Computational Statistics and Data Analysis 55 (2011) 199–212

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Simulation smoothing for state–space models: A computational efficiency analysis

William J. McCausland [a,b,*], Shirley Miller [a], Denis Pelletier [c]

[a] Département de sciences économiques, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7, Canada

[b] CIREQ and CIRANO, Montréal, QC, Canada

[c] Department of Economics, Campus Box 8110, North Carolina State University, Raleigh, 27695-8110, USA

ARTICLE INFO

Article history:
Received 1 December 2008
Received in revised form 9 July 2010
Accepted 11 July 2010
Available online 22 July 2010

Keywords:
State–space models
Markov chain Monte Carlo
Importance sampling
Count data
High frequency financial data

ABSTRACT

Simulation smoothing involves drawing state variables (or innovations) in discrete time state–space models from their conditional distribution given parameters and observations. Gaussian simulation smoothing is of particular interest, not only for the direct analysis of Gaussian linear models, but also for the indirect analysis of more general models. Several methods for Gaussian simulation smoothing exist, most of which are based on the Kalman filter. Since states in Gaussian linear state–space models are Gaussian Markov random fields, it is also possible to apply the Cholesky Factor Algorithm (CFA) to draw states. This algorithm takes advantage of the band diagonal structure of the Hessian matrix of the log density to make efficient draws. We show how to exploit the special structure of state–space models to draw latent states even more efficiently. We analyse the computational efficiency of Kalman-filter-based methods, the CFA, and our new method using counts of operations and computational experiments. We show that for many important cases, our method is most efficient. Gains are particularly large for cases where the dimension of observed variables is large or where one makes repeated draws of states for the same parameter values. We apply our method to a multivariate Poisson model with time-varying intensities, which we use to analyse financial market transaction count data.

1. Introduction

State–space models are time series models featuring both latent and observed variables. The latent variables have different interpretations according to the application. They may be the unobserved states of a system in biology, economics or engineering. They may be time-varying parameters of a model. They may be factors in dynamic factor models, capturing covariances among a large set of observed variables in a parsimonious way.

Gaussian linear state–space models are interesting in their own right, but they are also useful devices for the analysis of more general state–space models. In some cases, the model becomes a Gaussian linear state–space model, or a close approximation, once we condition on certain variables. These variables may be a natural part of the model, as in Carter and Kohn (1996), or they may be convenient but artificial devices, as in Kim et al. (1998), Stroud et al. (2003) and Frühwirth-Schnatter and Wagner (2006).

In other cases, one can approximate the conditional distribution of states in a non-Gaussian or non-linear model by its counterpart in a Gaussian linear model. If the approximation is close enough, one can use the latter for importance sampling,

* Corresponding author at: Département de sciences économiques, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7, Canada. Tel.: +1 514 343 7281; fax: +1 514 343 7221.

E-mail addresses: william.j.mccausland@umontreal.ca (W.J. McCausland), shirley.miller.lira@umontreal.ca (S. Miller), denis_pelletier@ncsu.edu (D. Pelletier).

URLs: http://www.cirano.qc.ca/~mccauslw (W.J. McCausland), http://www4.ncsu.edu/~dpellet (D. Pelletier).

0167-9473/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2010.07.009

as Durbin and Koopman (1997) do to compute likelihood functions, or as a proposal distribution in a Metropolis–Hastings update, as Shephard and Pitt (1997) do for posterior Markov chain Monte Carlo simulation.

To fix notation, consider the following Gaussian linear state–space model, expressed using notation from de Jong and Shephard (1995):

$$y_t = X_t\beta + Z_t\alpha_t + G_t u_t, \quad t = 1, \ldots, n, \tag{1}$$

$$\alpha_{t+1} = W_t\beta + T_t\alpha_t + H_t u_t, \quad t = 1, \ldots, n-1, \tag{2}$$

$$\alpha_1 \sim \mathrm{N}(a_1, P_1), \qquad u_t \sim \text{i.i.d. } \mathrm{N}(0, I_q), \tag{3}$$

where y_t is a $p \times 1$ vector of dependent variables, α_t is a $m \times 1$ vector of state variables, and β is a $k \times 1$ vector of coefficients. The matrices X_t, Z_t, G_t, W_t, T_t and H_t are known. Eq. (1) is the *measurement* equation and Eq. (2) is the *state* equation. Let $y \equiv (y_1^\top, \ldots, y_n^\top)^\top$ and $\alpha \equiv (\alpha_1^\top, \ldots, \alpha_n^\top)^\top$.

We will consider the familiar and important question of simulation smoothing, which is drawing α as a block from its conditional distribution given y. This is an important component of various sampling methods for learning about the posterior distribution of states, parameters and other functions of interest.

Several authors have proposed ways of drawing states in Gaussian linear state–space models using the Kalman filter, including Carter and Kohn (1994), Frühwirth-Schnatter (1994), de Jong and Shephard (1995), and Durbin and Koopman (2002).

Rue (2001) introduces the Cholesky Factor Algorithm (CFA), an efficient way to draw Gaussian Markov Random Fields (GMRFs) based on the Cholesky decomposition of the precision (inverse of variance) of the random field. He also recognizes that the conditional distribution of α given y in Gaussian linear state–space models is a special case of a GMRF. Knorr-Held and Rue (2002) comment on the relationship between the CFA and methods based on the Kalman filter.

Chan and Jeliazkov (2009) describe two empirical applications of the CFA algorithm for Bayesian inference in state–space macroeconomic models. One is a time-varying parameter vector autoregression model for output growth, unemployment, income and inflation. The other is a dynamic factor model for US post-war macroeconomic data.

The Kalman filter is used not only for simulation smoothing, but also to evaluate the likelihood function for Gaussian linear state–space models. We can do the same using the CFA and our method. Both give evaluations of $f(\alpha|y)$ for arbitrary α with little additional computation. We can then evaluate the likelihood as

$$f(y) = \frac{f(\alpha)f(y|\alpha)}{f(\alpha|y)}$$

for any value of α. A convenient choice is the conditional mean of α given y, since it is easy to obtain and simplifies the computation of $f(\alpha|y)$.

The Kalman filter also delivers intermediate quantities that are useful for computing filtering distributions, the conditional distributions of $\alpha_1, \ldots, \alpha_t$ given $y_1, \ldots, y_t$, for various values of t. While it is difficult to use the CFA algorithm to compute these distributions efficiently, it is fairly straightforward to do so using our method.

We make four main contributions in this paper. The first is a new method, outlined in Section 2, for drawing states in state–space models. Like the CFA, it uses the precision and co-vector (precision times mean) of the conditional distribution of α given y and does not use the Kalman filter. Unlike the CFA, it generates the conditional means $E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ and conditional variances $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ as a byproduct. These conditional moments turn out to be useful in an extension of the method, described in McCausland (2008), to non-Gaussian and non-linear state–space models with univariate states. This is because it facilitates Gaussian and other approximations to the conditional distribution of α_t given α_{t+1} and y. With little additional computation, one can also compute the conditional means $E[\alpha_t|y_1, \ldots, y_t]$ and variances $\mathrm{Var}[\alpha_t|y_1, \ldots, y_t]$, which together specify the filtering distributions, useful for sequential learning.

The second main contribution, described in Section 3, is a careful analysis of the computational efficiency of various methods for drawing states, showing that the CFA and our new method are much more computationally efficient than methods based on the Kalman filter when p is large or when repeated draws of α are required. For the important special case of state–space models, our new method is up to twice as fast as the CFA for large m. We find examples of applications with large p in recent work in macroeconomics and forecasting using "data-rich" environments, where a large number of observed time series is linked to a much smaller number of latent factors. See, for example, Boivin and Giannoni (2006), which estimates Dynamic Stochastic General Equilibrium (DSGE) models, or Stock and Watson (1999, 2002) and Forni et al. (2000), which shows that factor models with large numbers of variables give better forecasts than small-scale vector autoregressive (VAR) models do. Examples with large numbers of repeated draws of α include the evaluation of the likelihood function in non-linear or non-Gaussian state–space models using importance sampling, as in Durbin and Koopman (1997).

Our third contribution is to illustrate these simulation smoothing methods using an empirical application. In Section 4, we use them to approximate the likelihood function for a multivariate Poisson state–space model, using importance sampling. Latent states govern time-varying intensities. Observed data are transaction counts in financial markets.

The final contribution is the explicit derivation, in Appendix A, of the precision and co-vector of the conditional distribution of α given y in Gaussian linear state–space models. These two objects are the inputs to the CFA and our new method.

We conclude in Section 5.

2. Precision-based methods for simulation smoothing

In this section we discuss two methods for state smoothing using the precision Ω and co-vector c of the conditional distribution of α given y. The first method is due to Rue (2001), who considers the more general problem of drawing Gaussian Markov random fields. The second method, introduced here, offers new insights and more efficient draws for the special case of Gaussian linear state–space models. Both methods involve pre-computation, which one performs once for a given Ω and c, and computation that is repeated for each draw.

We will take Ω and c as given here. In Appendix A, we show how to compute Ω and c in terms of $X_t, Z_t, G_t, W_t, T_t, H_t, a_1$ and P_1, assuming that the stacked innovation $v_t \equiv ((G_t u_t)^\top, (H_t u_t)^\top)^\top$ has full rank.

The full rank condition is frequently, but not always, satisfied and we note that de Jong and Shephard (1995) and Durbin and Koopman (2002) do not require this assumption. The full rank condition is not as restrictive as it may appear, however, for two reasons pointed out by Rue (2001).

First, we can draw α conditional on the linear equality restriction $A\alpha + b$ by drawing $\tilde{\alpha}$ unconditionally and then "conditioning by Kriging" to obtain α. This gives $\alpha = \tilde{\alpha} - \Omega^{-1}A^\top(A\Omega^{-1}A^\top)^{-1}(A\tilde{\alpha} + b)$. One can pre-compute the columns of $\Omega^{-1}A^\top$ in the same way as we compute $\mu = \Omega^{-1}c$ in Appendix B, then pre-compute $A\Omega^{-1}A^\top$ and $-\Omega^{-1}A^\top(A\Omega^{-1}A^\top)^{-1}$.

Second, state–space models where the innovation has less than full rank are often more naturally expressed in another form, one that allows application of the CFA method. Take, for example, a model where a univariate latent variable α_t is an autoregressive process of order p and the measurement equation is given by (1). Such a model can be coerced into state–space form, with a p-dimensional state vector and an innovation variance of less than full rank. However, the conditional distribution of α given y is a GMRF and one can apply the CFA method directly.

Having repeated these points, we acknowledge that the full rank condition is still quite restrictive. Conditioning by Kriging is costly when A has $O(n)$ rows, and it seems to us that simulation smoothing in autoregressive moving average (ARMA) models is impractical using precision-based methods.

Rue (2001) introduces a simple procedure for drawing Gaussian random fields. We suppose that α is multivariate normal, with a band diagonal precision matrix Ω and co-vector c. We let N be the length of α and b be the number of non-zero subdiagonals in Ω. Ω is symmetric, so its bandwidth is $2b + 1$.

Pre-computation consists of computing the Cholesky decomposition $\Omega = LL^\top$ using an algorithm that exploits the band diagonal structure of Ω and then computing $L^{-1}c$ using band back-substitution. To draw $\alpha \sim N(\Omega^{-1}c, \Omega^{-1})$, one draws $\epsilon \sim N(0, I_N)$ and then computes $\alpha = (L^\top)^{-1}([L^{-1}c] + \epsilon)$ using band back-substitution. Here and elsewhere, we use square brackets to denote previously computed quantities. The decomposition and back-substitution operations are standard in commonly used numerical computation libraries: the LAPACK routine DPBTRF computes the Cholesky decomposition of band diagonal matrices, and the BLAS routine DTBSV solves banded triangular systems of equations using band back-substitution.

Rue (2001) recognizes that the vector of states α in Gaussian linear state–space models is an example of a Gaussian Markov random field. In Appendix A, we explicitly derive Ω and c. We note that for the state–space model defined in the introduction, $N = nm$ and $b = 2m - 1$.

We now introduce another method (MMP method hereafter) for drawing α based on the precision Ω and co-vector c of its conditional distribution given y. It is based on the following result, proved in Appendix B.

Result 2.1. *If $\alpha|y \sim N(\Omega^{-1}c, \Omega^{-1})$, where Ω has the block band structure of (13), then*

$$\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y \sim \mathrm{N}(m_t - \Sigma_t\Omega_{t,t+1}\alpha_{t+1}, \Sigma_t) \quad \textit{and} \quad E[\alpha|y] = (\mu_1^\top, \ldots, \mu_n^\top)^\top,$$

where

$$\Sigma_1 = (\Omega_{11})^{-1}, \qquad m_1 = \Sigma_1 c_1,$$

$$\Sigma_t = (\Omega_{tt} - \Omega_{t-1,t}^\top\Sigma_{t-1}\Omega_{t-1,t})^{-1}, \qquad m_t = \Sigma_t(c_t - \Omega_{t-1,t}^\top m_{t-1}),$$

$$\mu_n = m_n, \qquad \mu_t = m_t - \Sigma_t\Omega_{t,t+1}\mu_{t+1}.$$

The result is related to a Levinson-like algorithm introduced by Vandebril et al. (2007). Their algorithm solves the equation $Bx = y$, where B is an $n \times n$ symmetric band diagonal matrix and y is an $n \times 1$ vector. Result 2.1 extends the results in Vandebril et al. (2007) in two ways. First, we modify the algorithm to work with $m \times m$ submatrices of a block band diagonal matrix rather than individual elements of a band diagonal matrix. Second, we use intermediate quantities computed while solving the equation $\Omega\mu = c$ for $\mu = E[\alpha|y]$ in order to compute $E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ and $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$.

Pre-computation involves iterating the following computations for $t = 1, \ldots, n$:

1. Compute the Cholesky decomposition $\Sigma_t^{-1} = \Lambda_t\Lambda_t^\top$, where $\Sigma_1^{-1} = \Omega_{11}$ and $\Sigma_t^{-1} = \Omega_{tt} - [\Omega_{t-1,t}^\top\Sigma_{t-1}\Omega_{t-1,t}]$ for $t > 1$.
2. Compute $\Lambda_t^{-1}\Omega_{t,t+1}$ using triangular back-substitution.
3. Compute the symmetric matrix $\Omega_{t,t+1}^\top\Sigma_t\Omega_{t,t+1} = [\Lambda_t^{-1}\Omega_{t,t+1}]^\top[\Lambda_t^{-1}\Omega_{t,t+1}]$.

4. Compute m_t using triangular back-substitution twice, where $m_1 = (\Lambda_1^\top)^{-1}(\Lambda_1^{-1}c_1)$ and $m_t = (\Lambda_t^\top)^{-1}(\Lambda_t^{-1}(c_t - \Omega_{t-1,t}^\top m_{t-1}))$ for $t > 1$.

To draw $\alpha \sim N(\Omega^{-1}c, \Omega^{-1})$, we proceed backwards. For $t = n, \ldots, 1$:

1. Draw $\epsilon_t \sim \mathrm{N}(0, I_m)$.
2. Compute α_t using matrix–vector multiplication and back-substitution, where $\alpha_n = m_n + (\Lambda_n^\top)^{-1}\epsilon_n$ and $\alpha_t = m_t + (\Lambda_t^\top)^{-1}(\epsilon_t - [\Lambda_t^{-1}\Omega_{t,t+1}]\alpha_{t+1})$ for $t < n$.

We consider now the problem of computing the filtering distribution at time t, the conditional distribution of α_t given $y_1, \ldots, y_t$. Since α and y are jointly multivariate Gaussian, this distribution is also Gaussian and it is enough to compute the mean $E[\alpha_t|y_1, \ldots, y_t]$ and variance $\mathrm{Var}[\alpha_t|y_1, \ldots, y_t]$. It turns out that we can do this with very little additional computation.

Fix t and consider the two cases $n = t$ and $n > t$. It is easy to see (in Appendix A) that for $\tau = 1, \ldots, t-1$, the values of c_τ, $\Omega_{\tau\tau}$ and $\Omega_{\tau,\tau+1}$ do not differ between cases. Therefore the values of m_τ and Σ_τ do not vary from case to case either. We can use Eq. (14) (for Ω_{nn}, taking $n = t$) to compute

$$\tilde{\Omega}_{tt} \equiv Z_t^\top (G_t G_t^\top)^{-1} Z_t + A_{22,t-1},$$

and Eq. (15) (for c_n, taking $n = t$) to compute

$$\tilde{c}_t \equiv Z_t^\top (G_t G_t^\top)^{-1}(y_t - X_t\beta) - A_{21,t-1}(y_{t-1} - X_{t-1}\beta) + A_{22,t-1}(W_{t-1}\beta).$$

Then

$$\mathrm{Var}[\alpha_t|y_1, \ldots, y_t] = \tilde{\Sigma}_t \equiv (\tilde{\Omega}_{tt} - \Omega_{t-1,t}^\top \Sigma_{t-1}\Omega_{t-1,t})^{-1}$$

and

$$E[\alpha_t|y_1, \ldots, y_t] = \tilde{m}_t \equiv \tilde{\Sigma}_t(\tilde{c}_t - \Omega_{t-1,t}^\top m_{t-1}).$$

3. Efficiency analysis

We compare the computational efficiency of various methods for drawing $\alpha|y$. We do this using counts of operations and computational experiments with artificial data.

3.1. Counts of operations

We consider separately the fixed computational cost of pre-computation, which is incurred only once, no matter how many draws are needed, and the marginal computational cost required for an additional draw. We do this because there are some applications, such as Bayesian analysis of state–space models using Gibbs sampling, in which only one draw is needed and other applications, such as importance sampling in non-Gaussian models, where many draws are needed.

We compute the cost of various matrix operations in terms of the number of floating point multiplications required per observation. All the methods listed in the introduction have fixed costs that are third order polynomials in p and m. The methods of Rue (2001), Durbin and Koopman (2002) and the present paper all have marginal costs that are second order polynomials in p and m. We will ignore fixed cost terms of order less than three and marginal cost terms of order less than two. The marginal costs are important only when multiple draws are required.

We take the computational cost of multiplying an $N_1 \times N_2$ matrix by an $N_2 \times N_3$ matrix as $N_1N_2N_3$ scalar floating point multiplications. If the result is symmetric or if one of the matrices is triangular, we divide by two. It is possible to multiply matrices more efficiently, but the dimensions required before realizing savings are higher than those usually encountered in state–space models. We take the cost of the Cholesky decomposition of a full $N \times N$ matrix as $N^3/6$ scalar multiplications, which is the cost using the algorithm in Press et al. (1992, p. 97). When the matrix has bandwidth $2b + 1$, the cost is $Nb^2/2$. Solving a triangular system of N equations using back-substitution requires $N^2/2$ scalar multiplications. When the triangular system has bandwidth $b + 1$, only Nb multiplications are required.

3.1.1. Fixed costs

We first consider the cost of computing the precision Ω and co-vector c, which is required for the methods of Rue (2001) and the current paper.

The cost depends on how we specify the variance of v_t, the stacked innovation. The matrices G_t and H_t are more convenient for methods using the Kalman filter, while the precision A_t is most useful for the precision-based methods. We recognize that it is often easier to specify the innovation distribution in terms of G_t and H_t rather than A_t. In most cases, however, the matrices A_t are diagonal, constant, or take on one of a small number of values, and so the additional time required to compute them is negligible.

Table 1
Scalar multiplications needed for pre-computation.

Method	Operation	Scalar multiplications
Kalman	$P_t Z_t^\top$	$m^2 p$
	$Z_t[P_t Z_t^\top]$	$mp^2/2$
	$T_t[P_t Z_t^\top]$	$m^2 p$
	$D_t = \Upsilon_t \Upsilon_t^\top$ (Cholesky)	$p^3/6$
	$[T_t P_t Z_t^\top + H_t G_t^\top](\Upsilon_t^\top)^{-1}\Upsilon_t^{-1}$	mp^2
	$K_t Z_t$	$m^2 p$
	$T_t P_t$	m^3
	$[T_t P_t] L_t^\top$	m^3
	$[H_t G_t^\top] K_t$	$m^2 p$
CFA	$\Omega = LL^\top$	$2m^3$
MMP	$(\Omega_{tt} - \Omega_{t-1,t}^\top \Sigma_{t-1}\Omega_{t-1,t}) = \Lambda_t \Lambda_t^\top$ (Cholesky)	$m^3/6$
	$\Lambda_t^{-1}\Omega_{t,t+1}$	$m^3/2$
	$\Omega_{t,t+1}^\top \Sigma_t \Omega_{t,t+1} = [\Lambda_t^{-1}\Omega_{t,t+1}]^\top[\Lambda_t^{-1}\Omega_{t,t+1}]$	$m^3/2$

There is an important case where it is in fact more natural to provide the matrices A_t. When linear Gaussian state–space models are used as approximations of non-linear or non-Gaussian state–space models, the A_t are typically based on the Hessian matrix of the log observation density of the latter. See Durbin and Koopman (1997) and Section 4 of the present paper for examples.

In general, calculation of the Ω_{tt} and $\Omega_{t,t+1}$ is computationally demanding. However, in many cases of interest, A_t, Z_t and T_t are constant, or take on one of a small number of values. In these cases, the computational burden is a constant, not depending on n. We do need to compute each c_t, but provided that the matrix expressions in parentheses in the equations following (13) can be pre-computed, this involves matrix–vector multiplications, whose costs are only second order polynomials in p and m.

We now consider the cost of the Kalman filter, which is used in most methods for simulation smoothing. The computations are as follows:

$$e_t = y_t - [X_t\beta] - Z_t a_t, \qquad D_t = Z_t P_t Z_t^\top + [G_t G_t^\top],$$

$$K_t = (T_t P_t Z_t^\top + [H_t G_t^\top]) D_t^{-1}, \qquad L_t = T_t - K_t Z_t,$$

$$a_{t+1} = [W_t\beta] + T_t a_t + K_t e_t, \qquad P_{t+1} = [T_t P_t] L_t^\top + [H_t H_t^\top] + [H_t G_t^\top] K_t.$$

As before, we use square brackets for quantities, such as $[T_t P_t]$ above, that are computed in previous steps. Here and elsewhere, we also use them for quantities such as $[H_t H_t^\top]$ that usually are constant or take values in a small pre-computable set.

Table 1 lists the matrix–matrix multiplications, Cholesky decompositions, and solutions of triangular systems required for three high level operations: an iteration of the Kalman filter, the computation of $\Omega = LL^\top$ using standard methods for band diagonal Ω, and the computation of the Σ_t and m_t of Result 2.1. All simulation smoothing methods we are aware of use one of these high level operations. We represent the solution of triangular systems using notation for the inverse of a triangular matrix, but no actual matrix inversions are performed, as this is inefficient and less numerically reliable. Table 1 also gives the number of scalar multiplications for each operation as a function of p and m. Terms of less than third order are omitted, as we ignore matrix–vector multiplications, whose costs are mere second order monomials in m and p.

There are special cases where the Kalman filter computations are less costly. In some of these, the elements of T_t and Z_t are zero or one, and certain matrix multiplications do not require any scalar multiplications. In others, certain matrices are diagonal, reducing the number of multiplications by an order.

The relative efficiency of precision-based methods compared with Kalman-filter-based methods depends on various features of the application. We see that the precision-based methods have no third order monomials involving p. For the MMP method, the coefficient of the m^3 term is $7/6$, compared with 2 for the CFA and 2 for the Kalman filter if $T_t P_t$ is a general matrix multiplication. If T_t is diagonal or composed of zeros and ones, the coefficient of m^3 drops to 1 for the Kalman filter.

3.1.2. Marginal costs

Compared with the fixed cost of pre-processing, the marginal computational cost of an additional draw from $\alpha|y$ is negligible for all four methods we consider. In particular, no matrix–matrix multiplications, matrix inversions, or Cholesky decompositions are required. However, when large numbers of these additional draws are required, this marginal cost becomes important. It is here that the precision-based methods are clearly more efficient than those based on the Kalman filter. We use the methods of Durbin and Koopman (2002) and de Jong and Shephard (1995) as benchmarks.

Using the modified simulation smoothing algorithm in Section 2.3 of Durbin and Koopman (2002) (DK hereafter), an additional draw from $\alpha|y$ requires the following computations. We define $\epsilon_t \equiv G_t u_t$ and $\eta_t \equiv H_t u_t$, and assume that

$G_t^\top H_t = 0$ and $X_t\beta = 0$, recognizing that these assumptions can be easily relaxed. The first step is forward simulation using Eqs. (6) and (7) in that article.

$$x_1 \sim N(0, P_1), \qquad v_t^+ = Z_t x_t + \epsilon_t^+ \qquad x_{t+1} = T_t x_t - K_t v_t^+ + \eta_t^+,$$

where $\epsilon_t^+ \sim N(0, \Xi_t)$ and $\eta_t^+ \sim N(0, Q_t)$. The next step is the backwards recursion of Eq. (5):

$$r_n = 0, \qquad r_{t-1} = [Z_t D_t^{-1}] v_t^+ + L_t^\top r_t,$$

and the computation of residuals in Eq. (4):

$$\hat{\eta}_t^+ = Q_t r_t.$$

A draw $\tilde{\eta}$ from the conditional distribution of η given y is given by

$$\tilde{\eta} = \hat{\eta} - \hat{\eta}^+ + \eta^+,$$

where $\hat{\eta}$ is a pre-computed vector. To construct a draw $\tilde{\alpha}$ from the conditional distribution of α given y, we use

$$\tilde{\alpha}_1 = \hat{\alpha}_1 - P_1 r_0 + x_1, \qquad \tilde{\alpha}_{t+1} = T_t \tilde{\alpha}_t + \tilde{\eta}_t,$$

where $\hat{\alpha}_1$ is pre-computed.

de Jong and Shephard (1995) (DeJS hereafter) draw $\alpha|y$ using the following steps, given in Eq. (4) of their paper. First ϵ_t is drawn from $N(0, \sigma^2 C_t)$, where the Cholesky factor of $\sigma^2 C_t$ can be pre-computed. Then r_t is computed using the backwards recursion

$$r_{t-1} = [Z_t^\top D_t^{-1} e_t] + L_t^\top r_t - [V_t^\top C_t^{-1}]\epsilon_t.$$

Next, α_{t+1} is computed as

$$\alpha_{t+1} = [W_t\beta] + T_t\alpha_t + \Omega_t r_t + \epsilon_t.$$

In the MMP approach, we draw, for each observation, a vector $\epsilon_t \sim N(0, I_m)$ and compute

$$\alpha_t = m_t + (\Lambda_t^\top)^{-1}(\epsilon_t - [\Lambda_t^{-1}\Omega_{t,t+1}]\alpha_{t+1}).$$

The matrix–vector multiplication requires m^2 multiplications and the triangular back-substitution requires $m(m-1)/2$ multiplications and m floating point divisions. We can convert the divisions into less costly multiplications if we store the reciprocals of the diagonal elements of Λ_t, obtained during the pre-computation of $\Lambda_t^{-1}\Omega_{t,t+1}$.

The band back-substitution used by Rue (2001) is quite similar to this. However, it is a little less efficient if one is using standard band back-substitution algorithms. These do not take advantage of the special structure of state–space models, for which Ω has elements equal to zero in its first $2m-1$ subdiagonals.

3.2. Computational experiments

The performance of a simulation smoothing method does not only depend on the number of floating point multiplications. In this section, we perform computational experiments with artificial data to illustrate some of the other issues involved. The experiments reveal that these other issues are important.

One issue is whether the method is coded in a high level interpreted language such as Matlab or a lower level programming language such as C. Depending on the dimension of the problem, the number and depth of loops, and the availability and efficiency of relevant functions in the interpreted language, the cost of interpreting commands may dominate or be dominated by the cost of executing commands for numerical analysis.

Processing resources are also important, particularly the availability of multiple processor cores and an optimized math library that exploits them.

We use two different state–space models and generate two different artificial data sets for each one. The first model is a regression model with time-varying regression parameters. The measurement equation is

$$y_t = x_t\beta_t + \epsilon_t, \quad \epsilon_t \sim \text{i.i.d.N}(0, \sigma_\epsilon^2),$$

where y_t is a univariate observed dependent variable, x_t is an observed m-vector of explanatory variables, and β_t is an unobserved time-varying m-vector of regression coefficients. The dynamics of β_t are given by the state equation

$$\beta_1 \sim \text{N}(a, Q_1) \quad (\beta_{t+1} - \beta_t) \sim \text{i.i.d. N}(0, Q),$$

and the ϵ_t and β_t are mutually independent. We generate two artificial data sets from the model, one with $m = 4$ and the other with $m = 8$. In both cases, $n = 1000$, $a = 0_m$, $Q_1 = I_m$, $Q = (0.001)^2(\frac{1}{2}I_m + \frac{1}{2}\iota_m\iota_m^\top)$, and $\sigma_\epsilon^2 = 0.05$. I_m is the m-dimensional identity matrix and ι_m is the m-vector with unit elements. We generate the vector of explanatory variables according to $x_{t1} = 1$ and $x_{ti} \sim N(0, 1)$ for $i = 2, \ldots, m$.

Table 2
Costs in ms, time-varying parameter model.

Algorithm	Pre-computation, $m = 4$	Draw, $m = 4$	Pre-computation, $m = 8$	Draw, $m = 8$
MMP-M	126.0	28.2	132.7	29.7
MMP-C	1.178	0.812	5.21	1.74
CFA-M	66.6	0.853	87.7	1.65
CFA-C	2.08	0.737	8.36	1.64
DeJS-M	277.6	64.7	299.1	66.6

Table 3
Costs in ms, dynamic factor model.

Algorithm	$m = 4, p = 10$	$m = 10, p = 100$
MMP-M	123.3	140.5
MMP-C	1.95	11.07
CFA-M	39.6	79.9
CFA-C	2.81	16.4
DeJS-M	416	1165

The second state–space model is a dynamic factor model of the kind used in "data-rich" environments. The observation and state equations are

$$\begin{aligned} y_t &= Z\alpha_t + u_t, \quad u_t \sim N(0, D), \\ \alpha_1 &= a + v_1, \quad v_1 \sim N(0, Q_1), \\ \alpha_{t+1} &= T\alpha_t + v_t, \quad v_t \sim N(0, Q), \end{aligned}$$

where y_t is a p-vector of observable dependent variables, α_t is a m-vector of latent factors, a is a fixed vector, Z and T are fixed coefficient matrices and D, Q_1 and Q are fixed covariance matrices, D being diagonal. The u_t and v_t are mutually independent.

For the simulations, we set the following parameter values. We draw the elements of the factor loading matrix Z independently, with $Z_{ij} \sim N(0, (0.001)^2)$ for $i = 1, \ldots, m$ and $j = 1, \ldots, p$. We set $T = 0.9I_m$. We assign the following values to the covariance matrices: $D = I_p$, $a = 0_m$, $Q_1 = I_m$, $Q = (0.2)^2(\frac{1}{2}I_m + \frac{1}{2}\iota_m\iota_m^\top)$.

We generate two artificial data sets from the dynamic factor model. For the first, we use $m = 4$ and $p = 10$, which are relatively small. For the second, we use $m = 10$ and $p = 100$, more typical of data-rich environments. For each artificial data set we perform simulation smoothing for the following methods:

DeJS-M Method of de Jong and Shephard (1995), implemented in Matlab.
CFA-M Cholesky Factor Algorithm of Rue (2001), implemented in Matlab. The matrix Ω is stored as a sparse matrix and the Cholesky decomposition exploits the sparse structure.
MMP-M Method introduced in Section 2, implemented in Matlab.
CFA-C Cholesky Factor Algorithm, implemented in C. The matrix Ω is stored as a band triangular matrix according to the convention of LAPACK. We use the LAPACK routine DPBTRF to compute the Cholesky decomposition of band diagonal matrices, and the BLAS routine DTBSV for band back-substitution.
MMP-C Method introduced in Section 2, implemented in C.

We use Matlab R2009a running on a MacBook Pro with a 2.2 GHz Intel Core Duo processor running OS X 10.6.1. We measure running times for Matlab code using the Matlab profiler, and those for C code using the XCode profiler. Results for the time-varying parameter model are shown in Table 2. For each method, we measure the time required for pre-computation and the time required for each draw. Table 3 shows results for the dynamic factor model. Here costs are the total cost of pre-computation and a single draw. We do not report the marginal cost of a draw since importance sampling is impractical for higher-dimensional models.

Although we report results only for $n = 1000$, experiments not reported here suggest that timing is very close to linear in the number of observations n. This is hardly surprising, given that the numbers of operations required for interpretation and numerical computation both grow linearly in n.

We see clearly that the cost of interpretation dominates the cost of numerical computation for low-dimensional problems. This gives a clear advantage to the CFA method, as it does not require loops over t except for the construction of the Ω_{tt} and $\Omega_{t,t+1}$ and then the sparse matrix Ω.

When MMP and CFA are coded in C, there is no longer an interpretation cost. Here, we see that the MMP method is faster than the CFA.

Even for compiled code, we see that the relative costs of CFA and MMP do not exactly correspond to the relative numbers of floating point operations. Experiments not reported here suggest that this is because it is easier to exploit multiple cores for operations on larger matrices and vectors.

4. An empirical application to count models

Durbin and Koopman (1997) show how to compute an arbitrarily accurate evaluation of the likelihood function for a semi-Gaussian state–space model in which the state evolves according to Eq. (2), but the conditional distribution of observations given states is given by a general distribution with density (or mass) function $p(y|\alpha)$. To simplify, we suppress the notation for dependence on θ, the vector of parameters.

The approach is as follows. The likelihood function $L(\theta)$ we wish to evaluate is

$$L(\theta) = p(y) = \int p(y, \alpha)\mathrm{d}\alpha = \int p(y|\alpha)p(\alpha)\mathrm{d}\alpha. \tag{4}$$

Durbin and Koopman (1997) employ importance sampling to approximate this integral. The approximating Gaussian model has the same state density $p(\alpha)$, a Gaussian measurement density $g(y|\alpha)$ and likelihood

$$L_g(\theta) = g(y) = \frac{g(y|\alpha)p(\alpha)}{g(\alpha|y)}. \tag{5}$$

Substituting $p(\alpha)$ from (5) into (4) gives

$$L(\theta) = L_g(\theta) \int \frac{p(y|\alpha)}{g(y|\alpha)} g(\alpha|y)\mathrm{d}\alpha = L_g(\theta)E_g[w(\alpha)], \tag{6}$$

where

$$w(\alpha) \equiv \frac{p(y|\alpha)}{g(y|\alpha)}.$$

One can generate a random sample $\alpha^{(1)}, \ldots, \alpha^{(N_s)}$ from the density $g(\alpha|y)$ using any of the methods for drawing states in fully Gaussian models, then compute a Monte Carlo approximation of $L(\theta)$.

The approximating state–space model has the form

$$y_t = \mu_t + Z\alpha_t + \epsilon_t, \tag{7}$$

where the ϵ_t are independent $N(0, \Xi_t)$ and independent of the state equation innovations. The Gaussian measurement density $g(y|\alpha)$ is chosen such that the Hessian (with respect to α) of $\log g(y|\alpha)$ matches the Hessian of $\log p(y|\alpha)$ at $\hat{\alpha}$, the conditional mode of α given y. Durbin and Koopman (1997) use routine Kalman filtering and smoothing to find $\hat{\alpha}$.

4.1. Modifications to the algorithm for approximating $L(\theta)$

We propose here three modifications of the Durbin and Koopman (1997) method for approximating $L(\theta)$. The modified method does not involve Kalman filtering.

First, we use the MMP algorithm to draw α from its conditional distribution given y.

Second, we compute $L_g(\theta)$ as the extreme right hand side of Eq. (5). The equation holds for any value of α; convenient choices which simplify computations include the prior mean and the posterior mean. We use the prior mean.

Finally, we calculate $\hat{\alpha}$ using Result 2.1, as described in the rest of this section. As in Durbin and Koopman (1997), the method is essentially the Newton method. The difference lies in the implementation.

We iterate the following steps until convergence.

1. Using the current value of $\hat{\alpha}$, find the precision $\bar{\bar{H}}$ and co-vector $\bar{\bar{c}}$ of a Gaussian approximation to $p(\alpha|y)$ based on a second order Taylor expansion of $\log p(\alpha) + \log p(y|\alpha)$ around the point $\hat{\alpha}$.
2. Using the current values of $\bar{\bar{H}}$ and $\bar{\bar{c}}$, compute $\hat{\alpha} = \bar{\bar{H}}^{-1}\bar{\bar{c}}$, the mean of the Gaussian approximation, using Result 2.1.

We compute the precision $\bar{\bar{H}}$ as $\bar{H} + \tilde{H}$, and the co-vector $\bar{\bar{c}}$ as $\bar{c} + \tilde{c}$, where $\bar{H}$ and $\bar{c}$ are the precision and co-vector of the marginal distribution of α (detailed formulations are provided for our example in the next section), and $\tilde{H}$ and $\tilde{c}$ are the precision and co-vector of the Gaussian density with mean $\hat{\alpha}$ and variance equal to the negative inverse of the Hessian of $\log p(y|\alpha)$ at $\hat{\alpha}$. Since $\tilde{H}$ is block diagonal and $\bar{H}$ is block band diagonal, $\bar{\bar{H}}$ is also block band diagonal.

We compute $\tilde{H}$ and $\tilde{c}$ as follows. Let $a(\alpha_t) \equiv -2\log[p(y_t|\alpha_t)]$. We approximate $a(\alpha_t)$ by $\tilde{a}(\alpha_t)$, consisting of the first three terms of the Taylor expansion of $a(\alpha_t)$ around $\hat{\alpha}_t$:

$$a(\alpha_t) \approx \tilde{a}(\alpha_t) = a(\hat{\alpha}_t) + \frac{\partial a(\hat{\alpha}_t)}{\partial \alpha_t}(\alpha_t - \hat{\alpha}_t) + \frac{1}{2}(\alpha_t - \hat{\alpha}_t)^\top \frac{\partial^2 a(\hat{\alpha}_t)}{\partial \alpha_t \partial \alpha_t^\top}(\alpha_t - \hat{\alpha}_t).$$

If we complete the square, we obtain

$$\tilde{a}(\alpha_t) = (\alpha_t - h_t^{-1}c_t)^\top h_t(\alpha_t - h_t^{-1}c_t) + k,$$

where

$$h_t = \frac{1}{2}\frac{\partial^2 a(\hat{\alpha}_t)}{\partial \alpha_t \partial \alpha_t^\top}, \qquad c_t = h_t \hat{\alpha}_t - \frac{1}{2}\frac{\partial a(\hat{\alpha}_t)}{\partial \alpha_t},$$

and k does not depend on α_t. Note that h_t and c_t are the precision and co-vector of a multivariate normal distribution with density proportional to $\exp[-\frac{1}{2}\tilde{a}(\alpha_t)]$.

Since $\log p(y|\alpha)$ is additively separable in the elements of α, it means that it is reasonably well approximated, as a function of α, by $\prod_{t=1}^{n} \exp[-\frac{1}{2}\tilde{a}(\alpha_t)]$, which is proportional to a multivariate normal distribution with precision $\tilde{H}$ and co-vector $\tilde{c}$, given by

$$\tilde{H} \equiv \begin{bmatrix} h_1 & 0 & \cdots & 0 \\ 0 & h_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_n \end{bmatrix} \quad \text{and} \quad \tilde{c} \equiv \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}.$$

4.2. A multivariate Poisson model with time-varying intensities

As an example of a semi-Gaussian state–space model, let us consider a case where $y_t \equiv (y_{t1}, \ldots, y_{tp})$ is a vector of observed counts. We assume that the y_{ti} are conditionally independent Poisson with intensities λ_{ti}, so that the conditional density of y_t given $\lambda_{t1}, \ldots, \lambda_{tp}$ is

$$p(y_{t1}, \ldots, y_{tp}|\lambda_{t1}, \ldots, \lambda_{tp}) = \prod_{i=1}^{p} \frac{\exp(-\lambda_{ti})\lambda_{ti}^{y_{ti}}}{y_{ti}!}. \tag{8}$$

The latent count intensities $\lambda_{t1}, \ldots, \lambda_{tp}$ are assumed to follow a factor model:

$$\lambda_{ti} = \exp\left(\sum_{j=1}^{m} z_{ij}\alpha_{tj}\right), \quad i = 1, \ldots, n, \tag{9}$$

$$\alpha_{t+1,j} = (1-\phi_j)\bar{\alpha}_j + \phi_j\alpha_{tj} + \eta_{tj}, \quad j = 1, \ldots, m, \tag{10}$$

where the η_{tj} are independent $N(0, Q_j)$ and the distribution of α_1 is the stationary distribution, so that the $\alpha_{1,j}$ are independent $N(\bar{\alpha}_j, Q_j/(1-\phi_j^2))$.

Denote by Q the diagonal matrix $\mathrm{diag}(Q_1, \ldots, Q_m)$. The vector of model parameters is $\theta \equiv (\bar{\alpha}_j, \phi_j, Q_j, z_{ij})_{i\in\{1,\ldots,p\},\, j\in\{1,\ldots,m\}}$. To ensure identification,[1] we impose $z_{ii} = 1$ and $z_{ij} = 0$ for $j > i$.

We now turn to the problem of estimating the likelihood $L(\theta)$ of this particular semi-Gaussian model using the approach of Durbin and Koopman (1997). For this example, the precision $\bar{H}$ and co-vector $\bar{c}$, are given by

$$\bar{H} = \begin{bmatrix} \bar{H}_{11} & \bar{H}_{12} & 0 & \cdots & 0 & 0 \\ \bar{H}_{21} & \bar{H}_{22} & \bar{H}_{23} & \cdots & 0 & 0 \\ 0 & \bar{H}_{32} & \bar{H}_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \bar{H}_{n-1,n-1} & \bar{H}_{n-1,n} \\ 0 & 0 & 0 & \cdots & \bar{H}_{n,n-1} & \bar{H}_{nn} \end{bmatrix}, \qquad \bar{c} = \begin{bmatrix} \bar{c}_1 \\ \bar{c}_2 \\ \vdots \\ \bar{c}_{n-1} \\ \bar{c}_n \end{bmatrix},$$

where

$$\bar{H}_{11} = \bar{H}_{nn} = Q^{-1},$$

$$\bar{H}_{tt} = \begin{bmatrix} (1+\phi_1^2)/Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (1+\phi_m^2)/Q_m \end{bmatrix}, \quad t = 2, \ldots, n-1,$$

$$\bar{H}_{t,t+1} = \bar{H}_{t+1,t} = \begin{bmatrix} -\phi_1/Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -\phi_m/Q_m \end{bmatrix}, \quad t = 1, \ldots, n-1,$$

[1] See for example Heaton and Solo (2004).

Table 4
Computational costs per observation per additional draw of α_t.

Algorithm	×	$N_{0,1}$
DeJS	$(3p^2+p)/2+2mp+m^2$	p
DK	$(5p^2+p)/2+4mp+2m+m^2$	$p+m$
CFA	$2m^2+pm$	m
MMP	$(3m^2+m)/2+pm$	m

Table 5
Time cost of drawing $\alpha^{(i)}$ as a function of the number of draws N_s. The values are in 100ths of seconds.

Method	$N_s=1$	$N_s=10$	$N_s=50$	$N_s=150$	$N_s=250$
DeJS	8.14	8.43	8.89	10.61	11.78
DK	5.56	6.25	7.32	11.01	13.49
MMP-M	5.88	6.12	6.28	6.88	7.47
CFA-M	1.88	1.98	2.16	2.65	3.10
MMP-C	1.11	1.18	1.34	1.90	2.30
CFA-C	1.13	1.21	1.44	2.19	2.73

$$\bar{c}_1=\bar{c}_n=\begin{bmatrix}\bar{\alpha}_1(1-\phi_1)/Q_1\\ \vdots\\ \bar{\alpha}_m(1-\phi_m)/Q_m\end{bmatrix},$$

$$\bar{c}_t=\begin{bmatrix}\bar{\alpha}_1(1-\phi_1)^2/Q_1\\ \vdots\\ \bar{\alpha}_m(1-\phi_m)^2/Q_m\end{bmatrix},\quad t=2,\ldots,n-1.$$

We compare the computational efficiency of all the three methods for estimating the likelihood for this semi-Gaussian state–space model. We do so by counting operations and profiling code. Since a large number of draws from $g(\alpha|y)$ is required for a good approximation of $L(\theta)$, we focus on the marginal computational cost of an additional draw, the overhead associated with the first draw being small. For all the four methods, we compute $\hat{\alpha}$ using the fast method presented in Section 4.1.

We have already seen how to make an incremental draw using the various methods. For both MMP and CFA, we add $p\times m$ multiplications for each of the $Z\alpha_t$, which are required to evaluate $p(y|\alpha)$. The computational costs per observation for an additional draw of α_t are summarized in Table 4.

We profile code for all the four methods to see how they perform in practice. We use data from the New York Stock Exchange Trade and Quote database on the stocks of four gold mining companies: Agnico-Eagle Mines Limited, Barrick Gold Corporation, Gold Fields Limited and Goldcorp Inc. For each stock, we observe transaction counts for 195 consecutive two minute intervals covering trading hours on November 6, 2003. The data are plotted in Fig. 1.

For the case where the number of factors is equal to the number of series, that is $m=p=4$, and for various values of N_s, Table 5 gives the time cost in 100ths of seconds of generating N_s draws of α. All times are averaged over 10,000 replications.[2] We report results for two implementations of the MMP and CFA algorithms, one Matlab only (MMP-M, CFA-M) and a second where pre-computation (Cholesky decomposition of Ω for CFA, steps 1 and 2 of MMP) and draws (band back-substitution for CFA, steps 3 and 4 of MMP) are coded in C (MMP-C, CFA-C). The implementation of MMP in C gives a better comparison with the Matlab implementation of CFA that is able to use specialized libraries to compute the banded Cholesky decomposition and perform the band back-substitution.[3]

First, we see that for a single draw, DK is slightly faster than DeJS and MMP (Matlab). For larger numbers of draws, MMP is fastest. Second, these first three methods are dominated for every value of N_s by the CFA-M algorithm. This is the result of requiring less operations (compared to DeJS and DK) and being very efficiently implemented in Matlab. There are no loops over t, which reduces interpretation costs. Third, implementing CFA in C so that it uses LAPACK and BLAS routines for banded triangular matrices and systems is computationally more efficient than Matlab's built-in functions. Fourth, we see that MMP-C is faster than CFA-C. As a point of reference, Durbin and Koopman (1997) consider $N_s=200$ (combined with antithetic and control variables) as an acceptable value in an empirical example they consider.

We next discuss the results of the estimation of this multivariate count data model. The estimates, standard errors[4] and log-likelihood values are presented in Table 6 for different values of m, the number of latent factors. These results are obtained with $N_s=500$ and antithetic variables. To select a value for m we cannot use a test statistic such as the likelihood

2 The simulations were performed on a MacBook Pro 2.4 GHz with Matlab R2008b.

3 By declaring Ω to be a sparse matrix, Matlab can use cholmod, a sparse Cholesky factorization package.

4 See Durbin and Koopman (2001, Chapter 12).

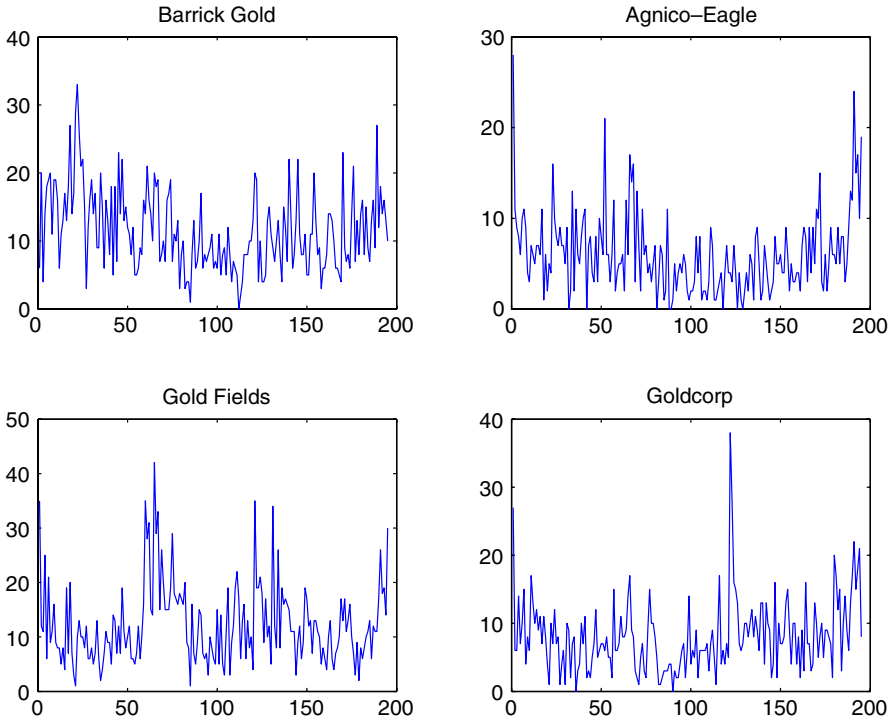

Fig. 1. Transaction data.

Table 6
Estimation results for the model for different values of m. The standard errors are in parentheses.

	$m=1$		$m=2$		$m=3$		$m=4$	
$\bar{\alpha}_1$	2.0569	(0.0101)	2.0207	(0.0050)	1.9999	(0.0140)	2.0013	(0.0049)
$\bar{\alpha}_2$			0.4022	(0.0718)	0.7311	(0.2105)	0.7004	(0.0821)
$\bar{\alpha}_3$					0.5360	(0.0218)	0.2939	(0.0184)
$\bar{\alpha}_4$							0.3858	(0.0489)
ϕ_1	0.7873	(0.0007)	0.7402	(0.0255)	0.7595	(0.0079)	0.7710	(0.0059)
ϕ_2			0.1780	(0.0549)	0.1233	(0.0353)	0.2829	(0.0144)
ϕ_3					0.0886	(0.0162)	0.0412	(0.0020)
ϕ_4							0.1182	(0.0025)
Q_1	0.1582	(0.0019)	0.2225	(0.0207)	0.2250	(0.0041)	0.2142	(0.0077)
Q_2			0.0321	(0.0030)	0.1316	(0.0628)	0.1378	(0.0133)
Q_3					0.1451	(0.0262)	0.1678	(0.0074)
Q_4							0.1405	(0.0120)
z_{21}	0.9928	(0.0064)	0.8170	(0.0358)	0.6626	(0.1000)	0.6714	(0.0388)
z_{31}	0.9965	(0.0065)	0.5830	(0.1243)	0.6449	(0.0380)	0.6738	(0.0242)
z_{41}	0.9882	(0.0065)	0.6073	(0.1192)	0.5785	(0.0148)	0.6098	(0.0359)
z_{32}			2.2228	(0.2558)	0.3012	(0.0665)	0.5203	(0.0078)
z_{42}			2.0586	(0.2555)	0.6674	(0.4069)	0.4394	(0.0247)
z_{43}					0.7747	(0.3121)	0.3856	(0.0105)
$\log L(\hat{\theta})$	−2432.71		−2378.44		−2350.67		−2324.22	

ratio test with the usual χ^2 limit distribution. For example, a likelihood ratio test for $m = 1$ versus $m = 2$ where we test $z_{32} = z_{42} = 0$ leaves the parameters $\bar{\alpha}_2$, ϕ_2 and Q_2 unidentified under the null. An alternative is to use an information criterion such as $AIC = -2\log L(\theta) + 2\dim(\theta)$ and $SIC = -2\log L(\theta) + \log(pn)\dim(\theta)$. See Song and Belin (2008) for an example. These two criteria both suggest that m should equal four. For the model with $m = 4$, we can see that the first factor is the most persistent with $\hat{\phi}_1 = 0.7710$. It is also the factor with the highest innovation variance and the highest factor loadings, the three largest z's being z_{21}, z_{31} and z_{41}.

5. Conclusions

In this paper we introduce a new method for drawing state variables in Gaussian state–space models from their conditional distribution given parameters and observations. The method is quite different from standard methods, such

as those of de Jong and Shephard (1995) and Durbin and Koopman (2002), that use Kalman filtering. It is much more in the spirit of Rue (2001), who describes an efficient method for drawing Gaussian random vectors with band diagonal precision matrices. As Rue (2001) recognizes, the distribution $\alpha|y$ in linear Gaussian state–space models is an example.

Our first contribution is computing Ω and c for a widely used and fairly flexible state–space model. These are required inputs for both the CFA of Rue (2001) and the method we described here.

Our second contribution is a new precision-based state smoothing algorithm. It is more computationally efficient for the special case of state–space models, and delivers the conditional means $E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ and conditional variances $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ as a byproduct. These conditional moments turn out to be very useful in an extension of the method, described in McCausland (2008), to non-linear and non-Gaussian state–space models with univariate states.

The algorithm is an extension of a Levinson-like algorithm introduced by Vandebril et al. (2007), for solving the equation $Bx = y$, where B is an $n \times n$ symmetric band diagonal matrix and y is a $n \times 1$ vector. The algorithm extends theirs in two ways. First, we modify the algorithm to work with $m \times m$ submatrices of a block band diagonal matrix rather than individual elements of a band diagonal matrix. Second, we use intermediate quantities computed while solving the equation $\Omega\mu = c$ for the mean μ given the precision Ω and co-vector c in order to compute the conditional means $E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$ and conditional variances $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y]$.

Our third contribution is a computational analysis of several state smoothing methods. One can often pre-compute the Ω_{tt} and $\Omega_{t,t+1}$, in which case the precision-based methods are more efficient than those based on the Kalman filter. The advantage is particularly strong when p is large or when several draws of α are required for each value of the parameters. Kalman filtering, which involves solving systems of p equations in p unknowns, requires $O(p^3)$ scalar multiplications. If the A_t can be pre-computed, or take on only a constant number of values, the precision-based methods require no operations of order higher than p^2, in p. If the Z_t and T_t can also be pre-computed, or take on only a constant number of values, the order drops to p. For large m, our method involves half as many scalar multiplications as CFA.

Illustrations with artificial data reveal that performance does not depend only on the number of floating point multiplications. Whether numerical computations are implemented in high level interpreted code or low level compiled code is important when m and p are small and, consequently, the relative burden of interpreting code in loops is high. Even when computations are performed in compiled code, operations on higher dimension vectors and matrices may be relatively more efficient if they can exploit multiple cores.

We consider an application of our methods to the evaluation of the log-likelihood function for a multivariate Poisson model with latent count intensities.

We have learned several things relevant to the choice of a simulation smoothing method for a given state–space model. It is clear that no method dominates the others in all cases, and that much depends on the details of the state–space model, its dimensions, whether the user is using a high level language such as Matlab or a low level language such as C, the number of draws required for each value of the parameters, and whether or not sequential learning is important.

The two precision-based methods are naturally suited for models with large values of p, such as those used in data-rich environments, or when one needs large numbers of repeated draws, as when one applies importance sampling for non-linear or non-Gaussian models. On the other hand, they are not well suited for state–space models such as ARMA models that cannot be expressed in a form where the variance of the stacked innovation term has full rank. They may also be less efficient than methods of de Jong and Shephard (1995) or Durbin and Koopman (2002), based on Kalman filtering, when the computation of the Ω_{tt} or $\Omega_{t,t+1}$ requires a number of operations that is third order in m and p. This is the case when Z_t or T_t or the innovation precision A_t are full matrices taking on different values at every value of t.

Of the two precision-based methods, the CFA method is best suited for low-dimensional models implemented in interpreted languages such as Matlab, provided that the language has routines for efficient Cholesky decomposition and back-substitution, either for sparse matrices or for banded matrices. The MMP method is better suited for larger dimensional models. It is for these models that the benefits of coding in a compiled language are greatest. Once the decision to use a compiled language is made, the MMP method offers further computational efficiency by avoiding multiplications by zero. The MMP method is also valuable when sequential learning is required.

Acknowledgement

Miller would like to thank the Fonds de recherche sur la société et la culture (FQRSC) for financial support.

Appendix A. Derivation of Ω and c

Here we derive expressions for the precision Ω and co-vector c of the conditional distribution of α given y, for the Gaussian linear state–space model described in Eqs. (1)–(3). The matrix Ω and vector c are required inputs for the CFA method and our new method.

Let v_t be the stacked period-t innovation:

$$v_t = \begin{bmatrix} G_t u_t \\ H_t u_t \end{bmatrix}.$$

We will assume that the variance of v_t has full rank.

We define the matrix A_t as the precision of v_t and then partition it as:

$$A_t \equiv \begin{bmatrix} G_t G_t^\top & G_t H_t^\top \\ H_t G_t^\top & H_t H_t^\top \end{bmatrix}^{-1} = \begin{bmatrix} A_{11,t} & A_{12,t} \\ A_{21,t} & A_{22,t} \end{bmatrix},$$

where $A_{11,t}$ is the leading $p \times p$ submatrix.

Clearly α and y are jointly Gaussian and therefore the conditional distribution of α given y is also Gaussian. We can write the log conditional density of α given y as

$$\log f(\alpha|y) = -\frac{1}{2}\left[\alpha^\top \Omega \alpha - 2c^\top \alpha\right] + k, \tag{11}$$

where k does not depend on α. Using the definition of the model in Eqs. (1)–(3) we can also write

$$\log f(\alpha|y) = \log f(\alpha, y) - \log f(y) = -\frac{1}{2} g(\alpha, y) + k', \tag{12}$$

where

$$\begin{aligned} g(\alpha, y) \;=\; & (\alpha_1 - a_1)^\top P_1^{-1}(\alpha_1 - a_1) + \sum_{t=1}^{n-1} \begin{bmatrix} y_t - X_t\beta - Z_t\alpha_t \\ \alpha_{t+1} - W_t\beta - T_t\alpha_t \end{bmatrix}^\top A_t \begin{bmatrix} y_t - X_t\beta - Z_t\alpha_t \\ \alpha_{t+1} - W_t\beta - T_t\alpha_t \end{bmatrix} \\ & + (y_n - X_n\beta - Z_n\alpha_n)^\top (G_n G_n^\top)^{-1}(y_n - X_n\beta - Z_n\alpha_n), \end{aligned}$$

and k' is a term not depending on α.

Matching linear and quadratic terms in the α_t between Eqs. (11) and (12), we obtain

$$\Omega \equiv \begin{bmatrix} \Omega_{11} & \Omega_{12} & 0 & \cdots & 0 \\ \Omega_{12}^\top & \Omega_{22} & \Omega_{23} & \ddots & \vdots \\ 0 & \Omega_{23}^\top & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \Omega_{n-1,n-1} & \Omega_{n-1,n} \\ 0 & \cdots & 0 & \Omega_{n-1,n}^\top & \Omega_{nn} \end{bmatrix} \qquad c \equiv \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \tag{13}$$

where

$$\begin{aligned} \Omega_{11} &\equiv Z_1^\top A_{11,1} Z_1 + Z_1^\top A_{12,1} T_1 + T_1^\top A_{21,1} Z_1 + T_1^\top A_{22,1} T_1 + P_1^{-1}, \\ \Omega_{tt} &\equiv Z_t^\top A_{11,t} Z_t + Z_t^\top A_{12,t} T_t + T_t^\top A_{21,t} Z_t + T_t^\top A_{22,t} T_t + A_{22,t-1}, \quad t = 2, \ldots, n-1, \\ \Omega_{nn} &\equiv Z_n^\top (G_n G_n^\top)^{-1} Z_n + A_{22,n-1}, \\ \Omega_{t,t+1} &\equiv -Z_t^\top A_{12,t} - T_t^\top A_{22,t}, \quad t = 1, \ldots, n-1, \end{aligned} \tag{14}$$

$$\begin{aligned} c_1 &\equiv (Z_1^\top A_{11,1} + T_1^\top A_{21,1})(y_1 - X_1\beta) - (Z_1^\top A_{12,1} + T_1^\top A_{22,1})(W_1\beta) + P_1^{-1} a_1, \\ c_t \;&\equiv\; (Z_t^\top A_{11,t} + T_t^\top A_{21,t})(y_t - X_t\beta) - (Z_t^\top A_{12,t} + T_t^\top A_{22,t})(W_t\beta) \\ &\quad - A_{21,t-1}(y_{t-1} - X_{t-1}\beta) + A_{22,t-1}(W_{t-1}\beta), \quad t = 2, \ldots, n-1, \end{aligned}$$

$$c_n \equiv Z_n^\top (G_n G_n^\top)^{-1}(y_n - X_n\beta) - A_{21,n-1}(y_{n-1} - X_{n-1}\beta) + A_{22,n-1}(W_{n-1}\beta). \tag{15}$$

Appendix B. Proof of Result 2.1

Suppose $\alpha|y \sim N(\Omega^{-1}c, \Omega^{-1})$ and define

$$\Sigma_1 = \Omega_{11}^{-1}, \qquad m_1 = \Sigma_1 c_1,$$

$$\Sigma_t = (\Omega_{tt} - \Omega_{t-1,t}^\top \Sigma_{t-1} \Omega_{t-1,t})^{-1}, \qquad m_t = \Sigma_t (c_t - \Omega_{t-1,t}^\top m_{t-1}).$$

Now let $\mu_n \equiv m_n$ and for $t = n-1, \ldots, 1$, let $\mu_t = m_t - \Sigma_t \Omega_{t,t+1} \mu_{t+1}$. Let $\mu = (\mu_1^\top, \ldots, \mu_n^\top)^\top$.

We first show that $\Omega\mu = c$, which means that $\mu = E[\alpha|y]$:

$$\begin{aligned} \Omega_{11}\mu_1 + \Omega_{12}\mu_2 &= \Omega_{11}(m_1 - \Sigma_1\Omega_{12}\mu_2) + \Omega_{12}\mu_2 \\ &= \Omega_{11}(\Omega_{11}^{-1}c_1 - \Omega_{11}^{-1}\Omega_{12}\mu_2) + \Omega_{12}\mu_2 = c_1. \end{aligned}$$

For $t = 2, \ldots, n-1$,

$$\begin{aligned}
\Omega_{t-1,t}^{\top}\mu_{t-1} + \Omega_{tt}\mu_t + \Omega_{t,t+1}\mu_{t+1} &= \Omega_{t-1,t}^{\top}(m_{t-1} - \Sigma_{t-1}\Omega_{t-1,t}\mu_t) + \Omega_{tt}\mu_t + \Omega_{t,t+1}\mu_{t+1} \\
&= \Omega_{t-1,t}^{\top}m_{t-1} + (\Omega_{tt} - \Omega_{t-1,t}^{\top}\Sigma_{t-1}\Omega_{t-1,t})\mu_t + \Omega_{t,t+1}\mu_{t+1} \\
&= \Omega_{t-1,t}^{\top}m_{t-1} + \Sigma_t^{-1}\mu_t + \Omega_{t,t+1}\mu_{t+1} \\
&= \Omega_{t-1,t}^{\top}m_{t-1} + \Sigma_t^{-1}(m_t - \Sigma_t\Omega_{t,t+1}\mu_{t+1}) + \Omega_{t,t+1}\mu_{t+1} \\
&= \Omega_{t-1,t}^{\top}m_{t-1} + (c_t - \Omega_{t-1,t}^{\top}m_{t-1}) = c_t.
\end{aligned}$$

$$\begin{aligned}
\Omega_{n,n-1}\mu_{n-1} + \Omega_{nn}\mu_n &= \Omega_{n,n-1}(m_{n-1} - \Sigma_{n-1}\Omega_{n-1,n}\mu_n) + \Omega_{nn}\mu_n \\
&= \Omega_{n,n-1}m_{n-1} + \Sigma_n^{-1}\mu_n \\
&= \Omega_{n,n-1}m_{n-1} + \Sigma_n^{-1}m_n \\
&= \Omega_{n,n-1}m_{n-1} + (c_n - \Omega_{n,n-1})m_{n-1} = c_n.
\end{aligned}$$

We will now prove that $E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y] = m_t - \Sigma_t\Omega_{t,t+1}\alpha_{t+1}$ and that $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y] = \Sigma_t$. We begin with the standard result

$$\alpha_{1:t}|\alpha_{t+1:n}, y \sim \mathrm{N}\left(\mu_{1:t} - \Omega_{1:t,1:t}^{-1}\Omega_{1:t,t+1:n}(\alpha_{t+1:n} - \mu_{t+1:n}),\, \Omega_{1:t,1:t}^{-1}\right),$$

where μ, α and Ω are partitioned as

$$\mu = \begin{bmatrix} \mu_{1:t} \\ \mu_{t+1:n} \end{bmatrix}, \qquad \alpha = \begin{bmatrix} \alpha_{1:t} \\ \alpha_{t+1:n} \end{bmatrix}, \qquad \Omega = \begin{bmatrix} \Omega_{1:t,1:t} & \Omega_{1:t,t+1:n} \\ \Omega_{t+1:n,1:t} & \Omega_{t+1:n,t+1:n} \end{bmatrix},$$

with $\mu_{1:t}$, $\alpha_{1:t}$ and $\Omega_{1:t,1:t}$ having dimensions $tm \times 1$, $tm \times 1$, and $tm \times tm$, respectively. Note that the only non-zero elements of $\Omega_{1:t,t+1:n}$ come from $\Omega_{t,t+1}$. We can therefore write the univariate conditional distribution $\alpha_t|\alpha_{t+1:n}$ as

$$\alpha_t|\alpha_{t+1:n} \sim \mathrm{N}(\mu_t - (\Omega_{1:t,1:t}^{-1})_{tt}\Omega_{t,t+1}(\alpha_{t+1} - \mu_{t+1}),\, (\Omega_{1:t,1:t}^{-1})_{tt}).$$

The following inductive proof establishes the result $\mathrm{Var}[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y] = \Sigma_t$:

$$\begin{aligned}
(\Omega_{11})^{-1} &= \Sigma_1 \\
(\Omega_{1:t,1:t}^{-1})_{tt} &= (\Omega_{tt} - \Omega_{t,1:t-1}\Omega_{1:t-1,1:t-1}^{-1}\Omega_{1:t-1,t})^{-1} \\
&= (\Omega_{tt} - \Omega_{t-1,t}^{\top}\Sigma_{t-1}\Omega_{t-1,t})^{-1} = \Sigma_t.
\end{aligned}$$

As for the conditional mean,

$$E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y] = \begin{cases} \mu_t - \Sigma_t\Omega_{t,t+1}(\alpha_{t+1} - \mu_{t+1}) & t = 1, \ldots, n-1 \\ \mu_n & t = n. \end{cases}$$

By the definition of μ_t, $m_t = \mu_t + \Sigma_t\Omega_{t,t+1}\mu_{t+1}$, so we obtain

$$E[\alpha_t|\alpha_{t+1}, \ldots, \alpha_n, y] = \begin{cases} m_t - \Sigma_t\Omega_{t,t+1}\alpha_{t+1} & t = 1, \ldots, n-1 \\ m_n & t = n. \end{cases}$$

References

Boivin, J., Giannoni, M., 2006. DSGE models in a data-rich environment. Working Paper 12772, National Bureau of Economic Research.

Carter, C.K., Kohn, R., 1994. On Gibbs sampling for state space models. Biometrika 81 (3), 541–553.

Carter, C.K., Kohn, R., 1996. Markov chain Monte Carlo in conditionally Gaussian state space models. Biometrika 83 (3), 589–601.

Chan, J.C.C., Jeliazkov, I., 2009. Efficient Simulation and Integrated Likelihood Estimation in State Space Models. Working paper.

de Jong, P., Shephard, N., 1995. The simulation smoother for time series models. Biometrika 82 (1), 339–350.

Durbin, J., Koopman, S.J., 1997. Monte Carlo maximum likelihood estimation for non-Gaussian state space models. Biometrika 84 (3), 669–684.

Durbin, J., Koopman, S.J., 2001. Time Series Analysis by State Space Methods. In: Oxford Statistical Science Series, vol. 24. Oxford University Press, Oxford.

Durbin, J., Koopman, S.J., 2002. A simple and efficient simulation smoother for state space time series analysis. Biometrika 89 (3), 603–615.

Forni, M., Hallin, M., Lippi, M., Reichlin, L., 2000. The generalized dynamic factor model: identification and estimation. Review of Economics and Statistics 82 (4), 540–554.

Frühwirth-Schnatter, S., 1994. Data augmentation and dynamic linear models. Journal of Time Series Analysis 15, 183–202.

Frühwirth-Schnatter, S., Wagner, H., 2006. Auxiliary mixture sampling for parameter-driven models of time series of counts with applications to state space modelling. Biometrika 93, 827–841.

Heaton, C., Solo, V., 2004. Identification of causal factor models of stationary time series. Econometrics Journal 7 (2), 618–627.

Kim, S., Shephard, N., Chib, S., 1998. Stochastic volatility: likelihood inference and comparison with ARCH models. Review of Economic Studies 65 (3), 361–393.

Knorr-Held, L., Rue, H., 2002. On block updating in Markov random field models for disease mapping. Scandinavian Journal of Statistics 29, 597–614.

McCausland, W.J., 2008. The HESSIAN Method (Highly Efficient State Smoothing, In A Nutshell). Cahiers de recherche du Département de sciences économiques. Université de Montréal, no. 2008–03.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992. The art of scientific computing. In: Numerical Recipes in C, second ed. Cambridge University Press, Cambridge.

Rue, H., 2001. Fast sampling of Gaussian Markov random fields with applications. Journal of the Royal Statistical Society Series B 63, 325–338.

Shephard, N., Pitt, M.K., 1997. Likelihood analysis of non-Gaussian measurement time series. Biometrika 84 (3), 653–667.

Song, J., Belin, T.R., 2008. Choosing an appropriate number of factors in factor analysis with incomplete data. Computational Statistics and Data Analysis 52 (7), 3560–3569.

Stock, J.H., Watson, M.W., 1999. Forecasting inflation. Journal of Monetary Economics 44, 293–335.

Stock, J.H., Watson, M.W., 2002. Macroeconomic forecasting using diffusion indexes. Journal of Business and Economic Statistics 20, 147–162.

Stroud, J.R., Müller, P., Polson, N.G., 2003. Nonlinear state–space models with state-dependent variances. Journal of the American Statistical Association 98, 377–386.

Vandebril, R., Mastronardi, N., Van Barel, M., 2007. A Levinson-like algorithm for symmetric strongly nonsingular higher order semiseparable plus band matrices. Journal of Computational and Applied Mathematics 198 (1), 74–97.