A New Algorithm for Rapid Parameter Learning in Linear Dynamical Systems

by

James Martens

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Copyright © 2009 by James Martens

Abstract

A New Algorithm for Rapid Parameter Learning in Linear Dynamical Systems

James Martens

Master of Science

Graduate Department of Computer Science

University of Toronto

2009

The linear dynamical system (LDS) is an important model of time-series data with many practical applications in fields such as engineering and mathematical finance. One commonly used iterative algorithm for learning the parameters of the LDS is known as expectation maximization (EM). EM has many desirably properties including a statistically formulated objective function (the log-likelihood), the ability to find local optima, and a graceful scaling of performance to high-dimensional time-series. Unfortunately it tends to be slow for long time-series because each iteration involves inference over all of T hidden states, where T is the length of the time-series.

This report develops two new algorithms, based on EM, for learning the parameters of an LDS. Their development is motivated first by the existence of the so-called "steady-state" approximation and second by the observation that the "M-step" of EM only requires certain second-order statistics over the hidden states and not individual estimates for each of them. The first algorithm, which we call Steady-State EM (SSEM), uses the steady-state approximation to simplify inference of the hidden states, thereby achieving greatly superior performance (asymptotically and in practice) over standard EM. The second algorithm, which we call Approximate EM (AEM), introduces several additional approximations in order to approximate the second-order statistics needed for the M-step without performing inference for each hidden state and in time *independent of T*. We show experimentally that SSEM and AEM achieve nearly identical solutions to those produced by standard EM, but with iterations that can be done much faster or in constant time, respectively. To our knowledge AEM is the only algorithm capable of finding closely approximated local optima of the log-likelihood function using constant-time iterations.

Contents

Notation 1

1 Introduction **2**

1.1 The Linear Dynamical System Model 2

1.2 Learning Algorithms for the LDS 3

1.3 Contribution 6

1.4 Sufficient Statistics for the M-step 7

1.5 The E-Step via Kalman Filtering/Smoothing 8

1.6 Steady State 9

1.7 The Steady State Approximation of EM 11

1.8 A Faster Approximation of the E-step 12

2 Details of the New Algorithm **16**

2.1 Three Simple Identities 16

2.2 Approximating the First-order Statistics 17

2.3 Approximating the Second-order Statistics 18

2.4 Computing The Log-Likelihood 25

2.5 Summary of Algorithm 26

3 Experimental Evaluation **32**

3.1 Experimental Setup 32

3.2 Experimental Results 33

3.3 Discussion of Results 40

4 Conclusions and Future Work 42

Bibliography 45

A Details of the Derivation of EM 48

A.1 Computing the EM Objective Function 48

A.2 Derivation of the M-step 49

B Algorithms for Solving Various Matrix Equation 53

B.1 Solving the DARE 53

B.2 Solving Lyapunov and Sylvester Equations 54

B.3 Solving Equation (2.9) 55

Notation

In this report we will use the following non-standard/less-commonly-seen notation:

- $a_{\leq t}$ will be short-hand for $a_1, a_2, ..., a_t$

- The transpose of a matrix A will be denoted by A'.

- The spectral radius of a continuous linear operator (or matrix) f will denoted by $\rho(f)$ where spectral radius is defined as:

$$\rho(f) = \sup\{|\lambda| : \lambda \text{ is in the spectrum of } f\ \}$$

- For lists of vectors $\{a_1, a_2, ..., a_T\} \subset \Re^n$ and $\{b_1, b_2, ..., b_T\} \subset \Re^m$ and $k \geq 0$ we define:

$$(a, b)_k \equiv \sum_{t=1}^{T-k} a_{t+k} b'_t$$

- For a function f, f^k will denote the function composed with itself k times

- We will use the shorthand $(\sum_{t=n}^{m} + \sum_{t=s}^{q}) \, a_t$ to mean $\sum_{t=n}^{m} a_t + \sum_{t=s}^{q} a_t$

- $A \otimes B$ will be the Kronecker product of the matrices A and B and $\text{vec}(A)$ the vectorization of A

Chapter 1

Introduction

1.1 The Linear Dynamical System Model

Consider a sequence of real-valued multivariate data $\{y_t \in \mathbb{R}^{N_y}\}_{t=1}^{T}$ exhibiting temporal relationships (a "time-series"). The discrete-time Linear Dynamical System (LDS) is a model of such time-series data that proposes a corresponding sequence of hidden states $\{x_t \in \mathbb{R}^{N_x}\}_{t=1}^{T}$ that encode the underlying phenomenon presumed to have generated the data (also called the 'output'). The model is conditioned on an observed input sequence or "control signal" $\{u_t \in \mathbb{R}^{N_u}\}_{t=1}^{T}$. In the time-invariant version, which is the only one considered in this report, the hidden states evolve according to the following linear dynamics:

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t \tag{1.1}$$

where $A \in \mathbb{R}^{N_x \times N_x}$, $B \in \mathbb{R}^{N_x \times N_u}$ and the error/noise terms $\{\epsilon_t\}_{t=1}^{T}$ are i.i.d. multivariate normal with mean 0 and covariance matrix Q. This is called time invariant because the parameter matrices A, B and Q do not depend on t.

The output y is generated linearly from the hidden states according to:

$$y_t = Cx_t + Du_t + \delta_t \tag{1.2}$$

where $C \in \mathbb{R}^{N_y \times N_x}$, $D \in \mathbb{R}^{N_y \times N_u}$ and the error/noise terms $\{\delta_t\}_{t=1}^{T}$ are also i.i.d. multivariate normal (and independent of $\{\epsilon_t\}_{t=1}^T$) with mean 0 and covariance matrix R. To complete the model we also specify the distribution on x_1, the initial state, as multivariate normal with mean π_1 and covariance matrix Π_1.

The joint probability of the outputs and hidden states conditioned on the model parameters $\theta \equiv \{A, B, C, D, Q, R, \pi_1, \Pi_1\}$ and inputs u is:

$$p(x, y|\theta, u) = p(x_1|\theta) \left(\prod_{t=1}^{T} p(y_t|x_t, \theta, u) \right) \left(\prod_{t=1}^{T-1} p(x_{t+1}|x_t, \theta, u) \right) \tag{1.3}$$

where,

$$\begin{aligned} p(y_t|x_t, \theta, u) &= n(y_t; Cx_t + Du_t, R), \\ p(x_{t+1}|x_t, \theta) &= n(x_{t+1}; Ax_t + Bu_t, Q), \\ p(x_1, \theta) &= n(x_1; \pi_1, \Pi_1) \end{aligned}$$

and $n(z; \hat{z}, \Sigma)$ denotes the multivariate normal probability density function for z with mean $\hat{z}$ and covariance matrix Σ.

Sometimes referred to as the "Kalman filter" or linear state space model, the LDS is arguably the most commonly used time-series model for real-world engineering and financial applications. This is due to its relative simplicity, it's mathematically predictable behavior, the existence of many physical systems that are known to be accurately modeled by it, and the fact that exact inference and prediction can done efficiently.

1.2 Learning Algorithms for the LDS

The objective function maximized during maximum likelihood learning is the log probability of the observation sequence y conditioned on the input u and the model parameters (also known as the log likelihood function). This can be obtained from the previous joint probability by integrating out the hidden states: $\log p(y|\theta, u) = \log \int_x p(x, y|\theta, u)$. While the gradient and Hessian of this function are very difficult to compute, it is relatively simple to compute the log joint probability and it's expectation given y for a particular setting of the parameters. The Expectation-Maximization algorithm (EM), which was first applied to LDS parameter learning by Shumway and Stoffer (1982), can indirectly optimize the log-likelihood by iteratively maximizing this later quantity, which we denote:

$$\mathcal{Q}_n(\theta) \equiv E_{\theta_n}[\log p(x, y|\theta, u)|y] = \int \log p(x, y|\theta, u) p(x|y, \theta_n, u)\, dx \tag{1.4}$$

The EM algorithm iteratively alternates between two phases called the "E-step" and the "M-step". For a given estimate of the parameters θ_n, the E-step computes the statistics over y and u which appear in the expression for $\mathcal{Q}_n(\theta)$, allowing it to be easily evaluated and optimized with respect to θ in the M-step. The M-step then computes the new parameter estimate θ_{n+1} as:

$$\theta_{n+1} = \arg\max_{\theta} \mathcal{Q}_n(\theta) \tag{1.5}$$

From (1.3), the joint probability $p(x, y|\theta, u)$ of observations and hidden states is multivariate normal and thus by standard results, the conditional distribution $p(x|\theta_n, y, u)$ is also multivariate normal. Moreover there is a reasonably efficient algorithm known as Kalman filtering and smoothing for inferring this distribution for a given setting of the parameters in time $O(TN_x^3)$ (where we assume for simplicity that $N_x \geq N_y, N_u$).

The EM algorithm is guaranteed to converge to a local maximum of the log-likelihood function (Dempster et al., 1977). However, the convergence rate is typically linear and so very many iterations can be required. This can be a problem for long time series because each iteration of EM involves recursive inference computations for all T hidden states.

One alternative learning algorithm to EM, known as subspace-identification, (a.k.a. 4SID, see Overschee and Moor, 1991) works by reducing the LDS learning problem to that of solving a large SVD, subject to some approximations. Because it is non-iterative, 4SID tends to be much more computationally efficient than EM. However, because it implicitly optimizes a sum of squared prediction-error objective function instead of the log likelihood (which are similar but not strictly the same) and implicitly uses a primitive inference procedure for the hidden states, 4SID is not statistically optimal. In particular, hidden state point estimates lacking covariance information are found by conditioning on only the past i data points, where i is a parameter known as the “prediction horizon”. In contrast, the EM algorithm estimates hidden states with mean and covariance information, conditioned on the entire observation sequence and the current estimate of θ. 4SID also tends to scale poorly to very long and high-dimensional time-series.

The general finding is that EM, when initialized properly, will find parameter settings with better log likelihoods and prediction errors than those found by 4SID. However, because of its speed, ability to estimate the dimension of the hidden state space (i.e. N_x), and the fact that it optimizes an objective which is similar to EM’s, 4SID is often considered an ideal method for initializing iterative learning algorithms such as EM. See Smith and Robinson (2000) and Smith et al. (1999) for theoretical and empirical comparative analyses of 4SID and EM.

Another classical algorithm for LDS learning is known as Prediction Error Minimization (PEM). PEM optimizes a sum of squared prediction error objective function by direct application of iterative non-linear optimization methods. Computing the gradient of the PEM objective function is very difficult and practical implementations use techniques such as finite-differences to achieve reasonable performance. When implemented with Newton or quasi-Newton methods, PEM requires significantly less iterations than EM to converge since EM’s convergence is linear while Newton and quasi-Newton methods give quadratic and superlinear convergence respectively. However, the performance of such implementations of PEM tend to scale very poorly with the dimension of y, and for very long time-series the per-iteration cost can become prohibitive. For example, the implementation of PEM included with MATLAB’s System Identification Toolbox breaks down for even modestly high-dimensioned data (e.g. $N_y = 10$).

1.3 Contribution

The main contribution of this report is the development of a new algorithm, based on EM, for learning the parameters of an LDS. It is motivated both by the existence of the well-known "steady-state" approximation, which simplifies inference of x, and by the observation that the M-step requires only a small set of expected second order statistics (sums over t of products of x, y and u) that could possibly be approximated without doing inference for each x_t, as is traditionally done.

Under the steady-state approximation, the recursive equations (known as the Kalman recursions) normally used to perform the inference of x may instead be used to derive equations that express the required second order statistics in terms of similar statistics that differ only in that they have larger "time-lags" (which we define as the difference in time index between the two terms of the products being summed). Such statistics can be expressed in terms of others with even larger time-lags, and so on. Eventually, statistics with very large time-lags may be approximated by their expectations under the (currently learned) model. Our experiments will show that the resulting approximate EM algorithm (which we will call AEM) performs nearly as well as EM given the same number of iterations, despite its use of approximations. But since the work required per iteration of AEM does not depend on T, AEM can be much more computationally efficient than EM when T is large. Thus AEM is a fast alternative to EM, which unlike 4SID, is able to find locally optimal estimates of the parameters (subject to a mild and adjustable approximation). It is particularly useful for learning from very long timeseries, where the performance gain can mean the difference between minutes and hours of training time.

1.4 Sufficient Statistics for the M-step

As a consequence of the linear and Gaussian nature of the LDS model, the function $\mathcal{Q}_n(\theta)$ can be written in terms of θ and sufficient statistics that are first and second order in x, y and u (and are highly non-linear in θ_n). With these statistics computed, optimizing $\mathcal{Q}_n(\theta)$ with respect to θ reduces to a straightforward application of matrix calculus and is similar to linear regression with an unknown covariance (see the appendix for details).

The following statistics form a complete list of those needed to compute and optimize $\mathcal{Q}_n(\theta)$ (where $(a, b)_k \equiv \sum_{t=1}^{T-k} a_{t+k} b'_t$):

$$(y, y)_0, \quad (y, u)_0, \quad (u, u)_0, \quad u_T u'_T, \quad E_{\theta_n}[(y, x)_0 \mid y, u], \quad E_{\theta_n}[(u, x)_0 \mid y, u]$$
$$E_{\theta_n}[(u, x)_1 \mid y, u], \quad E_{\theta_n}[(x, x)_0 \mid y, u], \quad E_{\theta_n}[(x, x)_1 \mid y, u], \quad E_{\theta_n}[x_1 \mid y, u],$$
$$E_{\theta_n}[x_1 {x_1}' \mid y, u], \quad E_{\theta_n}[x_T {x_T}' \mid y, u], \quad u_T E_{\theta_n}[x'_T \mid y, u]$$

where we have used the obvious identity $(a, b)_0 = (b, a)'_0$ to eliminate statistics that are the transpose of ones already in the list.

For the sake of brevity we will use the standard notation for the remainder of this report:

$$\begin{aligned} x_t^k &\equiv E_{\theta_n}[\, x_t \mid y_{\leq k}, u_{\leq k} \,] \\ V_{t,s}^k &\equiv \mathrm{Cov}_{\theta_n}[\, x_t, x_s \mid y_{\leq k}, u_{\leq k} \,] \\ \tilde{y}_t &\equiv y_t - E_{\theta_n}[\, y_t \mid y_{\leq t-1}, u_{\leq t} \,] \\ S_t &\equiv \mathrm{Cov}_{\theta_n}[\, \tilde{y}_t \mid y_{\leq t}, u_{\leq t} \,] \end{aligned}$$

By breaking down the second order statistics as the sum of the covariance and the product of the means, according to:

$$E_{\theta_n}[x_t x'_s \mid y_{\leq k}, u_{\leq k} \,] = x_t^T x_s^T + V_{t,s}^T$$

the statistics that depend on θ_n may be rewritten as:

$$(y, x^T)_0, \quad (u, x^T)_0, \quad (u, x^T)_1, \quad (x^T, x^T)_0 + \sum_{t=1}^{T} V_{t,t}^T, \quad (x^T, x^T)_1 + \sum_{t=1}^{T-1} V_{t+1,t}^T,$$

$$x_1^T, \quad x_1^T x_1^{T\prime} + V_{1,1}^T, \quad x_T^T x_T^{T\prime} + V_{T,T}^T, \quad u_1 x_1^T$$

1.5 The E-Step via Kalman Filtering/Smoothing

The Kalman recursions are a set of recursive relations that define a computational procedure for exact inference of the distribution over the hidden states x in the LDS model. The standard approach to computing the M-statistics is to apply this procedure to find x_t^T, $V_{t,t}^T$, $V_{t+1,t}^T$ for each value of t in succession and then compute the required sums. This has time complexity $O(TN_x^3)$. Using the steady-state approximation, which we discuss in the next section, we can reduce this to $O(TN_x^2)$ and in the next chapter we will show how with the addition of some more approximations we can reduce this even further to $O(k_{lim}N_x^3)$ where k_{lim} is a constant such that $k_{lim} << T$.

The complete set of recursions (Ghahramani and Hinton, 1996) consists of a forward pass (commonly referred to as filtering) and a backwards pass (commonly referred to as smoothing). The forward filtering pass infers the distribution for each hidden state x_t conditioned on all past data $y_{\leq t}$ (and $u_{\leq t}$). This can be described as a two stage process. In the first stage, the mean and covariance of predictions from strictly older data ($y_{\leq t-1}$) of x_t and y_t are computed as:

$$V_{t,t}^{t-1} = AV_{t-1,t-1}^{t-1}A' + Q \tag{1.6}$$

$$x_t^{t-1} = Ax_{t-1}^{t-1} + Bu_t \tag{1.7}$$

$$S_t = CV_{t,t}^{t-1}C' + R \tag{1.8}$$

$$\tilde{y}_t = y_t - Cx_t^{t-1} - Du_t \tag{1.9}$$

where $x_1^0 = \pi_1$ and $V_{1,1}^0 = V_1$.

In the second stage, the predictions for x_t are "corrected" based on the values of y_t and u_t according to:

$$\begin{aligned} K_t &= V_{t,t}^{t-1} C' S_t^{-1} && (1.10) \\ V_{t,t}^{t} &= V_{t,t}^{t-1} - K_t C V_{t,t}^{t-1} && (1.11) \\ x_t^t &= x_t^{t-1} + K_t \tilde{y}_t && (1.12) \end{aligned}$$

where K_t is known as the Kalman gain matrix.

In a similar fashion to the second stage of Kalman filtering, the backwards smoothing pass integrates information from future observations into the mean and covariances estimates of x_t, resulting in estimates that are conditioned on the complete time-series ($y_{\leq T}$). The backwards recursions are:

$$\begin{aligned} J_{t-1} &= V_{t-1,t-1}^{t-1} A' (V_{t,t}^{t-1})^{-1} && (1.13) \\ V_{t-1,t-1}^{T} &= V_{t-1,t-1}^{t-1} + J_{t-1}(V_{t,t}^{T} - V_{t,t}^{t-1}) J_{t-1}' && (1.14) \\ V_{t,t-1}^{T} &= V_{t,t}^{T} J_{t-1}' && (1.15) \\ x_{t-1}^{T} &= x_{t-1}^{t-1} + J_{t-1}(x_t^T - x_t^{t-1}) && (1.16) \end{aligned}$$

where the smoothing matrix J_t plays a similar role to that of K_t.

1.6 Steady State

The Kalman recursions compute the covariance matrices between hidden state vectors according to recursive formulas. Notably, these recursions depend only on the model parameters and are independent of the particular values of y and u.

From (1.6), (1.8), (1.10) and (1.11) we can express $V_{t+1,t+1}^{t}$ directly in terms of $V_{t,t}^{t-1}$ as:

$$V_{t+1,t+1}^{t} = f(V_{t,t}^{t-1}) \tag{1.17}$$

where,

$$f(X) = A\left(X - XC'(CXC' + R)^{-1} CX\right) A' + Q$$

Similarly, from (1.14) we can express $V^T_{t-1,t-1}$ directly in terms of $V^T_{t,t}$ and quantities computed by the forward recursions:

$$V^T_{t-1,t-1} = g_t(V^T_{t,t}) \tag{1.18}$$

where,

$$g_t(X) = V^{t-1}_{t-1,t-1} + J_{t-1}(X - V^{t-1}_{t,t})J'_{t-1}$$

and J_{t-1} is expressible in terms of quantities from the forward recursions according to (1.13)

For well-behaved values of the model parameters, $f^i(X)$ and $g^i_t(X)$ will converge as $i \to \infty$. When this happens (to within a reasonable precision) the system is said to have reached a steady state. In particular, $V^t_{t+1,t+1}$ converges to a constant matrix Λ^1_0 for large enough t. Similarly, $V^T_{t,t}$ converges to Λ_0 for values of t reasonably far away from 1 and T. This convergence is exponentially fast (but with a potentially small base) and has been studied extensive in the filtering and control theory literature (e.g. Goodwin and Sin, 1984). Note that the superscripts and subscripts for the steady-state matrices have significance as part of a more general notation: Λ^j_i is the steady state of $V^{t-j}_{t,t-i}$ and Λ_i is the steady state of $V^T_{t,t-i}$.

To find the steady-state matrix Λ^1_0 we can compute $f^i(0)$ by iteratively applying f until convergence is obtained, or we can attempt to solve the equation $X = f(X)$. This type of equation is known as a discrete algebraic Riccati equation (DARE) and there are several efficient algorithms for solving it. One of these algorithms, known as the "doubling algorithm", is discussed in the appendix.

The steady-state values of S_t, K_t, J_t and $V^t_{t,t}$, which we denote S, K, J and Λ^0_0 respectively, may be computed from Λ^1_0 via (1.8), (1.10), (1.13) and (1.11):

$$S = C\Lambda^1_0 C' + R \qquad K = \Lambda^1_0 C' S^{-1}$$

$$\Lambda^0_0 = \Lambda^1_0 - KC\Lambda^1_0 \qquad J = \Lambda^0_0 A'(\Lambda^1_0)^{-1}$$

Note that g_t depends on t only because it contains various quantities computed by the forward recursions. However, once the steady state of f is reached we may replace these with their steady-state counterparts so that the dependence on t vanishes. The resulting function, which we denote by g (without a subscript) is:

$$g(X) = \Lambda_0^0 + J(X - \Lambda_0^1)J' \tag{1.19}$$

As before we can compute the steady state Λ_0 of $V_{t,t}^T$ by solving the equation $X = g(X)$. This equation is known as a Lyapunov equation and may be viewed as a special case of the DARE, although there are simpler algorithms for solving it. See the appendix for further details.

Finally we may compute the steady-state value of $V_{t,t-1}^T$, which we denote by Λ_1, from Λ_0 and (1.15) as $\Lambda_1 = \Lambda_0 J'$.

1.7 The Steady State Approximation of EM

One way to dramatically simplify the EM algorithm is to replace the t-dependent matrices in both the E and M steps with their steady-state values. As we saw in the previous section, there are computationally efficient methods for finding these that are independent of T.

In other words we approximate:

$$K_t \approx K, \quad J_t \approx J, \quad V_{t,t}^T \approx \Lambda_0, \quad V_{t+1,t}^T \approx \Lambda_1$$

so that the Kalman recursions reduce to:

$$\begin{aligned} x_t^{t-1} &= Ax_{t-1}^{t-1} + Bu_t && (1.20)\\ x_t^t &= x_t^{t-1} + K(y_t - Cx_t^{t-1} - Du_t) && (1.21)\\ x_{t-1}^T &= x_{t-1}^{t-1} + J(x_t^T - x_t^{t-1}) && (1.22) \end{aligned}$$

and the covariance matrices are replaced with their steady-state values in the computation of the second-order statistics required by the M-step.

Because we only have to compute matrix-vector multiplications for each value of T, the time complexity of a single iteration under this scheme is just $O(TN_x^2)$. And fortunately, the approximations being made are relatively mild since the t-dependent matrices only deviate significantly from their steady-state values when t is close to either 1 or T. For large values of T this will represent an insignificant portion of the time-series and hence only introduce a minimal distortion into the objective function. Moreover, it is common-place in engineering applications to assume that an LDS has already reached steady state in order to simplify the computations needed for inference/prediction.

Other researchers (e.g. Fukumori et al., 1993) have used a steady-state approximations of the t-dependent matrices in Kalman filter/smoother to improve the efficiency of forecasting and prediction algorithms (where the model parameters are known). But the idea of using them to efficiently approximate the E-step in the EM algorithm appears to be novel.

1.8 A Faster Approximation of the E-step

The main contribution of this report is an algorithm for computing, in time *independent of* T, a robust approximation to the statistics required by the M-step. It exploits the fact that, under the steady-state approximation, the Kalman recursions for computing the means are identical for each t (i.e. the filtering and smoothing matrices are constant), allowing them to be approximately computed in parallel.

As a gentle introduction to our approach we will first show how the ideas we have outlined result in a much simpler computational scheme for a toy version of the problem we are trying to solve. In particular, suppose we wished to compute $\sum_{t=1}^{T} x_t^t$, which is a simple first-order statistic over x. We could run the steady-state (or conventional) Kalman filter to compute x_t^t for each t and then take the sum (this is analogous to what is traditionally done in the E-step). However, if we already have access to $\sum_{t=1}^{T} y_t$, $\sum_{t=1}^{T} u_t$,

x_1^1 and x_T^T there is a more efficient way to compute this statistic.

In the interest of simplicity we will rewrite the Kalman filtering equations (1.21), which hold for $2 \leq t \leq T$, as:

$$\begin{aligned} x_t^t &= Ax_{t-1}^{t-1} + Bu_{t-1} + K(y_t - CAx_{t-1}^{t-1} - CBu_{t-1} - Du_t) \\ &= Hx_{t-1}^{t-1} + Ky_t + Lu_{t-1} - KDu_t \end{aligned} \tag{1.23}$$

where we have defined $H \equiv A - KCA$ and $L \equiv B - KCB$. Summing both sides for $t = 2, 3, ...T$ we have:

$$\begin{aligned} \sum_{t=2}^{T} x_t^t &= \sum_{t=2}^{T} (Hx_{t-1}^{t-1} + Ky_t + Lu_t) \\ &= H \sum_{t=2}^{T} x_{t-1}^{t-1} + K \sum_{t=2}^{T} y_t + L \sum_{t=2}^{T} u_{t-1} - KD \sum_{t=2}^{T} u_t \end{aligned}$$

which we can rewrite as:

$$\begin{aligned} \sum_{t=1}^{T} x_t^t - x_1^1 = H \left(\sum_{t=1}^{T} x_t^t - x_T^T \right) + K \left(\sum_{t=1}^{T} y_t - y_T \right) + L \left(\sum_{t=1}^{T} u_t - u_T \right) \\ -KD \left(\sum_{t=1}^{T} u_t - u_1 \right) \end{aligned}$$

Then solving for $\sum_{t=1}^{T} x_t^t$ we have:

$$\begin{aligned} \sum_{t=1}^{T} x_t^t = (I - H)^{-1} \Bigg[-Hx_T^T + K \left(\sum_{t=1}^{T} y_t - y_T \right) + L \left(\sum_{t=1}^{T} u_t - u_T \right) \\ -KD \left(\sum_{t=1}^{T} u_t - u_1 \right) + x_1^1 \Bigg] \end{aligned}$$

While the computation of $\sum_{t=1}^{T} y_t$ and $\sum_{t=1}^{T} u_t$ requires $2T$ steps, we can pre-compute them before we know the parameters. And x_1^1 can be computed with a constant number of vector multiplications via the Kalman filter recursions. x_T^T is more difficult, but we can reasonably approximate it by running the steady-state Kalman filter starting from $t = T - k$ with $x_{T-k}^{T-k} = 0$ where k is some constant "lag" factor. The use of k is similar to the use of the horizon parameter in the 4SID algorithm. However, unlike with 4SID we are only applying the crude filtering approximation to one term of the sum $\sum_{t=1}^{T} x_t^t$ (namely at $t = T$).

Thus after pre-computing $\sum_{t=1}^{T} y_t$ and $\sum_{t=1}^{T} u_t$, computing $\sum_{t=1}^{T} x_t^t$ for different parameter settings can be done in constant time (relative to T). An intuitive explanation for why this works is that we are exploiting the linearity of the first-order relation (1.23) to perform the filtering operation for each state *in parallel*.

Unfortunately, when we try to apply this technique to second-order statistics we find that it doesn't directly yield an algorithm to efficiently compute them. Instead, it allows us to express the statistics we need in terms of related statistics that have a larger time-lag (which we define to be the value of k when the statistic is expressed with the notation $(\cdot, \cdot)_k$). In general, we can derive formulas that express a statistic of time-lag k in terms of already known or precomputed statistics (such as $(u, y)_k$) and an unknown statistic of time-lag $k + 1$. Then by approximating certain statistics of large time-lags with carefully chosen conditionally unbiased estimators we can eventually express everything we need as the solution to a finite set of matrix equations and recursive relations (the number of which are independent of T). These matrix equations and relations can then be solved efficiently by methods similar to those used to compute the steady-state covariances, yielding an efficient algorithm.

The time-lag threshold at which we will approximate statistics will be denoted by k_{lim} and is a meta-parameter of our algorithm which determines the trade off between the quality of its approximation and its computational efficiency. Under this scheme the set of pre-computed statistics we need will be just $(y, y)_k$, $(u, y)_k$, $(y, u)_k$ and $(u, u)_k$ for $0 \leq k \leq k_{lim} + 1$. In general, the optimal choice of k_{lim} will not depend on T and in practice, $k_{lim} << T$. The per-iteration time-complexity of our algorithm is $O(k_{lim} N_x^3)$ which critically doesn't depend on T. Only the pre-computation of the required statistics over y and u during the one-time initialization phase of our algorithm depends on T. And in practice this pre-computation can be done much faster than even a single iteration of standard EM.

Chapter 2

Details of the New Algorithm

2.1 Three Simple Identities

First we will present two identities that re-express general first order relations between lists of vectors as second order relations. Suppose that a, b and c^i are lists of vectors indexed over $1...T$ such that $a_t = \sum_i A_i c^i_{t+n_i} \; \forall U \leq t \leq W$ where n_i, U and W are integers such that this expression is well-defined (i.e. all indices are in range), and A_i are appropriately sized matrices. Note that (1.23) is a first-order relation of this form.

If the additional condition $n_i \geq -k \; \forall i$ is satisfied then we have:

$$(a,b)_k = \sum_i A_i \left((c^i,b)_{k+n_i} - \left(\sum_{t=1}^{U-k-1} + \sum_{t=W-k+1}^{T-k-n_i} \right) c^i_{t+k+n_i} b'_t \right) + \left(\sum_{t=1}^{U-k-1} + \sum_{t=W-k+1}^{T-k} \right) a_{t+k} b'_t \tag{2.1}$$

and similarly if the condition $n_i \leq k \; \forall i$ is satisfied,

$$(b,a)_k = \sum_i \left((b,c^i)_{k-n_i} - \left(\sum_{t=1}^{U+n_i-1} + \sum_{t=W+n_i+1}^{T-k+n_i} \right) b_{t+k-n_i} c^{i\prime}_t \right) A'_i + \left(\sum_{t=1}^{U-1} + \sum_{t=W+1}^{T-k} \right) b_{t+k} a'_t \tag{2.2}$$

Finally we have the trivial identity,

$$(a,b)_0 = (b,a)'_0 \tag{2.3}$$

For the interest of simplicity in our derivations we will rewrite the Kalman smoothing recursion (1.22), which holds for $1 \leq t \leq T-1$, as:

$$x_t^T = x_t^t + J(x_{t+1}^T - Ax_t^t - Bu_t) = Jx_{t+1}^T + Px_t^t - JBu_t \tag{2.4}$$

where we have defined $P \equiv I - JA$.

We will derive our approach to computing the second-order statistics required by for the M-step entirely from the simplified Kalman recursions (1.23) and (2.4), and the previous three identities.

2.2 Approximating the First-order Statistics

It will turn out that in order to make use of (2.1) and (2.2) we first need to compute the first order statistics x_t^t and x_t^T for the leading and trailing $k_{lim}+1$ time-steps (x_1^T and x_T^T are also required for the M-step). These computations can be done approximately for the leading time-steps by applying steady-state Kalman filtering/smoothing to the sub-series $y_1, y_2, ..., y_{k_{lag}}$, where $k_{lag} \geq k_{lim}+1$ is a constant s.t. $k_{lag} << T$, and using π_1 for the mean of the initial state distribution. For the trailing time-steps we can use a similar approach applied to the sub-series $y_{T-k_{lag}}, y_{T-k_{lag}+1}, ..., y_T$ but instead approximate the initial distribution as having mean 0 (since no other obvious candidate is available).

Both of these computations can be considered approximate in several senses. For $t \leq k_{lag}$, while x_t^t are computed properly (subject to the steady-state approximation) we are essentially approximating $x_t^T \approx x_t^{k_{lag}}$. And for $t \geq T - k_{lag}$ both x_t^t and x_t^T are being computed without conditioning on y_t for $t < T - k_{lag}$. Nevertheless, these approximations are similar to those used by 4SID except that we are only using them to compute the conditional means for the first and last k_{lim} states.

2.3 Approximating the Second-order Statistics

2.3.1 Computing $(u, x^*)_k$

In order to make use of our "$(\cdot,\cdot)_k$" notation for second-order statistics we define $x_t^* \equiv x_t^t$ for the remainder of this document.

Applying (2.2) to (1.23) we have that for $k \geq 0$ and b an arbitrary indexed list of vectors:

$$\begin{aligned}(b, x^*)_k = (b, x^*)_{k+1}H' + \left((b, y)_k - b_{1+k}y_1'\right)K' + (b, u)_{k+1}L' \\ - \left((b, u)_k - b_{1+k}u_1'\right)D'K' + b_{1+k}{x_1^*}' \end{aligned} \tag{2.5}$$

By taking $b = u$ this identity can be used to recursively compute $(u, x^*)_k$ for each $0 \leq k \leq k_{lim}$ provided we have access to $(u, x^*)_{k_{lim}+1}$ (or an approximation of it) as a starting point for the recursion.

2.3.2 Approximating $(u, x^*)_{k_{lim}+1}$

When we develop our approximation for $(y, x^*)_{k_{lim}+1}$ (in subsection 2.3.5) we will use the fact that y_{t+1} can be estimated (in an unbiased way) from previous values of y and we will apply this approximation to second-order statistics of time-lag $k_{lim}+1$. We can compute such estimates because the LDS model specifies a distribution over y, conditional on u and θ. But it specifies no such distribution on u and so we can make no analogous unbiased estimate for u_{t+1} by conditioning on past data.

One approximation we can use is $(u, x^*)_{k_{lim}+1} = (u, x^*)_{k_{lim}}$ which may be reasonable if $u_{t+1} \approx u_t$ for most values of t. To make use of this approximation we solve (2.5) for $k = k_{lim}$ and $b = u$, substituting $(u, x^*)_{k_{lim}+1}$ for $(u, x^*)_{k_{lim}}$ on the left hand side. This gives:

$$\begin{aligned}(u, x^*)_{k_{lim}+1} \approx \left[\,\left((u, y)_{k_{lim}} - u_{1+k_{lim}}y_1'\right)K' + (u, u)_{k_{lim}}L' \right. \\ \left. - \left((u, u)_{k_{lim}} - u_{1+k_{lim}}u_1'\right)D'K' + u_{1+k_{lim}}{x_1^*}'\right](I - H')^{-1}\end{aligned}$$

A simpler alternative, which works almost as well in practice, is to use $(u, x^*)_{k_{lim}+1} \approx 0$.

2.3.3 Computing $(x^*, u)_k$

Using (2.3) we have:

$$(x^*, u)_0 = (u, x^*)'_0 \tag{2.6}$$

Then applying (2.1) to (1.23) we have for $k \geq 1$ and b an arbitrary indexed list of vectors:

$$(x^*, b)_k = H\left((x^*, b)_{k-1} - x^*_T b'_{T-k+1}\right) + K(y, b)_k + L\left((u, b)_{k-1} - u_T b'_{T-k+1}\right) - KD(u, b)_k \tag{2.7}$$

Taking $b = u$ this can be used to recursively compute $(x^*, u)_k$ for $0 \leq k \leq k_{lim}$.

2.3.4 Computing $(y, x^*)_k$

Taking $b = y$, identity (2.5) can be used to recursively compute $(y, x^*)_k$ for each $0 \leq k \leq k_{lim}$ provided we have access to $(y, x^*)_{k_{lim}+1}$ (or an approximation of it) as a starting point for the recursion.

2.3.5 Approximating $(y, x^*)_{k_{lim}+1}$

We desire an approximation to $(y, x^*)_{k_{lim}+1}$ that is in some sense conditionally unbiased and available from the other statistics we have computed (or will compute).

Since $x^{t-k}_{t-k} \equiv E_{\theta_n}[\ x_{t-k} \mid y_{\leq t-k}, u_{\leq t-k}\ ]$ is only a function of $y_{\leq t-k}$ and $u_{\leq t-k}$ we may move it outside of the expectation in the following computation (which is valid for $1 \leq t \leq T-1$):

$$\begin{aligned} E_{\theta_n}[\, y_{t+1} {x^*_{t-k}}' \mid y_{\leq t}, u_{\leq t+1}\,] &= E_{\theta_n}[\, y_{t+1} \mid y_{\leq t}, u_{\leq t+1}\,] {x^*_{t-k}}' \\ &= (Cx^t_{t+1} + Du_{t+1}) {x^*_{t-k}}' \\ &= (CAx^t_t + CBu_t + Du_{t+1}) {x^*_{t-k}}' \end{aligned}$$

where the parameter value (θ_n) used to take the outer expectations is the same one used to compute the terms x_i^j and also corresponds to C and A as they appear here. It should be noted that this proof assumes that the y_t are generated by our model with parameters θ_n. In other words, the quality of the approximation depends on how well the model (with the current parameter estimates) explains the data.

Thus the approximation:

$$y_{t+1}{x^*_{t-k}}' \approx (CAx_t^t + CBu_t + Du_{t+1}){x^*_{t-k}}'$$

is conditionally unbiased.

Taking $k = k_{lim}$ and summing both sides we have:

$$\begin{aligned}(y, x^*)_{k_{lim}+1} \approx CA\left((x^*, x^*)_{k_{lim}} - x^*_T {x^*_{T-k_{lim}}}'\right) \\ + CB\left((u, x^*)_{k_{lim}} - u_T {x^*_{T-k_{lim}}}'\right) + D(u, x^*)_{k_{lim}+1}\end{aligned}$$

Before we can apply this approximation we must first be able to compute $(x^*, x^*)_{k_{lim}}$. However, it will turn out that the relations we have derived above will allow us to compute and solve a matrix-equation for $(x^*, x^*)_{k_{lim}}$.

Note that we could have used an approximation analogous to either of those considered in subsection 2.3.2 for $(u, x^*)_{k_{lim}+1}$. However, we have found that this works poorly in practice unless $\rho(H)^{k_{lim}} \approx 0$, in which case it (provably) doesn't matter what approximation we use. But when $\rho(H)^{k_{lim}}$ was sufficiently large, of all approximation we tried only the one we develop above led to symmetric estimates of the covariance matrices Q and R in the M-step.

2.3.6 Computing $(x^*, y)_k$

Using (2.3) we have:

$$(x^*, y)_0 = (y, x^*)_0' \tag{2.8}$$

Then taking $b = y$, identity (2.7) can be used to recursively compute $(x^*, y)_k$ for $0 \leq k \leq k_{lim}$.

2.3.7 Computing $(x^*, x^*)_{k_{lim}}$

In the previous sub-sections we derived recursions for approximately computing $(y, x^*)_k$ and $(x^*, y)_k$ given the value of $(x^*, x^*)_{k_{lim}}$. In this sub-section we will show that the approximations already given for $(y, x^*)_{k_{lim}+1}$ and $(u, x^*)_{k_{lim}+1}$ determine a unique solution for $(x^*, x^*)_{k_{lim}}$ which can (with some mathematical difficulty) be computed efficiently.

First we make explicit the dependency on $(x^*, x^*)_{k_{lim}}$ of $(y, x^*)_k$ and $(x^*, y)_k$. For $0 \leq k \leq k_{lim}$ we have:

$$(y, x^*)_k = (y, x^*)_k^\dagger + CA(x^*, x^*)_{k_{lim}} H^{k_{lim}+1-k'}$$

$$(x^*, y)_k = (x^*, y)_k^\dagger + H^{k_{lim}+1+k}(x^*, x^*)'_{k_{lim}} A'C'$$

where $(y, x^*)_k^\dagger$ and $(x^*, y)_k^\dagger$ are the values of $(y, x^*)_k$ and $(x^*, y)_k$ as computed by the recursions given in the previous sub-sections but starting with $(x^*, x^*)_{k_{lim}} = 0$ in the approximation for $(y, x^*)_{k_{lim}+1}$. This can be easily seen by tracing through said recursions and proved by a simple induction.

Taking $b = x^*$ and $k = k_{lim} + 1$ in identity (2.7), substituting our approximation for $(y, x^*)_{k_{lim}+1}$ and simplifying we have:

$$(x^*, x^*)_{k_{lim}+1} = A\left((x^*, x^*)_{k_{lim}} - x_T^* x_{T-k_{lim}}^*\right) + B\left((u, x^*)_{k_{lim}} - u_T x_{T-k_{lim}}^*\right)$$

Taking $b = x^*$ and $k = k_{lim}$ in (2.5), substituting the above expressions for $(x^*, x^*)_{k_{lim}+1}$ and $(x^*, y)_{k_{lim}}$ and isolating the terms involving $(x^*, x^*)_{k_{lim}}$ we have:

$$(x^*, x^*)_{k_{lim}} = A(x^*, x^*)_{k_{lim}} H' + H^{2k_{lim}+1}(x^*, x^*)_{k_{lim}}{}' A'C'K' + G \tag{2.9}$$

where,

$$G \equiv (-Ax_T^* x_{T-k_{lim}}^*{}' + B\left((u, x^*)_{k_{lim}} - u_T x_{T-k_{lim}}^*\right))H' + \left((x^*, y)_{k_{lim}}^\dagger - x_{1+k_{lim}}^* y_1'\right) K'$$
$$+ (x^*, u)_{k_{lim}+1} L' - \left((x^*, u)_{k_{lim}} - x_{1+k_{lim}}^* u_1'\right) D'K' + x_{1+k_{lim}}^* x_1^{*\prime}$$

In principle we can solve this for $(x^*, x^*)_{k_{lim}}$ by re-writing it as:

$$(I \otimes I - H \otimes A - (KCA \otimes H^{2k_{lim}+1})T_{N_x, N_x}) \operatorname{vec}((x^*, x^*)_{k_{lim}}) = \operatorname{vec}(G)$$

where T_{N_x,N_x} is a "transposition operator" defined by: $T_{N_x,N_x}\,\mathrm{vec}(Z) = \mathrm{vec}(Z')$ for an N_x-by-N_x matrix Z.

Unfortunately this approach is very inefficient and numerically unstable for even modestly large values of N_x since the linear system is N_x^2 by N_x^2. A more feasible possibility would be to iterate (2.9) until it converges to its fixed point, but this can require very many iterations in practice.

Since only simpler forms of this type of equation (e.g. Sylvester equations) have efficient exact algorithms in the literature we need to develop a specialized algorithm to solve this equation. Fortunately, we have found such an algorithm, a description and derivation of which is given in the appendix.

2.3.8 Computing $(x^*, x^*)_k$

Using $(x^*, x^*)_{k_{lim}}$ as a starting point we can recursively compute $(x^*, x^*)_k$ for $0 \leq k \leq k_{lim} - 1$ using identity (2.7) with $b = x^*$.

2.3.9 Computing $(x^T, x^*)_k$

Applying (2.1) to (2.4) we have that for $k \geq 0$ and b an arbitrary indexed list of vectors:

$$\begin{aligned}(x^T, b)_k = J(x^T, b)_{k+1} + P\left((x^*, b)_k - x_T^* b_{T-k}'\right)& \\ - JB\left((u, b)_k - u_T b_{T-k}'\right) + x_T^T b_{T-k}'& \qquad (2.10)\end{aligned}$$

By taking $b = x^*$ this identity can be used to recursively compute $(x^T, x^*)_k$ for each $0 \leq k \leq k_{lim} - 1$ provided we have access to $(x^T, x^*)_{k_{lim}}$ (or an approximation of it) as a starting point for the recursion.

2.3.10 Approximating $(x^T, x^*)_{k_{lim}}$

Like before we compute the conditional expectation:

$$\begin{aligned} E_{\theta_n}[\ x_t^T {x^*_{t-k}}' \mid y_{\leq t}, u_{\leq t}\ ] &= E_{\theta_n}[\ x_t^T \mid y_{\leq t}, u_{\leq t}\ ]{x^*_{t-k}}' \\ &= E_{\theta_n}[\ E_{\theta_n}[x_t | y, u] \mid y_{\leq t}, u_{\leq t}\ ]{x^*_{t-k}}' \\ &= E_{\theta_n}[\ x_t \mid y_{\leq t}, u_{\leq t}\ ]{x^*_{t-k}}' = x_t^* {x^*_{t-k}}' \end{aligned}$$

where the second line follows from the "law of iterated expectations". Thus the following approximation is conditionally unbiased:

$$x_t^T {x^*_{t-k}}' \approx x_t^t {x^*_{t-k}}'$$

Taking $k = k_{lim}$ and summing both sides we get:

$$(x^T, x^*)_{k_{lim}} \approx (x^*, x^*)_{k_{lim}}$$

2.3.11 Computing $(x^*, x^T)_0$

Using (2.3) we have:

$$(x^*, x^T)_0 = (x^T, x^*)_0'$$

2.3.12 Computing $(x^T, u)_k$

Taking $b = u$, identity (2.10) can be used to recursively compute $(x^T, u)_k$ for each $0 \leq k \leq k_{lim}$ provided we have access to $(x^T, u)_{k_{lim}}$ (or an approximation of it) as a starting point for the recursion.

2.3.13 Approximating $(x^T, u)_{k_{lim}}$

Using an identical argument to our proof that $E_{\theta_n}[\ x_t^T {x^*_{t-k}}' \mid y_{\leq t}, u_{\leq t}\ ] = x_t^* {x^*_{t-k}}'$ we have:

$$E_{\theta_n}[\ x_t^T {u_{t-k}}' \mid y_{\leq t}, u_{\leq t}\ ] = x_t^t {u_{t-k}}'$$

so that the following approximation is conditionally unbiased:

$$x_t^T {u_{t-k}}' \approx x_t^t {u_{t-k}}'$$

Taking $k = k_{lim}$ and summing both sides we get:

$$(x^T, u)_{k_{lim}} \approx (x^*, u)_{k_{lim}}$$

2.3.14 Computing $(u, x^T)_0$

Using (2.3) we have:

$$(u, x^T)_0 = (x^T, u)_0'$$

2.3.15 Computing $(x^T, x^T)_k$ for $k = 0, 1$

Applying (2.2) to (2.4) we have for $k \geq 1$ and b an arbitrary indexed list of vectors:

$$(b, x^T)_k = \left((b, x^T)_{k-1} - b_k x_1^T\right) J' + (b, x^*)_k P' - (b, u)_k B' J' \tag{2.11}$$

For $b = x^T$, identity (2.11) allows us to express $(x^T, x^T)_{k+1}$ in terms of $(x^T, x^T)_k$. Similarly for $b = x^T$, identity (2.10) allows us to express $(x^T, x^T)_{k+1}$ in terms of $(x^T, x^T)_k$. Substituting the first expression into the second we have:

$$\begin{aligned}(x^T, x^T)_k = J\Big(&((x^T, x^T)_k - x_{k+1}^T x_1^{T\prime})J' + (x^T, x^*)_{k+1} P' - (x^T, u)_{k+1} B' J'\Big) \\ &+ P\Big((x^*, x^T)_k - x_T^* {x_{T-k}^T}'\Big) - JB\Big((u, x^T)_k - u_T {x_{T-k}^T}'\Big) + x_T^T {x_{T-k}^T}'\end{aligned} \tag{2.12}$$

We can compute $(x^T, x^T)_0$ by solving this equation with $k = 0$. Fortunately, it has the form of a discrete Lyapunov equation and thus can efficiently solved using standard methods (see appendix).

Then taking $b = x^T$ and $k = 1$ in identity (2.11) we can compute $(x^T, x^T)_1$ from $(x^T, x^T)_0$.

2.3.16 Computing $(x^T, y)_k$

Taking $b = y$, identity (2.10) can be used to recursively compute $(x^T, y)_k$ for each $0 \leq k \leq k_{lim}$ provided we have access to $(x^T, y)_{k_{lim}}$ (or an approximation of it) as a starting point for the recursion.

2.3.17 Approximating $(x^T, y)_{k_{lim}}$

Again using an identical argument to our proof that $E_{\theta_n}[\ x_t^T x_{t-k}^*{}' \mid y_{\leq t}, u_{\leq t}\ ] = x_t^* x_{t-k}^*{}'$ we have:

$$E_{\theta_n}[\ x_t^T y_{t-k}{}' \mid y_{\leq t}, u_{\leq t}\ ] = x_t^t y_{t-k}{}'$$

so that the following approximation is conditionally unbiased:

$$x_t^T y_{t-k}{}' \approx x_t^t y_{t-k}{}'$$

Taking $k = k_{lim}$ and summing both sides we get:

$$(x^T, y)_{k_{lim}} \approx (x^*, y)_{k_{lim}}$$

2.4 Computing The Log-Likelihood

While we don't need to compute the log-likelihood function (ℓ) in order to optimize it with EM, it may be desirable to have it in order to monitor the progress and convergence of the algorithm. While it cannot be directly computed from the same statistics required by the M-step, it *can* be computed from some of the the intermediate statistics found during the approximate E-step.

Recalling that $S_t \equiv \mathrm{Cov}_\theta[\bar{y}_t \mid y_{\leq t}, u_{\leq t}]$ where $\bar{y}_t = E_{\theta_n}[y_t \mid y_{\leq t-1}, u_{\leq t}] = Cx_t^{t-1} + Du_t$, it can be easily shown that $p(y_t \mid y_{\leq t-1}, u_{\leq t}) = n(\bar{y}_t; 0, S_t)$. Note that $x_1^0 = \pi_1$ and for $t > 1$, $x_t^{t-1} = Ax_{t-1}^* + Bu_t$.

So under the steady-state approximation $S_t = S$ we have:

$$\begin{aligned}
\ell &= \sum_{t=1}^{T} \log p(y_t \mid y_{\leq t-1}, u_{\leq t}) = -\frac{1}{2}\sum_{t=1}^{T}(N_y \log 2\pi + \log|S| + (y_t - Cx_t^{t-1})' S^{-1}(y_t - Cx_t^{t-1})) \\
&= -\frac{1}{2}[TN_y \log 2\pi + T\log|S| + (y_t - C\pi_1 - Du_t)' S^{-1}(y_t - C\pi_1 - Du_t) \\
&\quad + \sum_{t=2}^{T}(y_t - CAx^*_{t-1} - CBu_{t-1} - Du_t)' S^{-1}(y_t - CAx^*_{t-1} - CBu_{t-1} - Du_t)]
\end{aligned}$$

where the large summation term can be written as:

$$\begin{aligned}
&\operatorname{tr}[S^{-1}\sum_{t=2}^{T}(y_t - CAx^*_{t-1} - CBu_{t-1} - Du_t)(y_t - CAx^*_{t-1} - CBu_{t-1} - Du_t)'] \\
&= \operatorname{tr}[S^{-1}\sum_{t=2}^{T} y_t y_t' - 2y_t {x^*_{t-1}}' A'C' + CAx^*_{t-1}{x^*_{t-1}}' A'C' - 2y_t u'_{t-1}B'C' + 2CAx^*_{t-1}u'_{t-1}B'C' \\
&\qquad + 2Du_t u'_{t-1}B'C' + CBu_{t-1}u'_{t-1}B'C' - 2y_t u'_t D' + 2Du_t x'_{t-1}A'C' + 2Du_t u'_{t-1}B'C' + Du_t u'_t D'] \\
&= \operatorname{tr}[S^{-1}((y,y)_0 - y_1 {y_1}' - 2(y,x^*)_1 A'C' + CA((x^*,x^*)_0 - x^*_T {x^*_T}')A'C' - 2(y,u)_1 B'C' \\
&\qquad + 2CA((x^*,u)_0 - x^*_T u'_T)B'C' + 2D(u,u)_1 B'C' + CB((u,u)_0 - u_T u'_T)B'C' \\
&\qquad - 2((y,u)_0 - y_1 u'_1)D' + 2D(u,x^*)_1 A'C' + 2D(u,u)_1 B'C' + D((u,u)_0 - u_1 u'_1)D')]
\end{aligned}$$

This final formula involves second-order statistics that are computed in the approximate E-step and thus can be evaluated efficiently. However, because these statistics are approximate, this way of computing ℓ should not be considered definitive. Indeed, when the approximation fails (due to, for example, k_{lim} being set too low) the approximately computed ℓ might be very far (in either direction) from ℓ's true value.

2.5 Summary of Algorithm

In this section we provide a code-like summary of the approximate EM algorithm we have developed in the previous sections. While we continue to use the same notation for the sake of consistency, for the purposes of implementation the symbols can be interpreted as meaningless "variable names".

2.5.1 Initialization

1. Compute $(y,y)_k$, $(y,u)_k$, $(u,y)_k$ and $(u,u)_k$ for $k := 0, 1, ..., k_{lim}+1$ by direct evaluation or more efficiently by zero-padding y and u and using the Circular Convolution Theorem

2. Initialize model parameters to random starting values satisfying:

    - $\rho(A) < 1$
    - R, Q, and Π_1 positive definite

2.5.2 Approximate E-step

1. Solve the following DARE for Λ_0^1 using the doubling algorithm or some other efficient method (see the appendix):

$$\Lambda_0^1 = A\left(\Lambda_0^1 - \Lambda_0^1 C'(C\Lambda_0^1 C' + R)^{-1} C\Lambda_0^1\right) A' + Q$$

2. Compute:

$$S := C\Lambda_0^1 C' + R \quad K := \Lambda_0^1 C' S^{-1} \quad \Lambda_0^0 := \Lambda_0^1 - KC\Lambda_0^1 \quad J := \Lambda_0^0 A'(\Lambda_0^1)^{-1}$$

3. Solve the discrete Lyapunov equation for Λ_0 using the doubling algorithm (see the appendix) or another efficient method:

$$\Lambda_0 = \Lambda_0^0 + J(\Lambda_0 - \Lambda_0^1)J'$$

4. Compute $\quad V_1^T := V_0^T J'$

5. Compute $\quad H := A - KCA$, $P := I - JA$ and $L := B - KCB$

6. Compute $\quad x_1^* := \pi_1 + K(y_1 - C\pi_1)$

7. For $k = 2,\ 3,\ ...,\ k_{lag}+1$

$$x_t^* := Hx_{t-1}^* + Ky_t + Lu_{t-1} - KDu_t$$

8. Approximate $\quad x^T_{1+k_{lag}} := x^*_{1+k_{lag}}$

9. For $k = k_{lag},\ k_{lag}-1,\ ...,\ 1$ compute:

$$x^T_t := Jx^T_{t+1} + Px^*_t - JBu_t$$

10. Approximate $\quad x^*_{T-k_{lim}-1} := 0$

11. For $k = T-k_{lag},\ T-k_{lag}+1,\ ...,\ T$

$$x^*_t := Hx^*_{t-1} + Ky_t + Lu_{t-1} - KDu_t$$

12. Approximate:

$$\begin{aligned}(u,x^*)_{k_{lim}+1} := [\,&((u,y)_{k_{lim}} - u_{1+k_{lim}}y_1')\,K' + (u,u)_{k_{lim}}L' \\ &- ((u,u)_{k_{lim}} - u_{1+k_{lim}}u_1')\,D'K' + u_{1+k_{lim}}x_1^{*\prime}](I-H')^{-1}\end{aligned}$$

13. For $k = k_{lim},\ k_{lim}-1,\ ...,\ 1,\ 0$ compute:

$$\begin{aligned}(u,x^*)_k := (u,x^*)_{k+1}H' + &((u,y)_k - u_{1+k}y_1')\,K' + (u,u)_{k+1}L' \\ &- ((u,u)_k - u_{1+k}u_1')\,D'K' + u_{1+k}x_1^{*\prime}\end{aligned}$$

14. Compute $\quad (x^*,u)_0 := (u,x^*)_0'$

15. For $k = 1,\ 2,\ ...,\ k_{lim}$ compute:

$$(x^*,u)_k := H\left((x^*,u)_{k-1} - x^*_T u'_{T-k+1}\right) + K(y,u)_k + L\left((u,u)_{k-1} - u_T u'_{T-k+1}\right) - KD(u,u)_k$$

16. Approximate:

$$(y,x^*)^\dagger_{k_{lim}+1} := -CAx^*_T x^*_{T-k_{lim}}{}' + CB\left((u,x^*)_{k_{lim}} - u_T x^*_{T-k_{lim}}{}'\right) + D(u,x^*)_{k_{lim}+1}$$

17. For $k = k_{lim},\ k_{lim}-1,\ ...,\ 1,\ 0$ compute:

$$\begin{aligned}(y,x^*)^\dagger_k := (y,x^*)^\dagger_{k+1}H' + &((y,y)_k - y_{1+k}y_1')\,K' + (y,u)_{k+1}L' \\ &- ((y,u)_k - y_{1+k}u_1')\,D'K' + y_{1+k}x_1^{*\prime}\end{aligned}$$

18. Compute $\quad (x^*, y)^\dagger_0 := (y, x^*)^{\dagger\,\prime}_0$

19. For $k = 1,\ 2,\ ...,\ k_{lim}$ compute:

$$(x^*, y)^\dagger_k := H\left((x^*, y)^\dagger_{k-1} - x^*_T y'_{T-k+1}\right) + K(y, y)_k + L\left((u, y)_{k-1} - u_T y'_{T-k+1}\right) - KD(u, y)_k$$

20. Compute:

$$G := (-Ax^*_T x^*_{T-k_{lim}}{}' + B\left((u, x^*)_{k_{lim}} - u_T x^*_{T-k_{lim}}\right))H' + \left((x^*, y)^\dagger_{k_{lim}} - x^*_{1+k_{lim}} y'_1\right) K' \\ + (x^*, u)_{k_{lim}+1} L' - \left((x^*, u)_{k_{lim}} - x^*_{1+k_{lim}} u'_1\right) D'K' + x^*_{1+k_{lim}} x^{*\prime}_1$$

21. Solve the following equation for $(x^*, x^*)_{k_{lim}}$ using the algorithm developed in the appendix:

$$(x^*, x^*)_{k_{lim}} = A(x^*, x^*)_{k_{lim}} H' + H^{2k_{lim}+1} (x^*, x^*)_{k_{lim}}{}' A'C'K' + G$$

22. Compute the following updates for $k \in \{0,\ 1,\ 2,\ ...,\ k_{lim}\}$:

$$(y, x^*)_k := (y, x^*)^\dagger_k + CA(x^*, x^*)_{k_{lim}} H^{k_{lim}+1-k'}$$

$$(x^*, y)_k := (x^*, y)^\dagger_k + H^{k_{lim}+1+k} (x^*, x^*)'_{k_{lim}} A'C'$$

23. For $k = k_{lim}-1,\ k_{lim}-2,\ ...,\ 1,\ 0$ compute:

$$(x^*, x^*)_k := (x^*, x^*)_{k+1} H' + \left((x^*, y)_k - x^*_{1+k} y'_1\right) K' + (x^*, u)_{k+1} L' \\ - \left((x^*, u)_k - x^*_{1+k} u'_1\right) D'K' + x^*_{1+k} x^{*\prime}_1$$

24. Approximate $\quad (x^T, x^*)_{k_{lim}} := (x^*, x^*)_{k_{lim}}$

25. For $k = k_{lim}-1,\ k_{lim}-2,\ ...,\ 1,\ 0$ compute:

$$(x^T, x^*)_k := J(x^T, x^*)_{k+1} + P\left((x^*, x^*)_k - x^*_T x^*_{T-k}{}'\right) - JB\left((u, x^*)_k - u_T x^*_{T-k}{}'\right) + x^T_T x^*_{T-k}{}'$$

26. Compute $\quad (x^*, x^T)_0 := (x^T, x^*)'_0$

27. Approximate $\quad (x^T, u)_{k_{lim}} := (x^*, u)_{k_{lim}}$

28. For $k = k_{lim}-1,\ k_{lim}-2,\ ...,\ 1,\ 0$ compute:

$$(x^T, u)_k := J(x^T, u)_{k+1} + P\left((x^*, u)_k - x_T^* u'_{T-k}\right) - JB\left((u, u)_k - u_T u'_{T-k}\right) + x_T^T u'_{T-k}$$

29. Compute $\quad (u, x^T)_0 := (x^T, u)'_0$

30. Solve the following Lyapunov equation for $(x^T, x^T)_0$ using an efficient method (see appendix):

$$\begin{aligned}(x^T, x^T)_0 = J &\Big(((x^T, x^T)_0 - x_1^T x_1^{T\prime})J' + (x^T, x^*)_1 P' - (x^T, u)_1 B'J'\Big) \\ &+ P\Big((x^*, x^T)_0 - x_T^* x_T^{T\prime}\Big) - JB\Big((u, x^T)_0 - u_T x_T^{T\prime}\Big) + x_T^T x_T^{T\prime}\end{aligned}$$

31. Compute $\quad (x^T, x^T)_1 := \left((x^T, x^T)_0 - x_1^T x_1^T\right) J' + (x^T, x^*)_1 P' - (x^T, u)_1 B'J'$

32. Approximate $\quad (x^T, y)_{k_{lim}} := (x^*, y)_{k_{lim}}$

33. For $k = k_{lim}-1,\ k_{lim}-2,\ ...,\ 1,\ 0$ compute:

$$(x^T, y)_k := J(x^T, y)_{k+1} + P\left((x^*, y)_k - x_T^* y'_{T-k}\right) - JB\left((u, y)_k - u_T y'_{T-k}\right) + x_T^T y'_{T-k}$$

34. Compute $\quad (y, x^T)_0 := (x^T, y)'_0$

2.5.3 M-step

1. Compute $E_{\theta_n}[(x, x)_0 \mid y, u] := (x^T, x^T)_0 + T\Lambda_0$ and $E_{\theta_n}[(x, x)_1 \mid y, u] := (x^T, x^T)_1 + (T-1)\Lambda_1$. We will write these as E_0 and E_1 respectively, for brevity.

2. Compute the parameter updates (see the appendix for a derivation):

$$
\begin{aligned}
C &:= \left((y, x^T)_0 - (y, u)_0 (u, u)_0^{-1} (u, x^T)_0\right) \left(E_0 - (x^T, u)_0 (u, u)_0^{-1} (u, x^T)_0\right)^{-1} \\
D &:= \left((y, u)_0 - C(x^T, u)_0\right) (u, u)_0^{-1} \\
R &:= \frac{1}{T} \left((y, y)_0 - C(x^T, y)_0 - D(u, y)_0\right) \\
A &:= \Big(E_1 - (x^T, u)_1 ((u, u)_0 - u_T u_T')^{-1} ((u, x^T)_0 - u_T x_T^{T'})\Big) \\
&\qquad \cdot \Big(E_0 - x_T^T x_T^{T'} - V_0^T - ((x^T, u)_0 - x_T^T u_T')((u, u)_0 - u_T u_T')^{-1} ((u, x^T)_0 - u_T x_T^{T'})\Big)^{-1} \\
B &:= \left((x^T, u)_1 - A((x^T, u)_0 - x_T^T u_T')\right) ((u, u)_0 - u_T u_T')^{-1} \\
Q &:= \frac{1}{T-1} ((x^T, x^T)_0 - x_1^T x_1^{T'} - \Lambda_0 - A(x^T, x^T)_1' - B(x^T, u)_1') \\
\pi_1 &:= x_1^T \\
\Pi_1 &:= x_1^T x_1^{T'} + \Lambda_0 - \pi_1 \pi_1'
\end{aligned}
$$

Chapter 3

Experimental Evaluation

3.1 Experimental Setup

Our experiments were designed to investigate the performance characteristics of our approximate EM algorithm while comparing it to EM with just the steady-state approximation (SSEM) and standard/classical EM. We also investigated the trade-off between solution quality and speed as a function of the meta-parameter k_{lim}.

Each algorithm was implemented efficiently in MATLAB and run on an Intel 3.2GHz quad-core machine with 4GB of DDR2 ram. Our implementations were carefully "vectorized" so that there was only a minimal performance degradation due to MATLAB's code execution overhead. Log-likelihoods were computed every 5 iterations as each algorithm ran. Each algorithm was given the same randomly chosen initial parameters and run for the same number of iterations (which was determined to be when they all approximately converged). Running times were calculated to ignore the extra time needed to compute the log likelihoods.

We used several datasets in our experiments. The first was a 3-dimensional time-series and corresponding 3-dimensional control signal of length 6305 which consisted of sensor readings from an industrial milk evaporator. This is a standard dataset used in system identification (e.g. Zhu et al., 1994) and is available online from the Database for the Identification of Systems (DaISy). Also from DaiSy we used a dataset containing a 1-dimensional times-series and corresponding 1-dimensional control signal of length 4000 which consisted of sensor readings from a "liquid-satured steam heat exchanger".

Another dataset we used was the first 10 dimensions of a 49-dimensional time-series of length 15300 (with no control signals) consisting of transformed sensor readings from a motion capture experiment. This dataset is available on-line from the Carnegie Mellon University Motion Capture Database and was preprocessed as in Taylor et al. (2007).

Finally, we used two very long datasets, also available online, from the Signal Processing Information Base (SPIB) based at Rice University. The first was a 48-dimensional time-series of length 301,056 consisting of sensor readings from a sonar experiment conducted by the NATO SACLANT Center. In particular, we used the data from the "350 Hz source" experiment. The second dataset we used from SPIB was a very long 1-dimensional time series consisting of the first 750,000 time-steps (38 seconds) of an audio recording taken in the "Operations Room" of a destroyer warship. We truncated this dataset because SSEM was taking too much time and memory to make our tests feasible (while standard EM was grossly infeasible).

3.2 Experimental Results

We will present our results as a series of graphs of log-likelihood versus iteration number, with the parameters and other important details of the experiment given in the captions. 'AEM-n' is our approximate EM algorithm with $k_{lim} = n$ and $k_{lag} = 2n + 1$. Running times are listed in brackets. Note that the point of these graphs is to show that despite its liberal use of approximations, AEM achieves a per-iteration performance that is comparable with EM and SSEM, as long as k_{lim} is set high enough. A graph of log-likelihood versus time would only obscure this point and would ultimately serve no purpose since in practice LDS learning algorithms are run until convergence and this happens after roughly the same number of iterations for each algorithm. Also note that while the graphs may visually suggest a large difference between the quality of the models learned by the different algorithms, special attention should be paid to the scale of the y-axis as the graphs have been carefully zoomed to emphasize differences that are often negligible.

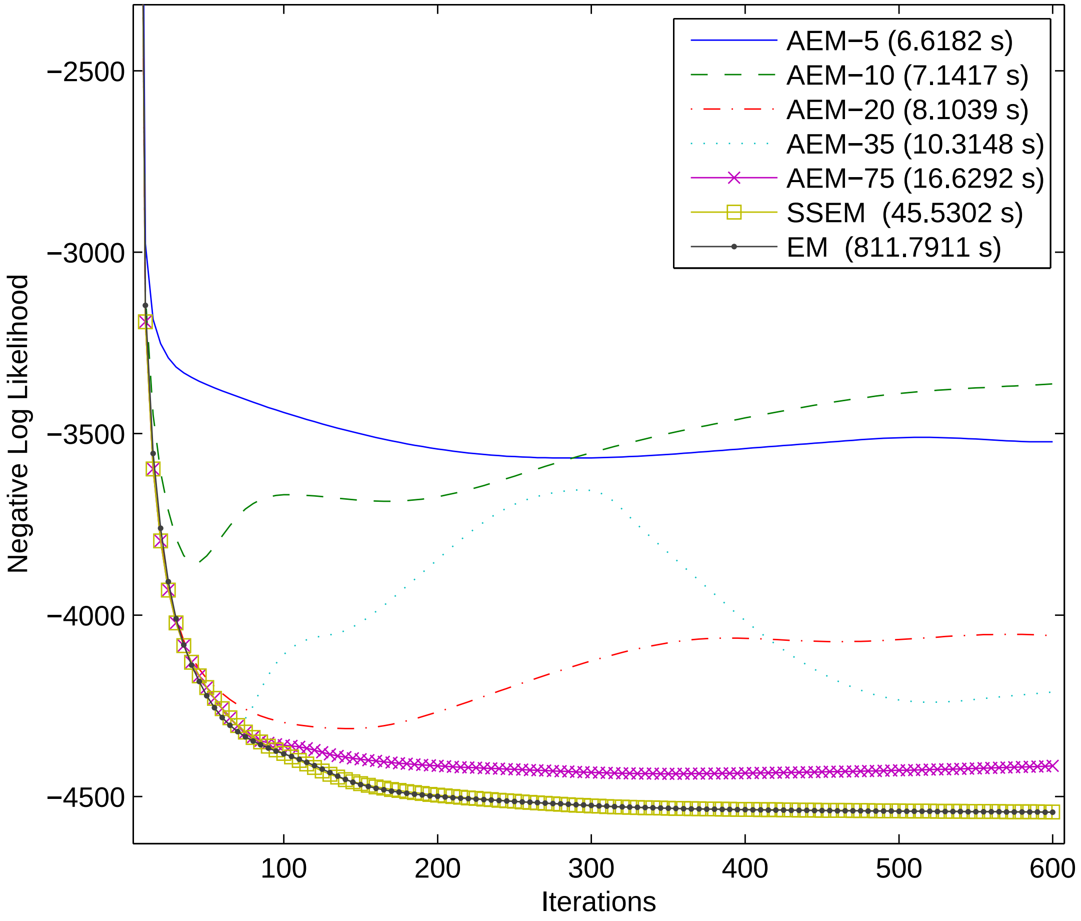

Figure 3.1: Results from the milk evaporator dataset with $N_x = 15$.

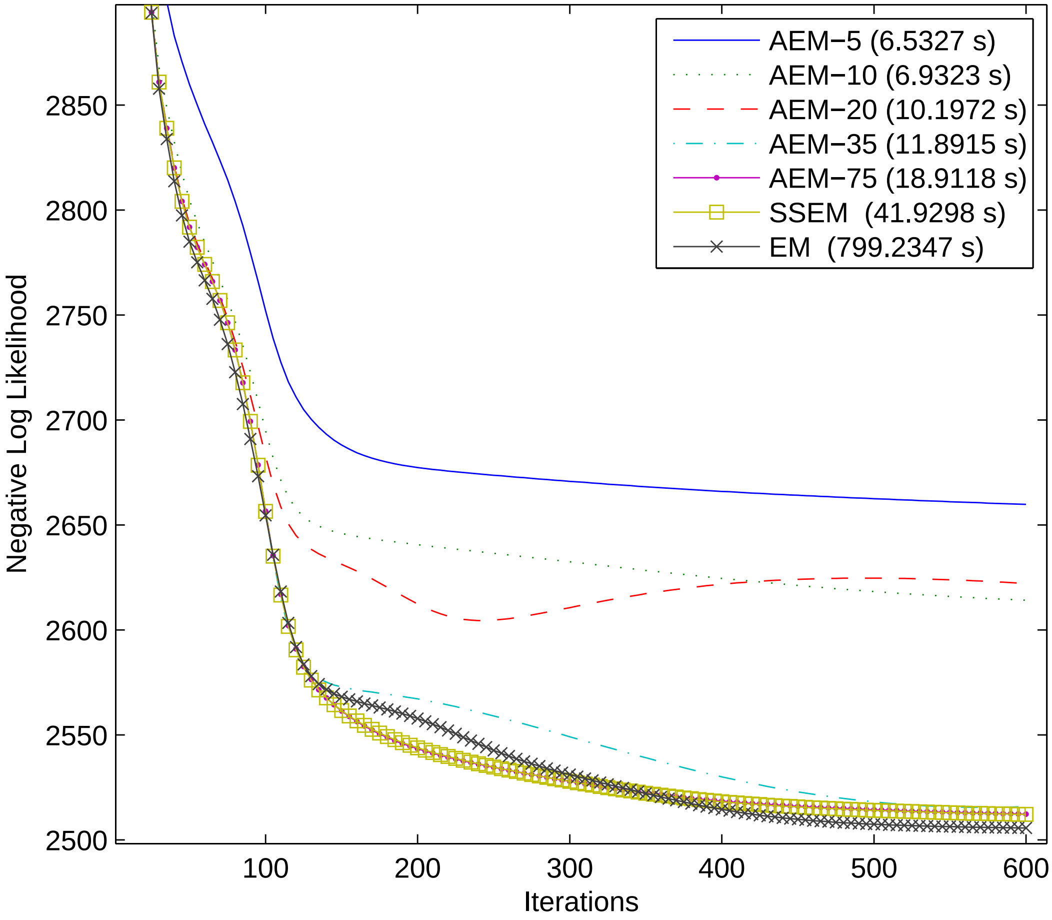

Figure 3.2: Results from the milk evaporator dataset with $N_x = 15$ and the input/control signals ignored.

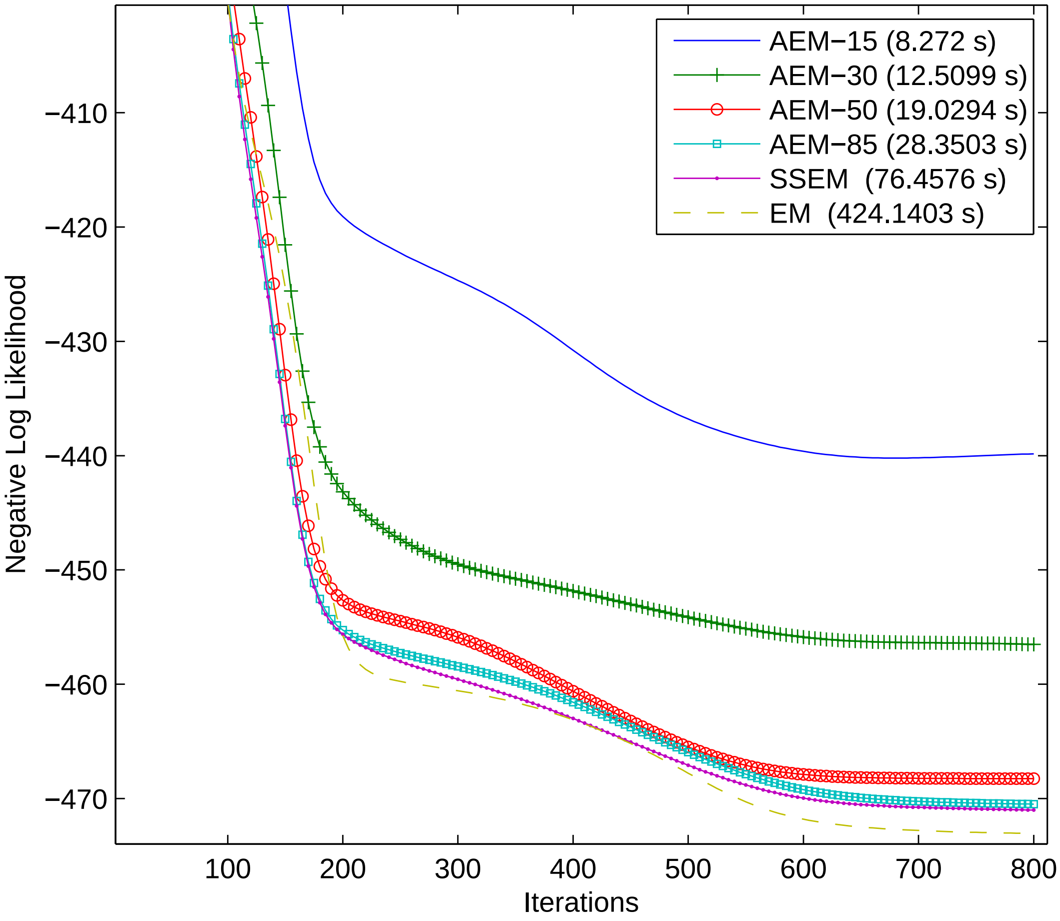

Figure 3.3: Results from the heat exchanger dataset with $N_x = 8$. Note that our implementation of AEM initially had problems with this dataset until we mean-subtracted it (which won't be a problem for most potential applications since the mean can always be added back in). Before mean subtraction the dataset has a very large fixed offset from 0 which seemed to be causing numerical issues.

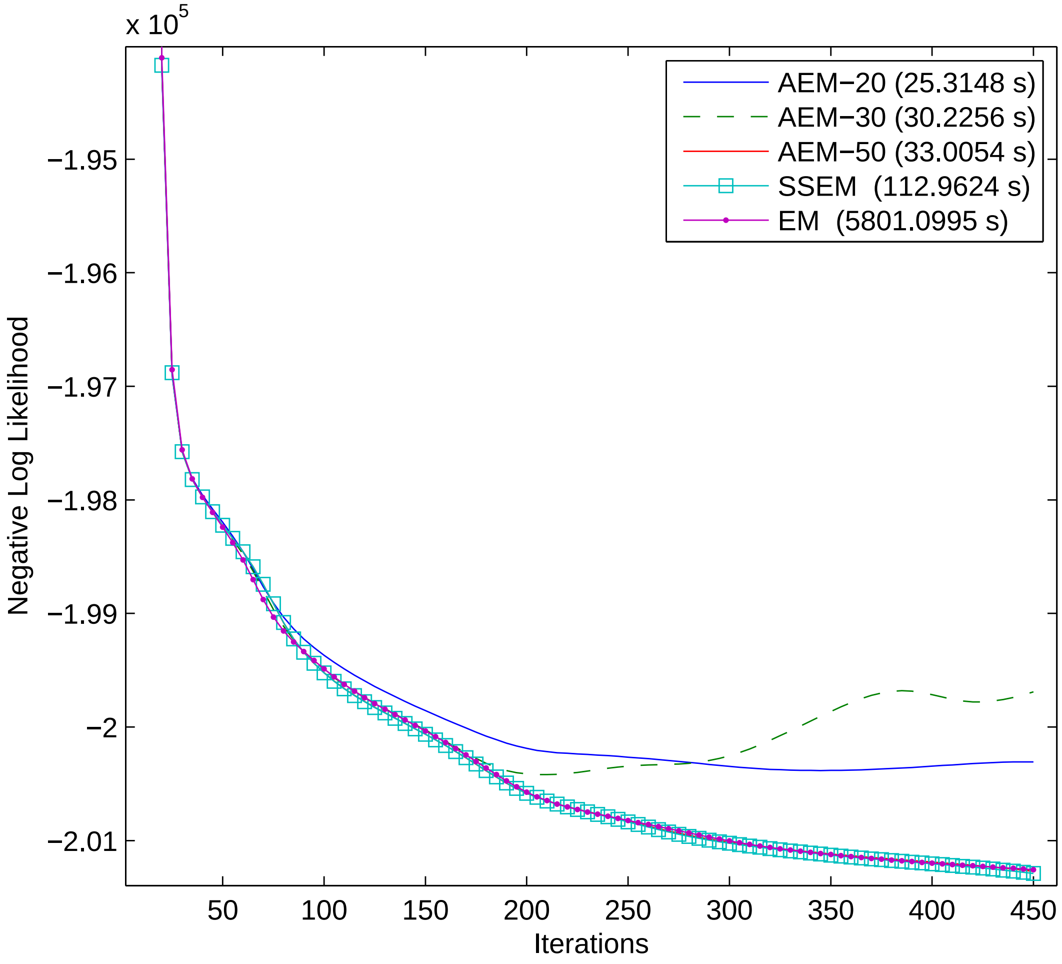

Figure 3.4: Results from the motion-capture dataset with $N_x = 40$.

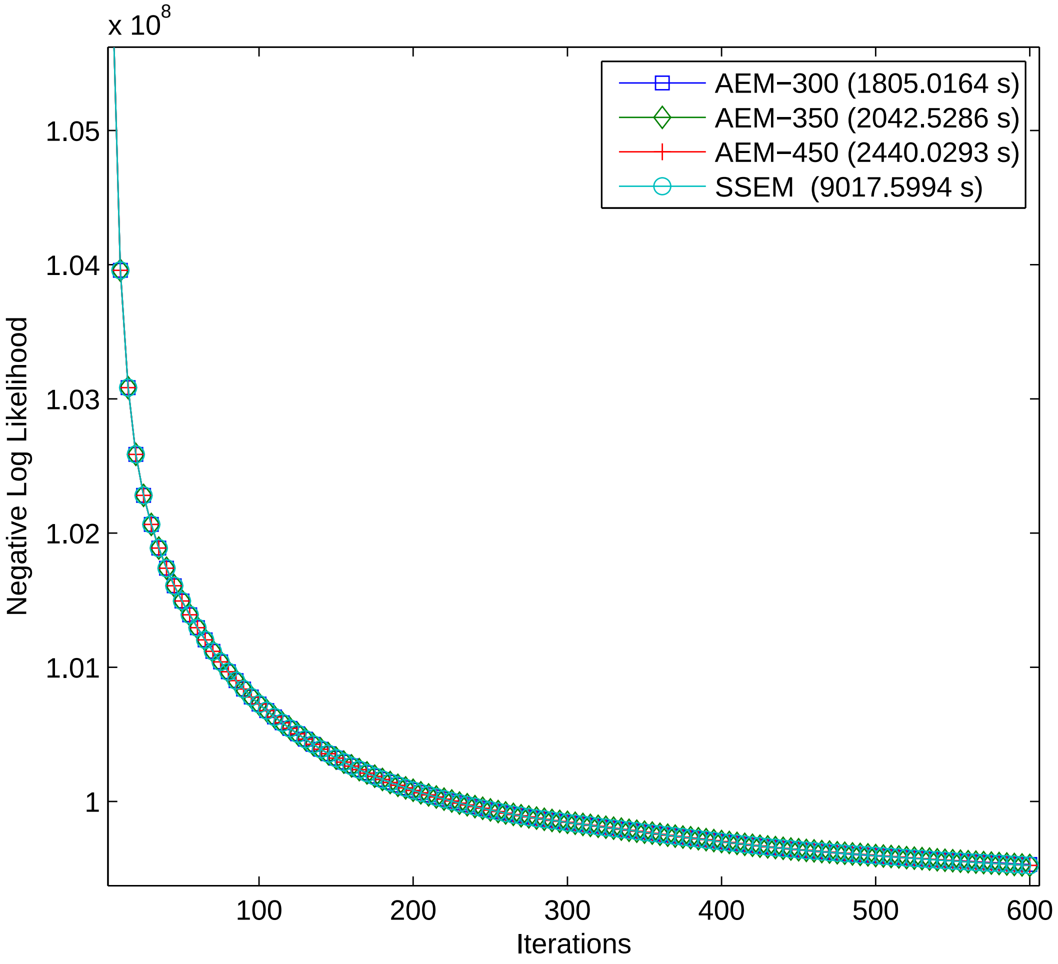

Figure 3.5: Results from the SACLANT dataset with $N_x = 150$. Standard EM is not included because it was infeasible with this dataset and the performance metrics (LL and RMS) were computed using the steady-state approximation for the same reason. Note that while graph seems to suggest using a smaller value of k_{lim} to achieve a better run-time, we found that doing so caused indefinite estimates of the covariance matrix Q.

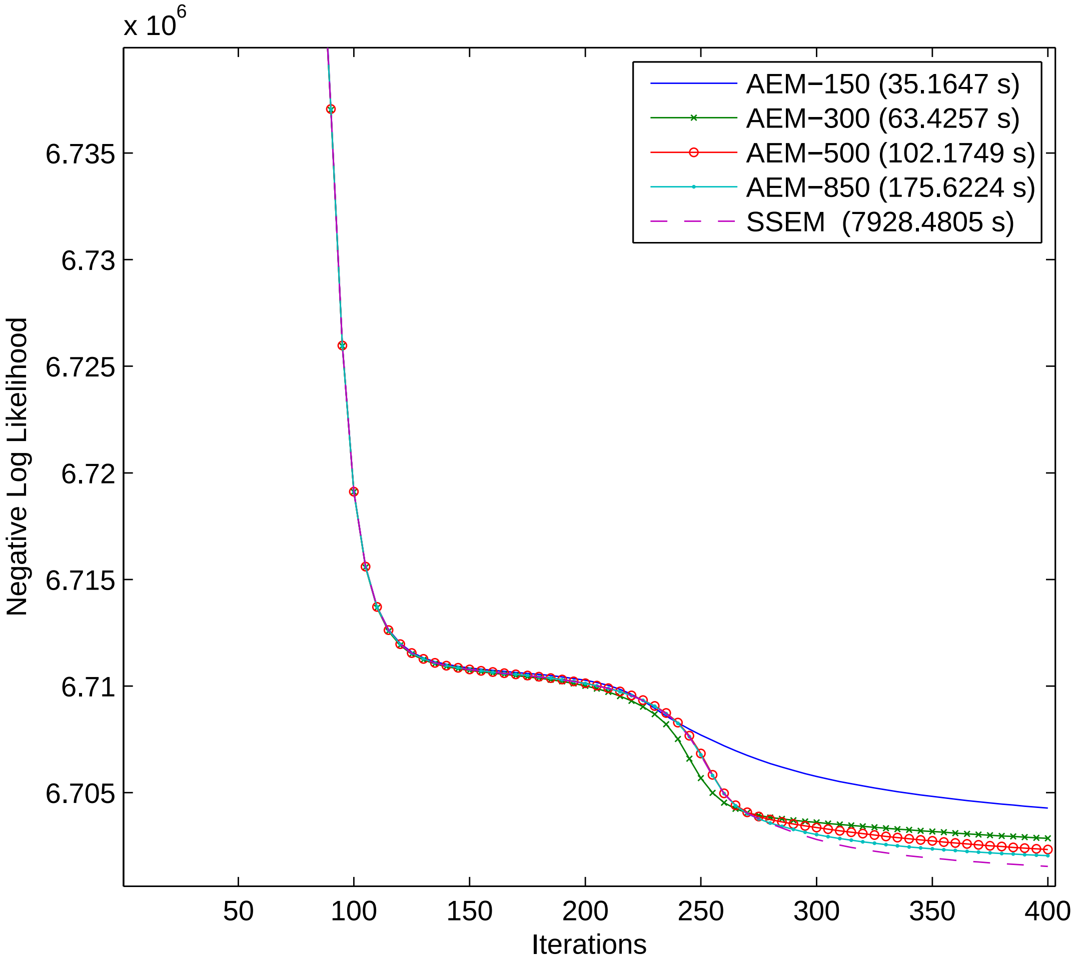

Figure 3.6: Results from the destroyer operations room audio dataset with $N_x = 20$. Standard EM is not included because it was infeasible with this dataset and the performance metrics (LL and RMS) were computed using the steady-state approximation for the same reason. Note that the run-time difference between SSEM and AEM was most dramatic here due to the extreme length of the dataset (which was a truncated version of a much longer dataset).

3.3 Discussion of Results

Our experiments indicate that SSEM and EM behave nearly identically (except for the former being a lot faster in implementation) and that AEM tends converge to parameter values with a slightly worse log-likelihood than those found by EM. However, as the value of k_{lim} is raised, the point at which AEM converges seems to approach that of SSEM and EM. Moreover, depending on the value of T, the k_{lim} value at which this occurs (to within a reasonable degree of precision) can yield an algorithm with a significantly better runtime than SSEM. We included the runtimes in our results only as a demonstration that AEM can achieve real performance improvements in a reasonable implementational setting (carefully vectorized MATLAB code) with commonly used datasets. Whether or not the reader accepts them as reasonable indicators of relative performance, the fact remains that AEM is asymptotically faster than either SSEM and EM per iteration since its iteration cost is independent of T (as discussed below).

Another conclusion from these experiments is that the steady-state approximation is indeed very good and seems to have little impact on the quality of inference. It is also apparent from the results for the first two datasets that AEM cannot guarantee a decrease in the log likelihood for each iteration. Most likely this is because AEM is implicitly optimizing an approximate objective function whose direction of increase doesn't necessarily correspond to that of the actual log likelihood, especially when k_{lim} is set too low.

Note that the value of k_{lim} needs to be high enough so that the model parameters are adequately specified by the matrix quantities pre-computed in the initialization phase. For example, if $k_{lim} = 5$, $N_x = 5$, $N_y = 1$ and $N_u = 0$ then the algorithm would have to learn the parameters (which include the A matrix with its 25 degrees of freedom) from only about 5 to 15 distinct real numbers. If k_{lim} wasn't set high enough to restrict the parameters, we found that the covariance matrices (R and Q) estimated in the M-step would often become indefinite after a few iterations. Additionally, the experiments demonstrate that if a parameter estimate of sufficient quality is to be obtained, the value of k_{lim} should be set higher than what is required to merely constrain the parameter values. But critically, while it depends on the values of N_x and N_y and the qualitative nature of the data, the appropriate value of k_{lim} does not seem to have any dependence on the length (T) of the data.

Chapter 4

Conclusions and Future Work

Our AEM algorithm is the first iterative algorithm that we know of for learning an LDS where the computational cost for each iteration does not depend on the length of the time-series used for training. Thus, with an unlimited amount of data our algorithm can exhibit an arbitrarily large performance advantage over other iterative algorithms such as standard EM.

The experiments discussed in the previous sections only hint at the potential of our approach since the datasets they use are only modestly long. Most datasets that are available for evaluating the performance of LDS learning algorithms (such as those in DaISy) are relatively short since length is usually an uninteresting factor when comparing algorithms. Our algorithm allows for the possibility of using extremely long datasets for practical learning applications.

The cost of pre-computing the statistics over y and u in the initialization phase of our algorithm is comparable to a single iteration of steady-state EM and is much less expensive than an iteration of classical EM. Naively it is $O(Tk_{lim}max\{N_y, N_u\})^2)$ but more careful implementations that use FFTs and the Circular Convolution Theorem can achieve $O(T\log_2 Tmax\{N_y, N_u\}^2)$ (with a very small constant factor thanks to highly optimized FFT packages). Most importantly, the cost of this pre-computation is independent of the number of iterations. And while it is hard to exactly quantify, the number of iterations is justifiably considered non-constant and dominant in asymptotic analysis due to EM's slow convergence. Furthermore, these pre-computations could be performed on-line as the data is collected, eliminating the need to ever store the entire dataset.

One possible extension of our algorithm is into the on-line learning setting in which new training data is continually being received and a learning algorithm must be able to process and then discard this data so that its computational requirements do not grow without bound. On-line learning algorithms are important for robotics, noise cancellation and other applications where learning must be performed in real-time as data is received. There are on-line versions of EM that are typically applied to non-temporal models (i.e. not LDSs) that work by storing and updating an approximate posterior over the model parameters, and incorporating (via Bayes' rule) the likelihood of each observation as it is processed. Thus previously observed data are only remembered through their contributions to the approximate posterior. Fortunately it is relatively simple to extend our approach into the on-line learning setting, eliminating the need to employ such a crude approximation to the posterior. This is because our approach only requires second order statistics of bounded time-lag over y and u (which can easily be computed online) and not the individual values of y_t and u_t for each t.

Another possible extension of our approach is to non-linear models/inference-schemes such as the Extended Kalman filter (EKF). The EKF is similar to the LDS but assumes a non-linear transfer function between hidden states, allowing them to be linearized (via a Taylor series expansion) around the current state estimates during inference and learning. An extension of AEM into to this setting would require a significant modification to our approach since a key assumption, that the covariance of each state does not depend on its mean during inference, would be violated. A second possible non-linear model where AEM could be employed would be a hierarchical version of the LDS where a non-linearity is applied to the output of each layer, which then acts as the input or control signal to the layer below. Since the optimization of the parameters of each layer would be equivalent to learning a standard LDS, our approach could be naturally applied in this setting.

While this report gave a complete derivation of the algorithm, many obvious theoretical questions were left unanswered such as:

- How can the expected error induced by the approximations be quantified?
- When are the approximated statistics computed in the E-step guaranteed to give well-defined parameter estimates in the M-step? For example, when is the estimate of Q guaranteed to be symmetric? Positive definite?
- Can it be mathematically proven that for sufficiently large values of k_{lim} our algorithm will produce identical results to those of SSEM? The answer, which we won't justify here, is yes, as long as $\rho(H)^{k_{lim}} \approx 0$ holds at every iteration there is such a provable guarantee. But in practice, even when this condition doesn't hold, the algorithm can perform very well.

These and other theoretical questions are good candidates for further investigations in future work.

In our development of the SSEM algorithm we applied the steady-state approximation to the Kalman recursions at each time-step. But since the steady-state assumptions are significantly violated near the beginning and end of the sequence it might be worthwhile to develop a hybrid algorithm that applies the standard Kalman recursions (computing full covariance information) at the beginning and end of the sequence, switching automatically to the steady-state version after some pre-specified number of time-steps (or once the covariance matrices converge close enough to their steady-state values). Moreover it would be a relatively simple task to adapt our constant-time approximation technique to this hybrid algorithm.

EM, as applied to general statistical models, is a well studied algorithm. There are many well-known techniques for accelerating EM (e.g. Jamshidian and Jennrich, 1997) that might become useful to employ in a direct comparison between EM (and by extension AEM) and other algorithms for learning the LDS such as 4SID and PEM.

Finally, it may be beneficial to augment our algorithm so that it automatically determines a good value for the meta-parameter k_{lim} and/or adjusts k_{lim} dynamically as required. Indeed our experiments demonstrate that with a lower value of k_{lim} our approximate EM algorithm will only diverge from SSEM during the latter stages of learning.

Bibliography

B.D.O. Anderson and J.B. Moore. *Optimal Filtering.* Prentice-Hall, 1979.

A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

I. Fukumori, J. Benveniste, C. Wunsch, and D.B. Haidvogel. Maximum likelihood from incomplete data via the em algorithm. *Journal of Physical Oceanography*, page 18311855, 1993.

Z. Ghahramani and G.E. Hinton. Parameter estimation for linear dynamical systems. Technical report, 1996.

G.H. Golub, S. Nash, and C. Van Loan. A hessenberg-schur method for the problem AX + XB = C. Technical report, Ithaca, NY, USA, 1978.

G.C. Goodwin and K.S. Sin. *Adaptive Filtering Prediction and Control.* Prentice-Hall, 1984.

M. Jamshidian and R.I. Jennrich. Acceleration of the em algorithm by using quasi-newton methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, 59(3):569–587, 1997.

P. Van Overschee and B. De Moor. Subspace algorithms for the stochastic identification problem. pages 1321–1326 vol.2, Dec 1991.

R.H. Shumway and D.S. Stoffer. An approach to time series smoothing and forecasting using the em algorithm. *Journal of Time Series Analysis*, 3:253–264, 1982.

G. Smith, J.F. De Freitas, M. Niranjan, and T. Robinson. Speech modelling using subspace and em techniques. In *In Advances in Neural Information Processing Systems 12*, pages 796–802, 1999.

G.A. Smith and A.J. Robinson. A comparison between the em and subspace identification algorithms for time-invariant linear dynamical systems. Technical report, Cambridge University, 2000.

G.W. Taylor, G.E. Hinton, and S. Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*. MIT Press, 2007.

Y. Zhu, P. Van Overschee, B. De Moor, and L. Ljung. Comparison of three classes of identification methods. In *Proc. of SYSID '94*, volume 1, pages 175–180, 1994.

Appendix A

Details of the Derivation of EM

A.1 Computing the EM Objective Function

The EM objective function is:

$$\begin{aligned}\mathcal{Q}_n(\theta) &= E_{\theta_n}[\log p(x,y|\theta)|y] \\ &= E_{\theta_n}\left[\sum_{t=1}^{T}\log p(y_t|x_t,\theta)+\sum_{t=1}^{T-1}\log p(x_{t+1}|x_t,\theta)+\log p(x_1|\theta) \;\mid\; y\right]\end{aligned}$$

Recalling that $p(y_t|x_t,\theta) = n(y_t; Cx_t + Du_t, R)$, we have:

$$\sum_{t=1}^{T}\log p(y_t|x_t) = -\frac{T}{2}(\log|R| + N_y \log 2\pi) - \frac{1}{2}\sum_{t=1}^{T}(y_t - Cx_t - Du_t)'R^{-1}(y_t - Cx_t - Du_t)$$

where the large summation term can be written as

$$\begin{aligned}&\operatorname{tr}\left[R^{-1}\sum_{t=1}^{T}(y_t - Cx_t - Du_t)(y_t - Cx_t - Du_t)'\right] \\ &= \operatorname{tr}[R^{-1}\sum_{t=1}^{T} y_t y_t' - y_t x_t' C' - Cx_t y_t' + Cx_t x_t' C' - y_t u_t' D' - Du_t y_t' + Du_t x_t' C' \\ &\qquad + Cx_t u_t' D' + Du_t u_t' D'] \\ &= \operatorname{tr}[R^{-1}((y,y)_0 - (y,x)_0 C' - C(x,y)_0 + C(x,x)_0 C' - (y,u)_0 D' - D(u,y)_0 + D(u,x)_0 C' \\ &\qquad + C(x,u)_0 D' + D(u,u)_0 D')]\end{aligned}$$

Recalling that $p(x_{t+1}|x_t, \theta) = n(x_{t+1}; Ax_t + Bu_{t+1}, Q)$, we have:

$$\sum_{t=1}^{T-1} \log p(x_{t+1}|x_t) = -\frac{T}{2}(\log|Q| + N_x \log 2\pi) - \frac{1}{2}\sum_{t=1}^{T-1}(x_{t+1} - Ax_t - Bu_t)'Q^{-1}(x_{t+1} - Ax_t - Bu_t)$$

where the large summation term can be written as:

$$\begin{aligned}
&\mathrm{tr}[Q^{-1}((x,x)_0 - x_1{x_1}' - (x,x)_1 A' - A(x,x)_1' + A((x,x)_0 - x_T x_T')A' - (x,u)_1 B' \\
&\quad - B(x,u)_1' + A((x,u)_0 - x_T u_T')B' + B((u,x)_0 - u_T x_T')A' + B((u,u)_0 - u_T u_T')B')]
\end{aligned}$$

Recalling that $p(x_1|\theta) = n(x_1; \pi_1, \Pi_1)$, we have:

$$\begin{aligned}
\log p(x_1) &= -\frac{1}{2}(\log|\Pi_1| + N_x \log 2\pi) - \frac{1}{2}(x_1 - \pi_1)'\Pi_1^{-1}(x_1 - \pi_1) \\
&= -\frac{1}{2}(\log|\Pi_1| + N_x \log 2\pi) - \frac{1}{2}\mathrm{tr}\left[\Pi_1^{-1}\left(x_1{x_1}' - \pi_1{x_1}' - x_1{\pi_1}' + \pi_1{\pi_1}'\right)\right]
\end{aligned}$$

Taking the conditional expectation of the three expressions derived above (and noting that it respects matrix multiplication and trace) we have that the objective function is:

$$\begin{aligned}
\mathcal{Q}_n(\theta) = -\frac{1}{2}\{\ &\mathrm{tr}[R^{-1}((y,y)_0 - E_{\theta_n}[(y,x)_0 \mid y,u]C' - CE_{\theta_n}[(x,y)_0 \mid y,u] + CE_{\theta_n}[(x,x)_0 \mid y,u]C' \\
&\quad - (y,u)_0 D' - D(u,y)_0 + DE_{\theta_n}[(u,x)_0 \mid y,u]C' + CE_{\theta_n}[(x,u)_0 \mid y,u]D' + D(u,u)_0 D')] \\
&+ \mathrm{tr}[Q^{-1}(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_1{x_1}' \mid y,u] - E_{\theta_n}[(x,x)_1 \mid y,u]A' - AE_{\theta_n}[(x,x)_1' \mid y,u] \\
&\quad + A(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_T{x_T}' \mid y,u])A' - E_{\theta_n}[(x,u)_1 \mid y,u]B' - BE_{\theta_n}[(x,u)_1' \mid y,u] \\
&\quad + A(E_{\theta_n}[(x,u)_0 \mid y,u] - E_{\theta_n}[x_T \mid y,u]u_T')B' + B(E_{\theta_n}[(u,x)_0 \mid y,u] - u_T E_{\theta_n}[x_T' \mid y,u])A' \\
&\quad + B((u,u)_0 - u_T{u_T}')B')] \\
&+ \mathrm{tr}\left[\Pi_1^{-1}\left(E_{\theta_n}[x_1{x_1}' \mid y,u] - \pi_1 E_{\theta_n}[{x_1}' \mid y,u] - E_{\theta_n}[x_1 \mid y,u]{\pi_1}' + \pi_1{\pi_1}'\right)\right] \\
&+ (T-1)\left(\log|Q| + N_x \log 2\pi\right) + T\left(\log|R| + N_y \log 2\pi\right) + \log|\Pi_1| + N_x \log 2\pi\ \}
\end{aligned}$$

A.2 Derivation of the M-step

To compute the M-step we will differentiate the objective function with respect to each parameter, set the result to zero, and solve the resulting systems.

The partial derivative w.r.t. D is:

$$\frac{\partial \mathcal{Q}_n(\theta)}{\partial D} = -R^{-1}\left(-(y,u)_0 + D(u,u)_0 + CE_{\theta_n}[(x,u)_0 \mid y,u]\right)$$

Setting this to zero and pre-multiplying by $-R$ we have:

$$-(y,u)_0 + D(u,u)_0 + CE_{\theta_n}[(x,u)_0 \mid y,u] = 0 \tag{A.1}$$

Solving for D we obtain:

$$D = \left((y,u)_0 - CE_{\theta_n}[(x,u)_0 \mid y,u]\right)(u,u)_0^{-1}$$

The partial derivative w.r.t. C is:

$$\frac{\partial \mathcal{Q}_n(\theta)}{\partial C} = -R^{-1}\left(-E_{\theta_n}[(y,x)_0 \mid y,u] + CE_{\theta_n}[(x,x)_0 \mid y,u] + DE_{\theta_n}[(u,x)_0 \mid y,u]\right)$$

Setting this to zero and pre-multiplying by $-Q$ we obtain:

$$-E_{\theta_n}[(y,x)_0 \mid y,u] + CE_{\theta_n}[(x,x)_0 \mid y,u] + DE_{\theta_n}[(u,x)_0 \mid y,u] = 0 \tag{A.2}$$

Then substituting the previous solution for D (which depends on C) gives us:

$$\begin{aligned}-E_{\theta_n}&[(y,x)_0 \mid y,u] + CE_{\theta_n}[(x,x)_0 \mid y,u]\\ &+ \left((y,u)_0 - CE_{\theta_n}[(x,u)_0 \mid y,u]\right)(u,u)_0^{-1}E_{\theta_n}[(u,x)_0 \mid y,u] = 0\end{aligned}$$

Solving this for C gives:

$$\begin{aligned}C = &\left(E_{\theta_n}[(y,x)_0 \mid y,u] - (y,u)_0(u,u)_0^{-1}E_{\theta_n}[(u,x)_0 \mid y,u]\right)\\ &\cdot\left(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[(x,u)_0 \mid y,u](u,u)_0^{-1}E_{\theta_n}[(u,x)_0 \mid y,u]\right)^{-1}\end{aligned}$$

Taking the partial derivative w.r.t. R^{-1} and using (A.1) post-multiplied by D' and (A.2) post-multiplied by C' to simplify the expression we obtain:

$$\frac{\partial \mathcal{Q}_n(\theta)}{\partial R^{-1}} = -\frac{1}{2}((y,y)_0 - CE_{\theta_n}[(x,y)_0 \mid y,u] - D(u,y)_0 - TR)$$

Setting this to zero and solving for R we obtain:

$$R = \frac{1}{T}\left((y,y)_0 - CE_{\theta_n}[(x,y)_0 \mid y,u] - D(u,y)_0\right)$$

The partial derivative w.r.t. B is:

$$\begin{aligned}\frac{\partial \mathcal{Q}_n(\theta)}{\partial B} = -Q^{-1}(-E_{\theta_n}[(x,u)_1 \mid y,u] + A(E_{\theta_n}[(x,u)_0 \mid y,u] - E_{\theta_n}[x_T \mid y,u]u'_T) \\ + B((u,u)_0 - u_T u'_T))\end{aligned}$$

Setting this to zero we have and pre-multiplying by $-Q$ we obtain:

$$-E_{\theta_n}[(x,u)_1 \mid y,u] + A(E_{\theta_n}[(x,u)_0 \mid y,u] - E_{\theta_n}[x_T \mid y,u]u'_T) + B((u,u)_0 - u_T u'_T) = 0 \tag{A.3}$$

And solving for B we obtain:

$$B = (E_{\theta_n}[(x,u)_1 \mid y,u] - A(E_{\theta_n}[(x,u)_0 \mid y,u] - E_{\theta_n}[x_T \mid y,u]u'_T))\left((u,u)_0 - u_T u'_T\right)^{-1}$$

The partial derivative w.r.t. A is:

$$\begin{aligned}\frac{\partial \mathcal{Q}_n(\theta)}{\partial A} = -Q^{-1}(-E_{\theta_n}[(x,x)_1 \mid y,u] + A(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_T x'_T \mid y,u]) \\ + B(E_{\theta_n}[(u,x)_0 \mid y,u] - u_T E_{\theta_n}[x'_T \mid y,u]))\end{aligned}$$

Setting this to zero and pre-multiplying by $-Q$ we obtain:

$$\begin{aligned}-E_{\theta_n}[(x,x)_1 \mid y,u] + A(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_T x'_T \mid y,u]) \\ + B(E_{\theta_n}[(u,x)_0 \mid y,u] - u_T E_{\theta_n}[x'_T \mid y,u]) = 0\end{aligned} \tag{A.4}$$

Then substituting the previous solution for B (which depends on A) and solving for A gives:

$$\begin{aligned}A = &(E_{\theta_n}[(x,x)_1 \mid y] - E_{\theta_n}[(x,u)_1 \mid y]M) \\ &\cdot (E_{\theta_n}[(x,x)_0 \mid y] - E_{\theta_n}[x_T x'_T \mid y] - (E_{\theta_n}[(x,u)_0 \mid y] - E_{\theta_n}[x_T \mid y]u'_T)M)^{-1}\end{aligned}$$

where $M \equiv ((u,u)_0 - u_T u'_T)^{-1}(E_{\theta_n}[(u,x)_0 \mid y] - u_T E_{\theta_n}[x'_T \mid y])$

Taking the partial derivative w.r.t. Q^{-1} and using (A.3) post-multiplied by B' and (A.4) post-multiplied by A' to simplify the expression we obtain:

$$\begin{aligned}\frac{\partial \mathcal{Q}_n(\theta)}{\partial Q^{-1}} = -\frac{1}{2}(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_1 {x_1}' \mid y,u] - AE_{\theta_n}[(x,x)_1' \mid y,u] \\ - BE_{\theta_n}[(x,u)_1' \mid y,u] - (T-1)Q)\end{aligned}$$

Setting this to zero and solving we obtain:

$$Q = \frac{1}{T-1}(E_{\theta_n}[(x,x)_0 \mid y,u] - E_{\theta_n}[x_1 x_1' \mid y,u] - AE_{\theta_n}[(x,x)_1' \mid y,u] - BE_{\theta_n}[(x,u)_1' \mid y,u])$$

The partial derivative w.r.t. π_1 is:

$$\frac{\partial \mathcal{Q}_n(\theta)}{\partial \pi_1} = -\Pi^{-1}\left(-E_{\theta_n}[x_1 \mid y,u] + \pi_1\right)$$

Setting this to zero and solving for π_1 we obtain:

$$\pi_1 = E_{\theta_n}[x_1 \mid y,u] \tag{A.5}$$

Taking the partial derivative w.r.t. Π_1^{-1} and using (A.5) is:

$$\frac{\partial \mathcal{Q}_n(\theta)}{\partial \Pi_1} = -\frac{1}{2}(E_{\theta_n}[x_1 x_1' \mid y,u] - \pi_1 \pi_1' - \Pi_1)$$

Setting this to zero and solving for Π_1 we obtain:

$$\Pi_1 = E_{\theta_n}[x_1 {x_1}' \mid y,u] - \pi_1 \pi_1'$$

Appendix B

Algorithms for Solving Various Matrix Equation

B.1 Solving the DARE

The discrete algebraic Riccati equation (DARE) is a matrix equation of the form $X = f(X)$ where,

$$f(X) \equiv E\left(X - XF'(FXF' + N)^{-1}FX\right)E' + M$$

and E, F, N and M are matrices such that $f^t(X)$ converges (reaches a steady state) for any X as $t \to \infty$.

The doubling algorithm (e.g. Anderson and Moore, 1979) works by simulating a very long sequence of repeated applications of f, starting from the zero matrix. It is able to simulate 2^t such applications in only t iterations, making it much more efficient than performing each application directly.

The algorithm is defined as the computation of a particular sequence of matrices according to the following recursions:

$$\begin{aligned}\Phi_{t+1} &= \Phi_t(I+\Psi_t\Theta_t)^{-1}\Phi_t \\ \Psi_{t+1} &= \Psi_t+\Phi_t(I+\Psi_t\Theta_t)^{-1}\Psi_t\Phi_t' \\ \Theta_{t+1} &= \Theta_t+\Phi_t'\Theta_t(I+\Psi_t\Theta_t)^{-1}\Phi_t\end{aligned}$$

with the starting points $\Phi_1 = E'$, $\Psi_1 = F'N^{-1}F$ and $\Theta_1 = M$. It can be shown (with some difficulty) that $\Theta_t = f^{2^t}(0)$.

B.2 Solving Lyapunov and Sylvester Equations

The discrete Lyapunov equation, which has the general form $X = f(X)$ where $f(X) = EXE' + M$, is a special case of the Sylvester equation, which has the form $X = f(X)$ where $f(X) = EXF + M$. The Sylvester equation can be solved efficiently by several well-known algorithms including an alternate version of the doubling algorithm defined by the recursions:

$$\Phi_{t+1} = \Phi_t^2 \qquad \Upsilon_{t+1} = \Upsilon_t^2 \qquad \Theta_{t+1} = \Theta_t + \Phi_t\Theta_t\Upsilon_t$$

with the starting points $\Phi_1 = E$, $\Upsilon_1 = F$ and $\Theta_1 = M$. As with the doubling algorithm for the DARE, $\Theta_t = f^{2^t}(0)$. This can be shown with a simple inductive argument and the observation that $f^t(0) = \sum_{i=0}^{t} E^iMF^i$. Thus if $\rho(F' \otimes E) = \rho(E)\rho(F) < 1$, Θ_t will converge to the solution of $X = f(X)$.

There are more sophisticated non-iterative algorithms for solving $X = f(X)$ that do not require $\rho(E)\rho(F) < 1$. One such algorithm is based on the reduction of E and F to Hessenberg and Schur forms (see Golub et al., 1978) and is implemented in MATLAB as the function ‘dlyap’.

Algorithm 1 Algorithm Solving (2.9)

```
Input: $A$, $C$, $K$, $H$, $G$
Initialize $X := 0$, $Y := G$.
while $Y$ has not converged to 0 do
    $Y :=$ Solution for Z of $(Z = AZH' + Y)$
    $X := X + Y$
    $Y := H^{2k_{lim}+1}Y'A'C'K'$
end while
```

B.3 Solving Equation (2.9)

Lemma 1. *Let V be a vector space, $f : V \longrightarrow V$ be a continuous linear function such that $\rho(f) < 1$. Then a solution to the equation $x = f(x) + y$ is given by:*

$$x_0 = \sum_{i=0}^{\infty} f^i(y) \tag{B.1}$$

Proof. The condition $\rho(f) < 1$ ensures that the series converges (and determines the rate of convergence).

Then,

$$x_0 = \sum_{i=0}^{\infty} f^i(y) = \sum_{i=1}^{\infty} f^i(y) + f^0(y) = \sum_{i=0}^{\infty} f \circ f^i(y) + y = f(\sum_{i=0}^{\infty} f^i(y)) + y = f(x_0) + y$$

where we have used the fact that f is both continuous and linear so that it respect the infinite sum. □

Now let $f_1(X) = X - AXH'$, $f_2(X) = H^{2k_{lim}+1}X'A'C'K'$ and $y = G$ where A, H, C, K and G are defined in the context of section 2.3.7. These functions are clearly linear and continuous. Then the solution of $f_1(X) = f_2(X) + G$ is the solution of (2.9). Taking f_1^{-1} of both sides yields $X = f_1^{-1} \circ f_2(X) + f_1^{-1}(G)$ which is the form of the equation solved in the previous lemma with $f = f_1^{-1} \circ f_2$ and $y = f_1^{-1}(G)$.

Conjecture 1. *For all $k_{lim} \geq 0$, $f_1^{-1} \circ f_2$ is a continuous linear function with $\rho(f_1^{-1} \circ f_2) <$* 1

In practice, $\rho(f_1^{-1} \circ f_2)$ will be a significantly less than 1 when k_{lim} is large enough (even when $\rho(H)$ is close to 1) which implies rapid convergence of the series defined in (B.1).

Algorithm 1 computes this series term-by-term and so by the previous lemmas and rapid converge property it is an efficient method for solving (2.9).