Markov models can be extended to a seasonal form that yields more representative models of non-stationary seasonal time series.

The forward-backward methods cannot be applied to the SHMMs. This problem required the development of techniques to fit SHMMs to a data sequence. Such methods may be used to adapt HMMs to other non-stationary dependencies and to other training functions.
Appendix A  Program Files

A1  shmmrpow Makefile

# shmmrpow  Makefile
# Powell's method with restarts

CC = gcc
CFLAGS = -g -0 $(WARNS) $(INCL) $(PMPS) $(FD)

WARNS = -Wall

LDFLAGS = -lmps -lm -Llib
INCL = -Iinclude -Isrc
INCLUDES = nrutil.h utilmps.h shmmcg.h

LIB = lib

SHMMRPOWOBJS = shmmrpow.o smps.o smpsinit.o readpowr.o writmod.o
writans.o \func.o xv2optv.o powcomb.o acalc.o randxmat.o

shmmrpow: $(SHMMRPOWOBJS) $(LIB)/libmps.a
   $(CC) $(CFLAGS) -o shmmrpow $(SHMMRPOWOBJS) $(LDFLAGS)

shmmrpow.o : src/shmmrpow.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o shmmrpow.o -c src/shmmrpow.c

smps.o : src/smps.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o smps.o -c src/smps.c

acalc.o : src/acalc.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o acalc.o -c src/acalc.c

randxmat.o : src/randxmat.c $(INCLUDES)
$(CC) $(CFLAGS) -DSHMM -o randxmat.o -c src/randxmat.c

smpsinit.o : src/smpsinit.c $(INCLUDES)
  $(CC) $(CFLAGS) -c src/smpsinit.c

readpowr.o : src/readpowr.c $(INCLUDES)
  $(CC) $(CFLAGS) -DSHMM -o readpowr.o -c src/readpowr.c

writmod.o : src/writmod.c $(INCLUDES)
  $(CC) $(CFLAGS) -DSHMM -o writmod.o -c src/writmod.c

writans.o : src/writans.c $(INCLUDES)
  $(CC) $(CFLAGS) -DSHMM -o writans.o -c src/writans.c

func.o : src/func.c $(INCLUDES)
  $(CC) $(CFLAGS) $(PMPS) -c src/func.c

xv2optv.o : src/xv2optv.c $(INCLUDES)
  $(CC) $(CFLAGS) -c src/xv2optv.c

powcomb.o : src/powcomb.c src/brent.c src/fldim.c src/mnbrak.c \ src/linmin.c src/powell.c
  $(CC) $(CFLAGS) -c src/powcomb.c

$(INCLUDES) :

# mps library
$(LIB)/libmps.a : src/utilmps.c src/nrutil.c $(INCLUDES)
  $(CC) $(CFLAGS) -c src/nrutil.c
  $(CC) $(CFLAGS) -c src/utilmps.c
  $(CC) $(CFLAGS) -c src/ranl.c
  ar r $(LIB)/libmps.a ranl.o
  ar r $(LIB)/libmps.a utilmps.o
  ar r $(LIB)/libmps.a nrutil.o
  ranlib $(LIB)/libmps.a
  -rm -f utilmps.o nrutil.o

clean:
  rm -f *.o *run # core gmon.out

realclean:
  rm -f *.o *run user_out asa_out* gmon.out # core

cleanlib :
  -rm -f lib/*
A2 shmmrpow shmm.h include file

/****** BEGIN shmm.h FILE ******/
/* shmm.h header file for shmmrpow.c */

#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "syscalls.h"
#include "utilmps.h"
#include "nrutil.h"

#define float double

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#define TWOPI 6.283185307179586476925
#define GMAX 10.0
#define SMALL 1.0e-18

#ifndef NR_END
#define NR-END 1
#endif

long tstart, tstop;

/* model parameters */
int S; /* number of states */
int K; /* number of observables */
long per; /* period length */
long T; /* sequence length */

double gmax;
long t;

double **xmat, /* matrix address of mps() vector xv */
*xv, /*SHMM model [1..npar]:
a0[S], c[S][S], b[K][S], tp[S], g[S] */
*a0, /* [1..S] */
**c, /* [1..S] [1..S] */
**b, /* [1..K] [1..S] */
*tp, /* [1..S] */
*g, /* [1..S] */
*psy, /* [1..S] */
*pst, /* [1..S] */
*ppst, /* [1..S] */
**h, /* [0..T-1][1..S] */
***a; /* [0..T-1][1..S][1..S] */

#endif

int *y; /* observable sequence y[1..T] integers */
#else
float *y; /* observable sequence y[1..T] floats */
#endif

int yt;
int npar, nrows;
double *ovec, *dovec, *ht, **xi;

FILE *fpmod, *fpseq, *fpout;
char *outf;

int start, nstarts;
char *seqnam, *filhead;
char readmodbuf[258];

void readmod(int argc, char *argv[]);
void readpow(int argc, char *argv[]);
void readpars();
void writmod(double *x, FILE *fp);

double mps();
void mpsinit();
void acalc();

/* for powell */
#define FTOL le-6
double *pow;
void powell(float *p, float **xi, int n, float fort, int *iter, float *fret,
        float (*func)(float []));
double func( double *ovec );

/* bounded/unbounded transformations */
void xv2optv( double *xv, double *optv);
void optv2xv(double *optv, double *xv);

void randomxmat( double **xmat, int K, int S, double gmax, long *seed);

void writans (FILE *fpout, double value, int iter,
int argc, char *argv[], char *filnam);

/* for Numerical Recipes random number generator ranl() */
long *seed;
float ranl(long *idum);

/******* END shmm.h FILE *******/
A3 shmmrpow source files

```c
#include "shmmrpow.h"

int main(int argc, char *argv[])
{
    int repeat, iter, j;
    double value;
    char filnam[256];

    readpow( argc, argv);

    if(NULL == (fpseq = fopen(seqnam,"rt"))) {
        fprintf(stderr,
            "Can't open sequence data file %s
", argv[1]);
        exit(5);
    }

    #ifndef YFLOAT
    int *pv;
    #else
    float *pv;
    #endif
    
    if(1 != fscanf(fpseq,"%d", ++pv) ) {
        fprintf(stderr,
            "Can't read sequence data from file %s \n", argv[1]);
        exit(2);
    }

    #else
    y = (int *) ivector(l,T);
    for (pv=y, t=T; t; t--)
        if(1 != fscanf(fpseq,"%d", ++pv) ) {
            fprintf(stderr,
                "Can't read sequence data from file %s \n", argv[1]);
            exit(2);
        }

    #endif
    
    for (pv=y, t=T; t; t--)
        if(1 != fscanf(fpseq,"%lg", ++pv) ) {
            fprintf(stderr,
                "Can't read sequence data from file %s \n", argv[1]);
            exit(2);
        }
```
exit(2);
}
#endif
fclose(fpseq);

for(start=1; start<=nstarts; start++) {
#ifdef YFLOAT
K = 0;
#endif
randomxmat( xmat, K, S, gmmax, seed);

for(repeat=1; repeat <= 9; repeat++) {
    if(0 == strcmp(argv[7],"stdout"))
        fpout = stdout;
    else {
        sprintf(filnam,"%-d.%-d", filhead, start, repeat);
        fprintf(stderr,"Can't open output file %s",filnam);
        break;
    }
}

xv2optv( xv, powv);

filldvec(xi[1], 0.0, npar*npar);
for (j=1;j<=npar;j++)
    xi[j][j] = 1.0;
/* initialize xi = identity matrtix */
powell( powv, xi, npar, FTOL, &iter, &value, func );

writans( fpout, value, iter, argc, argv, filnam );
fclose( fpout );
if ( iter <= 1 || repeat >= 9 )
    break;
}
return 0;
}

/******* END shmmrpow.c FILE *******/

/******* BEGIN readpowr.c FILE *******
/* readpowr.c initiation routine, for
 * shmmrpow()'s mps.c
* reads starting model file
* allocates memory for model data
* & sequence data
* reads sequence data file
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "nrutil.h"
#include "shrnrnpow.h"

void readmoderr() {
    fprintf(stderr,"Can't read model file\n");
    exit(1);
}

void readpow(int argc, char *argv[])
{
    if(argc<9) {
        fprintf(stderr,
            " USAGE: %s seq_file T P K S grnax filhead nstarts
[seed]\n" "\tseq_file = data file to model\n" "\tT = sequence length (time steps)\n" "\tP = period steps\n" "\tK = # outputs\n" "\tS = # states\n" "\tgmax = upper bound on g\n" "\tfilhead = start of output file names\n" "\tnstarts = number of fresh starts\n" "\tseed = random number generator seed (default=1)\n",
        argv[9]);
        exit(3);
    }

    seqnam = argv[1];
    T = atoi(argv[2]);
    per = atoi(argv[3]);
    K = atoi(argv[4]);
    S = atoi(argv[5]);
    grnax = atof(argv[6]);
    filhead = argv[7];
    nstarts = atoi(argv[8]);

    seed = malloc(sizeof(long));
    if(argc<10)
        *seed = 1L;
    else
        *seed = atol(argv[9]);

    nrows = 3+S+K;

npar = S*nrows;
mpsinit();
}

/******** END readpwr.c FILE ********/

/******** BEGIN smpsinit.c FILE ********/
/* smpsinit.c initializes array variables for smps.c shmmpow.c */

#include "nrutil.h"
#include "utilmps.h"
#include "shmmpow.h"

void mpsinit() {
    int j;
    
xmat = dmatrix( 1, S+K+3, 1, S);
xv = xmat[1];
a0 = xv;
c = xmat + 1;
b = xmat + S+1;
tp = xmat[2];
g = xmat[S+K+3];
s = d3d( 0, per-1, 1, S, 1, S);

npar = S*(3+S+K);

psy = dvector( 1, S);
pst = dvector( 1, S);
ppst = dvector( 1, S);
powv = dvector( 1, npar);
xi = dmatrix(l,npar,1,npar);
filldvec(xi[l], 0.0, npar*npar);
for (j=1;j<=npar;j++)
    xi[j][j] = 1.0; /* xi = identity matrrix */
}

/******** END smpsinit.c FILE ********/

/******** BEGIN randxmat.c FILE ********/
/* randxmat.c loads mps(xv)'s model vector xv with a random model */
/* call with K = 0 for continuous output probability models */
/* with no b[] matrix */

#ifndef NR_END
#define NR-END 1
#endif

float ranl(long *idum);
void randomxmat( double **xmat, int K, int S,  
    double gmax, long *seed) {  

    int n;
    double *vec = xmat[1] + NR_END;
    float temp;

    for( n= S*(S+K+2); n; n--, vec++) {
        /* aO[0], c[][], b[][], tp[] */
        temp = ranl(seed);
        *vec = temp;
    }

    for( n=S; n; n--, vec++) {
        temp = ranl(seed);
        *vec = gmax * temp;
    }

    /* initial normalization:  
       1 a0 row, S c columns, S b columns  
    */
    normdvec( xmat[1], 1, S);

    vec = xmat[2];
    for(n=S; n; n--, vec++)
        normdvec(vec, S, S);

    vec = xmat[2+S];
    for(n=S; n; n--, vec++)
        normdvec(vec, S, K);

} /* END randxmat.c */

******* BEGIN FILE *******
/* xv2optv.c routines for one-to-one  
* transformations between a SHMM parameter  
* vector xvec[] and vector optv[] of infinite  
* range parameters.  
*  
* A routine to optimizer gradient derivatives  
* dovec[] from the xvec derivatives is also here.  
*  
* The vector and matrix are in Numerical Recipes  
* formats.  
* The vector is suitable for manipulation by  
* Numerical Recipes' optimizer routines  
* fail with unbounded ranges.  
*  
* The transformation function is a composite of  
* two rectangular hyperbolic segments with  
* matching values and slopes at the transition  
* point. Each hyperbolic segment is asymptotic to  
* a finite range limit.  
*/
* The xvec to cg transformation scales the
  * probabilities to properly sum to 1
  *
  * Minimum *xvec is SMALL
  */

#include <math.h>
#include "shmmpow.h"

inline double fin2inf(double f) {
    return f>0.5 ? (0.5/(1.0-f)-1.0) : (1.0-0.5/f);
}

inline double inf2fin(double i) {
    return i>0.0 ? 1.0-0.5/(1.0+i) : 0.5/(1.0-i);
}

inline double fin2dfdi(double f) {
    if (f>0.5)
        f -= 1;
    return 2.0*f*f;
}

void xv2optv( double *xv, double *optv) {
    double *pc, *pb;
    int j;
    normdvec(aO, 1, S);
    pc = c[1];
    pb = b[1];
    for( j=0; j<S; j++ ) {
        normdvec( pc++, S, S);
        normdvec( pb++, S, K);
    }
    optv += NR_END;
    xv += NR_END;
    for( j=npar-S-S; j--; optv++, xv++ )
        /* from range (0,1) */
        *optv = fin2inf( *xv );
    for( j=S ; j--; optv++, xv++ )
        /* phases, no bounds */
        *optv = *xv;
    for( j=S ; j--; optv++, xv++ )
        /* g range (0,gmax) */
        if( *xv < SMALL )
            *xv=SMALL;
        *optv= fin2inf( *xv/gmax );
}
void optv2xv(double *optv, double *xv)
{
    double *pc, *pb;
    int j;

    optv += NR_END;
    xv += NR_END;
    for( j=npar-S-S; j; j--, optv++, xv++ )
        /* a0[],c[],b[] range [0,1] */
        \*xv = inf2fin(\*optv);
    for( j=S; j; j--, optv++, xv++ )
        /* tp[] phase ranges [0,1] */
        /* \*xv = fmod( \*optv, 1.0 ); */
        \*xv = *optv;
    for( j=S; j; j--, optv++, xv++ ) /* g[] range (0,gmax) */
        \*xv = gmax * inf2fin( \*optv);
}

#include "powcomb.c"

/******** END powcomb.c FILE ********/

/******** BEGIN powcomb.c FILE ********/
/* powcomb.c a collection of powell source code 
* with all floats redefined as doubles!
*/
/\* convert Numerical Recipes floats to doubles */
#define float double
#define vector dvector
#define free vector free dvector

static float maxarg1,maxarg2;
#define MAX(a,b) maxarg1=(a),maxarg2=(b), \n        (maxarg1) > (maxarg2) ? (maxarg1) : (maxarg2))

#include "shmpow.h"
#include "brent.c"
#include "f1dim.c"
#include "mnbrak.c"
#include "linmin.c"
#include "powell.c"

/******** END powcomb.c FILE ********/
/******** BEGIN writans.c FILE ********/
/* writans.c
 * saves results of an optimization sequence
 */
#include "shmmpow.h"

void writans(FILE *fpout, double value, int iter,
             int argc, char *argv[], char *filnam) { int j;
    optv2xv(powv, xv);
    writmod(xv, fpout);
    fprintf(fpout,
            "# mps %-8g 1/%-8g bits %-8g iters %2d\n",
            -value, -1.0/value,
            -log(-value)/log(2.0), iter);
    fprintf(fpout,"# %s",argv[0]);
    for(j=1; j<argc; ++j)
        fprintf(fpout," %s",argv[j]);
    fprintf(fpout," %s\n",filnam);
    fflush(fpout);
}
/******** END writans.c FILE ********/

/******* BEGIN writmod.c FILE *******/
/* writmod.c prints model parameters in
 * vector xv to stdout
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "shmmpow.h"

inline void printdvec(FILE *fp, double *xv, int i)
{
    for( ; i--; xv++)
        fprintf(fp,
                *xv>0.0010 ? "\t%-6.4f" : *xv!=0.0 ? "\t%-7.1e" : "\t0",
                *xv);
    fprintf(fp, "\n");
}

void writmod(double *xv, FILE *fp)
{
int j;

xv += NR_END;

fprintf(fp,
" T %ld P %ld K %d S %d
",
T, per, K, S);

fprintf(fp,"a0");
printdvec(fp,xv,S); xv += S;

fprintf(fp,"c ");
for(j=S; j; j--) {
  printdvec(fp,xv,S); xv += S; j
}

fprintf(fp,"b ");
for(j=K; j; j--) {
  printdvec(fp,xv,S); xv += S; j
}

fprintf(fp,"f ");
printdvec(fp,xv,S); xv += S;

fprintf(fp,"g ");
printdvec(fp,xv,S);
}

/ * END writmod. FILE */

/********** BEGIN func.c FILE **********/
/* func.c translates vector pow to vector xmat
* evaluates & returns -mps(xmat)
*/

#include "shmmpow.h"

double func( double *ovec )
{
  double value;

  optv2xv( ovec, xv ); /* changed to local vectptr pointer ovec */
  value = mps(xv);

  #ifdef MPSOUT
    fprintf(fpout, " func mps %-16.14f\n",value);
  #endif /* MPSOUT */

  return -value;
}

/********** END func.c FILE **********/
calculates 1/mps for SHMM seasonal hidden Markov model

mps = ( P(y[1],...y[T] | SHMM) )^(1/T)

geometric mean probability per sequence step

If mps() is starting at a time tstart other than the time used in calculating a given model, the a0[] initial state probabilities should be calculated as the eigenvector of the a[] state transition probability vector for eigenvalue = 1.

SHMM parameters:

a0[state] = P(state at t=1)

c[state][pred] = scalar weighting pred_state to state transition frequency

b[yt][state] = P(output yt | state)

tp[state] = phase of maximum of state seasonal factor h[t][j]

g[state] = "variance" parameter of h[t][state]

where

a[state][pred] = P(state at t ; pred_state at t-1) = c[state][pred] * h[t][state] / sum_i (c[state][pred]*h[t][state])

h[t][state][tp[state],g[state]] = periodic factor weighting of pred_state to state frequency

a0, c, b, tp, g arrays are mapped to x, a vector of npar SHMM parameters

pst[state] = P(state | y[t], SHMM)

= a0[state] at t=1

= sum_pred( a[state][pred] * psy[pred] ) for 2 <= t <=T

psy[state] = P(state, y[t])

= P(state) * P(y / state)

pyt = P(y[t]) = sum_over_states(psy[state])

ppst[state] = P(state | y[t])

/

#include <math.h>
#include "utilmps.h"
#include "shmmpow.h"

double mps(double *xv)
{
    double lnp, pyt, value;
    long t;

    acalc(); /* pre-calc a[][] */
```c
#ifndef TSTARTSTOP
  tstart = l;
  tstop = T;
#endif

#define A2A0

a2a0();
#endif

void a2a0() {
  copydvec(a0, pst, S);
  lnp = 0.0;
  for (t = tstart; TRUE; t++) {
    pyt = ddotprod(b[y[t]], pst, psy, S);
    /* ddotprod also yields vector psy of
term by term products */
    /* pyt = P(y[t]), psy[j] = P(state j, y[t]) */
    lnp += log(pyt);
    if (t >= tstop)
      break;
    scaledvec! psy, 1.0/pyt, S; /* psy[j] = P(state, y[t] =
psy[j]/pyt */
    dmatxvec(a[t % per], psy, pst, S, S); /* pst[j]=P(state j @ t+1 |
y[t]) */
  }
}

void a2a0() {
  long t, row, col;
  double **at, theta, *acol, h;
  for(t=0; t<per; t++) {
    at = a[t];
    for (row=1; row<=S; row++) {
      theta = ((double)t / (double)per - tp[row]) * TWOPI;
      h = exp(g[row] * (cos(theta) - 1.0));
      dvxscal( c[row], h, at[row], S);
      /* at[j][i] <== c[j][i]*h[j] */
  }
```
for( acol=at[1], col=0; col<S; col++)
    normdvec( acol++, S, S);
    /* normalize at[j][i] */
A4 shmmbrpo.c Makefile

/******* BEGIN shmmbrpo Makefile FILE *******/

# Makefile - powell's method
# shmmbrpo multiple binary sequence representation

CC = gcc
CFLAGS = -g -o $(WARNS) $(INCL) $(PMPS) $(FD)
WARNS = -Wall
LDFLAGS = -lmps -lm -Llib
INCL = -Iinclude -Isrc
INCLUDES = nrutil.h utilmps.h shmmcg.h
LIB = lib
CHKOPT = -DA2A0 -DTSTARTSTOP

SHMMBRPOP0OBJS = shmrnpow.o smpsb.o smpscinit.o \
    readmod.o writmod.o writans.o \
    func.o xv2optvb.o powcomb.o acalc.o randxmat.o

shmmbrpo: $(SHMMBRPOP0OBJS) $(LIB)/libmps.a \
    $(CC) $(CFLAGS) -o shmmbrpo $(SHMMBRPOP0OBJS) $(LDFLAGS)

chkshmmb : $(CHKSHPMMOBJS) $(LIB)/libmps.a \
    $(CC) $(CFLAGS) $(CHKOPT) -o chkshmmb $(CHKSHPMMOBJS) $(LDFLAGS)

chkacalc : src/chkacalc.c readcg.o smpsarcg.o src/acalc2.c \
    $(LIB)/libmps.a \
    $(CC) $(CFLAGS) -DSHMM -o chkacalc src/chkacalc.c readcg.o \
    smpsarcg.o src/acalc2.c $(LDFLAGS)

shmmbrpo.o : src/shmmbrpo.c $(INCLUDES) \
    $(CC) $(CFLAGS) -DSHMM -o shmmbrpo.o -c \src/shmmbrpo.c

smpsb.o : src/smpsb.c $(INCLUDES) \
    $(CC) $(CFLAGS) -DSHMM -o smpsb.o -c \src/smpsb.c

smpsc.o : src/smpsc.c $(INCLUDES) \
    $(CC) $(CFLAGS) -DSHMM -DYFLOAT -o smpsc.o \-c src/smpsc.c

acalc.o : src/acalc.c $(INCLUDES) \
    $(CC) $(CFLAGS) -DSHMM -o acalc.o -c \src/acalc.c
randxmat.o : src/randxmat.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o randxmat.o -c src/randxmat.c

smpsinit.o : src/smpsinit.c $(INCLUDES)
   $(CC) $(CFLAGS) -c src/smpsinit.c

readmod.o : src/readmod.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o readmod.o -c src/readmod.c

writmod.o : src/writmod.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o writmod.o -c src/writmod.c

writans.o : src/writans.c $(INCLUDES)
   $(CC) $(CFLAGS) -DSHMM -o writans.o -c src/writans.c

func.o : src/func.c $(INCLUDES)
   $(CC) $(CFLAGS) $(PMPS) -c src/func.c

xv2optvb.o : src/xv2optvb.c $(INCLUDES)
   $(CC) $(CFLAGS) -c src/xv2optvb.c

powcomb.o : src/powcomb.c src/brent.c \ 
   src/fldim.c src/mnbrak.c src/linmin.c \ 
   src/powell.c
   $(CC) $(CFLAGS) -c src/powcomb.c

chkshmm.o : src/chkshmm.c $(INCLUDES)
   $(CC) $(CFLAGS) $(CHKOPT) -o chkshmm.o \ 
   -c src/chkshmm.c

chkrdmod.o : src/chkrdmod.c $(INCLUDES)
   $(CC) $(CFLAGS) $(CHKOPT) -o chkrdmod.o \ 
   -c src/chkrdmod.c

a2a0.o : src/a2a0.c $(LIB)/libmps.a $(INCLUDES)
   $(CC) $(CFLAGS) -o a2a0.o -c src/a2a0.c

$(INCLUDES) :

# mps library
$(LIB)/libmps.a : src/utilmps.c src/nrutil.c $(INCLUDES)
   $(CC) $(CFLAGS) -c src/nrutil.c
   $(CC) $(CFLAGS) -c src/utilmps.c
   $(CC) $(CFLAGS) -c src/ran1.c
   ar r $(LIB)/libmps.a ran1.o
   ar r $(LIB)/libmps.a utilmps.o
   ar r $(LIB)/libmps.a nrutil.o
   ranlib $(LIB)/libmps.a
   -rm -f utilmps.o nrutil.o
clean:
    rm -f *.o *run # core gmon.out

realclean:
    rm -f *.o *run user_out asa_out* gmon.out # core

cleanlib :
    -rm -f lib/*
/******* END shmbro Makefile FILE *******/
A5 shmmbrpo shmm.h include file

/****** BEGIN shmm.h FILE ******/

/* shmm.h header file for smpsb.c */

#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "syscalls.h"
#include "utilmps.h"
#include "nrutil.h"

#define float double

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define TWOPI 6.283185307179586476925
#define GMAX 10.0
#define SMALL 1.0e-18

#endif NR-END

long tstart, tstop;

/* model parameters */
int S, /* number of states */
K; /* number of observables */
long per, /* period length */
T; /* sequence length */

double gmax;
long t;

double **xmat, /* matrix addressing of mps() vector xv */
*xv, /* SHMM model [1..npar]: a0[S], c[S][S], b[K][S], tp[S],
g[S] */
*a0, /* [1..S] */
**c, /* [1..S] [1..S] */
**b, /* [1..K] [1..S] */
*tp, /* [1..S] */
*g, /* [1..S] */
*psy, /* [1..S] */
*pst, /* [1..S] */
*ppst, /* 1..S */
**h, /* [0..T-1][1..S] */
***a; /* [0..T-1][1..S][1..S] */

#ifndef YFLOAT
int *y; /* observable sequence y[1..T] integers */
#else
float *y; /* observable sequence y[1..T] floats */
#endif
int yt;
int npar, nrows;
double *cvec, *dvec, *ht, **xi;
FILE *fpmod, *fpseq, *fpout;
char *outf;

int start, nstarts;
char *seqnam, *filhead;
char readmodbuf[258];

void readmod(int argc, char *argv[]);
void readpars();
void writmod(double *x, FILE *fp);
double mps();
void mpsinit();
void acalc();

/* the following are special for powell */
#define FTOL 1e-6

double *powv;

void xv2optv(double *xv, double *optv);
void optv2xv(double *optv, double *xv);
void powell(float *p, float **xi, int n, float ftol, int *iter, float *fret,
           float (*func)(float[]));

double func(double *ovec);
void randomxmat(double **xmat, int K, int S, double gmax, long *seed);

void writans(FILE *fpout, double value, int iter,
             int argc, char *argv[], char *filnam);

/* for random number generator ran1() */
long *seed;
float ranl(long *idum);

/* special for smpsc.c */
double *ystdev, *ymean;

/* for chkshmmb */
void a2a0();

/***** END shmmb.h FILE *******/*
A6  shmmbrpo  source files

(differing from shmmmmpow source files)

/****** BEGIN xv2optvb.c FILE ******/

/* xv2optvb.c
*  routines for one-to-one
*  transformations between a SHMM
*  parameter vector xvec[] and vector
*  optv[] of infinite range parameters.
*
*  A routine to optimize gradient derivatives
dovec[] from the xvec derivatives is also
*  included here.
*
*  The vector and matrix are in Numerical
*  Recipes formats. The vector optv[] is suitable
*  for manipulation by Numerical Recipes'
*  optimizer routines which fail with unbounded
*  ranges.
*
*  The transformation function is a composite of
*  two rectangular hyperbolic segments with
*  matching values and slopes at the transition
*  point. Each hyperbolic segment is asymptotic
*  to a finite range limit.
*
*  The xvec to cg transformation scales the
*  probabilities to properly sum to 1
*
*  Minimum *xvec is SMALL
*/

#include <math.h>
#include "shmmpow.h"

#include <math.h>
#include "shmmpow.h"

inline double fin2inf(double f) {
    return f>0.5 ? (0.5/(1.0-f)-1.0) : (1.0-0.5/f);
}

inline double inf2fin(double i) {
    return i>0.0 ? 1.0-0.5/(1.0+i) : 0.5/(1.0-i);
}

inline double fin2dfdi(double f) {
    if( f>0.5)
        f -= 1;
    return 2.0*f*f;
}

void xv2optv( double *xv, double *optv)
double *pc;
int j;

normdvec(a0, 1, S);
pc = c[1];
for (j=0; j<S; j++)
    normdvec(pc++, S, S);

optv += NR_END;
xv += NR_END;
for( j=npar-S-S; j; j--, optv++, xv++ ) /* from range (0,1) */
    *optv = fin2inf(*xv);

for( j=S ; j--; optv++, xv++ ; /* phases, no bounds */
    *optv = *xv;

for( j=S ; j--; optv++, xv++ ) { /* g range (0,gmax) */
    *optv = gmax<=0.0 ? 0.0 : fin2inf(*xv/gmax);
}

void optv2xv(double *optv, double *xv)
{
    double *pd;
    int j;

    pd = xv;    /* pd = start a0 vector */
    optv += NR_END;
xv += NR_END;
for( j=npar-S-S; j; j--, optv++, xv++ ) /* a0[1], c[i][j], b[j][i] range [0,1] */
    *xv = inf2fin(*optv);

for( j=S ; j--; optv++, xv++ )/* tp[j] phase ranges [0,1] */
    /* *xv = fmod( *optv, 1.0 ); */
    *xv = *optv;

for( j=S ; j--; optv++, xv++ )/* g[j] range (0,gmax) */
    *xv = gmax<=0.0 ? 0.0 : gmax * inf2fin(*optv);

normdvec(pd, 1, S);    /* normalize a0 */
pd += S;    /* pd = start c matrix */
for( j=S; j--; pd++ )
    normdvec(pd, S, S);    /* normalize c[i][j] columns */
}

/******** END xv2optv.c FILE ********/
/********* BEGIN smpsb.c FILE *********
/* smpsb.c calculates l/mps for SHMM seasonal hidden Markov model
* mps = ( P(y[1],...y[T] | SHMM) )^1/T
* = geometric mean probability per
* sequence step
* smpsb interprets y[t] as an encoding of K binary
* sequences
* b[k][state] =
* P(binary sequence k) = 1 | state)
* y[t][state] = 16 level encoding of state
* NOTE: This modified mps function requires a
* x2xmatb.c in place of x2xmat.c to transform
* between the now independent b[]][]'s and the
* optimization parameters.
* If mps() is starting at a time tstart other
* than the time used in calculating a model, the
* a0[] initial state probabilities should be
* calculated as the eigenvector of the
* a[]][] state transition probability vector for
* the eigenvalue = 1.
* SHMM parameters:
* S = number of states
* K = number of sites = 4
* per = period of sequence
* T = length of sequence
* a0[state] = P(state at t=1)
* c[state][pred] = scalar weighting predecessor
* to state transition frequency
* b[site][state] = P(rain at site | state)
* tp[state] = phase of maximum of state
* g[state] = "variance" parameter of
* seasonal factor h[t][state]
* calculated:
* a[succ][pred] =
* P(state at t | predecessor state at t-1)
* = c[succ][pred] * h[t][state] /
* sum over pred (c[succ][pred] * h[t][state])
* h[t][state] = periodic factor weighting of predecessor to
* state transition frequency
* pst[state] = P(state | t, SHMM)
* = a0[state] for t=1
* = sum_pred{ a[state][pred] *
* psy[pred] for 2 <= t <= T
* psy[state] = P(state, yt | t) at start of *
* iteration
* = P(state | y[t]) at end of iteration
* pyt = P(y[t]); density
#include <math.h>
#include "shmmpow.h"

void acalc();

double mps ()
{
    double psty, /* P(state, y[t]) */
    pyt, /* P( y[t] ) */
    lnp;
    long t;
    int state, site, *rainyt;
    static int rain[17][5] =
    {{9,9,9,9,9}, {9,0,0,0,0}, {9,1,0,0,0}, {9,0,1,0,0}, {9,0,0,1,0},
    {9,0,0,0,1}, {9,1,1,0,0}, {9,1,0,1,0}, {9,1,0,0,1},
    {9,0,1,0,1}, {9,0,0,1,1}, {9,0,0,0,1}, {9,1,1,1,0},
    {9,1,1,0,1}, {9,1,0,1,1}, {9,0,1,1,1}, {9,1,1,1,1}};
    /* rain[y[t]][site] site encoding */

    acalc();

    ifndef TSTARTSTOP
    tstart=1;
    tstop = T;
    #endif

    ifndef A2A0
    a2a0();
    #endif

    copydvec(a0, pst, S);

    lnp = 0.0;
    for( t=tstart; t<=tstop; t++) {  /****** t = tstart to tstop ******/
        rainyt = rain[y[t]];
        for( pyt = 0.0, state=S; state; state--) {
            psty = pst[state];
            for( site=K; site; site--)
                psty *= rainyt[site] ? b[site][state] : (1.0-b[site][state]);
            pyt += psty; /* psy[state] = P(state, y[t]) */
        }
        lnp += log(pyt); /* accumulate ln(pyt = P(y[t])) */
    }
    scaledvec( psy, 1.0/pyt, S); /* psy[j] = P(state j @ t = y[t]) */
    dmatxvec( a[t % per], psy, pst, S, S); /* pst[state] = P( state @ t+1 | y[t] ) */
    return exp(lnp/(tstop-tstart+1));
}

/****** END smseb.c FILE *****/
A7 seqlcalc  Makefile, source & include files

/****** BEGIN MAKEFILE **********/
# Makefile - seqlcalc utilities

CC = gcc
CFLAGS = -g $(WARNS) $(INCL) $(PMPS)

WARNS = -Wall

LDFLAGS = -lmps -lm -Llib
INCL = -Iinclude -Isrc
INCLUDES = nrutil.h utilmps.h shmmcg.h

LIB = lib

seqlcalc: seqlcalc.o acalc.o smpsinit.o seqrdmod.o a2a0.o
$(LIB)/libmps.a
 $(CC) $(CFLAGS) -o seqlcalc seqlcalc.o acalc.o smpsinit.o \ seqrdmod.o a2a0.o $(LDFLAGS)

seqbcalc: seqbcalc.o acalc.o smpsinit.o seqrdmod.o a2a0.o
$(LIB)/libmps.a
 $(CC) $(CFLAGS) -o seqbcalc seqbcalc.o acalc.o smpsinit.o \ seqrdmod.o a2a0.o $(LDFLAGS)

seqlcalc.o : src/seqlcalc.c $(INCLUDES)
 $(CC) $(CFLAGS) -DSHMM -o seqlcalc.o -c src/seqlcalc.c

seqbcalc.o : src/seqbcalc.c $(INCLUDES)
 $(CC) $(CFLAGS) -DSHMM -o seqbcalc.o -c src/seqbcalc.c

acalc.o : src/acalc.c $(INCLUDES)
 $(CC) $(CFLAGS) -DSHMM -o acalc.o -c src/acalc.c

a2a0.o : src/a2a0.c $(INCLUDES)
 $(CC) $(CFLAGS) -DSHMM -o a2a0.o -c src/a2a0.c

smpsinit.o : src/smpsinit.c $(INCLUDES)
 $(CC) $(CFLAGS) -c src/smpsinit.c

seqrdmod.o : src/seqrdmod.c $(INCLUDES)
 $(CC) $(CFLAGS) -DSHMM -o seqrdmod.o -c src/seqrdmod.c

seqpairs : src/seqpairs.c $(LIB)/libmps.a
 $(CC) $(CFLAGS) -o seqpairs src/seqpairs.c $(LDFLAGS)

$(INCLUDES) :

# mps library
$(LIB)/libmps.a : src/utilmps.c src/nrutil.c $(INCLUDES)
  $(CC) $(CFLAGS) -c src/nrutil.c
  $(CC) $(CFLAGS) -c src/utilmps.c
  $(CC) $(CFLAGS) -c src/ran1.c
  ar r $(LIB)/libmps.a ran1.o
  ar r $(LIB)/libmps.a utilmps.o
  ar r $(LIB)/libmps.a nrutil.o
  ranlib $(LIB)/libmps.a
  -rm -f utilmps.o nrutil.o
  
clean:
    rm -f *\.o *run # core gmon.out
  
realclean:
    rm -f *\.o *run user_out asa_out* gmon.out # core
  
cleanlib :
    -rm -f lib/*
  
/******** END seqcalc utilities */

/******** BEGIN shmmseq.h FILE ************/
/* shmmseq.h header file for secalc.c seqbcalc.c */

#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "syscalls.h"
#include "utilmps.h"
#include "nrutil.h"

/*
#define float double
*/

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define TWOPI 6.283185307179586476925
#define GMAX 10.0
#define SMALL 1.0e-18

#ifndef NR_END
#define NR_END 1
#endif
long
tstart, tstop;

/* model parameters */
int
S, /* number of states */
K; /* number of observables */
long
per, /* period length */
T; /* sequence length */
double gmax;
long t;

double
**xmat, /* matrix address of mps() vector xv */
*xv, /*SHMM model [1..npar]: a0[S], c[S][S], b[K][S], tp[S],
g[S] */
*a0, /* [1..S] */
**c, /* [1..S] [1..S] */
**b, /* [1..K] [1..S] */
*tp, /* [1..S] */
*g, /* [1..S] */
*psy, /* [1..S] */
*pst, /* [i..S] */
*ppst, /* 1..S */
**h, /* [0..T-1][1..S] */
***a; /* [0..T-1][1..S][1..S] */
#endif

int *y; /* observable sequence y[1..T] integers */

FILE *fpmod, *fpseq, *fpout;
char *outf;

int start, nstarts;
char *seqnam, *filhead;

char reamodbuf[258];
void readmod(int argc, char *argv[]);
void readpars();
void writmod(double *x, FILE *fp);

double mps();
void mpsinit();
void acalc();

/* the following are special for powell */
#define FTOL 1e-6

double *powv;

void xv2optv(double *xv, double *optv);
void optv2xv(double *optv, double *xv);

void powell(float *p, float **xi, int n, float ftol, int *iter, float *fret,
            float (*func)(float []));

double func(double *ovec);
void randomxmat(double **xmat, int K, int S, double gmax, long *seed);

void writans(FILE *fpout, double value, int iter,
             int argc, char *argv[], char *filnam);

/* for Numerical Recipes
   random number generator ranl() */
long *seed;
float ranl(long *idum);

/* special for smpsc.c */
double *ystdev, *ymean;

void a2a0();

/****** END shmmseq.h header file */

/****** BEGIN FILE calcseq.c *******/
/* calcseq.c read a model file
   * compose a typical sequence based on
   * random number generator
   */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "nrutil.h"
#include "shmmseq.h"
int main(int argc, char *argv[]) {
    double **at;
    float sum, value;
    long tstop, seed[1];
    int y, state, succ;

    readmod(argc, argv);

    tstop = atol(argv[2]);

    if ( argc>3 )
        *seed = atol(argv[3]);
    else
        *seed = 1L;

    acalc();

    a2a0();

    value = (double) ranl(seed);
    for (state=1, sum=a0[state]; sum<value; )
        sum += a0[++state];

    for (t=1; t<=tstop; t++) {
        value = (double) ranl(seed);
        for (y=1, sum = b[y][state];
            sum<value && y<K;
            sum += b[++y][state];
        printf("%d\n", y);

            at = a[t%per];
        value = (double) ranl(seed);
        for ( succ=1, sum = at[succ][state];
            sum<value && succ<S;
            sum += at[++succ][state];
        state = succ;
    }

    return 0;
}

/******** END calcseq.c FILE ***********/

/******* BEGIN seqbcalc.c FILE ************/

/******* BEGIN seqbcalc.c FILE *************/

/* seqbcalc.c read a multiple binary model file
 * compose a typical sequence based on
 * random number generator
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "nrutil.h"
#include "shmmseq.h"

int main(int argc, char *argv[])
{
    double **at;
    float sum, ranval;
    long tstop, seed[2], yout[16];
    int bin, yt, state, site, sitebit, succ;

    int yout[16] =
        {1, 2, 3, 6, 4, 7, 8, 12, 5, 9, 10, 13, 11, 14, 15, 16};

    readmod(argc, argv);
    tstop = atol(argv[2]);
    if (argc>3)
        *seed = atol(argv[3]);
    else
        *seed = 1L;
    acalc();
    a2a0();

    ranval = (double) ranl(seed);
    for(state = 1, sum = a0[state];
        sum < ranval && state < S; )
        sum += a0[++state];

    for( t = 1; t <= tstop; t++) {
        for(bin = 0, sitebit = 8; site = K;
            site--, sitebit >>= 1) {
            ranval = (double) ranl(seed);
            if (ranval <= b[site][state])
                bin += sitebit;
        }
        yt = yout[bin];
        printf("%d
", yt);

        at = a[t%per];
        ranval = (double) ranl(seed);
        for( succ = 1, sum = at[1][state];
            sum < ranval && succ < S; )
            sum += at[++succ][state];
        state = succ;
    }

    return 0;
}
#include <math.h>
#include "shmmseq.h"

void acalc() {
    long t, row, col;
    double **at, theta, *acol, h;

    for(t=0; t<per; t++) {
        at = a[t];
        for(row=1; row<=S; row++) {
            theta = ((double)t / (double) per) - tp[row] * TWOPI;
            h = exp( g[row] * (cos(theta) - 1.0) );
            dvxscal(c[row], h, at[row], S); /* at[j][i] <= c[j][i]*h[j] */
        }
        for( acol=at[1], col=0; col<S; col++)
            normdvec(acol++, S, S);
    }
}

#include <math.h>
#include "shmmseq.h"

void matsq(double **m1, double **m2, int n);

double **ml, **m2, sum, theta, h;
int i, j;

ml = dmatrix(1,S,1,S);

m2 = dmatrix(1,S,1,S);

/* calc. state transition prob. matrix as m1 */
for (j=S; j; j--)
    theta = ((double)tstart / (double)per * tp[j]) * TWOPI;
    h = exp(g[j] * (cos(theta) - 1.0));
    for (i=S; i; i--)
        ml[j][i] = c[j][i] * h;

for (i=S; i; i--)
    for (j=S, sum=0.0; j; j--)
        sum += ml[j][i];
    sum = 1.0/sum;
    for (j=S; j; j--)
        ml[j][i] *= sum;

for (j=10; j--; ) {
    matsq(ml, m2, S);
    matsq(m2, ml, S);
}

for (sum=0.0, j=S; j; j--)
    sum += a0[j] = ml[j][1];
sum = 1.0/sum;
for (j=S; j; j--)
    a0[j] *= sum;

free_dmatrix(m2, 1, S, 1, S);
free_dmatrix(ml, 1, S, 1, S);

void matsq(double **ml, double **m2, int n)
/* computes m2 = ml*ml */
{
    double sum;
    int i, j, k;

    for(i=n; i; i--)
        for(j=n; j; j--)
            for(sum=0.0, k=n; k; k--)
                sum += ml[i][k]*ml[k][j];
        m2[i][j] = sum;
}

/******* END a2a0.c FILE *******/

/******* BEGIN smpsinit.c FILE *******/
/* smpsinit.c initializes array variables
   * for smps.c shmmpow.c
   */

#include "nrutil.h"
#include "utilmps.h"
#include "shmmseq.h"

void mpsinit ()
{
    int j;

    xmat = dmatrix(1, S+K+3, 1, S);
    xv = xmat[1];
    a0 = xv;
    c = xmat + 1;
    b = xmat + S+1;
    tp = xmat[S+K+2];
    g = xmat[S+K+3];
    a = d3d(0, per-1, 1, S, 1, S);

    npar = S*(3+S+K);

    psy = dvector(1, S);
    pst = dvector(1, S);
    ppst = dvector(1, S);

    powv = dvector(1, npar);
    xi = dmatrix(1, npar, 1, npar);
    filldvec(xi[1], 0.0, npar*npar);
    for (j=1; j<npar; j++)
        xi[j][j] = 1.0; /* xi = identity matrix */
}

/******* END smpsinit.c FILE *******/

/******* BEGIN seqrddmod.c FILE *******/
/* seqrddmod.c initiation routine for smps.c
   * reads starting model file
   * allocates memory for model data
   * & sequence data
   * reads sequence data file
   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "shmmseq.h"

void readmoderr() {
    fprintf(stderr,"Can't read model file\n");
    exit(1);
}

/******* END seqrddmod.c FILE *******/
#define scanvec(v,k) \
   (pd=v;n=k; \ 
   while(n--) \ 
      if(1!=fscanf(fp,"%lg", (++pd))) \ 
         readmoderr(); }

FILE *fpmod;

void readpars();

void readmod(int argc, char *argv[])
{
   if (argc<3 ) {
      fprintf(stderr, " USAGE: %s modelfilnam timesteps [seed]\n", argv[0]);
      exit(3); }
   if(NULL == (fpmod = fopen(argv[1],"rt"))) {
      fprintf(stderr, "Can't open model file %s\n",argv[1]);
      exit(4); }
   readpars(fpmod);
   fclose(fpmod);
}

void readpars(FILE *fp)
   /* obtain starting model from a file */
{
   int n;
   double *pd;
   unsigned int readmore;
   char buf[256];
   readmore = Ox1ff; /* model reading check bits: PTKSzabfg */
   while (readmore > 0x20) {
      if (1!= fscanf(fp," %s",buf)) {
         fprintf(stderr, "Can't find data in model file\n");
         exit (2);
      }
      if (!strcmp(buf,"T")) { 
         if (1!= fscanf(fp,"%ld",&T)) readmoderr();
         readmore &= ~0x80;
      }
      else if (!strcmp(buf,"P")) { 
         if (1!= fscanf(fp,"%ld",&per)) readmoderr();
      }
   }
readmore &= ~0x100;
}
else if (!strcmp(buf,"K")) {
  if (1!= fscanf(fp,"%d",&K))
    readmoderr();
  readmore &= ~0x40;
}
else if (!strcmp(buf,"S")) {
  if (1!= fscanf(fp,"%d",&S))
    readmoderr();
  readmore &= ~0x20;
}  
else {  
  do
    fscanf(fp,"%c",buf);
  while( '
' != *buf );
  if (EOF == buf[0]) readmoderr();
}

nrows = 3+S+K;
npar = S*nrows;

mpsinit();

while (readmore) {
  if (1 != fscanf(fp,"%s",buf)) {
    fprintf(stderr,
      "Can't find data in model file\n"");
    exit (2);
  }
  if ( buf[0]=='z' || !strcmp(buf,"a0")) {
    scanvec(a0,S);
    readmore &= ~0x10;
  }
  else if(buf[0]=='f' || !strcmp(buf,"tp")) {
    scanvec(tp,S);
    readmore &= ~0x2;
  }
  else if (buf[0]=='a' || buf[0]=='c') {
    scanvec(c[l],S*S);
    readmore &= ~0x8;
  }
  else if (buf[0]=='b') {
    scanvec(b[l],S*K);
    readmore &= ~0x4;
  }
  else if (buf[0]=='g') {
    scanvec(g,S);
    readmore &= ~0x01;
  }
  else if (EOF == buf[0]) readmoderr();
}
/****** END seqrdmod.c FILE ******/

/****** BEGIN seqpairs.c FILE ******/
/* seqpairs.c opens and reads a sequence file
 * tabulates sequence pairs from
 * tstart to tstop-1
 * prints results to stdout
 */

#include <stdio.h>
#include <stdlib.h>
#include "nrutil.h"

int main(int argc, char *argv[])
{
    long t, tstart, tstop;
    int y, pred, row, col, K, **pairs, sum;
    FILE *fp;

    if ( argc<5 ) {
        fprintf(stderr, " USAGE %s sequencefilename #outputs tstart
tstop\n", argv[0]);
        exit(1);
    }
    if( NULL== (fp=fopen(argv[1], "r")) ){
        fprintf(stderr,
            " Can't open sequence file %s
", argv[1]);
        exit(2);
    }
    K = atoi(argv[2]);
    tstart = atol(argv[3]);
    tstop = atol(argv[4]);

    pairs = imatrix(1,K,1,K);
    for(row=K; row; row--)
        for(col=K; col; col--)
            pairs[row][col] = 0;

    for( t=1; t<=tstart; t++ )
        fscanf(fp, "%d", &y );
    while( (++t)<=tstop ) {
        pred = y;
        fscanf(fp, "%d", &y);
        pairs[pred][y] += 1;
    }

    printf(" y[1]");
    for(col=1; col<=K; col++)
        printf("%4d",col);
    printf(" rowsum\n");
    for(row=1; row<=K; row++) {
        printf("%2d " ,row);
        ...
sum = 0;
for(col=1; col<=k; col++) {
    sum += pairs[row][col];
    printf("%4d", pairs[row][col]);
}
printf("%4d\n", sum);

printf("sequence pair frequencies: row first, column second\n");
printf("%s sequence file %s tstart %ld tstop %ld\n", 
        argv[0], argv[1], tstart, tstop);
return 0;

/***** END seqpairs.c FILE ******/
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "nrutil.h"

#define NR_END 1
#define FREE_ARG char*

void copydvec( double *source, double *dest, long n) {
    source += NR_END;
    dest += NR_END;
    for( ; n--; dest++, source++)
        *dest = *source;
}

void filldvec( double *vec, double value, long n) {
    vec += NR_END;
    for( ; n--; vec++)
        *vec = value;
}

double sumdvec( double *v, long nsteps) {
    double sum = 0.0;
    v += NR_END;
    for( ; nsteps--; v++)
        sum += *v;
    return sum;
}

double sumdvecstep( double *v, long incr, long nsteps) {
    double sum = 0.0;
    v += NR_END;
    for( ; nsteps--; v += incr)
        sum += *v;
    return sum;
}

void scaledvec( double *vec, double scfactor, long n) {
    vec += NR_END;
    for( ; n--; vec++)
        *vec *= scfactor;
}
void dvxscal( double *vec, double scfactor, double *prod, long n) {
    vec += NR_END;
    prod += NR_END;
    for( ; n--; vec++, prod++)
        *prod = *vec * scfactor;
}

double ddotprod( double *vl, double *v2, double *pv, long n) {
    v1 += NR_END;
    v2 += NR_END;
    pv += NR_END;
    for( ; n--; vl++, v2++, pv++)
        sum += *pv = *vl * *v2;
    return sum;
}

double ddot( double *vl, double *v2, long n) {
    double sum=0.0;
    vl += NR_END;
    v2 += NR_END;
    while( n--)
        sum+= *(vl++) * *(v2++);
    return sum;
}

void dmatxvec( double **m, double *v, double *prodv, long rows, long cols) {
    m += NR_END;
    prodv += NR_END;
    while( rows--)
        *(prodv++) = ddot( *(m++), v, cols);
}

double ddotstep2( double *vl, double *v2, long step, long n) {
    double sum=0.0;
    v1 += NR_END;
    v2 += NR_END;
    for( ; n--; vl++, v2 += step )
        sum += *vl * *v2;
    return sum;
}

void dmxm( double **ml, double **m2, double **prod, long r, long n, long c) {
    long row, col;
    for( row=1; row<=r; row++)
for( col=1; col <=c; col++)
    prod[row][col] = ddotstep2(m1[row], m2[1], col, n);

void normdvec(double *vec, long step, long nsteps) {
    double sum=0.0;
    int n;

    vec += NR_END;
    for( n=nsteps; n--; vec += step)
        sum += *vec;
    vec -= step;
    sum = 1.0/sum;
    for( n=nsteps; n--; vec -= step)
        *vec *= sum;
}

void normdmat(double **mat, long nrow, long ncol) {
    double *col;
    long n=ncol;

    col = mat[1];
    while(n--)
        normdvec(col++, ncol, nrow);
}

/* allocate a NR style 3d double array */
double *** d3d(long nrl, long nrh, long ncl, long nch, long ndl, long ndh) {
    long i, j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
    double **t;

    /* allocate pointers to pointers to rows */
    t=(double **) malloc ((size_t)
        ((nrow+NR_END)*sizeof(double**)));
    if (!t) nrerror("allocation failure 1 in d3d()");
    t += NR_END;
    t -= nrl;

    /* allocate pointers and set pointers to rows */
    t[nrl]=(double **) malloc (size_t)
        ((nrow*ncol+NR_END)*sizeof(double**)));
    if (!t[nrl])
        nrerror("allocation failure 2 in d3d()");
    t[nrl] += NR_END;
    t[nrl] -= ncl;

    /* allocate rows and set pointers to them */
    t[nrl][ncl]=(double *) malloc (size_t)
        ((nrow*ncol+ndep+NR_END)*sizeof(double)));
    if (!t[nrl][ncl])
        nrerror("allocation failure 3 in d3d()");
    t[nrl][ncl] += NR_END;
    t[nrl][ncl] -= ndl;
for (j=ncl+1; j<=nch; j++) t[nrl][j] = t[nrl][j-1] + ndep;
for (i=nrl+1; i<=nrh; i++) {
    t[i] = t[i-1] + ncol;
    t[i][ncl] = t[i-1][ncl] + ncol*ndep;
    for (j=ncl+1; j<=nch; j++) t[i][j] = t[i][j-1] + ndep;
}
/* return pointer to array of pointers to rows */
return t;

/* free entire d3d 3d array */
void free_d3d(double ***t, long nrl, long nrh,
              long ncl, long nch, long ndl, long ndh)
    /* free a double d3d allocated by d3d() */
    {
        free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
        free((FREE_ARG) (t[nrl]+ncl-NR_END));
        free((FREE_ARG) (t+nrl-NR_END));
    }

/******* END utilmps.c FILE ***********/

/******* BEGIN utilmps.h FILE ***********/
/* utilmps.h for utilmps.c */

void copydvec( double *source, double *dest, long n);
void filldvec( double *vec, double value, long n);
double sumdvec( double *v, long nsteps);
void scaledvec( double *vec, double scfactor, long n);
void dvxscal( double *vec, double scfactor,
              double *prod, long n);
double ddot( double *v1, double *v2, long n);
double ddotprod( double *v1, double *v2,
                 double *pv, long n);
void dmatxvec( double **m, double *v,
              double *prodv, long rows, long cols);
double ddotstep2( double *v1, double *v2, long c,
                 long n);
void dmatxmat( double **m1, double **m2,
              double **prod, long r, long n, long c);
void normdvec(double *vec, long step, long nsteps);
void free_d3d(double ***t, long nrl, long nrh,
long ncl, long nch, long ndl, long ndh);

double *** d3d(long nrl, long nrh, long ncl,
            long nch, long ndl, long ndh);

/************* END utilmps.h FILE ************/
Appendix B Parameters of Optimized SHMM

T 1825  P 365  K 16  S 2
a0 1.0000  3.3e-11
   0.8222  0.1022
   0.1778  0.8978
b  0.5751  0.0028
   0.0977  0.0282
   0.1366  0.0501
   0.0488  0.0385
   0.0519  0.0102
   0.0206  0.0494
   0.0139  0.0395
   0.0112  0.0458
   0.0061  0.0207
   0.0317  0.0352
   1.7e-14  0.0204
   4.6e-14  0.1859
   1.6e-15  0.0668
   1.2e-15  0.0227
   0.0064  0.0510
   2.3e-13  0.3318
f  8.6795  8.3602
g  0.1638  0.5334
# mps 0.122086  1/8.19095  bits 3.03403  iters 8
# ./.pow/shmmpow ./.y16 1825 365 16 2 1.0 y5s2g1i1 9 1 y5s2g1i18.1
<table>
<thead>
<tr>
<th>a0</th>
<th>1.0000</th>
<th>1.2e-12</th>
<th>2.7e-12</th>
<th>1.2e-12</th>
<th>1.0e-12</th>
<th>2.7e-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0.8913</td>
<td>2.7e-10</td>
<td>0.4422</td>
<td>0.314</td>
<td>0.4631</td>
<td>0.1155</td>
</tr>
<tr>
<td></td>
<td>0.0773</td>
<td>0.5369</td>
<td>0.4422</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0.5858</td>
<td>4.0e-13</td>
<td>4.9e-13</td>
<td>0.1262</td>
<td>1.3e-12</td>
<td>4.6e-11</td>
</tr>
<tr>
<td></td>
<td>0.1322</td>
<td>2.7e-14</td>
<td>0.0915</td>
<td>0.0458</td>
<td>9.8e-13</td>
<td>0.0680</td>
</tr>
<tr>
<td></td>
<td>0.0468</td>
<td>4.0e-16</td>
<td>0.0264</td>
<td>0.0142</td>
<td>7.1e-13</td>
<td>0.0904</td>
</tr>
<tr>
<td></td>
<td>0.0120</td>
<td>6.4e-13</td>
<td>0.0670</td>
<td>0.0077</td>
<td>5.9e-13</td>
<td>0.0796</td>
</tr>
<tr>
<td></td>
<td>2.2e-14</td>
<td>1.1e-13</td>
<td>0.0435</td>
<td>0.0273</td>
<td>8.9e-15</td>
<td>0.0648</td>
</tr>
<tr>
<td></td>
<td>4.8e-11</td>
<td>1.8e-13</td>
<td>0.0326</td>
<td>5.1e-15</td>
<td>0.5000</td>
<td>0.0312</td>
</tr>
<tr>
<td></td>
<td>6.1e-12</td>
<td>6.5e-13</td>
<td>0.1070</td>
<td>4.4e-14</td>
<td>6.9e-13</td>
<td>0.0363</td>
</tr>
<tr>
<td></td>
<td>0.0012</td>
<td>3.7e-12</td>
<td>0.0905</td>
<td>1.6e-13</td>
<td>0.5000</td>
<td>0.1710</td>
</tr>
<tr>
<td>f</td>
<td>18.8422</td>
<td>15.1089</td>
<td>14.8139</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0.9402</td>
<td>0.2833</td>
<td>0.4591</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# mps 0.124111 1/8.05711 bits 3.0103 iters 12
# ./pow/shmmrpow ..//y16 1825 365 16 3 1.0 y5s3gls1 g 1 y5s3gls18.1

<table>
<thead>
<tr>
<th>a0</th>
<th>1.0000</th>
<th>8.8e-12</th>
<th>6.9e-08</th>
<th>8.3e-08</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>3.5e-14</td>
<td>2.6e-14</td>
<td>1.7e-14</td>
<td>1.2e-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.3e-08</td>
<td>2.0e-13</td>
<td>2.9e-13</td>
<td>0.0549</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.2e-13</td>
<td>1.0000</td>
<td>0.7861</td>
<td>0.0554</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0000</td>
<td>5.7e-12</td>
<td>0.2139</td>
<td>0.8897</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1.0000</td>
<td>2.9e-13</td>
<td>0.5916</td>
<td>4.6e-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4e-12</td>
<td>0.1350</td>
<td>0.0936</td>
<td>0.0290</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.2e-12</td>
<td>0.2124</td>
<td>0.1366</td>
<td>0.0441</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7e-12</td>
<td>4.4e-13</td>
<td>0.0483</td>
<td>0.0415</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.0e-08</td>
<td>4.5e-15</td>
<td>0.0500</td>
<td>0.0141</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.3e-13</td>
<td>0.3089</td>
<td>0.0159</td>
<td>0.0400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.9e-12</td>
<td>4.2e-11</td>
<td>0.0143</td>
<td>0.0405</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.1e-11</td>
<td>6.9e-13</td>
<td>0.0119</td>
<td>0.0466</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.5e-12</td>
<td>0.2012</td>
<td>0.0024</td>
<td>0.0148</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.9e-13</td>
<td>0.1425</td>
<td>0.0288</td>
<td>0.0327</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0e-12</td>
<td>2.2e-13</td>
<td>1.1e-14</td>
<td>0.0209</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.5e-11</td>
<td>2.8e-12</td>
<td>1.7e-13</td>
<td>0.1917</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.6e-12</td>
<td>3.4e-13</td>
<td>7.2e-14</td>
<td>0.0685</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4e-12</td>
<td>4.6e-13</td>
<td>1.0e-13</td>
<td>0.0232</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.5e-13</td>
<td>1.5e-13</td>
<td>0.0667</td>
<td>0.0521</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.0e-11</td>
<td>2.6e-11</td>
<td>8.8e-09</td>
<td>0.3403</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>281.5533</td>
<td>176.3247</td>
<td>200.5197</td>
<td>224.3718</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0.9990</td>
<td>1.0000</td>
<td>0.3945</td>
<td>0.9408</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# mps 0.123524 1/8.09562 bits 3.01714 iters 1
# ./pow/shmmrpow ..//y16 1825 365 16 4 1.0 y5s4gls1 g 1 y5s4gls118.4
T 1825  p 365  K 16  S 5
a0 5.7e-12 8.3e-12 3.2e-12 1.0000 3.0e-12
 0.4665 0.0965 0.4105 0.1047 1.2e-12
 6.1e-12 0.4968 0.1193 2.7e-13 0.3401
 0.0209 0.4166 0.4206 3.7e-12 0.5082
 0.2358 4.3e-14 2.9e-14 0.7823 1.4e-14
 0.2269 3.0e-13 0.0496 0.1130 0.1517
b 0.3326 1.7e-12 3.6e-13 0.7347 2.9e-12
 0.1910 5.0e-13 3.5e-12 0.0352 0.0913
 0.1276 2.3e-12 0.0623 0.1451 0.0954
 0.0876 3.8e-13 0.0389 0.0213 0.0944
 0.1058 1.2e-12 2.3e-12 0.0143 0.0315
 0.0133 2.8e-13 0.1171 0.0214 0.0357
 0.0453 1.5e-12 0.0443 1.4e-11 0.0722
 8.9e-15 6.9e-13 0.0804 0.0110 0.0908
 1.8e-14 6.9e-13 0.0564 0.0060 0.0109
 0.0968 2.2e-11 0.0641 2.6e-13 4.8e-13
 4.3e-15 6.7e-13 0.0493 6.7e-12 0.0108
 1.4e-12 0.3304 7.7e-12 2.0e-13 0.2789
 4.8e-13 2.9e-12 0.1613 4.3e-13 0.0303
 5.3e-15 0.0147 0.0372 4.2e-12 0.0147
 5.7e-13 0.0088 0.1346 0.0074 3.2e-12
 4.1e-13 0.6461 0.1553 0.0036 0.1431
f 116.9454 120.9922 196.8734 105.0517
 139.1846
g 1.0000 0.7536 1.0000 0.8704 1.0000
# mps 0.129407 1/7.72756 bits 2.95001 iters 4
# ../pow/shmmrpow ../y16 1825 365 16 5 1.0 y5s5gls1 9 1
y5s5gls12.8
<table>
<thead>
<tr>
<th>T 1825</th>
<th>P 365</th>
<th>K 16</th>
<th>S 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>1.0000</td>
<td>1.7e-07</td>
<td>9.0e-08</td>
</tr>
<tr>
<td>c</td>
<td>0.5429</td>
<td>0.5334</td>
<td>1.0e-11</td>
</tr>
<tr>
<td></td>
<td>0.2034</td>
<td>0.3742</td>
<td>4.8e-13</td>
</tr>
<tr>
<td></td>
<td>1.0e-13</td>
<td>1.9e-12</td>
<td>0.6939</td>
</tr>
<tr>
<td></td>
<td>0.0428</td>
<td>1.0e-11</td>
<td>5.0e-13</td>
</tr>
<tr>
<td></td>
<td>0.2109</td>
<td>4.2e-12</td>
<td>0.0273</td>
</tr>
<tr>
<td></td>
<td>3.1e-12</td>
<td>0.0924</td>
<td>0.2767</td>
</tr>
<tr>
<td>b</td>
<td>0.7789</td>
<td>0.6121</td>
<td>1.3e-13</td>
</tr>
<tr>
<td></td>
<td>0.0926</td>
<td>0.0037</td>
<td>0.0047</td>
</tr>
<tr>
<td></td>
<td>4.4e-11</td>
<td>0.3301</td>
<td>8.4e-12</td>
</tr>
<tr>
<td></td>
<td>0.0138</td>
<td>1.6e-12</td>
<td>3.9e-11</td>
</tr>
<tr>
<td></td>
<td>0.0480</td>
<td>7.8e-12</td>
<td>4.4e-13</td>
</tr>
<tr>
<td></td>
<td>9.2e-12</td>
<td>0.0540</td>
<td>0.0176</td>
</tr>
<tr>
<td></td>
<td>3.4e-13</td>
<td>5.8e-14</td>
<td>5.5e-12</td>
</tr>
<tr>
<td></td>
<td>1.1e-12</td>
<td>2.0e-13</td>
<td>7.2e-12</td>
</tr>
<tr>
<td></td>
<td>9.5e-13</td>
<td>3.2e-12</td>
<td>0.0148</td>
</tr>
<tr>
<td></td>
<td>0.0668</td>
<td>3.4e-13</td>
<td>0.0110</td>
</tr>
<tr>
<td></td>
<td>2.9e-13</td>
<td>1.6e-13</td>
<td>1.8e-13</td>
</tr>
<tr>
<td></td>
<td>1.0e-13</td>
<td>1.4e-13</td>
<td>0.2815</td>
</tr>
<tr>
<td></td>
<td>6.0e-13</td>
<td>4.1e-13</td>
<td>3.4e-11</td>
</tr>
<tr>
<td></td>
<td>6.8e-14</td>
<td>9.9e-15</td>
<td>2.7e-14</td>
</tr>
<tr>
<td></td>
<td>7.1e-14</td>
<td>2.2e-13</td>
<td>0.0086</td>
</tr>
<tr>
<td></td>
<td>3.9e-13</td>
<td>1.0e-13</td>
<td>0.6618</td>
</tr>
<tr>
<td>f</td>
<td>59.7919</td>
<td>78.6533</td>
<td>63.7530</td>
</tr>
<tr>
<td></td>
<td>1.0000</td>
<td>0.2499</td>
<td>0.6270</td>
</tr>
</tbody>
</table>

# mps 0.129971 1/7.69404  bits 2.94374  iters 11
# .../pow/shmmpow .../y16 1825 365 16 6 1.0 y5s6g1sl 9 1
y5s6g1sl3.3
Appendix C  Mean Probabilities, All 9 Runs, 16 Pattern SHMMs

Sequence pair counts - 16 pattern data sequence

**FIRST 5 YEARS:**
```
<table>
<thead>
<tr>
<th>y[1]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>y[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>314</td>
<td>41</td>
<td>54</td>
<td>31</td>
<td>19</td>
<td>13</td>
<td>7</td>
<td>11</td>
<td>4</td>
<td>13</td>
<td>2</td>
<td>18</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>18</td>
<td>11</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>69</td>
<td>8</td>
<td>34</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>45</td>
<td>11</td>
<td>3</td>
<td>5</td>
<td>59</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>7</td>
<td>9</td>
<td>16</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>11</td>
<td>1</td>
<td>42</td>
<td>18</td>
<td>4</td>
<td>14</td>
<td>24</td>
</tr>
</tbody>
</table>
```

Sequence pair frequencies: row first, column second
```
seqpairs sequence file ../test/y16 tstart 1 tstop 1825
```
LAST 4 YEARS:

<table>
<thead>
<tr>
<th>y[1]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>y[0]</td>
<td>328</td>
<td>26</td>
<td>49</td>
<td>26</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>14</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td>14</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>5</td>
<td>18</td>
<td>13</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>15</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>54</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>31</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>105</td>
</tr>
</tbody>
</table>

sequence pair frequencies: row first, column second

seqpairs sequence file ../test/y16_tstart 1826 tstop 3285

ALL 9 YEARS:

<table>
<thead>
<tr>
<th>y[1]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>y[0]</td>
<td>642</td>
<td>67</td>
<td>103</td>
<td>57</td>
<td>29</td>
<td>19</td>
<td>19</td>
<td>14</td>
<td>14</td>
<td>5</td>
<td>16</td>
<td>3</td>
<td>32</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>73</td>
<td>32</td>
<td>20</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>13</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>16</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>113</td>
<td>13</td>
<td>52</td>
<td>18</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>51</td>
<td>21</td>
<td>12</td>
<td>15</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>5</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>9</td>
<td>13</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>16</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>19</td>
<td>1</td>
<td>13</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>8</td>
<td>17</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>7</td>
<td>13</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>99</td>
<td>15</td>
<td>5</td>
<td>8</td>
<td>97</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>12</td>
<td>18</td>
<td>10</td>
<td>23</td>
<td>16</td>
<td>15</td>
<td>19</td>
<td>4</td>
<td>73</td>
<td>26</td>
<td>7</td>
<td>18</td>
<td>229</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sequence pair frequencies: row first, column second

seqpairs sequence file ../test/y16_tstart 1 tstop 3285
References


