bssm: Bayesian Inference of Non-linear/Non-Gaussian State Space Models in R

Jouni Helske and Matti Vihola
University of Jyväskylä, Department of Mathematics and Statistics, Finland

June 12, 2017

Introduction

The `R` (R Core Team 2016) package `bssm` is designed for Bayesian inference of general state space models with Gaussian state dynamics and non-Gaussian and/or non-linear observational distribution. The package aims to provide easy-to-use and efficient functions for fully Bayesian inference of common time series models such basic structural time series model (BSM) (Harvey 1989) with exogenous covariates, making it straighforward and efficient to make predictions and other inference in a Bayesian and non-Bayesian setting.

The motivation behind the `bssm` package is (Vihola, Helske, and Franks 2016) which suggests a new computationally efficient approach for Bayesian inference of state space models. The core methodology relies on Markov chain Monte Carlo (MCMC) approach with adaptive random walk Metropolis updating, using RAM algorithm (Vihola 2012). In addition to the two-step procedure based on importance sampling type correction introduced in (Vihola, Helske, and Franks 2016), pseudo-marginal MCMC based on particle filtering, optionally with delayed acceptance, is also supported, as well as Kalman filter based inference of linear-Gaussian models.

In this vignette we will first introduce the basic state space modelling framework used in `bssm`, and the relevant algorithms. We then give illustrations how to use `bssm` in practice.

State space models with linear-Gaussian dynamics

Denote a sequence of observations $(y_1, \ldots, y_T)$ as $y_{1:T}$, and sequence of latent state variables $(\alpha_1, \ldots, \alpha_T)$ as $\alpha_{1:T}$. Note that in general both the observations and the states can be multivariate, but currently the main algorithms of `bssm` support only univariate observations. A general state space model consists of two parts: observation level densities $g_t(y_t|\alpha_t)$ and latent state transition densities $\mu_t(\alpha_{t+1}|\alpha_t)$. In this vignette we focus on case where the state transitions are linear-Gaussian:

$$\alpha_{t+1} = c_t + T_t\alpha_t + R_t\eta_t,$$

where c_t is known input vector (typically omitted), and T_t and R_t are a system matrices which can depend on unknown parameters. Also, $\eta_t \sim N(0, I_k)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other. For non-linear transition equations, see vignette `growth_model`. For observation level density g_t, the `bssm` package currently supports basic stochastic volatility model and general exponential family state space models.

For exponential family models, the observation equation has a general form

$$g_t(y_t|d_t + Z_t\alpha_t, x_t'\beta, \phi, u_t),$$

where d_t is a again known input, x_t contains the exogenous covariate values at time t, with β corresponding to the regression coefficients. Parameter ϕ and the known vector u_t are distribution specific and can be omitted in some cases. Currently, following observational level distributions are supported:

- Gaussian distribution: $g_t(y_t|Z_t\alpha_t, x_t'\beta) = x_t'\beta + Z_t\alpha_t + H_t\epsilon_t$ with $\epsilon_t \sim N(0, 1)$.

- Poisson distribution: $g_t(y_t|Z_t\alpha_t, x_t'\beta, u_t) = \text{Poisson}(u_t \exp(x_t'\beta + Z_t\alpha_t))$, where u_t is the known exposure at time t.
- Binomial distribution: $g_t(y_t|Z_t\alpha_t, x_t'\beta, u_t) = \text{binomial}(u_t, \exp(x_t'\beta + Z_t\alpha_t)/(1+\exp(x_t'\beta + Z_t\alpha_t)))$, where u_t is the size and $\exp(x_t\beta + Z_t\alpha_t)/(1 + \exp(x_t'\beta + Z_t\alpha_t))$ is the probability of the success.
- Negative binomial distribution: $g_t(y_t|Z_t\alpha_t, x_t'\beta, \phi, u_t) = \text{negative binomial}(\exp(x_t'\beta + Z_t\alpha_t), \phi, u_t)$, where $u_t \exp(x_t'\beta + Z_t\alpha_t)$ is the expected value and ϕ is the dispersion parameter (u_t is again exposure term).

For stochastic volatility model, there are two possible parameterizations available. In general for we have

$$y_t = x_t'\beta + \sigma \exp(\alpha_t/2)\epsilon_t, \quad \epsilon_t \sim N(0,1),$$

and

$$\alpha_{t+1} = \mu + \rho(\alpha_t - \mu) + \sigma_\eta \eta_t,$$

with $\alpha_1 \sim N(\mu, \sigma_\eta^2/(1-\rho^2))$. For identifiability purposes we must either choose $\sigma = 1$ or $\mu = 0$. Although analytically identical, the parameterization with μ is often preferable in terms of computational efficiency.

Typically some of the model components such as β, T_t or R_t depend on unknown parameter vector θ, so $g_t(y_t|\alpha_t)$ and $\mu_t(\alpha_{t+1}|\alpha_t)$ depend implicitly on θ. Our goal is to perform Bayesian inference of the joint posterior of $\alpha_{1:T}$ and θ.

MCMC for Gaussian state space models

Given the prior $p(\theta)$, the joint posterior of θ and $\alpha_{1:T}$ is given as

$$p(\alpha_{1:T}, \theta|y_{1:T}) \propto p(\theta)p(\alpha_{1:T}, y_{1:T}|\theta) = p(\theta)p(y|\theta)p(\alpha_{1:T}|y_{1:T}, \theta)$$

where $p(y_{1:T}|\theta)$ is the marginal likelihood, and $p(\alpha_{1:T}|y_{1:T}, \theta)$ is often referred as a smoothing distribution. However, instead of targeting this joint posterior, it is more efficient to target the marginal posterior $p(\theta|y)$, and then given the sample $\{\theta^i\}_{i=1}^n$ from this marginal posterior, simulate states $\alpha_{1:T}^i$ from the smoothing distribution $p(\alpha_{1:T}|y_{1:T}, \theta^i)$ for $i = 1 \ldots, n$.

For Gaussian models given the parameters θ, the marginal likelihood $p(y_{1:T}|\theta)$ can be computed using the well known Kalman filter recursions, and there are several algorithms for simulating the states $\alpha_{1:T}$ from the smoothing distribution $p(\alpha_{1:T}|y_{1:T})$ (see for example Durbin and Koopman (2012)). Therefore we can straightforwardly (at least in principle) apply standard MCMC algoritms. In `bssm`, we use an adaptive random walk Metropolis algorithm based on RAM (Vihola 2012) where we fix the target acceptance rate beforehand. The complete adaptive MCMC algorithm of `bssm` for Gaussian models is as follows.

Given the target acceptance rate a^* (e.g. 0.234) and $\gamma \in (0, 1]$ (the default 2/3 works well in practice), at iteration i:

1. Compute the proposal $\theta' = \theta^{i-1} + S_{i-1}u^i$, where u_i is simulated from the standard d-dimensional Gaussian distribution and S_{i-1} is a lower diagonal matrix with positive diagonal elements.
2. Accept the proposal with probability $a^i := \min\{1, \frac{p(\theta')p(y_{1:T}|\theta')}{p(\theta^{i-1})p(y_{1:T}|\theta^{i-1})\}}$.
3. If the proposal θ' is accepted, set $\theta^i = \theta'$ and simulate a realization (or multiple realizations) of the states $\alpha_{1:T}$ from $p(\alpha_{1:T}|y_{1:T}, \theta^i)$ using the simulation smoothing algorithm by Durbin and Koopman (2002). Otherwise, set $\theta^i = \theta^{i-1}$ and $\alpha_{1:T}^i = \alpha_{1:T}^{i-1}$.
4. Compute (using Cholesky update or downdate algorithm) the Cholesky factor matrix S^i satisfying the equation

$$S_i S_i^T = S_{i-1}\left(I + \min\{1, di^{-\gamma}\}(a^i - a^*)\frac{u_i u_i^T}{\|u_i\|^2}\right)S_{i-1}^T.$$

If the interest is in the posterior means and variances of the states, we can replace the simulation smoothing in step 3 with standard fixed interval smoothing which gives the smoothed estimates (expected values and variances) of the states given the data and the model parameters. From these, the posterior means and variances of the states can be computed straightforwardly.

Non-Gaussian models

For non-linear/non-Gaussian models, the marginal likelihood $p(y_{1:T}|\theta)$ is typically not available in closed form. Thus we need to resort to simulation methods, which leads to pseudo-marginal MCMC algorithm (Lin, Liu, and Sloan 2000, Beaumont (2003), Andrieu and Roberts (2009)). The observational densities of our non-linear/non-Gaussian models are all twice differentiable, so we can straightforwardly use the Laplace approximation based on (Durbin and Koopman 2000). This gives us an approximating Gaussian model which has the same mode of $p(\alpha_{1:T}|y_{1:T},\theta)$ as the original model. Often this approximating Gaussian model works well as such, and thus we can use it in MCMC scheme directly, which results in an approximate Bayesian inference. We can also use the approximating model together with importance sampling or particle filtering, which produces exact Bayesian inference on $p(\alpha_{1:T},\theta|y_{1:T})$.

We will illustrate our approach using simple importance sampling. We can factor the likelihood of the non-Gaussian model as (Durbin and Koopman 2012)

$$\begin{aligned} p(y_{1:T}|\theta) &= \int g(\alpha_{1:T}, y_{1:T}|\theta)\mathrm{d}\alpha \\ &= g(y_{1:T}|\theta)E_g\left[\frac{g(y_{1:T}|\alpha_{1:T},\theta)}{\tilde{g}(y_{1:T}|\alpha_{1:T},\theta)}\right], \end{aligned}$$

where $\tilde{g}(y_{1:T}|\theta)$ is the likelihood of the Gaussian approximating model and the expectation is taken with respect to the Gaussian density $g(\alpha|y,\theta)$. Equivalently we can write

$$\begin{aligned} \log p(y_{1:T}|\theta) &= \log g(y_{1:T}|\theta) + \log E_g\left[\frac{g(y_{1:T}|\alpha_{1:T},\theta)}{\tilde{g}(y_{1:T}|\alpha_{1:T},\theta)}\right] \\ &= \log g(y_{1:T}|\theta) + \log \frac{g(y_{1:T}|\hat{\alpha}_{1:T},\theta)}{\tilde{g}(y_{1:T}|\hat{\alpha}_{1:T},\theta)} + \log E_g\left[\frac{g(y_{1:T}|\alpha,\theta)/g(y_{1:T}|\hat{\alpha}_{1:T},\theta)}{\tilde{g}(y_{1:T}|\alpha_{1:T},\theta)/\tilde{g}(y_{1:T}|\hat{\alpha}_{1:T},\theta)}\right] \\ &= \log g(y|\theta) + \log \hat{w} + \log E_g w^* \\ &\approx \log g(y|\theta) + \log \hat{w} + \log \frac{1}{N}\sum_{j=1}^{N} w_j^*, \end{aligned}$$

where $\hat{\alpha}_{1:T}$ is the conditional mode estimate obtained from the approximating Gaussian model. For approximating inference, we simply omit the term $\log \frac{1}{N}\sum_{j=1}^{N} w_j^*$.

In principle, when using the exact Bayesian inference we should simulate multiple realizations of the states $\alpha_{1:T}$ in each iteration of MCMC in order to compute $\log \frac{1}{N}\sum_{j=1}^{N} w_j^*$. Fortunately, we can use so called delayed acceptance (DA) approach (Christen and Fox 2005; Banterle et al. 2015) which speeds up the computation considerably. Instead of single acceptance step we use two-stage approach as follows.

1. Make initial acceptance of the given proposal θ' with probability $\min\left\{1, \frac{p(y_{1:T}|\theta')\hat{w}'}{p(y_{1:T}|\theta^{i-1})\hat{w}^{i-1}}\right\}$.
2. If accepted, perform the importance sampling of the states $\alpha_{1:T}$ and make the delayed acceptance with probability $\min\{1, \sum_{j=1}^{N} w_j^{*,'} / \sum_{j=1}^{N} w_j^{*,i-1}\}$.
3. If the delayed acceptance is successful, set $\theta^i = \theta'$ and sample one (or multiple) realization of the previously simulated states with weights $w_j^i, j = 1,\ldots,N$ (with replacement in case of multiple samples are stored). Otherwise, set $\theta^i = \theta^{i-1}$ and similarly for the states.

If our approximation is good, then most of the times when we accept in the first stage we also accept in seconds stage, and thus we often need to simulate the states only for each accepted state. Compared to

standard pseudo-marginal approach where we need to simulate the states for each proposal, DA can provide substantial computational benefits.

However, the simple importance approach does not scale well with the data, leading to large variance in importance weights. Thus it is more efficient to use particle filtering based simulation methods for the marginal likelihood estimation and state simulation. Although `bssm` supports standard bootstrap particle filter (Gordon, Salmond, and Smith 1993), we recommend using more efficient ψ-particle filter (Vihola, Helske, and Franks 2016) which makes use of our approximating Gaussian model. With ψ-PF, we typically need only a very few particles (say 10) for relatively accurate likelihood estimate, which again speeds up the computations.

In addition to standard pseudo-marginal MCMC or its DA variant, `bssm` also supports the importance sampling type correction method presented in Vihola, Helske, and Franks (2016). Here the MCMC algorithm targets the approximating posterior (based on the Gaussian model), and the correction to actual target posterior is made in offline fashion. This gives some additional computational benefits over DA we need to run the particle filter only for each accepted value of the Markov chain (after burnin), and the weight computations are straightforwardly parallelisable. For details, see Vihola, Helske, and Franks (2016).

For all MCMC algorithms, `bssm` uses so-called jump chain representation of the Markov chain $X_1, \ldots, X_n$, where we only store each accepted X_k and the number of steps we stayed on the same state. So for example if $X_{1:n} = (1, 2, 2, 1, 1, 1)$, we present such chain as $\tilde{X} = (1, 2, 1)$, $N = (1, 2, 3)$. This approach reduces the storage space, and makes it more efficient to use importance sampling type correction algorithms. One drawback of this approach is that the results from the MCMC runs correspond to weighted samples from the target posterior, so some of the typical postprocessing tools need to be adjusted. Of course in case of other methods than IS-correction, the simplest option is to just expand the samples using the stored counts N instead.

Example

Here is a short example. First we build our model using `bsm` function:

```
library("bssm")
set.seed(123)

init_sd <- 0.1 * sd(log10(UKgas))
prior <- halfnormal(init_sd, 1)
model <- bsm(log10(UKgas), sd_y = prior, sd_level = prior,
  sd_slope = prior, sd_seasonal =  prior)
```

And run MCMC and print some summary statistics (there is also a `summary` method with similar output):

```
mcmc_out <- run_mcmc(model, n_iter = 1e5)
mcmc_out
```

```
##
## Call:
## run_mcmc.bsm(object = model, n_iter = 1e+05)
##
## Iterations = 50001:1e+05
## Thinning interval = 1
## Length of the final jump chain = 11463
##
## Acceptance rate after the burn-in period:  0.22924
##
## Summary for theta:
##
```

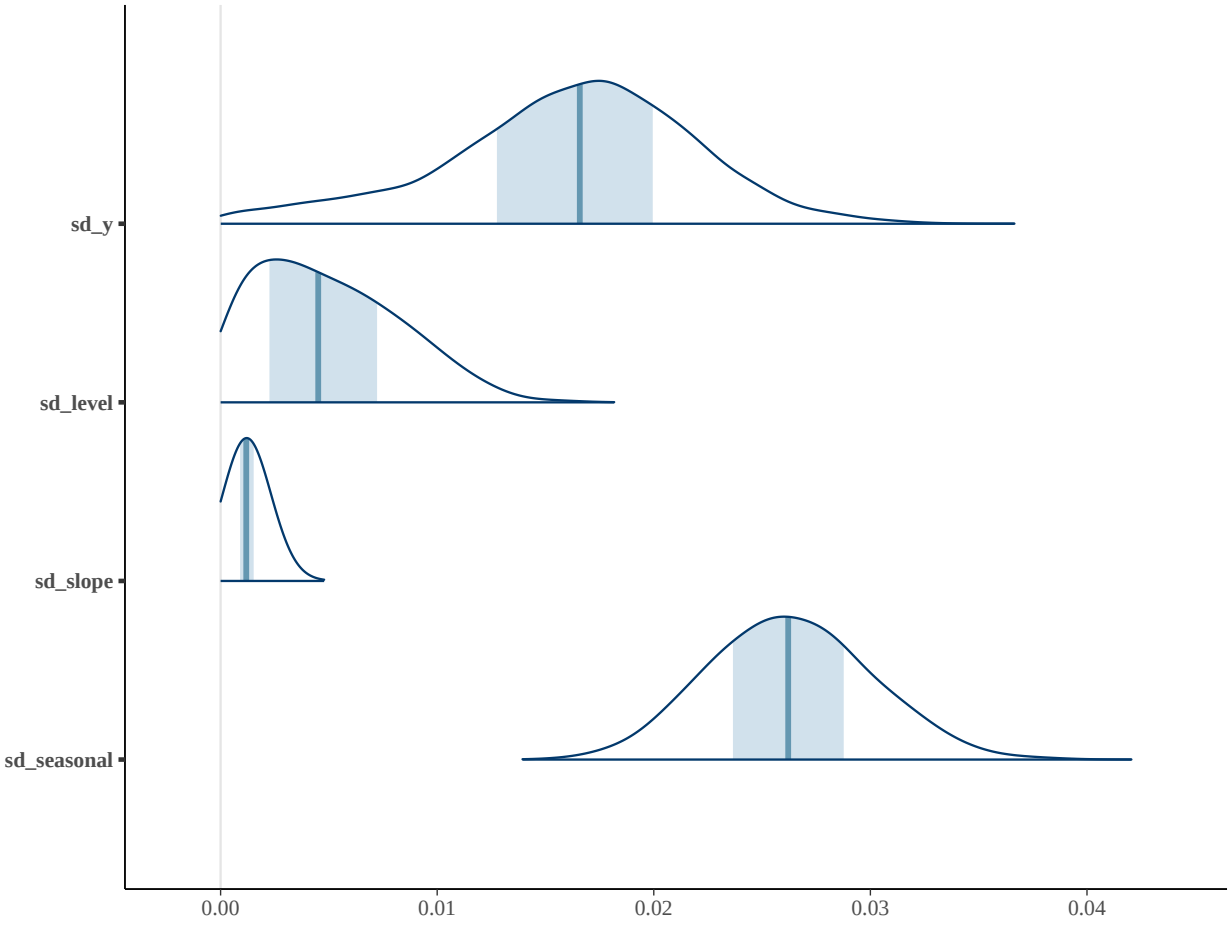
```
##                    Mean           SD        SE-IS        SE-AR
## sd_y        0.016092156 0.0057266809 7.113534e-05 1.107057e-04
## sd_level    0.004937246 0.0032716025 4.297507e-05 6.839495e-05
## sd_slope    0.001228371 0.0005186896 6.083203e-06 9.511893e-06
## sd_seasonal 0.026286522 0.0037522119 4.589120e-05 7.757001e-05
##                       SE
## sd_y        1.315902e-04
## sd_level    8.077578e-05
## sd_slope    1.129077e-05
## sd_seasonal 9.012829e-05
##
## Effective sample sizes for theta:
##
##               ESS-IS   ESS-AR
## sd_y        4813.643 2675.879
## sd_level    5054.692 2288.089
## sd_slope    5135.614 2973.591
## sd_seasonal 5639.565 2339.845
##
## Summary for alpha_108:
##
##                    Mean          SD        SE-IS        SE-AR           SE
## level       2.835461428 0.013478216 1.737592e-04 1.765963e-04 2.477469e-04
## slope       0.009848634 0.003684994 4.671918e-05 4.910402e-05 6.777822e-05
## seasonal_1  0.061691196 0.017976854 2.421116e-04 2.470094e-04 3.458781e-04
## seasonal_2 -0.294918227 0.015252940 2.002608e-04 2.035056e-04 2.855152e-04
## seasonal_3 -0.034554528 0.014357334 1.944550e-04 1.944635e-04 2.750069e-04
##
## Effective sample sizes for alpha:
##
##              ESS-IS   ESS-AR
## level      5621.276 5825.073
## slope      4912.816 5631.699
## seasonal_1 5078.434 5296.642
## seasonal_2 5593.020 5617.645
## seasonal_3 4816.931 5450.939
##
## Run time:
##    user  system elapsed
##  12.420   0.028  12.709
```

For plotting purposes, we'll use `bayesplot` package for the figures after expanding our jump chain representation. Note that the API of the `expand_sample` function is likely change in future.

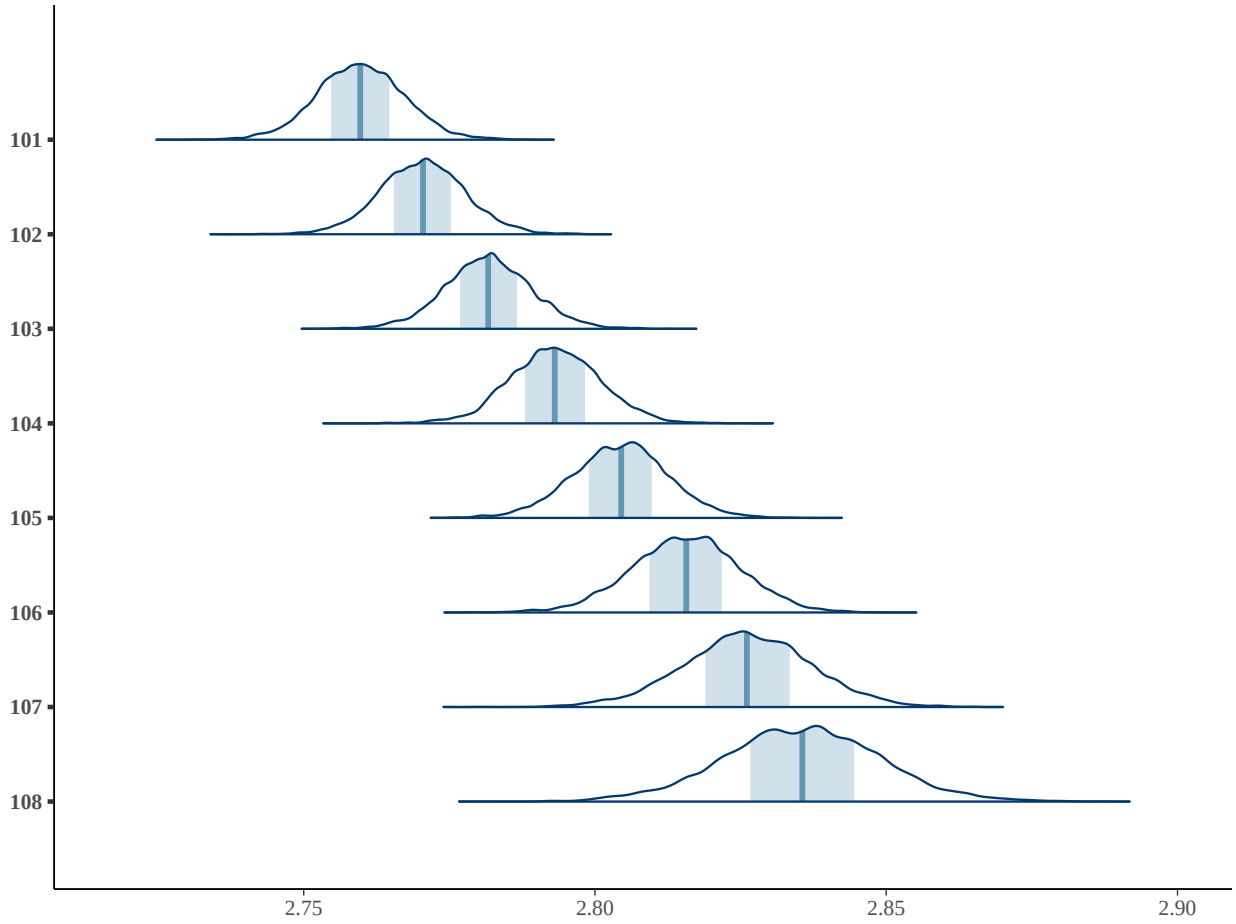
```
theta <- as.matrix(expand_sample(mcmc_out, "theta")) ## until bayesplot is updated
library("bayesplot")
```

```
## This is bayesplot version 1.2.0
```

```
mcmc_areas(theta, bw = 0.001)
```




```
level <- expand_sample(mcmc_out, "alpha",  times = 101:108, states = 1)
mcmc_areas(level)
```




```
# posterior mode estimates
mcmc_out$theta[which.max(mcmc_out$posterior), ]
```

```
##         sd_y     sd_level     sd_slope  sd_seasonal
## 0.0180045419 0.0007440553 0.0012899432 0.0250702911
```

Smoothed trend:

```
level <- expand_sample(mcmc_out, "alpha", states = 1)
# or using summary method:
sumr <- summary(mcmc_out)
level <- sumr$alpha_mean[, 1]
lwr <- level - 1.96 * sumr$alpha_sd[, 1]
upr <- level + 1.96 * sumr$alpha_sd[, 1]
ts.plot(model$y, cbind(level, lwr, upr), col = c(1, 2, 2, 2), lty = c(1, 1, 2, 2))
```

For prediction intervals, we first build a model for the future time points, and then use the previously obtained posterior samples for the prediction:

```
future_model <- model
future_model$y <- ts(rep(NA, 24), start = end(model$y) + c(0, 1), frequency = 4)
pred <- predict(mcmc_out, future_model, probs = c(0.025, 0.1, 0.9, 0.975),
  return_MCSE = FALSE)
ts.plot(log10(UKgas), pred$mean, pred$intervals[, -3],
  col = c(1, 2, c(3, 4, 4, 3)), lty = c(1, 1, rep(2, 4)))
```

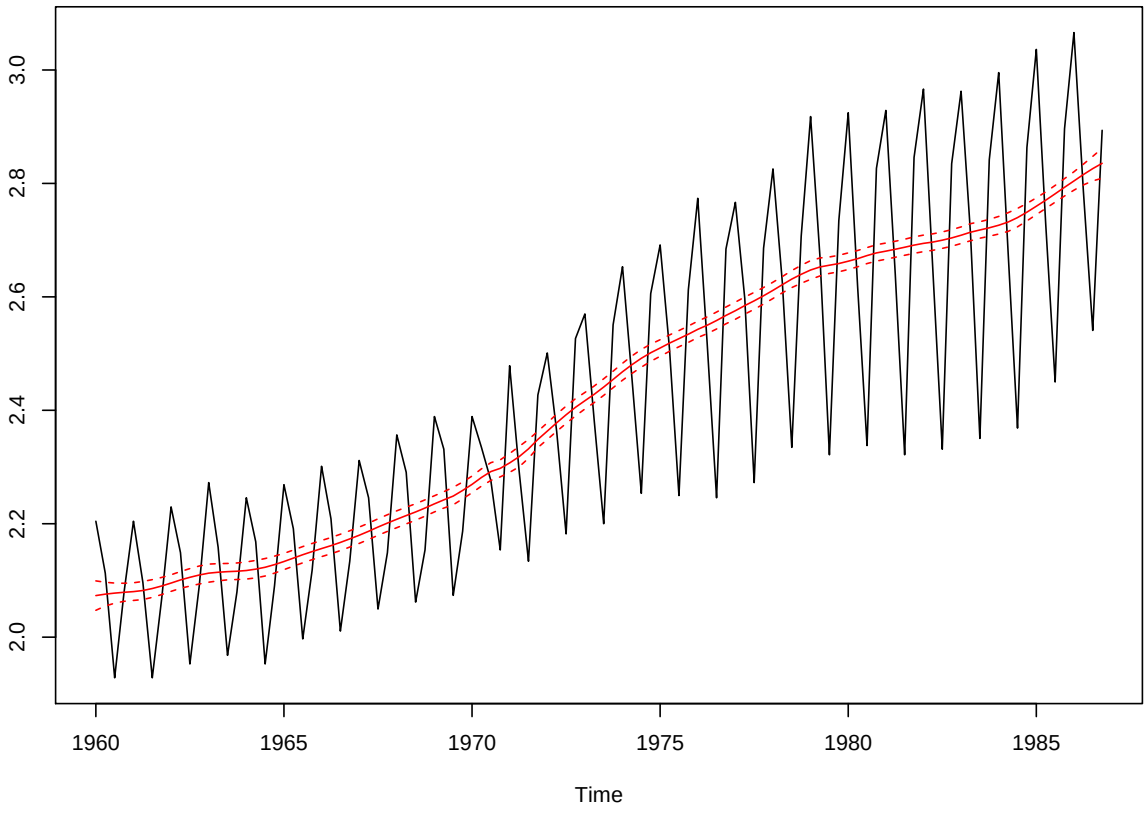


Figure 1: Smoothed trend component with 95% intervals.

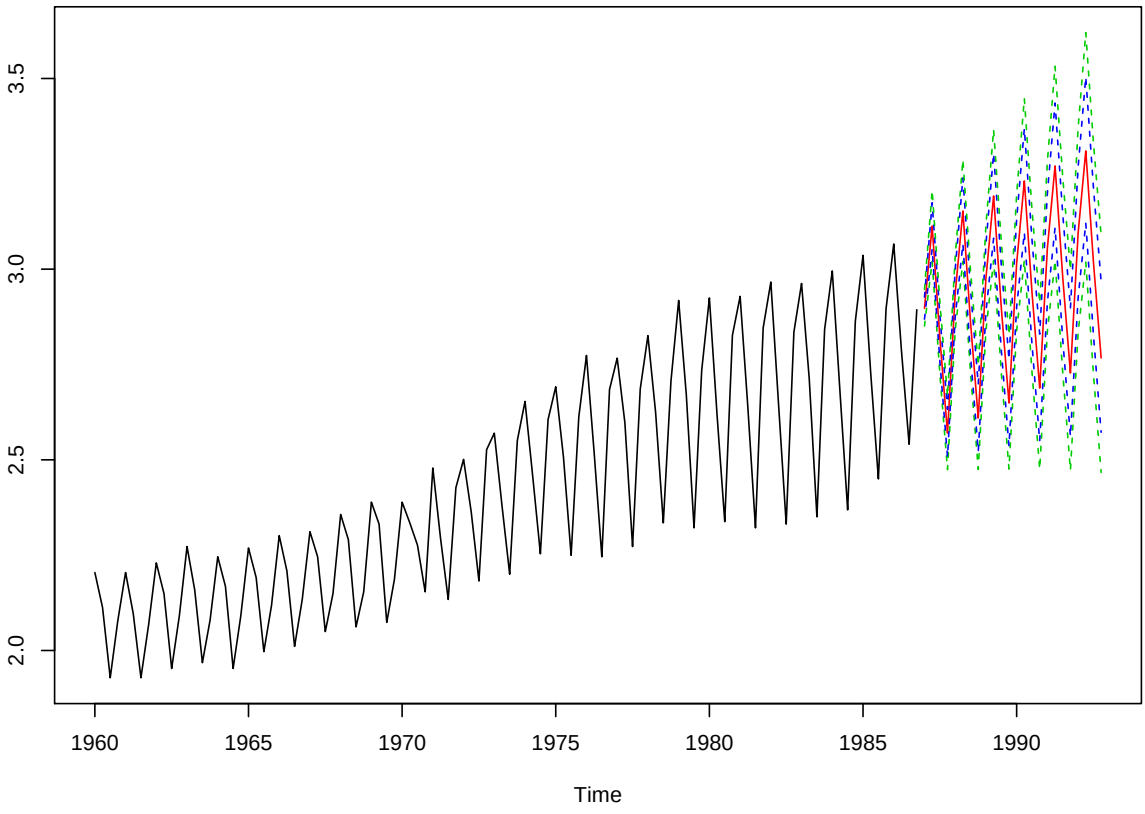


Figure 2: Mean predictions and prediction intervals.

Now same with ggplot2:

```
require("ggplot2")
```

```
## Loading required package: ggplot2
```

```
level_fit <- ts(colMeans(expand_sample(mcmc_out, "alpha")$level), start = start(model$y),
  frequency = 4)
autoplot(pred, y = model$y, fit = level_fit, interval_color = "red", alpha_fill = 0.2)
```

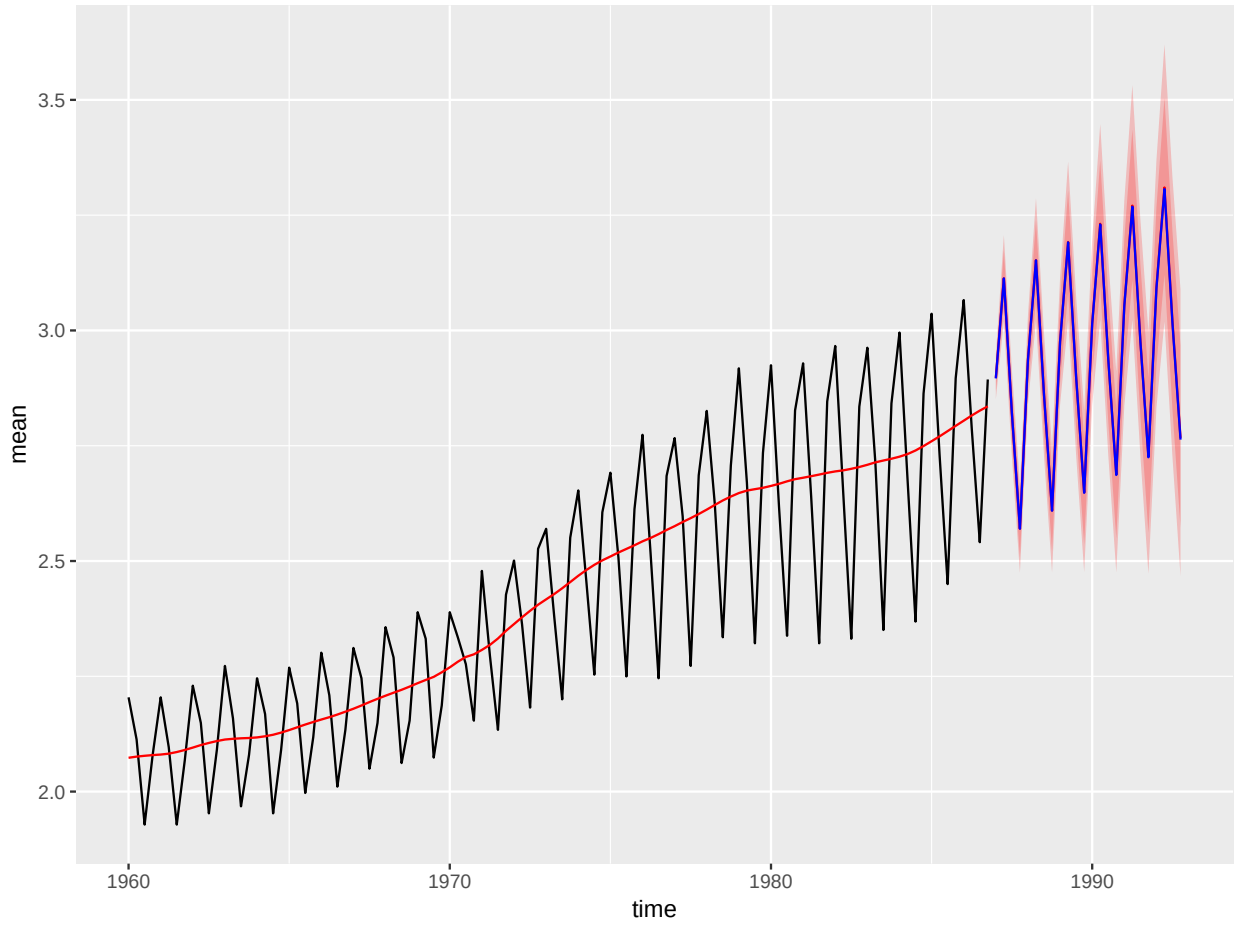


Figure 3: Prediction plots with ggplot2.

We can also obtain prediction in terms of individual components of the state vector:

```
pred_state <- predict(mcmc_out, future_model, probs = c(0.025, 0.1, 0.9, 0.975),
  return_MCSE = FALSE, type = "state")
ts.plot(log10(UKgas), level_fit, pred_state$mean[,"level"], pred_state$intervals$level[, -3],
  col = c(1, 2, 2, c(3, 4, 4, 3)), lty = c(1, 1, 1, rep(2, 4)))
```

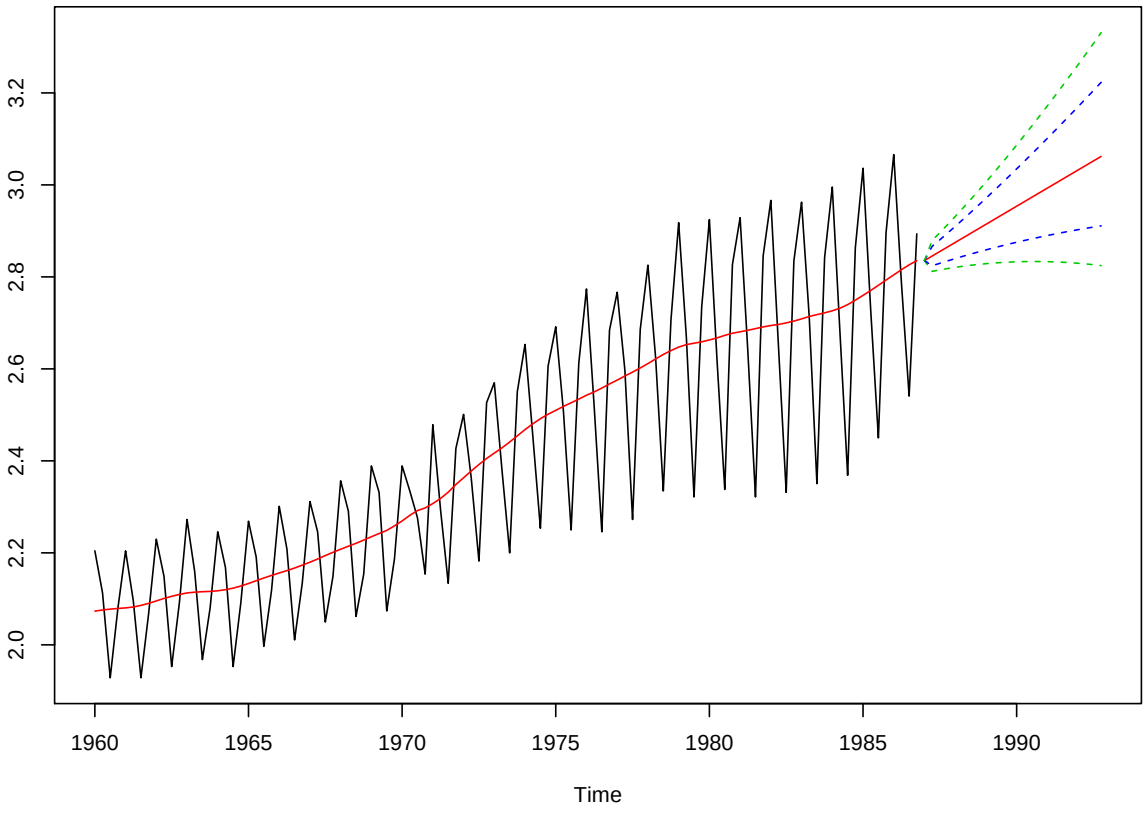


Figure 4: State prediction.

References

Andrieu, Christophe, and Gareth O. Roberts. 2009. "The Pseudo-Marginal Approach for Efficient Monte Carlo Computations." *Annstat* 37 (2): 697–725.

Banterle, M., C. Grazian, A. Lee, and C. P. Robert. 2015. "Accelerating Metropolis-Hastings algorithms by Delayed Acceptance." *ArXiv E-Prints*, March. http://arxiv.org/abs/1503.00996.

Beaumont, Mark A. 2003. "Estimation of Population Growth or Decline in Genetically Monitored Populations." *Genetics* 164: 1139–60.

Christen, J. Andrés, and Colin Fox. 2005. "Markov Chain Monte Carlo Using an Approximation." *Journal of Computational and Graphical Statistics* 14 (4): 795–810. doi:10.1198/106186005X76983.

Durbin, J., and S. J. Koopman. 2000. "Time Series Analysis of Non-Gaussian Observations Based on State Space Models from Both Classical and Bayesian Perspectives." *Journal of Royal Statistical Society B* 62: 3–56.

———. 2002. "A Simple and Efficient Simulation Smoother for State Space Time Series Analysis." *Biometrika* 89: 603–15.

———. 2012. *Time Series Analysis by State Space Methods*. 2nd ed. New York: Oxford University Press.

Gordon, Neil J., D. J. Salmond, and A. F. M. Smith. 1993. "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation." *IEE Proceedings-F* 140 (2): 107–13.

Harvey, A. C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Lin, L., K.F. Liu, and J. Sloan. 2000. "A Noisy Monte Carlo Algorithm." *Physical Review D* 61.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Vihola, M., J. Helske, and J. Franks. 2016. "Importance sampling type correction of Markov chain Monte Carlo and exact approximations." *ArXiv E-Prints*, September.

Vihola, Matti. 2012. "Robust Adaptive Metropolis Algorithm with Coerced Acceptance Rate." *Statistics and Computing* 22 (5): 997–1008. doi:10.1007/s11222-011-9269-5.