An EM algorithm for maximum likelihood estimation given corrupted observations.

E. E. Holmes, National Marine Fisheries Service

Introduction

EM algorithms extend likelihood estimation to cases with hidden states, such as when observations are corrupted and the true population size is unobserved. The following EM algorithm is based on the summary by Ghahramani and Hinton (1996) of the algorithm originally by Shumway and Stoffer (1982) for estimating the parameters of linear dynamical systems from corrupted observations. The algorithm consists of an estimation step ("E step"), which estimates the true state using a Kalman-Rauch filter, combined with a measurement step ("M step"), which gives the maximum likelihood estimates of the parameters given the data and the estimate of the true state.

EM algorithms and the Kalman filter are well-known and heavily used in engineering and computer science applications. For some general background on EM algorithms the reader is referred to McLachlan (1996) and to Harvey (1991) for EM algorithms for time series data. There are a multitude of books on the Kalman filter. One of the more penetrable introductions is chapter 1 of Maybeck (1979).

This presentation of an EM algorithm closely follows Ghahramani and Hinton's notation and derivation but adapts and extends it to the stochastic population model case.

The state-space model

The diffusion approximation for a stochastic exponential growth model can be written as a linear state space model (written in the notation familiar in the engineering literature):

$$x_{t+1} = x_t + B + w_t, \quad \text{where} \quad w_t \sim \text{Normal}(0, Q) \qquad [1]$$

$$y_t = x_t + v_t, \quad \text{where} \quad v_t \sim \text{f}(0, R) \qquad [2]$$

where $x_t = \log N_t$ is the true log population size and $y_t = \log O_t$ is the log observations of the population size. B is μ, the mean population growth rate. Q is the σ^2, otherwise known as the process error or environmental variability. R is the variability associated with sampling error or other non-process error. Only y_t, is observed; the underlying parameters, B, Q, and R, and the underlying true population size, x_t, is hidden. If we make the assumption that v_t is normally distributed, then the model is a linear Gaussian state-space model, and we can use the algorithm by Shumway and Stoffer (1982) for estimating the parameters of the state-space model from y_t alone.

From Eqns. 1 and 2 and the assumption that f( ) is normal, we can write the following conditional probabilities:

$$P(y_t \mid x_t) = \exp\left\{ -\frac{(y_t - x_t)^2}{2R} \right\} (2\pi |R|)^{-1/2} \qquad [3]$$

$$P(x_t \mid x_{t-1}) = \exp\left\{-\frac{(x_t - (x_{t-1} + B))^2}{2Q}\right\}(2\pi|Q|)^{-1/2} \qquad [4]$$

Given Eqn. 1 and that we are essentially observing an ongoing stochastic process, which we happen to begin observing at $t = 1$, then x_1 is itself a random variable that is normally distributed with some mean π_1 and variance V_1, and thus

$$P(x_1) = \exp\left\{-\frac{(x_1 - \pi_1)^2}{2V_1}\right\}(2\pi|V_1|)^{-1/2} \qquad [5]$$

Using the Markov property implicit in the model, the joint probability of the observed time series, $\{y\}_1^T = \{y_1, y_2, \ldots, y_T\}$ and the true state, $\{x\}_1^T = \{x_1, x_2, \ldots, x_T\}$, is

$$P(\{x\}_1^T, \{y\}_1^T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1})\prod_{t=1}^{T} P(y_t \mid x_t)\,. \qquad [6]$$

The joint log likelihood of $\{y\}_1^T$, $\{x\}_1^T$ is:

$$\begin{aligned}\log P(\{x\}_1^T, \{y\}_1^T) = &-\sum_{t=1}^{T}\frac{(y_t - x_t)^2}{2R} - \frac{T}{2}\log|R| \\ &-\sum_{t=2}^{T}\frac{(x_t - (x_{t-1} + B))^2}{2Q} - \frac{T-1}{2}\log|Q| \\ &-\frac{(x_1 - \pi_1)^2}{2V_1} - \frac{1}{2}\log|V_1| - T\log 2\pi\end{aligned} \qquad [7]$$

The goal is to find the estimates of x_t, B, R, Q, π_1 and V_1 that maximize log P($\{x\}_1^T, \{y\}_1^T$). The following EM algorithm does this.

The EM algorithm

This EM algorithm, an extension of the Shumway and Stoffer (1982) algorithm, has four basic steps:

0) Compute some initial parameter estimates, $\hat{R}, \hat{Q}, \hat{B}, \hat{\pi}_1, \hat{V}_1$, from which to start the algorithm.

1) Generate an estimate of x_t by estimating $E(x_t \mid \{y\}_1^T)$ using the Kalman-Raush recursion which gives the maximum likelihood estimates of $E(x_t \mid \{y\}_1^T)$ given $\hat{R}, \hat{Q}, \hat{B}, \hat{\pi}_1, \hat{V}_1$, and y_t. The maximum likelihood estimate of $E(x_t \mid \{y\}_1^T)$ is denoted $\hat{x}_t$.

2) Update the $\hat{R}, \hat{Q}, \hat{B}, \hat{\pi}_1, \hat{V}_1$ given the new $\hat{x}_t$ by finding the estimates which maximize the updated expected log likelihood (using the updated $\hat{x}_t$): $\psi = E(\log P(\{x\}_1^T, \{y\}_1^T) \mid \{y\}_1^T]$.
3) Check to see if ψ has converged and no longer increases. If not converged, return to step 1.

Writing out and describing the algorithm is rather tedious and long, however the actual code is quite trivial and encompasses about a page of text, minus the comments. Matlab code is given at the end of this write-up.

Step 0. Compute initial parameter estimates

To get good final estimates one needs to start the algorithm with reasonable initial parameter estimates. I use the following initial estimates which are based on the parameter estimates presented in Holmes and Fagan (2002):

$$\tilde{y}_t = \sum_{i=t}^{t+3} y_i$$

$$\hat{B} = \frac{(\tilde{y}_{T-3} - \tilde{y}_1)}{T-4}$$

$$\hat{Q} = \frac{1}{3}\left(\text{Var}(\tilde{y}_{t+4} - \tilde{y}_t) - \text{Var}(\tilde{y}_{t+1} - \tilde{y}_t)\right)$$

$$\hat{R} = \frac{1}{2}\left(\text{Var}(y_{t+1} - y_t) - \hat{Q}\right)$$

The estimate of $\hat{R}$ is based on the estimate of the non-process error presented briefly in the appendix of Holmes and Fagan (2002). Initial estimates of π_1 and V_1 are also needed. I used $\hat{\pi}_1 = y_1$ and $\hat{V}_1 = 0.1$.

Step 1. The Kalman-Raush recursion

The first part uses the Kalman recursion to estimate $E(x_t \mid \{y\}_1^t)$. This is a forward recursion since we work forward to generate it. The second part uses the Raush recursion to work backwards and compute $E(x_t \mid \{y\}_1^T)$ from $E(x_t \mid \{y\}_1^t)$. First, some notation:

$$
\begin{aligned}
\{y\}_1^\tau &\equiv \{y_1, y_2, \ldots, y_\tau\} \\
\hat{x}_t &\equiv E[x_t \mid \{y\}_1^T] \\
x_t^\tau &\equiv E[x_t \mid \{y\}_1^\tau] \\
V_t^\tau &\equiv \mathrm{Var}[x_t \mid \{y\}_1^\tau] \\
V_{t,t-1}^\tau &\equiv \mathrm{Cov}[x_t, x_{t-1} \mid \{y\}_1^\tau] \\
P_t &\equiv E[x_t x_t \mid \{y\}_1^T] \equiv V_t^T + x_t^T x_t^T \\
P_{t,t-1} &\equiv E[x_t x_{t-1} \mid \{y\}_1^T] \equiv V_{t,t-1}^T + x_t^T x_{t-1}^T
\end{aligned}
$$

The ultimate goal of these recursions is to compute $\hat{x}_t$, P_t, and $P_{t,t\text{-}1}$, which will be needed for step 2 of the algorithm.

The Kalman recursion
To compute x_t^t and V_t^t, start at $t = 1$ and step forward to T. At each step, compute:

$$
\begin{aligned}
x_t^{t-1} &= \begin{cases} \hat{\pi}_1 & \text{for } t = 1 \\ x_{t-1}^{t-1} + \hat{B} & \text{for } t > 1 \end{cases} \\
V_t^{t-1} &= \begin{cases} \hat{V}_1 & \text{for } t = 1 \\ V_{t-1}^{t-1} + \hat{Q} & \text{for } t > 1 \end{cases} \\
K_t &= \frac{V_t^{t-1}}{\left(V_t^{t-1} + \hat{R}\right)} \\
x_t^t &= x_t^{t-1} + K_t\left(y_t - x_t^{t-1}\right) \\
V_t^t &= V_t^{t-1} - K_t V_t^{t-1}
\end{aligned}
$$

This is the well-known Kalman filter, but it looks a little different than what you'll see in engineering texts. First generally it is assumed that y_t is a series of measurements from multiple instruments, thus the Kalman filter is always written in matrix form. Here since y_t is one measurement, it can be written in scalar form. Second, the Kalman filter is usually presented for the model $x_{t+1} = Ax_t + Bu_t + w_t$, $y_t = Cx_t + v_t$. In this application, A=1, C=1 and u_t =1, so the filter is simplified quite a bit.

The Rauch recursion
Next we work backwards from $t = T$ back to $t =$ 2, to compute x_t^T and V_t^T. This recursion requires the x_t^t, V_t^t, and V_t^{t-1} that were generated during the Kalman recursion.

$$
\begin{aligned}
J_{t-1} &= \frac{V_{t-1}^{t-1}}{V_t^{t-1}} \\
x_{t-1}^T &= x_{t-1}^{t-1} + J_{t-1}\left(x_t^T - \left(x_{t-1}^{t-1} + B\right)\right) \\
V_{t-1}^T &= V_{t-1}^{t-1} + J_{t-1}\left(V_t^T - V_t^{t-1}\right)J_{t-1}
\end{aligned}
$$

One more recursion
Using J_t from the Rauch recursion with K_t and V_t^t from the Kalman recursion, we do another backwards recursion to compute, $V_{t,t-1}^T$. Starting from $t = T$ work backwards to $t = 2$, and at each step compute

$$
V_{t,t-1}^T = \begin{cases} (1 - K_T)V_{T-1}^{T-1} & \text{for } t = T \\ V_{t-1}^{t-1}J_{t-2} + J_{t-1}\left(V_{t,t-1}^T \text{ - } V_{t-1}^{t-1}\right)J_{t-2} & \text{for } t < T \end{cases}
$$

Putting it all together
Using the three recursions, we can then compute the following, which are needed for step 2 of the algorithm.

$$
\begin{aligned}
\hat{x}_t &= x_t^T \\
P_t &= V_t^T + x_t^T x_t^T \\
P_{t,t-1} &= V_{t,t-1}^T + x_t^T x_{t-1}^T
\end{aligned}
$$

Step 2 Generate new parameter estimates
The new expected log likelihood function is given by Eqn. 7 with the new x_t estimate, $\hat{x}_t$:

$$
\psi = E(\log P(\{x\}_1^T, \{y\}_1^T) \mid \{y\}_1^T] \text{ using new } \hat{x}_t.
$$

To compute the new parameter estimates, we find the new $\hat{R}, \hat{Q}, \hat{B}, \hat{\pi}_1, \hat{V}_1$ that maximize the new ψ. To do this, we take the partial derivative of ψ with respect to each parameter, set the derivative to zero and solve for the maximizing parameter:

$$
\frac{\partial \psi}{\partial R^{-1}} = -\sum_{t=1}^{T} R^{-1} y_t \hat{x}_t + \sum_{t=1}^{T} R^{-1} P_t = 0
$$

$$
\hat{C}_{new} = \frac{\sum_{t=1}^{T} y_t \hat{x}_t}{\sum_{t=1}^{T} P_t}
$$

$$\frac{\partial\psi}{\partial B} = -\sum_{t=2}^{T} Q^{-1}\hat{x}_t + \sum_{t=2}^{T} Q^{-1}\hat{x}_{t-1} + Q^{-1}B = 0$$

$$\hat{B}_{new} = \sum_{t=2}^{T}\left(\hat{x}_t - \hat{x}_{t-1}\right) = \frac{\hat{x}_T - \hat{x}_1}{T-1}$$

$$\frac{\partial\psi}{\partial Q^{-1}} = \frac{T-1}{2}Q - \frac{1}{2}\sum_{t=2}^{T}\left(\hat{x}_t^2 - 2\hat{x}_t(\hat{x}_{t-1}+B) + (\hat{x}_{t-1}+B)^2\right)$$

$$= \frac{T-1}{2}Q - \frac{1}{2}\sum_{t=2}^{T}\left(\hat{x}_t^2 - 2\hat{x}_t\hat{x}_{t-1} - 2\hat{x}_t B) = +\hat{x}_{t-1}^2 + 2B\hat{x}_{t-1} + B^2\right)$$

$$= \frac{T-1}{2}Q - \frac{1}{2}\sum_{t=2}^{T}\left(\hat{x}_t^2 - 2\hat{x}_t\hat{x}_{t-1} + \hat{x}_{t-1}^2 - 2B(\hat{x}_t - \hat{x}_{t-1}) + B^2\right)$$

$$\text{given that } \hat{B} = \sum_{t=2}^{T}(\hat{x}_t - \hat{x}_{t-1})$$

$$\hat{Q}_{new} = \frac{1}{T-1}\sum_{t=2}^{T}\left(P_t - 2P_{t,t-1} + P_{t-1} - \hat{B}_{new}^2\right)$$

$$\frac{\partial\psi}{\partial\pi_1} = \frac{(\hat{x}_1 - \pi_1)}{V_1} = 0$$

$$\hat{\pi}_{1,new} = \hat{x}_1$$

$$\frac{\partial\psi}{\partial V_1^{-1}} = \frac{1}{2}V_1 - \frac{1}{2}\left(\hat{x}_1^2 - 2\hat{x}_1\pi_1 + \pi_{1}^2\right) = 0$$

$$\hat{V}_{1,new} = P_1 - \pi_{1,new}^2$$

Step 3 Check for convergence
A simple way to do this is to compare the new ψ to the previously estimated ψ and check if the difference is less than some threshold. Once the log likelihood converges, you're done.

References

Ghahramani, Z. and G. E. Hinton. 1996. Parameter estimation for linear dynamical systems. Technical report CRG-TR-96-2. University of Toronto, Department of Computer Science, Toronto, Canada.

Harvey, A. C. 1991. Forecasting, structural time series models and the Kalman filter. Cambridge University Press, Cambridge, UK.

Holmes, E. E. and W. F. Fagan. 2002. Validating population viability analysis for corrupted data sets. Ecology 83: 2379-2386.

McLachlan, G. M. 1996. The EM algorithm and extensions. Wiley, USA.

Maybeck, P. S. 1979. Stochastic models, estimation and control. Volume 1. Academic Press, New York, USA.

Shumway, R. H. and Stoffer, D. S. 1982. An approach to time series smoothing and forecasting using the EM algorithm. Journal of Time Series Analysis 3(4):253-264.

Matlab code

```
function [B, Q, R, initx, V1, LL] = PVAKalman(logdata)
%PVAKalman Find the ML parameters of a stochastic corrupted exponential growth
%time series using EM
%
% [B, Q, R, initx, initV, LL] = PVAKalman(logdata)
% fits the parameters which are defined as follows
%   x(t+1) = x(t) + B + w(t), w ~ N(0, Q), x(0) ~ N(initx, initV)
%   y(t)   = x(t) + v(t), v ~ N(0,R)
% logdata(t) is a 1 x (1:T) vector of the logged observations; no missing years
% LL is the vector of the log likelihood values at each iteration; the idea is
% to maximize this.
%
% The algorithm used here is an extension of the method described in
% Shumway, R. H. and Stoffer, D.S. 1982. An approach to time series smoothing and
%forecasting using the EM algorithm.  Journal of Time Series Analysis 3(4): 253-264.
% which is described in Ghahramani, Z. and Hinton, G.E. 1996. Parameter Estimation
%for LDS.  U. Toronto technical report CRG-TR-96-2.
% The notation used here follows Ghahramani and Hinton except that I drop the
%"new" subscripts.

%STEP 0 set the initial estimates
%Initialize by getting D-H estimates
T=length(logdata);
data=exp(logdata);
runsum = data(1:(T-3))+data(2:(T-2))+data(3:(T-1))+data(4:T);
B = mean(log(runsum(2:end)./runsum(1:(end-1))));
totvar = var(logdata(2:end)-logdata(1:(end-1)));
Q = (1/3)*(var(log(runsum(4:end)./runsum(1:(end-3))))-...
   var(log(runsum(2:end)./runsum(1:(end-1)))));
if(Q < 0) Q = 0.0001; end
R = (totvar - Q)/2;
if(R < 0) R = 0.0001; end
initx = logdata(1); %pi is Ghahramani and Hinton's notation, but pi reserved
V1 = 0.1; %variance of initx
y = logdata;

LL=[];
converged = 0;
previous_loglik = -Inf;
max_iter = 500;
num_iter = 0;

%run until the max log likelihood is found

while ~converged & (num_iter <= max_iter)

%STEP 1 using these initial estimates generate an estimate of x(t)
%using a forward and backward pass of the Kalman filter.  This gives
%you the ML estimate of x(t) given y(1:T) and the parameter estimates.

%initialize
xtt=zeros(1,T);  Vtt=zeros(1,T); xtt1=zeros(1,T); Vtt1=zeros(1,T);
xtT=zeros(1,T);  VtT=zeros(1,T); J=zeros(1,T); Vtt1T=zeros(1,T);

%forward pass gets you E[x(t) given y(1:t)]
x10=initx;
```

```
V10=V1;
for(t=1:T)
   if(t==1)
    xtt1(1) = initx; %denotes x_1^0
      Vtt1(1) = V1; %denotes V_1^0
   else
      xtt1(t) = xtt(t-1) + B; %xtt1 denotes x_t^(t-1)
      Vtt1(t) = Vtt(t-1) + Q;
   end
   Kt = Vtt1(t)/(Vtt1(t)+R);
   xtt(t) = xtt1(t) + Kt*(y(t) - xtt1(t));
   Vtt(t) = Vtt1(t)-Kt*Vtt1(t);
end
KT = Kt;

%backward pass gets you E[x(t)|y(1:T)] from E[x(t)|y(1:t)]
xtT(T) = xtt(T);
VtT(T) = Vtt(T);
for(t=T:-1:2)
   J(t-1) = Vtt(t-1)/Vtt1(t);
   xtT(t-1) = xtt(t-1) + J(t-1)*(xtT(t)-(xtt(t-1)+B));
   VtT(t-1) = Vtt(t-1) + J(t-1)*(VtT(t)-Vtt1(t))*J(t-1);
end
xhat = xtT; %estimate of x(t)
Pt = VtT + xtT.*xtT; %E(x^2 | y]

%run another backward recursion to get E[x(t)x(t-1)|y(T)]
Vtt1T(T) = (1 - KT)*Vtt(T-1); %this is Var(x(T)x(T-1)|y(T))
for(t=T:-1:3)
   Vtt1T(t-1) = Vtt(t-1)*J(t-2) + J(t-1)*(Vtt1T(t)-Vtt(t-1))*J(t-2);
end
Ptt1=[NaN Vtt1T(2:T)+xtT(2:T).*xtT(1:(T-1))]; %Ptt1(1) = NA since 1-1 = 0

%Calculate negative log likelihood for this xhat + B,Q,R,initx,initV1 combo
loglik = - sum((y-xhat).^2)/(2*R) - T*log(abs(R))/2 ...
   - sum((xhat(2:T)-(xhat(1:(T-1))+B)).^2)/(2*Q) - (T-1)*log(abs(Q))/2 ...
   - (xhat(1)-initx)^2/(2*V1) - log(abs(V1))/2 - T*log(2*pi);
LL=[LL loglik];

%STEP 2 Re-estimate B,Q,R,initx,initV1 via ML given x(t) estimate
R = (1/T)*sum(y.*y - xhat.*y);
B = (xhat(T)-xhat(1))/(T-1);
Q = sum(Pt(2:T) - 2*Ptt1(2:T) + Pt(1:(T-1)) - B^2)/(T-1);
initx = xhat(1);
V1 = Pt(1)-xhat(1)*xhat(1);

%STEP 3 check for convergence
num_iter = num_iter+1;
converged = em_converged(loglik, previous_loglik); %subfunction below
previous_loglik = loglik;

end %while not converged

function converged = em_converged(loglik, previous_loglik, threshold)
% EM_CONVERGED Has EM converged?
% [converged, decrease] = em_converged(loglik, previous_loglik, threshold)
%
% We have converged if
%   |f(t) - f(t-1)| / avg < threshold,
% where avg = (|f(t)| + |f(t-1)|)/2 and f is log lik.
% threshold defaults to 1e-4.
% This stopping criterion is from Numerical Recipes in C p423
```

```
if nargin < 3
  threshold = 1e-4;
end

%log likelihood should increase
if loglik - previous_loglik < -1e-3 % allow for a little imprecision
  fprintf(1, '******likelihood decreased from %6.4f to %6.4f!\n', previous_loglik,
loglik);
end

delta_loglik = abs(loglik - previous_loglik);
avg_loglik = (abs(loglik) + abs(previous_loglik) + eps)/2;
if (delta_loglik / avg_loglik) < threshold
   converged = 1;
else converged = 0; end
```