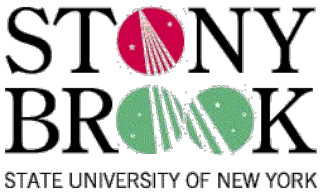


Tutorial:

Hidden Markov Models with Univariate Gaussian Outcomes

Robert J. Frey, Research Professor
Stony Brook University, Applied Mathematics and Statistics

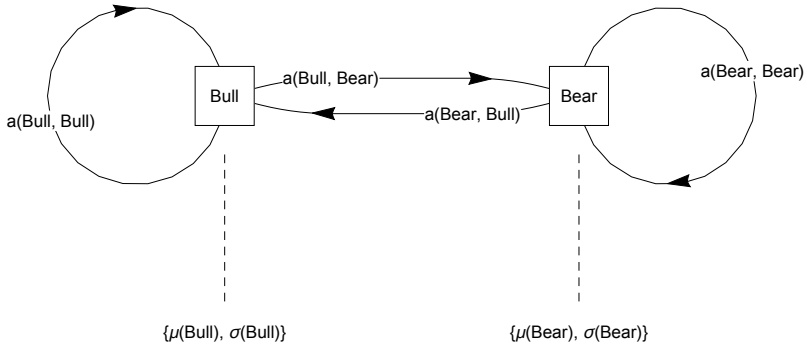
frey@ams.sunysb.edu
http://www.ams.sunysb.edu/~frey

22 December 2009

Hidden Markov Models

Consider a system which evolves according to a Markov process, which may be either continuous- or discrete-time, and may have finitely or infinitely many states. This underlying Markov process is not directly observable; *i.e.*, the system states are occult or hidden. Associated with each state is a stochastic process which produces outcomes which are, however, observable. As the system evolves over time in accordance with the occult Markov chain it produces these observable outcomes in a state-dependent fashion.

For a concrete example, take a simple model of the monthly returns of a stock market. The market has two states, a "Bull" state characterized by "good" performance and a "Bear" state characterized by "bad" performance. The transition from one state to another is controlled by a two-state Markov chain. Given the states $i, j \in$ {Bull. Bear} of the market in a given month, the probability the market transitions from state i to state j in the next month is $a(i, j)$. If the market is in some state i then its log return is Normally distributed with state-dependent mean $\mu(i)$ and standard deviation $\sigma(i)$. This simple market model is illustrated below:



Not what one would call the last word in market models. Limiting the granularity to monthly leaves out a great deal of information. The market dynamics are undoubtedly complex and two states may not be enough to capture this. Even during periods when a market appears to be reasonably characterized as bull or bear, the log returns tend to be more kurtotic than a Normal distribution would predict.

On the other hand, models always leave out stuff. That's the point. As demonstrated below, this simple model captures some interesting aspects of real stock market behavior. The question we address is: Given a set of observations and the framework of a model such as that above, how does one estimate its parameters?

Objectives of This Tutorial

"The road to learning by precept is long, but by example short and effective." —*Seneca*

Lawrence Rabiner's excellent "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition" [Rabiner 1989] is available here as a PDF from his faculty website (as of 09 Dec 2009) at UC Santa Barbara. The tutorial covers the maximum likelihood estimation (MLE) of hidden Markov models with both discrete and continuous outputs as well as several extensions, such as autoregressive and non-ergodic models. It also covers alternatives to MLE, such as estimating Viterbi (maximum probability) sequences. Finally, it covers important issues of implementation such as initializing parameters and scaling computations to avoid under- or overflow conditions.

However, Rabiner's paper focuses on the theory as it pertains to discrete outcome models used in speech recognition. It defers the details necessary for practical implementation until fairly late in the presentation. Where it does discuss continuous outcome models, it does so using the most general of cases: finite mixtures of multivariate Gaussians. It also does not present an end-to-end, step-by-step description of the practical implementation of the algorithm. Although it discusses applications, it does not present actual working examples.

The objectives of this tutorial are to:

- Present the general theory but narrow the focus to discrete-time models with continuous outcomes, which are of more interest in quantitative finance,

- Discuss a simple--yet interesting and useful--case, a discrete time HMM with univariate Gaussian outcomes, and thereby provide a somewhat more accessible introduction,
- Set out a complete, detailed sequence of the computations required and discuss how the list/vector processing facilities of languages such as *Mathematica* and MATLAB can be exploited in that implementation,
- Provide a set of working tools and apply them to actual data in an active document, *i.e.*, a *Mathematica* notebook, that students can use to run examples.
- Cover more recent developments and references, especially to those that pertain to financial time series.

None of these comments, however, should be interpreted as a criticism of [Rabiner 1989] and the student is encouraged to read and work through that paper and the other references provided.

Model Overview and Theory

Although the notation of [Rabiner 1989] is followed in most instances, there are a few modifications. The variable subscript is reserved for time indexing, while state indexing is done parenthetically with the aims that it makes the mathematics both easier to follow and more straightforward to transcribe into program data structures. Where Rabiner's original notation was at variance with that which might be more familiar to the student, the more common usage was adopted; *e.g.*, θ is used instead of λ to indicate the parameter set. Finally, certain variables were used with different meanings in different places, *e.g.*, c_t for the scaling constant [Rabiner 1989, (91)] and $c_{j,m}$ for the finite mixture of multivariate Gaussians proportions [Rabiner 1989; (50)]; herein, v_t is used for the scaling constant.

Description of the HMMUG Model

The hidden Markov model with univariate Gaussian outcomes (HMMUG) is the focus here. At a given time t we are presented with a discrete-time system that can be in one of N states:

$$S = \{s(1), \ldots, s(i), \ldots, s(N)\}$$

The states are not observable and are represented by:

$$Q = \{q_1, \ldots, q_t, \ldots, q_T\}$$

The states evolve over time according to a finite-state, ergodic Markov chain characterized by its intial state and transition matrix:

$$\imath(i) = \text{Prob}[q_1 = s(i)]$$

$$a(i, j) = \text{Prob}[q_{t+1} = s(j) \mid q_t = s(i)]$$

An observation is generated at each point in time in a manner that is conditioned on the state of the system. Here we assume that the observations are generated from a univariate Gaussian distribution with state-dependent mean and standard deviation:

$$\text{Density}[x_t \mid q_t = s(i)] = \psi[x \mid \mu(i), \sigma(i)]$$

The symbol θ will be used to refer to the parameters of the model collectively; implicit in the parameters is the number of states of the system:

$$\theta = \{\imath,\ a,\ \mu,\ \sigma\}$$

We are given a time series of T periodic observations:

$$X = \{x_1,\ \ldots,\ x_t,\ \ldots,\ x_T\}$$

and our objective is to fit parameters θ given the observations X.

The model will be fit using maximum likelihood estimation (MLE), *i.e.*, by determining the values of the parameters which maximizes the likelihood of observing the data:

$$\mathcal{L}[\theta_{\text{MLE}} \mid X] = \underset{\theta}{\text{argmax}}\left[\text{Prob}[X \mid \theta]\right]$$

EM Algorithm

In the HMMUG model, the state of the system q_t cannot be directly observed. If we had such state information, then estimating the parameters θ would be trivial.

The EM algorithm [Dempster *et al.*, 1977] can be used for MLE when there are occult (*i.e.*, hidden, latent, censored, or missing) data or when a problem can be reformulated in those terms. Informally, the EM algorithm takes the observed data and a current estimate of the parameters and uses them to estimate the missing data; it then takes the observed data and the estimated missing data and uses them to produce a new estimate of the parameters. Remarkably, under suitable conditions, this iterative process converges to a local maximum of the likelihood function [Dempster *et al.* 1977].

More formally, at the η^{th} iteration given a complete data set X, occult data Z, and a working estimate of parameters $\theta^{(\eta)}$, the expectation (E) step calculates the expected value of the log likelihood function of the conditional distribution of Z given X and $\theta^{(\eta)}$.

$$Q\left[\theta, \theta^{(\eta)}\right] = \text{E}\left[\log \mathcal{L}[Z \mid X, \theta] \mid X, \theta^{(\eta)}\right]$$

The maximization (M) step then finds a successor parameter set $\theta^{(\eta+1)}$ which maximizes that expectation.

$$\theta^{(\eta+1)} = \underset{\theta}{\text{argmax}}\, Q\left[\theta, \theta^{(\eta)}\right]$$

The parameter set $\theta^{(\eta+1}$ then becomes the new working parameter set. EM iterations continue until log $\mathcal{L}[\theta \mid X]$ converges. For the HMMUG model and many other cases of interest, the E-step reduces to estimating sufficient statistics for Z given X and θ and the M-Step to maximizing $\mathcal{L}[\theta \mid X, Z]$ using the "completed" data.

Each EM iteration results in a non-decreasing likelihood, but the EM algorithm converges sublinearly and may find a local maximum. There may be material numerical instabilities affecting convergence in practice. The algorithm may set certain elements of θ to explain a single point, and this drives the likelihood towards infinity. Single-point fits mean either that a less complex model is appropriate or that the offending data point is an outlier and must be removed from the sample [Dempster *et al.*, 1977] [McLachlan 2008].

In the HMMUG model, the state of the system q_t cannot be directly observed. If we had such state information, then estimating the parameters would be trivial. Thus, fitting an HMMUG model can be viewed as a missing data problem in which one is given the observations X but are missing the states Q.

E-Step

The E-step will involve the estimation of the following quantities [Baum *et al.* 1970]:

$$\xi_t(i, j) = \text{Prob}[q_t = s(i), q_{t+1} = s(j) \mid X, \theta]$$

How does ξ_t represent the missing state data? Clearly, if we knew ξ_t with certainty, we could use it to separate the outcomes into subsets depending upon state and use each subset to estimate μ and σ for each state. It would also allow us to count the frequency of transitions from one state to another and thereby estimate a. As it is, the best we can do is estimate ξ_t probabilistically given X and $\theta^{(\eta)}$, *i.e.*, as the solution to the expectation $Q\left[\theta \mid \theta^{(\eta)}\right]$.

M-Step

For the HMMUG model, the M-Step, the re-estimation of θ based on the completed data, will be accomplished for $\mu(i)$ and $\sigma(i)$ by weighting each observation proportionally to the probability that it is in a given state at a given time and for the transition matrix by summing the ξ_t across time and normalizing the rows to sum to unity [Baum *et al.* 1970].

Likelihood Function

Direct calculation of $\mathcal{L}[\theta \mid X]$ is not computationally tractable [Baum *et al.* 1970] [Rabiner 1989]. Let Q denote an arbitrary state-sequence, then

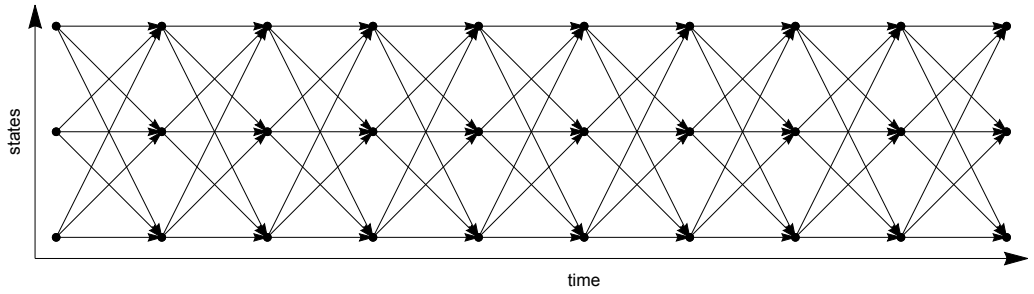
$$\mathcal{L}[\theta \mid X] = \sum_{\forall Q} \text{Prob}[X \mid Q, \theta] \times \text{Prob}[Q \mid \theta]$$

The computation of the likelihood for any one state sequence given the observed data and parameters is straightforward; however, there are N^T such state sequences. Consider a modest problem in which we wish to fit a two-state model to one-year of daily returns. The total number of possible state sequences would be $2^{250} \simeq 1.8 \times 10^{75}$. In the naïve form above, the computation of the likelihood is not practical.

Baum-Welch Algorithm

Forunately, the forward-backward, or Baum-Welch, algorithm [Baum *et al.* 1970] employs a clever recursion to compute both $Q\left[\theta, \theta^{(\eta)}\right]$ and $\mathcal{L}[\theta \mid X]$ efficiently; that paper also represents an early example of the application of the EM algorithm years before it was more generally and formally codified in [Dempster *et al.*, 1977].

Consider a case in which $N = 3$ and $T = 10$, *i.e.*, three states and ten periods. Even in a toy problem such as this there are $3^9 = 19{,}683$ paths. However, as illustrated below, paths are recombinant, forming a lattice.

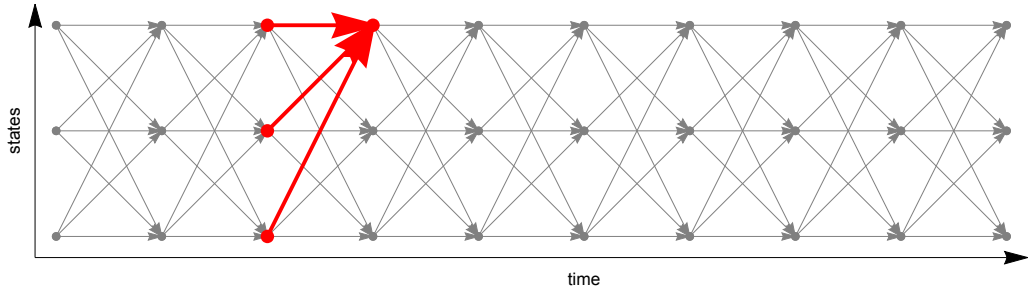


The forward-backward algorithm exploits this topology to efficiently estimate the probabilities required to compute the both the conditional expectation $Q\left[\theta, \theta^{(\eta)}\right]$ and the log likelihood $\log \mathcal{L}[X \mid \theta]$. The algorithm is a form of dynamic programming.

Forward-Backward Recursions

In presenting the forward-backward calculations, the variables are colored coded to cross-reference them with their associated specifications as probabilities in the hope that this is more illuminating than distracting. The exposition in no way depends on this color coding, however.

Forward Recursion: For a forward recursion, consider the example below in which we wish to compute $\text{Prob}[x_1, \ldots, x_4, q_4 = s(3) \mid \theta]$ and know the associated probabilities of the *N predecessor* states. Shown below are the target $\{q_4 = s(3)\}$ and it predecessors $\{q_3 = s(1), q_3 = s(2), q_3 = s(3)\}$ with the arrows meant to show the flow of computation.



From the density $\text{Prob}[x_4 \mid q_4 = s(1), \theta]$, the vector of predecessor forward probabilities $\text{Prob}[x_1, \ldots, x_3, q_3 = s(k) \mid \theta]$, and the appropriate column of the transition matrix $\text{Prob}[q_4 = 1 \mid q_3 = s(k)]$ the required calculation expressed generally is:

$$\text{Prob}[x_1, \ldots, x_t, q_t = s(i) \mid \theta] =$$

$$\text{Prob}[x_t \mid q_t = s(i), \theta] \times \sum_{k=1}^{N} \text{Prob}\left[x_1, \ldots, x_{t-1}, q_{t-1} = s(k) \mid \theta\right] \times \text{Prob}\left[q_t = i \mid q_{t-1} = s(k)\right]$$

If one defines the following variables:

$$b_t(i) = \text{Prob}[x_t \mid q_t = s(i), \theta]$$

$$\alpha_t(i) = \text{Prob}[x_1, \ldots, x_t, q_t = s(i) \mid \theta]$$

then:

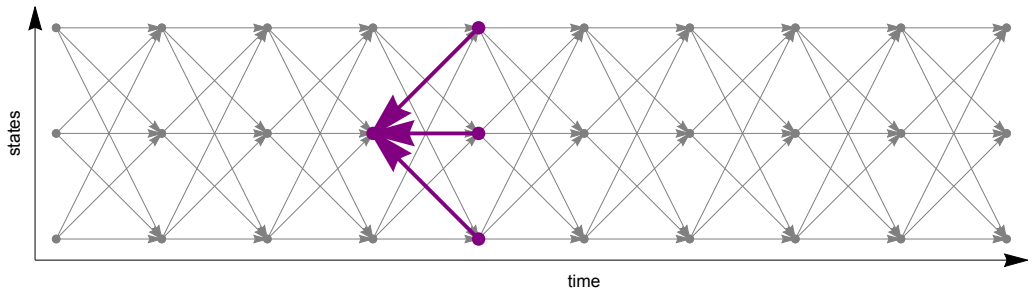
$$\alpha_t(i) = b_t(i) \sum_{k=1}^{N} \alpha_{t-1}(k)\, a(k, i)$$

An obvious initial condition to start the forward recursion is:

$$\alpha_1(i) = \pi(i)\, b_1(i)$$

The forward recursion starts at time $t = 1$ and recursively calculates the forward probabilities across states for each time period up to $t = T$.

Backward Recursion: For the backward recursion, consider the example below in which we wish to compute $\text{Prob}[x_5, \ldots, x_T \mid q_4 = s(2), \theta]$ and know the associated probabilities of its N *successor* states $\text{Prob}[x_6, \ldots, x_{10} \mid q_5 = s(k), \theta]$. Shown below are the target $\{q_4 = s(2)\}$ and it successors $\{q_5 = s(1), q_5 = s(2), q_5 = s(3)\}$ with the arrows again meant to show the flow of computation.



From the transition probabilities $\text{Prob}[q_5 = s(k) \mid q_4 = s(2)]$, the densities $\text{Prob}[x_5 \mid q_5 = s(k), \theta]$, and successor backward probabilities $\text{Prob}[x_6, \ldots, x_{10} \mid q_5 = s(k), \theta]$ the required calculation expressed generally is:

$$\text{Prob}[x_{t+1}, \ldots, x_T \mid q_t = s(i), \theta] =$$

$$\sum_{k=1}^{N} \text{Prob}[q_{t+1} = s(k) \mid q_t = s(i)] \times \text{Prob}[x_{t+1} \mid q_{t+1} = s(k), \theta] \times \text{Prob}[x_{t+2}, \ldots, x_T \mid q_{t+1} = s(k), \theta]$$

If one defines the following variables:

$$\beta_t(i) = \text{Prob}[x_{t+1}, \ldots, x_T \mid q_t = s(i), \theta]$$

then:

$$\beta_t(i) = \sum_{k=1}^{M} a(i, k)\, b_{t+1}(k)\, \beta_{t+1}(k)$$

The initial condition to start the backward recursion is *arbitrarily* chosen as:

$$\beta_T(i) = 1$$

The backward recursion starts at time $t = T$ and recursively calculates the backward probabilities acrosss states for each time period down to $t = 1$.

Sufficient Statistics for the Occult Data, Updating θ, and Calculating $\log \mathcal{L}[X \mid \theta]$

With the forward probabilities $\alpha_t(i)$ and backward probabilities $\beta_t(i)$ one can compute the probability that the system in a specific state at a specific time given the observed data and the working

parameters:

$$\text{Prob}[q_t = s(i) \mid X, \theta] \propto \text{Prob}[x_1, \ldots, x_t, q_t = s(i) \mid \theta] \times \text{Prob}[x_{t+1}, \ldots, x_T \mid q_t = s(i), \theta]$$

Define:

$$\gamma_t(i) = \text{Prob}[q_t = s(i) \mid X, \theta]$$

then:

$$\gamma_t(i) \propto \alpha_t(i)\, \beta_t(i)$$

Similarly, we can calculate the probability that the system transits from state $s(i)$ to state $s(j)$ at time t:

$$\text{Prob}[q_t = s(i), q_{t+1} = s(j) \mid X, \theta] \propto$$
$$\text{Prob}\left[x_1, \ldots, x_{t-1}, q_{t-1} = s(k) \,\middle|\, \theta\right] \times \text{Prob}[q_{t+1} = s(j) \mid q_t = s(i)] \times$$
$$\text{Prob}\left[x_{t+1} \mid q_{t+1} = s(k), \theta\right] \times \text{Prob}\left[x_{t+2}, \ldots, x_T \mid q_{t+1} = s(k), \theta\right]$$

Recalling the definition of $\xi_t(i, j)$:

$$\xi_t(i, j) = \text{Prob}[q_t - s(i), q_{t+1} = s(j) \mid X, \theta]$$

then:

$$\xi_t(i, j) \propto \alpha_t(i)\, a(i, j)\, b_{t+1}(j)\, \beta_{t+1}(j)$$

Note that ξ_t is *not* defined for $t = T$. The values of γ_t and ξ_t are normalized so that they represent proper probability measures:

$$\sum_{k=1}^{N} \gamma_t(k) = 1$$

$$\sum_{k=1}^{N} \sum_{l=1}^{N} \xi_t(k, l) = 1$$

Again, note that, consistent with the definitions of the variables:

$$\gamma_t(i) = \sum_{k=1}^{N} \xi_t(i, k)$$

As described earlier, the parameter set θ is updated using x_t, γ_t and ξ_t. The log likelihood can be computed by summing $\alpha_T(i)$ over the states:

$$\log \mathcal{L}[\theta \mid X] = \log \sum_{i=1}^{N} \text{Prob}[X, q_T = s(i) \mid \theta] = \log \sum_{i=1}^{N} \alpha_T(i)$$

MLE Implementation

Implementation Issues

There are several implementation issues associated with actually fitting a model to data. We deal with three here:

- Selecting the number of states.

- Initializing and terminating the algorithm. This has three sub-issues:
  - Generating an intial estimate of the parameters $\theta^{(0)}$.
  - Terminating the algorithm.
  - Avoiding local maxima of the log likelihood.
- Scaling to avoid under- and over-flow conditions in the forward-backward recursions.

Selecting the Number of States:

In MLE one can always increase the likelihood by adding parameters, but, as one adds parameters, the risk of overfitting is also increased. A trade-off mechanism is needed.

The Bayesian Information Criterion (BIC) is the log likelihood penalized for the number of free parameters [McLachlan 2000]. The BIC is not a test of significance (*i.e.*, it is not used to accept or reject models), but it does provide a means of ranking models that have been fit on the same data. The model with the smallest (*i.e.*, usually most negative) BIC is preferred.

The BIC given the likelihood $\mathcal{L}$, the number of free parameters κ, and the number of observations n is

$$\text{BIC}[X, \theta] = -2 \log \mathcal{L} + \kappa \log n$$

In the HMMUG model the number of observations is T, and the number of *free* parameters for ι, a, μ, and σ, respectively, is

$$\kappa[\theta] = (N - 1) + N(N - 1) + N + N = N(N + 2) - 1$$

The HMMUG model's BIC is, therefore,

$$\text{BIC}[X, \theta] = -2 \log \mathcal{L}[\theta \mid X] + (N(N + 2) - 1) \log T$$

There are alternatives to the BIC [McLachlan 2000] that involve similar concepts of log likelihood penalized for model complexity.

Initialization and Termination

The EM algorithm starts with an initial guess of the parameters θ. The algorithm can get stuck in a local maximum, so the choice of an initial θ is more than just an issue of efficiency. Several approaches have been suggested [Finch *et al.* 1989] [Karlis 2001] [Karlis *et al.* 2003]. Most termination criteria do not detect convergence *per se* but lack of progress, and the likelihood function has "flat" regions that can lead to premature termination [Karlis 2001].

Therefore, it makes sense to make the termination criteria reasonably strict, and it also makes sense to start the algorithm at multiple starting points. [Karlis 2001] [Karlis *et al.* 2003]. An approach suggested in [Karlis *et al.* 2003] is to run multiple starting points for a limited number of iterations, pick the one with the highest likelihood and then run that choice using a fairly tight termination tolerance. This is the approach taken in the demonstrations below.

Scaling

Consider the forward recursion:

$$\alpha_t(i) = b_t(i) \sum_{k=1}^{N} \alpha_{t-1}(k)\, a(k, i)$$

Repeated multiplication by $b_t(i)$ at each time step can cause serious computational problems. In the discrete case as discussed in [Rabiner 1989] $b_t(i) << 1$ and the computation of $\alpha_t(i)$ is driven exponentially towards zero. In the continuous case, however, $b_t(i)$ may take on any value and in the Gaussian case the expected value of $b_t(i)$ is $1/\sigma\sqrt{2\pi}$. Thus, $\alpha_t(i)$ may be driven to 0 or ∞ depending upon σ. In the case of time series of financial returns $\sigma << 1 \Big/ \sqrt{2\pi}$; hence, $b_t(i)$ tends to be >>1 and the problem is more one of over-flow than under-flow.

The solution, as discussed in [Rabiner 1989], is to scale the computations in a manner which will still allow one to use the same forward-backward recursions. For each α_t compute a normalization constant v_t and apply it to α_t to produce a normalized $\tilde{\alpha}_t$ that sums to 1:

$$v_t = 1 \Big/ \sum_{k=1}^{N} \alpha_t(k)$$

$$\tilde{\alpha}_t(i) = v_t\, \alpha_t(i)$$

At each recursion use the normalized $\tilde{\alpha}_t$ to compute an unnormalized α_{t+1} which is then itself normalized and used in the next iteration. Note that as the recursion progresses:

$$\tilde{\alpha}_t(i) = \alpha_t(i) \prod_{u=1}^{t} v_u$$

On the backward recursion apply the same normalization constant so that a normalized $\tilde{\beta}_t$ is produced with the same scaling as $\tilde{\alpha}_t$. Note that:

$$\tilde{\beta}_t(i) = \beta_t(i) \prod_{u=t}^{T} v_u$$

As [Rabiner 1989] shows, the effects of the normalization constants cancel out in the numerators and denominators of the computations of γ_t and ξ_t.

Note that the true value of α_T can be recovered from the scaled values:

$$1 = \sum_{k=1}^{N} \tilde{\alpha}_T(k) = \left(\prod_{t=1}^{T} v_t\right) \sum_{k=1}^{N} \alpha_T(k) \implies \sum_{i=1}^{N} \alpha_T(i) = 1 \Big/ \prod_{t=1}^{T} v_t$$

and used to compute the log likelihood

$$\log \mathcal{L}[\theta \mid X] = \log \sum_{i=1}^{N} \alpha_T(i) = -\sum_{t=1}^{T} \log v_t$$

At this point, the complete algorithm can be set out.

EM (Baum-Welch) Algorithm for the HMMUG

This subsection presents a complete end-to-end description of the computations of one iteration of the EM-algorithm and a brief description of the criteria for termination. Parameters without superscript represents the current estimates (*i.e.*, θ instead of $\theta^{(\eta)}$, with the successor estimates for the next iteration by a "+" superscript (*i.e.*, θ^{+} instead of $\theta^{(\eta+1)}$).

E-Step

For observation and state, the density is computed:

$$b_t(i) = \psi[x_t \mid \mu(i), \sigma(i)]$$

The forward recursions are initialized to:

$$\alpha_1(i) = \imath(i)\, b_1(i)$$

then scaled:

$$v_1 = 1 \Big/ \sum_{k=1}^{N} \alpha_1(k)$$

$$\tilde{\alpha}_1(i) = v_1\, \alpha_1(i)$$

The forward recursions continue for $t = 2, \ldots, T$ computing $\alpha_t(i)$ then scaling it to $\tilde{\alpha}_t(i)$:

$$\alpha_t(i) = b_t(i) \sum_{k=1}^{N} \tilde{\alpha}_{t-1}(k)\, a(k, i)$$

$$v_t = 1 \Big/ \sum_{k=1}^{N} \alpha_t(k)$$

$$\tilde{\alpha}_t(i) = v_t\, \alpha_t(i)$$

The backward recursion is intialized and proceeds backward from $t = T, \ldots, 1$:

$$\tilde{\beta}_T(i) = v_T$$

$$\tilde{\beta}_t(i) = v_t \sum_{k=1}^{N} a(i, k)\, b_{t+1}(k)\, \tilde{\beta}_{t+1}(k)$$

The values of γ and ξ are estimated using the scaled forward-backward values:

$$\gamma_t(i) = \tilde{\alpha}_t(i)\, \tilde{\beta}_t(i) \Big/ \sum_{k=1}^{N} \tilde{\alpha}_t(k)\, \tilde{\beta}_t(k)$$

$$\xi_t(i, j) = \tilde{\alpha}_t(i)\, a(i, j)\, b_{t+1}(j)\, \tilde{\beta}_{t+1}(j) \Big/ \sum_{k=1}^{N} \sum_{l=1}^{N} \tilde{\alpha}_t(k)\, a(k, l)\, b_{t+1}(l)\, \tilde{\beta}_{t+1}(l)$$

M-Step

The updated parameters θ^{+} are:

$$\imath^{+}(i) = \gamma_1(i)$$

$$a^{+}(i, j) = \sum_{t=1}^{T-1} \xi_t(i, j) \Big/ \sum_{t=1}^{T-1} \gamma_t(i)$$

$$\mu^+(i) = \sum_{t=1}^{T} \gamma_t(i)\, x_t \Big/ \sum_{t=1}^{T} \gamma_t(i)$$

$$\sigma^+(i) = \sqrt{\sum_{t=1}^{T} \gamma_t(i)\, (x_t - \mu^+(i))^2 \Big/ \sum_{t=1}^{T} \gamma_t(i)}$$

Log Likelihood

As noted earlier the log likelihood is computed from the scaling constants:

$$\log \mathcal{L}[\theta \mid X] = -\sum_{t=1}^{T} \log v_t$$

Termination Criteria

In the code developed here the relative change in log likelihood is used as the primary termination criterion. For some positive constant $\tau << 1$ the algorithm is terminated when for the $(\eta + 1)^{\text{th}}$ iteration:

$$\log \mathcal{L}\left[\theta^{(\eta+1)} \mid X\right] / \log \mathcal{L}\left[\theta^{(\eta)} \mid X\right] - 1 \leq \tau$$

Other choices of termination criteria are covered in [Karlis 2001] and [Karlis 2003].

In addition, a maximum iteration limit is set and at that point the algorithm terminates even if the log likelihood tolerance has not been achieved; one can look at the convergence of the log likelihood function and accept the solution or restart the algorithm using either the final parameter estimates from the prior run or a new initialization.

Programming Considerations

This tutorial uses *Mathematica* to implement the Baum-Welch algorithm along with some useful supporting functions.

xHMMUG[*data*, $\theta^{(0)}$]

The primay function is xHMMUG[] which performs the actual MLE. It's structure can be summarized as follows:

- input data vector X and initial $\theta^{(0)}$
- resolve options and set up working variables
- compute initial α, v, b, and and log $\mathcal{L}$
- begin loop
  - E-step: compute current β, γ, and ξ
  - M-step: update $\theta = \{\iota, a, \mu, \text{and } \sigma\}$
  - log likelihood: compute next α, v, b, and and log $\mathcal{L}$
  - append log $\mathcal{L}$ history vector
  - break out of loop if $\mathcal{L}$ has converged or max iterations reached

- end loop
- compute BIC
- return results: θ, γ, BIC, and log $\mathcal{L}$ history

Note that an initial log likelihood is computed before the main loop starts. Normally, the log likelihood is computed immediately after the M-step but the forward-backward algorithm also uses the αs in the E-step, These computations are organized so that the log likelihood reported at termination refers to the most recently updated θ.

Exploiting List Processing

Mathematica, in common with other modern tools such as R, MATLAB, and others, has a syntax which supports list or array processing. Using these capabilities usually results in code which is simpler and more efficient. For example, consider the expression for $\alpha_t(i)$:

$$\alpha_t(i) = b_t(i) \sum_{k=1}^{N} \alpha_{t-1}(k)\, a(k, i)$$

Using $\underline{\times}$ to denote element-by-element multiplication, the expression above can be stated as:

$$\alpha_t = b_t \underline{\times} (\alpha_{t-1}\, a)$$

Similarly, consider the expression for $\xi_t(i, j)$:

$$\xi_t(i, j) \propto \alpha_t(i)\, a(i, j)\, b_{t+1}(j)\, \beta_{t+1}(j)$$

Using $\otimes$ to denote the outer or Kronecker product, this can be restated as:

$$\xi_t \propto a \underline{\times} \left(\alpha_t \bigotimes (b_{t+1} \underline{\times} \beta_{t+1})\right)$$

It is not necessary to hold onto ξ_t for each iteration. The code in xHMMUG[] computes, normalizes, and accumulates ξ in one statement:

```
mnXi += #/Total[Flatten[#]]& [mnA KroneckerProduct[mnAlpha[[t]], mnBeta[[t + 1]] mnB[[t + 1]]]]
```

Unfortunately, the list or array conventions vary from one system to the next. Care needs to be taken if one tries, *e.g.*, to transcribe *Mathematica* code into R or MATLAB. At least one of the references below [Zucchini 2009] contains samples of R code.

Equilibrium Distribution of the Markov Chain

The equilibrium distribution for the Markov chain, π(i), is the long-run probability that the system finds itself in state $s(i)$:

$$\pi(i) = \sum_{k=1}^{N} a(i,\ k)\, \pi(i)$$

or

$$\mathbf{A}^T \boldsymbol{\pi} = \boldsymbol{\pi}$$

For finite-state Markov chains, the equilibrium distribution can be found by solving the following

equations, where **O** is a matrix of 1s, $\mathbb{1}$ is a vector of 1s, and **0** is a vector or 0s:

$$\mathbf{A}^T \boldsymbol{\pi} - \mathbf{I}\,\boldsymbol{\pi} = \mathbf{0} \text{ and } \mathbf{O}\,\boldsymbol{\pi} = \mathbb{1} \implies \left(\mathbf{A}^T + \mathbf{O} - \mathbf{I}\right)\pi = \mathbb{1}$$

The general problem of finding the equilibrium distribution of a Markov chain is quite difficult, but the approach above often works for small problems.

Mathematica Functions

The functions presented below are teaching and demonstration tools, lacking even rudimentary error checking and handling. It would be easy to add the required features, but that would greatly increase the number of lines of code and obscure the algorithms. They are, however, perfectly usable if one is careful about the inputs.

Equilibrium Distribution for a Finite State Markov Chain

Description

Compute the equilibrium distribution for an ergodic, finite state Markov chain.

Input

The transition matrix of the Markov chain, such that
mnTransitionMatrix[[i, j]] = Prob[$q_{t+1} = s(j) \mid q_t = s(i)$]. There are no options.

Output

The equilibrium distribution as a numeric vector.

Note

This function is of limited generality but is quite useful for small problems.

Code

```
xMarkovChainEquilibrium[mnTransitionMatrix_] := Inverse[
    Transpose[mnTransitionMatrix] - IdentityMatrix[Length[mnTransitionMatrix]] + 1].
  ConstantArray[1, Length[mnTransitionMatrix]];
```

Random θ Initializer

Description

Generate a random θ to intialize the EM algorithm for HMMUG models.

Input

There are two inputs: the data and the number of states:

- the data are represented by a numeric vector, and
- the number of states is a positive integer.

There are no options.

Output

The result is returned as a list of data representing θ; it contains the following four components:

- ι - initial state probability vector (numeric vector),
- a - transition matrix of the Markov chain (numeric square matrix),
- μ - state means (numeric vector), and
- σ - state standard deviations (numeric vector).

Note

The output is in a form in which in can be used directly as the second argument to xHMMUG[], *i.e.*, $\theta^{(0)}$.

The term "random" must be taken with a grain of salt. What is produced randomly is a candidate state transition matrix. The intial state is estimated from its equilibrium distribution. A random spread for the mean and sdev vectors is then generated based on the total sample mean and sdev.

```
xHMMUGRandomInitial[vnData_, iStates_] := Module[
  {mnA, i, vnGamma, vnIota, vnM, vnQ, vnS, t, mnW},
  (* Generate a random transition matrix (a) *)
  mnA = (# / Total[#]) & /@ (RandomReal[{0.01, 0.99}, {iStates, iStates}]);
  (* Compute (ι) from (a) *)
  vnIota = xMarkovChainEquilibrium[mnA];
  (* Mean (μ) and sdev (σ) for each state *)
  vnM = ((# / Total[#]) &[RandomReal[{-0.99, 0.99}, iStates]]) Mean[vnData];
  vnS = Sqrt[((# / Total[#]) &[RandomReal[{0.1, 0.99}, iStates]]) Variance[vnData]];
  (* return θ *)
  {vnIota, mnA, vnM, vnS}
 ];
```

EM Algorithm for HHMs with Univariate Gaussian Outcomes

Description

MLE fit of a HMMUG model using the Baum-Welch, or forward-backward, version of the EM algorithm.

Input

There are two inputs: the raw data and intial parameters:

- the raw data as a numeric vector, and
- the parameters θ as a list containing $\{\iota, a, \mu, \sigma\}$.

There are two options controlling termination:

- "LikelihoodTolerance" representing the minimum change in the log likelihood function required to terminate the EM iterations, and
- "MaxIterations" which are the maximum number of EM iterations before the function terminates.

The *lhs* of the option rules are strings.

Output

The result is returned as a list of rules containing:

- "ι" → the initial state probability vector,
- "a" → the transition matrix,
- "μ" → the mean vector,
- "σ" → the standard deviation vector,
- "γ" → the vector of periodic state probabilities, *i.e*, $\gamma[[t, i]] = \text{Prob}[q_t = s(i)]$,
- "BIC" → the Bayesian Information Criterion for the current fit, and
- "LL" → the log likelihood history, *i.e.*, a list of $\mathcal{L}[\theta \mid X]$ after each EM iteration.

The *lhs* of the rules above are strings.

Note

Restarting the fit from the prior run is straightforward. If vxH is the result of the prior run, then {"ι" , "a", "μ", "σ"} /. vxH will pull out θ which can be used to restart the algorithm at the point at which it terminated. Typically, one should consider changing the iteration limit or log likelihood tolerance options when restarting, although this is not always necessary.

Code

```
(* Options for xHMMUG function *)
Options[xHMMUG] = {"LikelihoodTolerance" → 10.^-7, "MaxIterations" → 400};

(* Input X, {ι, a, μ, σ}, and optionally a tolerance and iteration limit. *)
xHMMUG[vnData_, {vnInitialIota_, mnInitialA_, vnInitialMean_, vnInitialSdev_},
    OptionsPattern[]] := Module[
    {mnA, mnAlpha, mnB, mnBeta, nBIC, mnGamma, vnIota, vnLogLikelihood,
     iMaxIter, vnMean, iN, vnNu, vnSdev, t, iT, nTol, mnWeights, mnXi},
    (* Resolve options *)
    nTol = OptionValue["LikelihoodTolerance"];
    iMaxIter = OptionValue["MaxIterations"];
    (* Initialize variables *)
    iT = Length[vnData];
    iN = Length[mnInitialA];
    vnIota = vnInitialIota;
    mnA = mnInitialA;
    vnMean = vnInitialMean;
    vnSdev = vnInitialSdev;
    (* Initial log ℒ *)
    (* --- b *)
    mnB = Table[
      PDF[NormalDistribution[vnMean[[#]], vnSdev[[#]]], vnData[[t]]] & /@ Range[iN],
      {t, 1, iT}
     ];
    (* --- α and ν *)
    mnAlpha = Array[0. &, {iT, iN}];
    vnNu = Array[0. &, iT];
    mnAlpha[[1]] = vnIota mnB[[1]];
    vnNu[[1]] = 1 / Total[mnAlpha[[1]]];
    mnAlpha[[1]] *= vnNu[[1]];
    For[t = 2, t ≤ iT, t++,
     mnAlpha[[t]] = (mnAlpha[[t - 1]].mnA) mnB[[t]];
     vnNu[[t]] = 1 / Total[mnAlpha[[t]]];
     mnAlpha[[t]] *= vnNu[[t]];
    ];
    (* --- log ℒ *)
    vnLogLikelihood = {-Total[Log[vnNu]]};
    (* Main Loop *)
    Do[
     (* --- E-Step *)
     (* --- --- β *)
```

```
mnBeta = Array[0. &, {iT, iN}];
mnBeta[[iT, ;;]] = vnNu[[iT]];
For[t = iT - 1, t ≥ 1, t--,
 mnBeta[[t]] = mnA.(mnBeta[[t + 1]] mnB[[t + 1]]) vnNu[[t]];
];
(* --- --- γ *)
mnGamma = (# / Total[#]) & /@ (mnAlpha mnBeta);
(* --- --- ξ; note that we do not need the individual ξt s *)
mnXi = Array[0. &, {iN, iN}];
For[t = 1, t ≤ iT - 1, t++,
 mnXi += (# / Total[Flatten[#]]) &[
   mnA KroneckerProduct[mnAlpha[[t]], mnBeta[[t + 1]] mnB[[t + 1]]]];
];
(* --- M-Step*)
(* --- --- a *)
mnA = (# / Total[#]) & /@ mnXi;
(* --- --- ι *)
vnIota = mnGamma[[1]];
(* --- --- observation weights *)
mnWeights = (# / Total[#]) & /@ (mnGamma^T);
(* --- --- μ and σ *)
vnMean = mnWeights.vnData;
vnSdev = √(Total /@ (mnWeights (vnData - # & /@ vnMean)^2));
(* --- Log Likelihood *)
(* --- --- b *)
mnB = Table[
  PDF[NormalDistribution[vnMean[[#]], vnSdev[[#]]], vnData[[t]]] & /@ Range[iN],
  {t, 1, iT}
 ];
(* --- --- α and ν *)
mnAlpha = Array[0. &, {iT, iN}];
vnNu = Array[0. &, iT];
mnAlpha[[1]] = vnIota mnB[[1]];
vnNu[[1]] = 1 / Total[mnAlpha[[1]]];
mnAlpha[[1]] *= vnNu[[1]];
For[t = 2, t ≤ iT, t++,
 mnAlpha[[t]] = (mnAlpha[[t - 1]].mnA) mnB[[t]];
 vnNu[[t]] = 1 / Total[mnAlpha[[t]]];
 mnAlpha[[t]] *= vnNu[[t]];
```

```
  ];
  (* --- --- log ℒ *)
  vnLogLikelihood = Append[vnLogLikelihood, -Total[Log[vnNu]]];
  (* --- --- likelihood test for early Break[] out of Do[] *)
  If[vnLogLikelihood[[-1]] / vnLogLikelihood[[-2]] - 1 ≤ nTol, Break[]],
  (* --- Max iterations for Do[] *)
  {iMaxIter}
 ];
 (* BIC *)
 nBIC = -2 vnLogLikelihood[[-1]] + (iN (iN + 2) - 1) Log[iT];
 (* Return ι, a, μ, σ, γ, BIC, log ℒ as rule vector *)
 {"ι" -> vnIota, "a" → mnA, "μ" → vnMean,
  "σ" → vnSdev, "γ" → mnGamma, "BIC" → nBIC, "LL" → vnLogLikelihood}
];
```

---

xHMMUG Report

Description

Produce a summary report of the results of an xHMMUG[] fit.

Input

A vector of rules, typically as produced from xHMMUG[]. There are no options.

Output

The function does produce a result but Print[]s out the summary to the notebook as a side-effect.

Code

```
xHMMUGReport[hmmug_] := Module[
  {},
  Print["ι̂ = ", MatrixForm["ι" /. hmmug]];
  Print["â = ", MatrixForm["a" /. hmmug]];
  Print["π̂ = ", MatrixForm[xMarkovChainEquilibrium["a" /. hmmug]]];
  Print["μ̂ = ", MatrixForm["μ" /. hmmug]];
  Print["σ̂ = ", MatrixForm["σ" /. hmmug]];
  Print["ℒ[θ̂|X]= ", Last["LL" /. hmmug]];
  Print["τ = ", #[[-1]] / #[[-2]] - 1 &["LL" /. hmmug]];
  Print["# = ", Length["LL" /. hmmug]];
  Print["BIC = ", "BIC" /. hmmug];
 ];
```

HMMUG Simulator

Description

Simulate an HMMUG model with specified parameters for a specified number of periods.

Input

There are two inputs:

- θ itself as a list containing the parameters $\{\iota, \alpha, \mu, \sigma\}$, and
- a positive integer simulation length.

There are no options.

Output

Returns a 2-list;

- the vector of hidden states expressed as integers and
- the vector of simulated data.

Note

The entire simulation function is a single line of *Mathematica*.

Code

```
xSimHMMUG[{vnStart_, mnMarkovChain_, vnMean_, vnSdev_}, iSimLength_] :=
  {#, RandomReal[NormalDistribution[vnMean[[#]], vnSdev[[#]]] & /@ #} &[
   NestList[
    RandomChoice[mnMarkovChain[[#]] → Range[Length[mnMarkovChain]]] &,
    RandomChoice[vnStart → Range[Length[mnMarkovChain]]],
    iSimLength - 1
   ]
  ];
```

S&P 500 Demonstration

The dataset used to demonstrate the code above is the monthly log returns for the S&P 500 from Jan 1969 to the last full month of the current date.

Running the code below in mid-December 2009 results in just under 40-years of monthly log returns for the S&P 500 index. The the ticker "^GSPC" is price only, dividends are not included. For the purposes of this exposition, this is good enough.

Note that this period include a great deal of variety in the market: the sideways markets of the 1970s, the extended bull market that with hindsight appears to have ended in the early 2000s, and the difficult markets of the first decade of the century. It includes several interesting events such as the explosion of interest rates in the 1980s, the 1987 crash, the tech bubble, the housing bubble, *etc.*

```
FinancialData["^GSPC", "Name"]
```

```
S And P 500 Indexrth
```

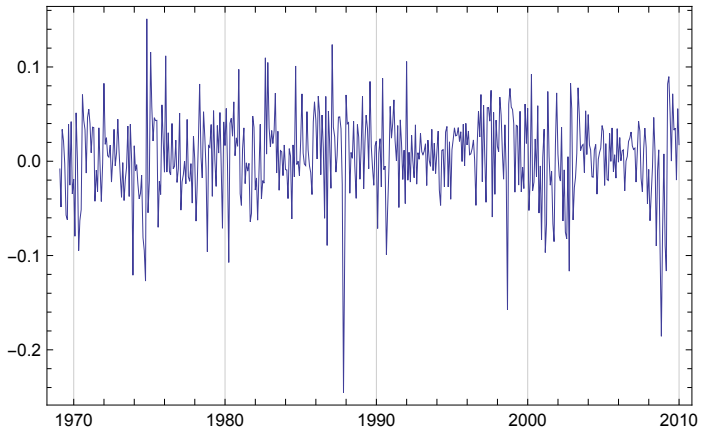
```
mxSP500 = FinancialData["^GSPC", {1968, 12, 1}];
mxSP500 = Most[Last /@ Split[mxSP500, #1[[1, 2]] == #2[[1, 2]] &]];
mxSP500LogReturns = {Rest[First /@ mxSP500], Differences[Log[Last /@ mxSP500]]}^T;
```

```
Print["Date Range:       ", {First@First[#], First@Last[#]} &[mxSP500LogReturns]];
Print["Number of Months: ", Length[mxSP500LogReturns]]
```

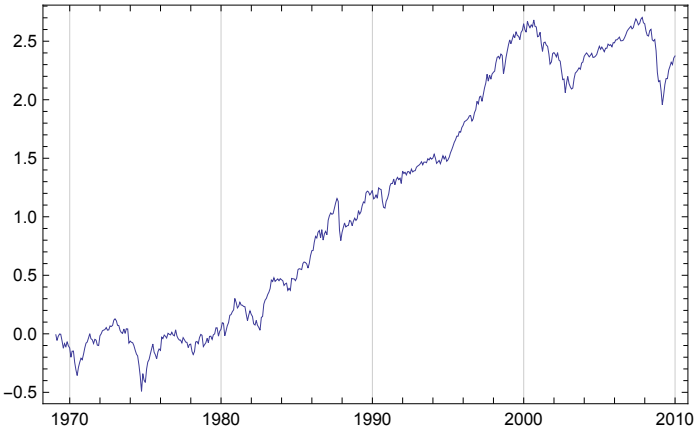
```
Date Range:       {{1969, 1, 31}, {2009, 12, 31}}
Number of Months: 492
```

Plots of the period and cumulative returns and a histogram of returns are:

```
DateListPlot[mxSP500LogReturns, Joined → True, PlotRange → All]
```



```
DateListPlot[{First /@ #, Accumulate[Last /@ #]}ᵀ &[mxSP500LogReturns],
 Joined → True, PlotRange → All]
```



The log returns are noticeably negatively skewed and leptokurtotic (heavy-tailed):

```
Through[{Mean, StandardDeviation, Skewness, Kurtosis}[mxSP500LogReturns[[All, 2]]]]
```

```
{0.0048245, 0.0452431, -0.709973, 5.58923}
```

Model Fits

Before fitting fancy hidden Markov models, it would be wise to evaluate the simple assumption of log Normality. For the S&P 500's log returns, the log likelihood is computed as follows:

Univariate Gaussian Model

For a univariate Gaussian distribution the log density of a single observation is:

$$\log \psi[x \mid \mu, \sigma] = -\frac{1}{2}\log[2\pi] - \log[\sigma] - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$$

hence, for the observations X:

$$\mathcal{L}[\theta \mid X] = \sum_{t=1}^{T} \log \psi[x_t \mid \mu, \sigma] = -T\left(\frac{1}{2}\log[2\pi] + \log[\sigma]\right) - \sum_{t=1}^{T}\frac{1}{2}\left(\frac{x_t-\mu}{\sigma}\right)^2$$

Note that we are using the MLE of the standard deviation.

```
nMean = Mean[mxSP500LogReturns[[All, 2]]]
nSdev = √(Mean[mxSP500LogReturns[[All, 2]]^2] - nMean^2)
iSize = Length[mxSP500LogReturns[[All, 2]]]
```

```
0.0048245
```

```
0.0451971
```

```
492
```

The log likelihood is:

```
nLogLikelihood =
 -iSize (Log[2 π]/2 + Log[nSdev]) - 1/2 Total[((mxSP500LogReturns[[All, 2]] - nMean)/nSdev)^2]
```

```
825.47
```

The BIC is:

```
nBIC = -2 nLogLikelihood + 2 Log[iSize]
```

```
-1638.54
```

Two-State Model

As mentioned earlier, it's a good idea to initialize the algorithm at different points. Because some of these initial values may be *really* bad, *Mathematica* may display underflow warnings. It's usually okay at this stage to ignore them. A set of 20 intitial guesses are run for five iterations...

```
vxInitialTest["SP500", 2] =
  xHMMUG[mxSP500LogReturns[[All, 2]], #, "MaxIterations" → 5] & /@
   Table[xHMMUGRandomInitial[mxSP500LogReturns[[All, 2]], 2], {20}];
```

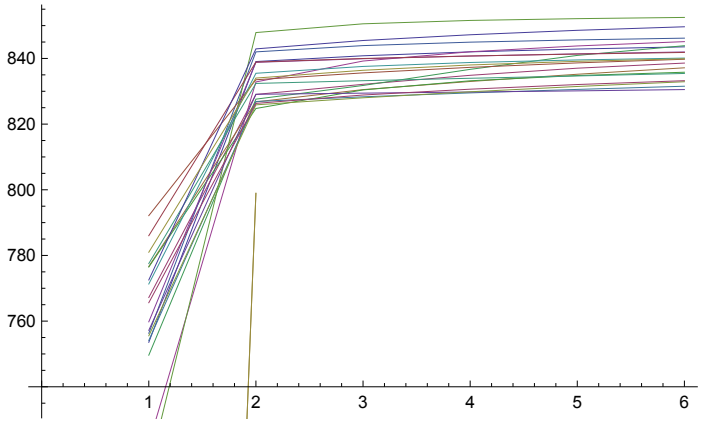
... then the one with the highest likelihood is selected...

```
iBest["SP500", 2] =
 Position[#, Max[#]] &[Last["LL" /. #] & /@ vxInitialTest["SP500", 2]][[1, 1]]
```

```
12
```

It's a good idea to examine the ensemble graphically to see if the number of iterations allowed for each candidate is reasonable:

```
ListPlot["LL" /. # & /@ vxInitialTest["SP500", 2], Joined → True]
```

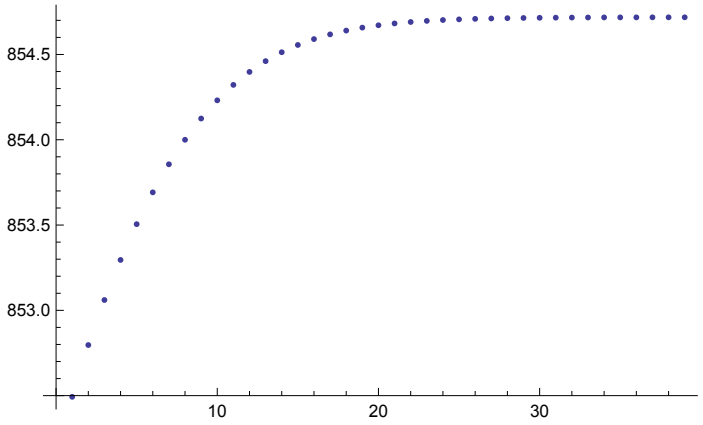


The best candidate is then used to intialize the model fit. Here the default tolerance and iteration limit is used:

```
vxHMMUG["SP500", 2] = xHMMUG[
   mxSP500LogReturns[[All, 2]],
   {"ι", "a", "μ", "σ"} /. (vxInitialTest["SP500", 2][[iBest["SP500", 2]]])
  ];
```

The convergence of the likelihood function should be examined:

```
ListPlot["LL" /. vxHMMUG["SP500", 2], PlotRange → All]
```



A report summarizing the result is:

```
xHMMUGReport[vxHMMUG["SP500", 2]]
```

$\hat{\iota} = \begin{pmatrix} 5.76112 \times 10^{-10} \\ 1. \end{pmatrix}$

$\hat{a} = \begin{pmatrix} 0.812364 & 0.187636 \\ 0.044969 & 0.955031 \end{pmatrix}$

$\hat{\pi} = \begin{pmatrix} 0.193328 \\ 0.806672 \end{pmatrix}$

$\hat{\mu} = \begin{pmatrix} -0.0188743 \\ 0.0104782 \end{pmatrix}$

$\hat{\sigma} = \begin{pmatrix} 0.0691238 \\ 0.0349897 \end{pmatrix}$

$\mathcal{L}[\hat{\theta} | X] = 854.718$

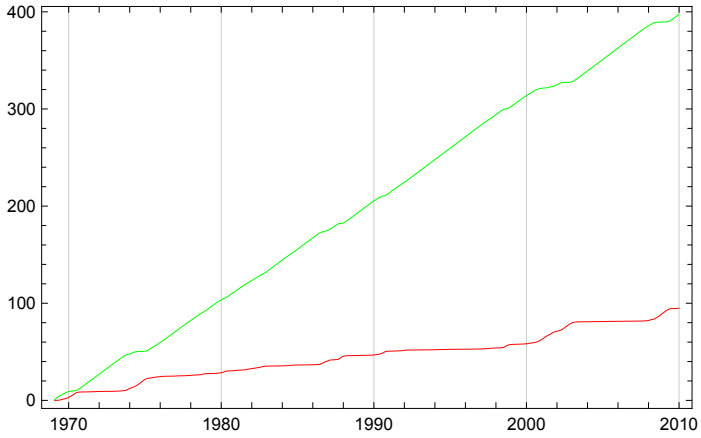
$\tau = 9.38355 \times 10^{-8}$

$\# = 39$

BIC = −1666.05

Finally, the values of γ_t are cumulatively summed. This makes the interplay across states easier to see compared with plotting the state probabilities:

```
DateListPlot[
 {mxSP500LogReturns[[All, 1]], #}^T & /@ (Accumulate /@ (("γ" /. vxHMMUG["SP500", 2])^T)),
 PlotStyle → {{Red}, {Green}}, Joined → True]
```



If there are doubts about the solution, then the number of initial candidates or the number of iterations set for the candidates can be varied. Once a candidate is selected the tolerance and iteration limit of the main fit may be adjusted. It's also reasonable to repeat the entire process above multiple times to check the convergence of the log likelihood. Finally, no claim is made that the method of generating initial candidates is optimal; it may be wise to consider alternatives.

Three-State Model

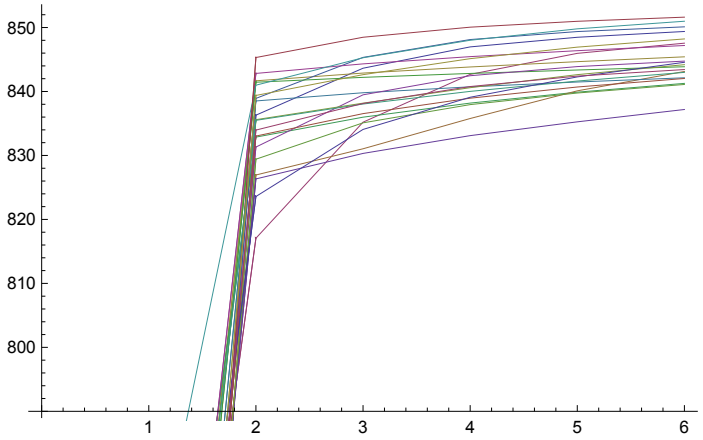
Here is the same analysis for a three-state model:

```
vxInitialTest["SP500", 3] =
  xHMMUG[mxSP500LogReturns[[All, 2]], #, "MaxIterations" → 5] & /@
   Table[xHMMUGRandomInitial[mxSP500LogReturns[[All, 2]], 3], {20}];
```

```
iBest["SP500", 3] =
 Position[#, Max[#]] &[Last["LL" /. #] & /@ vxInitialTest["SP500", 3]][[1, 1]]
```

```
15
```

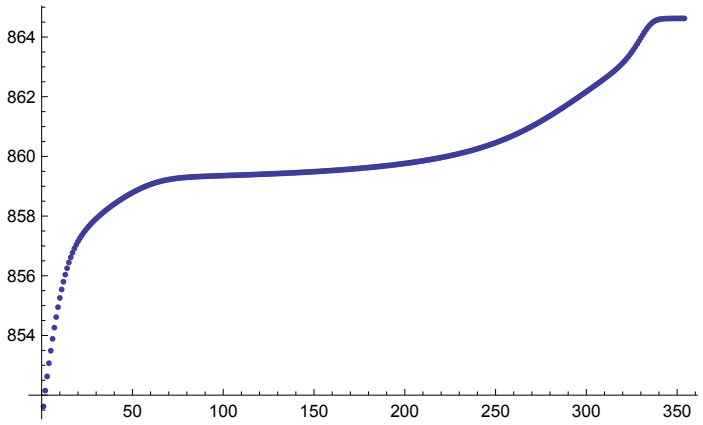
```
ListPlot["LL" /. # & /@ vxInitialTest["SP500", 3], Joined → True]
```



Previous trials have indicated that the iteration limit *sometimes* must be increased to achieve the desired tolerance.

```
vxHMMUG["SP500", 3] = xHMMUG[
   mxSP500LogReturns[[All, 2]],
   {"ι", "a", "μ", "σ"} /. (vxInitialTest["SP500", 3][[iBest["SP500", 3]]]),
   "MaxIterations" → 1000
  ];
```

```
ListPlot["LL" /. vxHMMUG["SP500", 3], PlotRange → All]
```




```
xHMMUGReport[vxHMMUG["SP500", 3]]
```

$$\hat{\iota} = \begin{pmatrix} 2.76515 \times 10^{-259} \\ 1. \\ 1.22483 \times 10^{-137} \end{pmatrix}$$

$$\hat{a} = \begin{pmatrix} 0.541067 & 0.458933 & 4.91812 \times 10^{-29} \\ 1.28261 \times 10^{-6} & 0.969118 & 0.0308804 \\ 0.195081 & 8.44613 \times 10^{-33} & 0.804919 \end{pmatrix}$$

$$\hat{\pi} = \begin{pmatrix} 0.0549045 \\ 0.815937 \\ 0.129159 \end{pmatrix}$$

$$\hat{\mu} = \begin{pmatrix} 0.0536356 \\ 0.00809838 \\ -0.0366716 \end{pmatrix}$$

$$\hat{\sigma} = \begin{pmatrix} 0.0204549 \\ 0.0356294 \\ 0.0695324 \end{pmatrix}$$

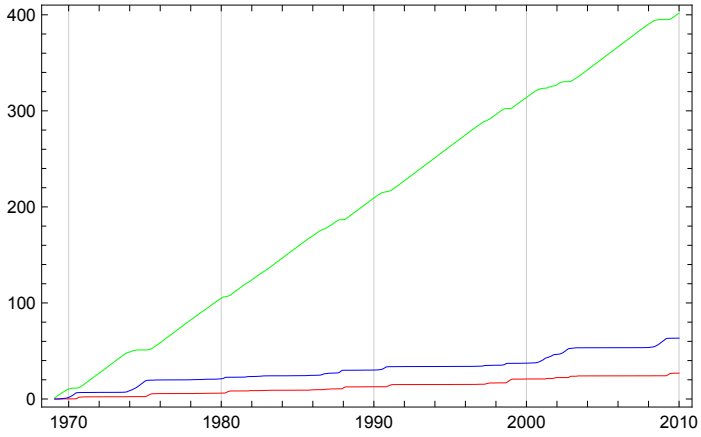
$\mathcal{L}[\hat{\theta}|X] = 864.624$

$\tau = 7.87853 \times 10^{-8}$

$\# = 354$

BIC = −1642.47

```
DateListPlot[
 {mxSP500LogReturns[[All, 1]], #}^T & /@ (Accumulate /@ (("γ" /. vxHMMUG["SP500", 3])^T)),
 PlotStyle → {{Red}, {Green}, {Blue}},
 Joined → True]
```



Summary

Summarizing results by comparing the BICs for each model:

```
TableForm[
 {{1, 2, 3},
   Join[{nBIC}, "BIC" /. # & /@ {vxHMMUG["SP500", 2], vxHMMUG["SP500", 3]}]}^T,
 TableHeadings → {None, {"#States", "BIC"}}, TableAlignments → Center]
```

#States	BIC
1	−1638.54
2	−1666.05
3	−1642.47

Run as of 21-Dec-2009, the two-state model is selected on the basis of its BIC.

A Few Observations

Consider the "best" model above, the two-state model:

```
xHMMUGReport[vxHMMUG["SP500", 2]]
```

$\hat{\iota} = \begin{pmatrix} 5.76112 \times 10^{-10} \\ 1. \end{pmatrix}$

$\hat{a} = \begin{pmatrix} 0.812364 & 0.187636 \\ 0.044969 & 0.955031 \end{pmatrix}$

$\hat{\pi} = \begin{pmatrix} 0.193328 \\ 0.806672 \end{pmatrix}$

$\hat{\mu} = \begin{pmatrix} -0.0188743 \\ 0.0104782 \end{pmatrix}$

$\hat{\sigma} = \begin{pmatrix} 0.0691238 \\ 0.0349897 \end{pmatrix}$

$\mathcal{L}[\hat{\theta}|X] = 854.718$

$\tau = 9.38355 \times 10^{-8}$

$\# = 39$

$\text{BIC} = -1666.05$

Based on monthly returns, the market appears to be in a "low volatility-moderately positive return" or "bull" state about 80% of the time and in a "high volatility-highly negative return" or "bear" state the remaining 20% of the time. The large transition probabilities on the main diagonal of the state transition matrix *a* indicate that the states are somewhat "sticky". Thus, once the market finds itself in a bull or bear state, it tends to stay there for a time.

This is the simple bull-bear model used as the initial example in the first section of the tutorial.

Simulation

Generating Simulated Data

A simulation, with timing, over the same time horizon as the original data with the two-state model's parameters is:

```
{nTime, {vnStates, vnSim}} = Timing[xSimHMMUG[
    {"ι", "a", "μ", "σ"} /. vxHMMUG["SP500", 2], Length[mxSP500LogReturns]]];
```

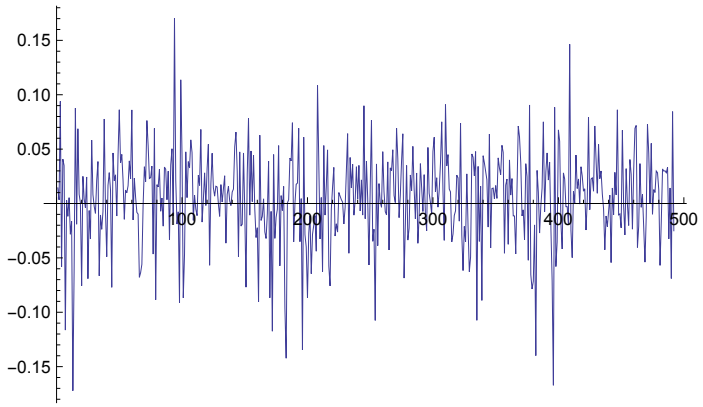
On a 2×2.93 GHz quad-core Intel Nehalem-based Xeon system, the time is < 5 milliseconds. Your time may be faster or slower but even on a relatively slow processor the code should run fairly quickly.

```
nTime
```

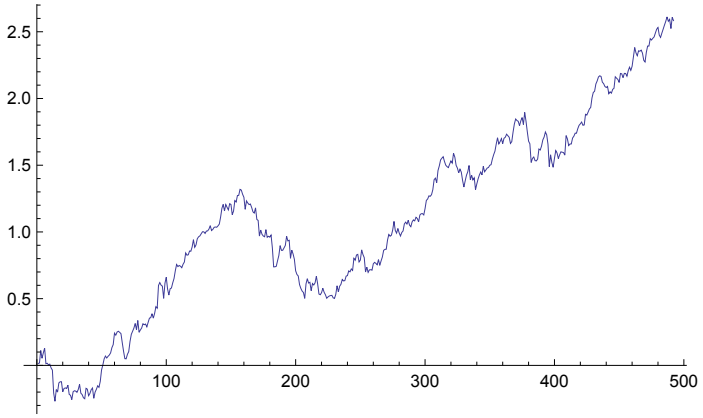
```
0.004454
```

Plots of the period and cumulative returns and a histogram of returns are:

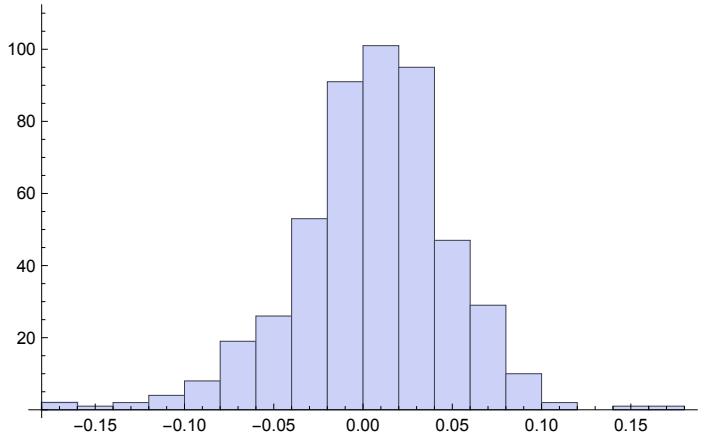
```
ListPlot[vnSim, Joined → True, PlotRange → All]
```




```
ListPlot[Accumulate[vnSim], Joined → True, PlotRange → All]
```



```
Histogram[vnSim]
```



The distribution should be noticeably negatively skewed and leptokurtotic:

```
Through[{Mean, StandardDeviation, Skewness, Kurtosis}[vnSim]]
```

```
{0.00525115, 0.0436044, -0.472083, 4.60125}
```

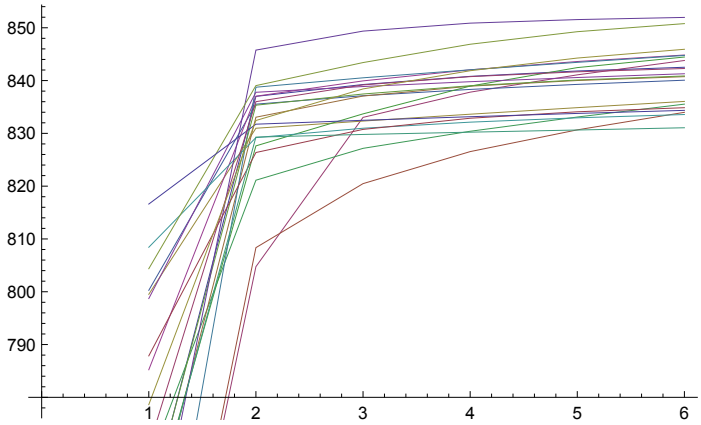
Fitting a Two-State HMMUG Model to the Simulation

```
vxInitialTest["Sim", 2] =
  xHMMUG[mxSP500LogReturns[[All, 2]], #, "MaxIterations" → 5] & /@
   Table[xHMMUGRandomInitial[mxSP500LogReturns[[All, 2]], 2], {20}];
```

```
iBest["Sim", 2] =
 Position[#, Max[#]] &[Last["LL" /. #] & /@ vxInitialTest["Sim", 2]][[1, 1]]
```

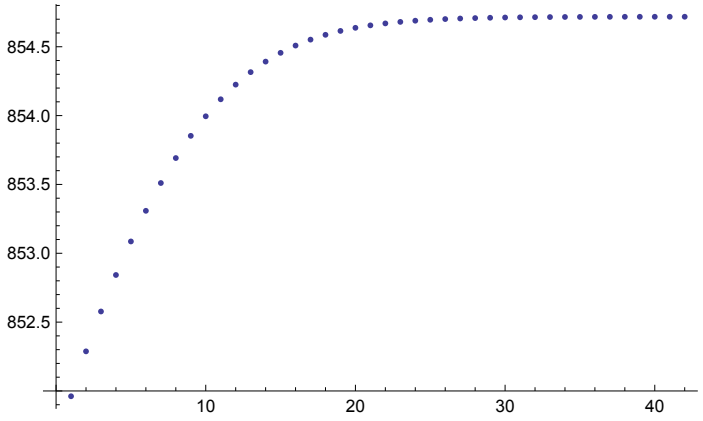
```
14
```

```
ListPlot["LL" /. # & /@ vxInitialTest["Sim", 2], Joined → True]
```



```
vxHMMUG["Sim", 2] = xHMMUG[
    mxSP500LogReturns[[All, 2]],
    {"ι", "a", "μ", "σ"} /. (vxInitialTest["Sim", 2][[iBest["Sim", 2]]])
  ];
```

```
ListPlot["LL" /. vxHMMUG["Sim", 2], PlotRange → All]
```



Evaluating the Model

Note that the assignment of parameters to states is arbitrary and dependent upon initial conditions.

The original model fit is:

```
xHMMUGReport[vxHMMUG["SP500", 2]]
```

$\hat{\iota} = \begin{pmatrix} 5.76112 \times 10^{-10} \\ 1. \end{pmatrix}$

$\hat{a} = \begin{pmatrix} 0.812364 & 0.187636 \\ 0.044969 & 0.955031 \end{pmatrix}$

$\hat{\pi} = \begin{pmatrix} 0.193328 \\ 0.806672 \end{pmatrix}$

$\hat{\mu} = \begin{pmatrix} -0.0188743 \\ 0.0104782 \end{pmatrix}$

$\hat{\sigma} = \begin{pmatrix} 0.0691238 \\ 0.0349897 \end{pmatrix}$

$\mathcal{L}[\hat{\theta} | X] = 854.718$

$\tau = 9.38355 \times 10^{-8}$

$\# = 39$

BIC = −1666.05

The fit on the data simulated using the above as parameters is:

```
xHMMUGReport[vxHMMUG["Sim", 2]]
```

$$\hat{\iota} = \begin{pmatrix} 2.62911 \times 10^{-10} \\ 1. \end{pmatrix}$$

$$\hat{a} = \begin{pmatrix} 0.811942 & 0.188058 \\ 0.044902 & 0.955098 \end{pmatrix}$$

$$\hat{\pi} = \begin{pmatrix} 0.192746 \\ 0.807254 \end{pmatrix}$$

$$\hat{\mu} = \begin{pmatrix} -0.0189478 \\ 0.0104746 \end{pmatrix}$$

$$\hat{\sigma} = \begin{pmatrix} 0.069162 \\ 0.0350023 \end{pmatrix}$$

$\mathcal{L}[\hat{\theta}|X] = 854.718$

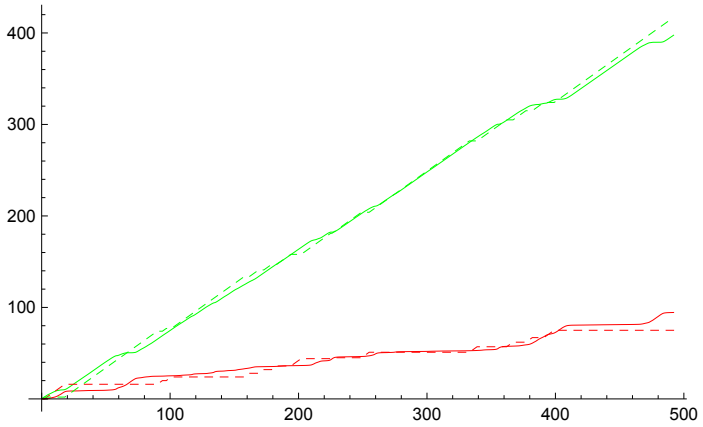
$\tau = 7.83003 \times 10^{-8}$

$\# = 42$

$BIC = -1666.05$

The cumulative γ-plots for the true hidden states produced by the simulation (dashed) and the fit (solid) are below. The assignment of the underlying states positionally is dependent upon intitialization.

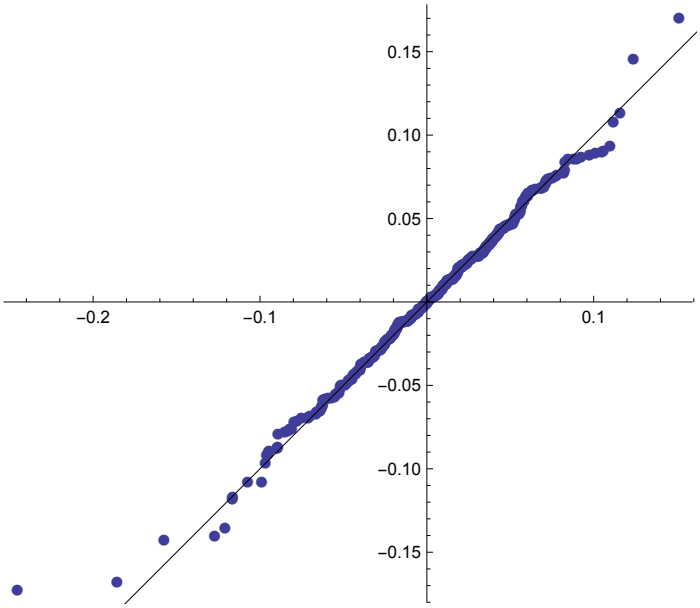
```
ListPlot[Join[Accumulate /@ Transpose["γ" /. vxHMMUG["Sim", 2]],
  Accumulate /@ Transpose[If[# == 1, {1, 0}, {0, 1}] & /@ vnStates]],
 PlotStyle → {{Red}, {Green}, {Red, Dashed}, {Green, Dashed}}, Joined → True]
```



Using a qq-plot or quantile plot to compare the distribution of the original and simulated data requires that the Statistical Plots package be loaded:

```
Needs["StatisticalPlots`"]
```

```
QuantilePlot[mxSP500LogReturns[[All, 2]], vnSim]
```



It's a good idea to run the above simulation several times.

A Few More Observations

If one proceeds with the working assumption that the two-state model fit on the S&P 500 above is a reasonable approximation of reality, then the simulation code can be used to study some interesting questions.

Annual Log Returns

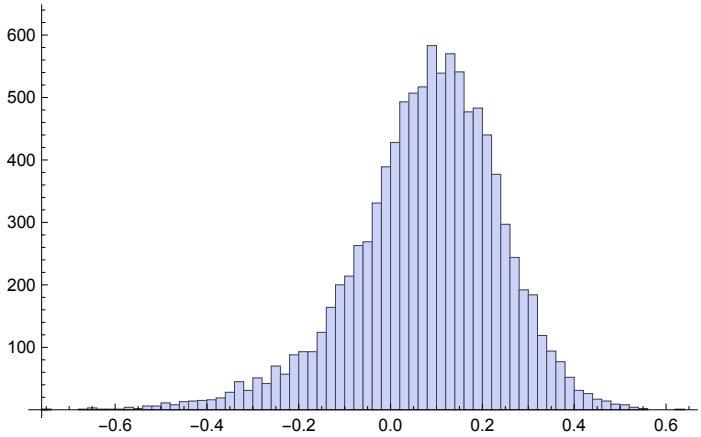
Although the distribution at the monthly level is clearly non-Gaussian (both the observed data and the resulting model), one may wonder how fast the Central Limit Theorem might kick in and cause the log returns to look more Gaussian. We can use the two-state model to simulate 12-month log returns for 10,000 samples:

```
vnYearOut = Table[
   Total[Last[xSimHMMUG[{"ι", "a", "μ", "σ"} /. vxHMMUG["SP500", 2], 12]]],
   {10 000}
  ];
```

```
Through[{Mean, StandardDeviation, Skewness, Kurtosis}[vnYearOut]]
```

```
{0.0799498, 0.159211, -0.612109, 4.04489}
```

```
Histogram[vnYearOut]
```



Even at 12-months, the distribution of annual returns is still noticeably negatively skewed and leptokurtotic, although both are somewhat attenuated compared to the monthly. The often applied simplifying assumption that returns are log Normally distributed is clearly not tenable even over year-long periods. This has significance for financial applications such as options pricing or investment management.

Decade Log Returns

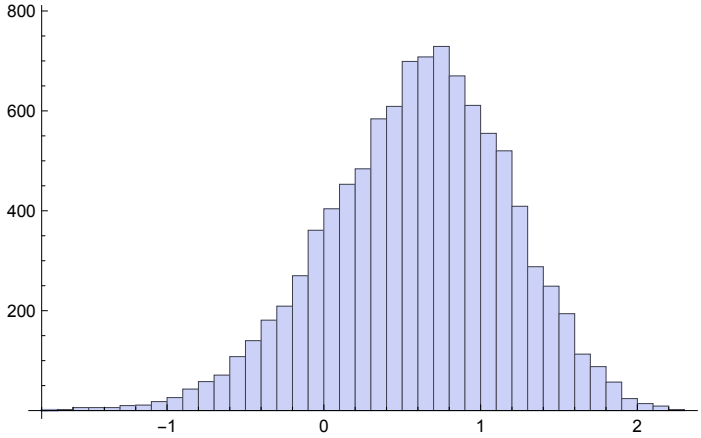
One might wonder if things settle down after a longer period. A decade-long simulation study appears below:

```
vnDecadeOut = Table[
   Total[Last[xSimHMMUG[{"ι", "a", "μ", "σ"} /. vxHMMUG["SP500", 2], 120]]],
   {10 000}
  ];
```

```
Through[{Mean, StandardDeviation, Skewness, Kurtosis}[vnDecadeOut]]
```

```
{0.599441, 0.577925, -0.316765, 3.10633}
```

```
Histogram[vnDecadeOut]
```



The kurtosis has approached the more Gaussian value of 3 but there is still significant negative skewness in the log returns. The expected return is $e^{0.6} - 1 \simeq 82 \times$ % for cases run as of 21 Dec 2009, but material draw downs are still an issue. Again, assumptions of log Normality appear wide of the mark even for decade-long returns.

Naive Bootstrap of Decade from Annual

The Markov chain operates at the monthly level. After a year there is not much "memory" of the state the system was in a year ago:

```
Print["\na12 = Prob[qt+12 = s(j)|qt = s(i)] = ",
 MatrixForm[MatrixPower[("a" /. vxHMMUG["SP500", 2])ᵀ, 12]ᵀ]]
```

$$a_{12} = \text{Prob}[q_{t+12} = s(j) | q_t = s(i)] = \begin{pmatrix} 0.226973 & 0.773027 \\ 0.185264 & 0.814736 \end{pmatrix}$$

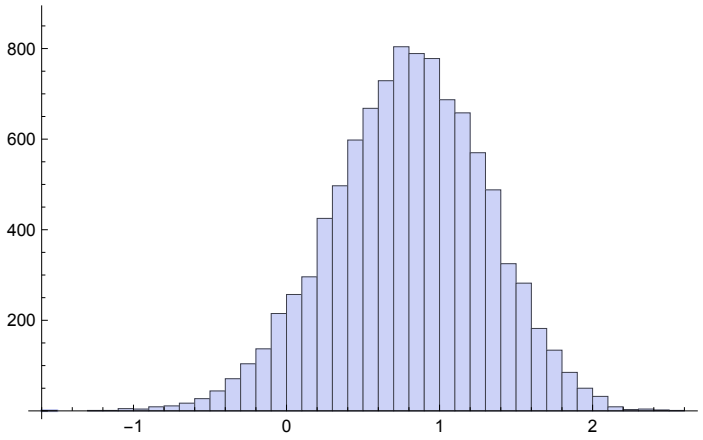
The rows are close the equilibrium distribution. This might cause one to think that after a year one could largely ignore the dynamics of the Markov chain and generate a sample of decade-long returns by direct random sampling from the annual returns. In other words, the annual simulation results are sampled and the effect of the Markov chain is otherwise ignored to produce decade-long simulations:

```
vnDecadeFromAnnual = Table[Total[RandomChoice[vnYearOut, 10]], {10 000}];
```

```
Through[{Mean, StandardDeviation, Skewness, Kurtosis}[vnDecadeFromAnnual]]
```

```
{0.795806, 0.506324, -0.192783, 3.06076}
```

```
Histogram[vnDecadeFromAnnual]
```



Not only is the mean log return significantly higher but the distribution is significantly less negatively skewed. Even out to a decade the dynamics captured by the HMM are important and must be accounted for. This "long memory" effect for a model that captures only monthly dynamics is interesting.

Next Steps

First of all, experiment. Try looking at different financial instruments at different time scales. Compare different periods.

Try different models. One obvious choice: a simple finite mixture of Guassian rather than a HMM. The BIC can provide guidance as to whether the extra parameters are justified in the HMM.

The form of the EM algorithm presented above is the most basic. There are methods for accelerating its convergence. There are also alternatives that involve direct maximization of the likelihood function [McLachlan 2008]. Once comfortable with the material in this tutorial, the student should study those alternatives.

The basic routine to fit HMMUG models can be easily extended to more complex cases. For example, if x_t is a vector with state-dependent multi-varitate Gaussian densities, then one has to replace the univariate PDF with the multivariate in b_t and update the state dependent mean vectors and covariance matrices using the same weights based on γ_t. In [Rabiner 1989] the more general case of using state-dependent finite mixtures of multivariate Guassian is covered.

The former change is trivial; the latter is slightly more difficult but still straightforward. The code above could have easily been written to accommodate them. However, the objectives here were more pedagogical than utilitarian, and these tasks are, therefore, left as an exercise for the student.

Special structures can be incorporated into the transition matrix or the model can be extended to a second-order Markov chain, *i.e.*, one in which the probability of the next state is conditioned on the prior two states. The Markov chain can be expressed in continuous time and the price dynamics expressed as regime-switching stochastic differential equations [Bhar *et al.* 2004] [Zucchini *et al.* 2009].

Densities other than Gaussian are accommodated in most instances by simply using them to compute $b_t(i)$ for forward-backward recursions, although the distributions must be ones for which the M-step can be accomplished in something at least approaching closed form. Parameter updates in the E-step are then done using MLE computations appropriate to the distributions at hand.

In some cases the M-step cannot be accomplished easily, and the conditional expectation must be computed by Monte Carlo. This is an example of Gibbs sampling and is a simple form of Markov chain Monte Carlo [Gilks 1995]. The HMM can also be viewed as a simple dynamic Bayesian network [Neapolitan 2003]. Thus, study of HMMs provides a gateway to powerful statistical and machine learning techniques that have arisen with the past two decades.

References

- Baum, L. E., T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains", *Annals of Mathematical Statistics*, Vol. 41, No. 1, 1970.
- Bhar, Ramaprasad, and Shigeyuki Hanori, *Hidden Markov Models: Applications to Financial Economics*, Kluwer, 2004.

- Dempster, A.P., N. M. Liard, and D. B. Rubin, "Maximum-Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society*, *Series B*, 39, 1977.
- Finch, S., N. Mendell, and H. Thode, "Probabilistic Measures of Adequacy of a Numerical Search for a Global Maximum." *Journal of the American Statistical Association*, 84, 1989.
- Gilks, W. R., S. Richardson, and D. Spiegelhalter (Eds.), *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*, Chapman & Hall/CRC Interdisciplinary Statistics Series, 1995.
- Karlis, D., "A Cautionary Note About the EM Algorithm for Finite Exponential Mixtures," *Technical Report No 150,* Department of Statisitics, Athens University of Economics and Business, 2001.
- Karlis, D., and E. Xekalaki, "Choosing Values for the EM Algorithm for Finite Mixtures," *Computational Statistics & Data Analysis*, 41, 2003.
- McLachlan, G. J., and D. Peel, *Finite Mixture Models*, Wiley-Interscience, 2000.
- McLachlan, G. J., and T. Krishnan, *The EM Algorithm and Extensions*, *2nd Ed.*, Wiley-Interscience, 2008.
- Neapolitan, R. E., *Learning Bayesian Networks*, Prentice-Hall, 2003.
- Rabiner, L. R., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceedings of the IEEE*, No. 2, February 1989.
- *Wikipedia, the free encyclopedia*, "Hidden Markov Model", retrieved from http://en.wikipedia.org/wiki/Hidden_Markov_model, 2009-12-22.
- Zucchini, W., and I. L. MacDonald, *Hidden Markov Models for Time Series: An Introduction Using R*, CRC Press, 2009.