VectorBiTE Methods Training
Bayesian State Space Modeling for Time Series Data

The VectorBiTE Team
(John W. Smith, Virginia Tech)

Summer 2021

Fitting State Space Models

Fitting State Space Models

In this section we will:

- Discuss basic fitting methods for Linear Gaussian State Space Models (NDLMs)

Fitting State Space Models

In this section we will:

- Discuss basic fitting methods for Linear Gaussian State Space Models (NDLMs)
- Discuss differences between smoothing and forward filtering

Fitting State Space Models

In this section we will:

- Discuss basic fitting methods for Linear Gaussian State Space Models (NDLMs)
- Discuss differences between smoothing and forward filtering
- Fit NDLMs in `JAGS`

Fitting State Space Models

In this section we will:

- Discuss basic fitting methods for Linear Gaussian State Space Models (NDLMs)
- Discuss differences between smoothing and forward filtering
- Fit NDLMs in `JAGS`
- Examine some diagnostics and applications in `JAGS`

Linear Gaussian State Estimation: Forward in Time

Linear Gaussian State Estimation: Forward in Time

- Suppose that we are interested in the current value of x_t, given the previous $t-1$ estimates $x_{1:t-1}$, and the observations $y_{1:t}$.

Linear Gaussian State Estimation: Forward in Time

- Suppose that we are interested in the current value of x_t, given the previous $t-1$ estimates $x_{1:t-1}$, and the observations $y_{1:t}$.
- The model is linear and Gaussian, so the evolution function and observation density function will have the form

$$x_t \sim N(A_t x_{t-1} + b_t, \phi)$$
$$y_t \sim N(\alpha_t x_t + \beta_t, \tau)$$

Linear Gaussian State Estimation: Forward in Time

- Suppose that we are interested in the current value of x_t, given the previous $t-1$ estimates $x_{1:t-1}$, and the observations $y_{1:t}$.
- The model is linear and Gaussian, so the evolution function and observation density function will have the form

$$x_t \sim N(A_t x_{t-1} + b_t, \phi)$$

$$y_t \sim N(\alpha_t x_t + \beta_t, \tau)$$

- Is there an analytic full conditional distribution for x_t?

Linear Gaussian State Estimation: Forward in Time

- Suppose that we are interested in the current value of x_t, given the previous $t-1$ estimates $x_{1:t-1}$, and the observations $y_{1:t}$.
- The model is linear and Gaussian, so the evolution function and observation density function will have the form

$$x_t \sim N(A_t x_{t-1} + b_t, \phi)$$
$$y_t \sim N(\alpha_t x_t + \beta_t, \tau)$$

- Is there an analytic full conditional distribution for x_t?
  - Yes there is, and its a familiar friend

Linear Gaussian State Estimation: Forward in Time

- If we write out the equation for the full conditional distribution,

$$\pi(x_t|x_{1:t-1}, y_{1:t}) \propto \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big) \exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big),$$

Linear Gaussian State Estimation: Forward in Time

- If we write out the equation for the full conditional distribution,

$$\pi(x_t|x_{1:t-1}, y_{1:t}) \propto \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big)$$
$$\exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big),$$

we can do a little algebra to come to the conclusion that

$$\pi(x_t|\cdot) \sim N(\mu^* = \frac{\phi(A_t x_{t-1} + b_t) + \tau\alpha_t(y_t - \beta_t))}{\phi + \tau\alpha_t^2}, \phi^* = \phi + \tau\alpha_t^2)$$

Linear Gaussian State Estimation: Forward in Time

- If we write out the equation for the full conditional distribution,

$$\pi(x_t|x_{1:t-1}, y_{1:t}) \propto \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big)$$
$$\exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big),$$

we can do a little algebra to come to the conclusion that

$$\pi(x_t|\cdot) \sim N(\mu^* = \frac{\phi(A_t x_{t-1} + b_t) + \tau\alpha_t(y_t - \beta_t))}{\phi + \tau\alpha_t^2}, \phi^* = \phi + \tau\alpha_t^2)$$

- This is the Kalman filter solution for updating the states

Linear Gaussian State Estimation: Smoothing

Linear Gaussian State Estimation: Smoothing

- What if instead of estimating the current value of x_t given the previous $t-1$ estimates $x_{1:t-1}$ and the observations $y_{1:t}$, we wanted to use all of the data, $x_{1:T}$ and $y_{1:T}$?

Linear Gaussian State Estimation: Smoothing

- What if instead of estimating the current value of x_t given the previous $t-1$ estimates $x_{1:t-1}$ and the observations $y_{1:t}$, we wanted to use all of the data, $x_{1:T}$ and $y_{1:T}$?
- Why would we want to do this?

Linear Gaussian State Estimation: Smoothing

- What if instead of estimating the current value of x_t given the previous $t-1$ estimates $x_{1:t-1}$ and the observations $y_{1:t}$, we wanted to use all of the data, $x_{1:T}$ and $y_{1:T}$?
- Why would we want to do this?
  - We get more information about the latent states by using all of the data

Linear Gaussian State Estimation: Smoothing

- What if instead of estimating the current value of x_t given the previous $t-1$ estimates $x_{1:t-1}$ and the observations $y_{1:t}$, we wanted to use all of the data, $x_{1:T}$ and $y_{1:T}$?
- Why would we want to do this?
  - We get more information about the latent states by using all of the data
  - Better estimation of Θ

Linear Gaussian State Estimation: Smoothing

- What if instead of estimating the current value of x_t given the previous $t-1$ estimates $x_{1:t-1}$ and the observations $y_{1:t}$, we wanted to use all of the data, $x_{1:T}$ and $y_{1:T}$?
- Why would we want to do this?
  - We get more information about the latent states by using all of the data
  - Better estimation of Θ
- The process of using all of the data at once to estimate the latent states is commonly called *smoothing*

Linear Gaussian State Estimation: Smoothing

Linear Gaussian State Estimation: Smoothing

- If we write out the equation for the full conditional distribution,

$$\pi(x_t | x_{1:t-1}, x_{t+1:T}, y_{1:T}) \propto \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big)$$
$$\exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big)$$
$$\exp\Big(-\frac{\phi}{2}(x_{t+1} - A_{t+1} x_t - b_{t+1})^2\Big),$$

Linear Gaussian State Estimation: Smoothing

- If we write out the equation for the full conditional distribution,

$$\pi(x_t|x_{1:t-1}, x_{t+1:T}, y_{1:T}) \propto \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big)$$
$$\exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big)$$
$$\exp\Big(-\frac{\phi}{2}(x_{t+1} - A_{t+1} x_t - b_{t+1})^2\Big),$$

we can do a little algebra to come to the conclusion that

$$\pi(x_t|\cdot) \sim N(\mu^* = \frac{\phi(A_t x_{t-1} + b_t + A_{t+1}(x_{t+1} - b_{t+1})) + \tau\alpha_t(y_t - \beta_t)}{\phi(1 + A_{t+1}^2) + \tau\alpha_t^2},$$
$$\phi^* = \phi(1 + A_{t+1}^2) + \tau\alpha_t^2)$$

Linear Gaussian State Estimation: Smoothing

- If we write out the equation for the full conditional distribution,

$$\begin{aligned}\pi(x_t|x_{1:t-1}, x_{t+1:T}, y_{1:T}) \propto & \exp\Big(-\frac{\phi}{2}(x_t - A_t x_{t-1} - b_t)^2\Big) \\ & \exp\Big(-\frac{\tau}{2}(y_t - \alpha_t x_t - \beta_t)^2\Big) \\ & \exp\Big(-\frac{\phi}{2}(x_{t+1} - A_{t+1} x_t - b_{t+1})^2\Big),\end{aligned}$$

we can do a little algebra to come to the conclusion that

$$\begin{aligned}\pi(x_t|\cdot) \sim N(\mu^* = & \frac{\phi(A_t x_{t-1} + b_t + A_{t+1}(x_{t+1} - b_{t+1})) + \tau\alpha_t(y_t - \beta_t)}{\phi(1 + A_{t+1}^2) + \tau\alpha_t^2}, \\ \phi^* = & \phi(1 + A_{t+1}^2) + \tau\alpha_t^2)\end{aligned}$$

- This is the smoothing solution for updating the states

Forward Filtering vs Smoothing

Forward Filtering vs Smoothing

- The key difference between forward filtering vs smoothing is that forwarding filtering uses only the data *before* time step t, while smoothing uses all of the data

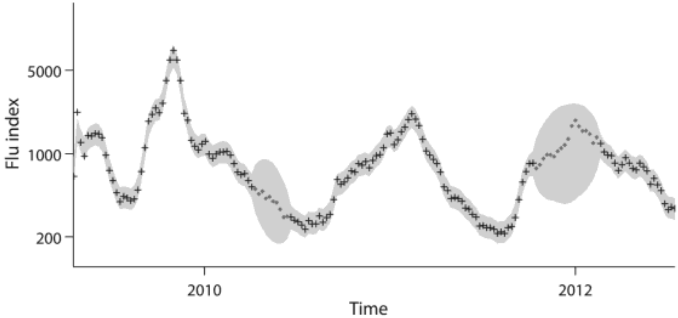
Forward Filtering vs Smoothing

- The key difference between forward filtering vs smoothing is that forwarding filtering uses only the data *before* time step t, while smoothing uses all of the data
- Forward filtering is frequently used in *real time forecasting* applications, where a process is being monitored and predicted, usually on short timescales

Forward Filtering vs Smoothing

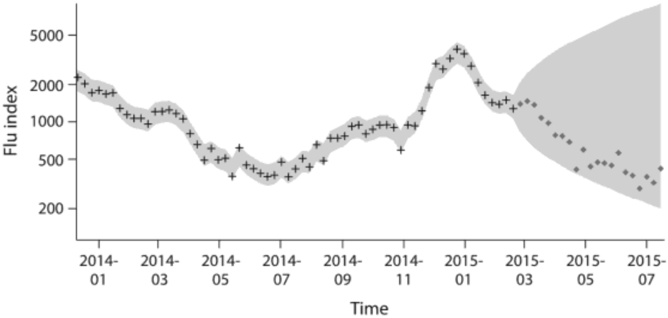
- The key difference between forward filtering vs smoothing is that forwarding filtering uses only the data *before* time step t, while smoothing uses all of the data
- Forward filtering is frequently used in *real time forecasting* applications, where a process is being monitored and predicted, usually on short timescales
- Applications include robotics, weather forecasting

Forward Filtering vs Smoothing



Example of smoothing taken from Dietze 2018. Note that the uncertainty is highest in the center of the missing observations

Forward Filtering vs Smoothing



Example of forward filtering taken from Dietze 2018. The uncertainty grows increasingly large because we are only looking forward in time

Example: Simple Linear Gaussian Model

- Let's start with a simple state space model of the form
$$x_t \sim N(A_t x_{t-1} + b_t, \phi)$$
$$y_t \sim N(\alpha_t x_t + \beta_t, \tau)$$
- Lets assume $A, b, \alpha, \beta, \phi, \tau$ are all known
- Suppose $A = .99, b = .5, \alpha = .95, \beta = 1, \phi = 4, \tau = 4$
- Let $x_1 = 50$

Example: Simple Linear Gaussian Model

```
## set seed for consistent results
library(matrixStats)
set.seed(123)

## set values for parameters
t <- 50
y <- x <- rep(0, t)
A <- .99
b <- .5
phi <- 1/.5^2
tau <- 4
alpha <- .95
beta <- 1
```

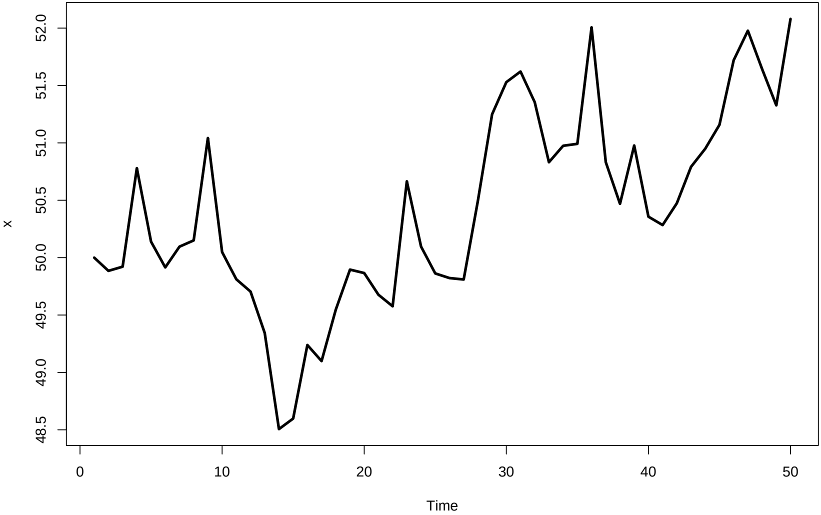
Example: Simple Linear Gaussian Model

```r
## set initial x, y values
x[1] = 50
y[1] = alpha*x[1] + beta + rnorm(1, 0, sqrt(1/tau))

## generate latent states and observations
for (i in 2:t){
  x[i] = A*x[i-1] + b + rnorm(1, 0, sqrt(1/phi))
  y[i] = alpha*x[i] + beta + rnorm(1, 0, sqrt(1/tau))
}
```

Example: Simple Linear Gaussian Model

```
plot(x, type = 'l', xlab = 'Time', lwd = 3)
```

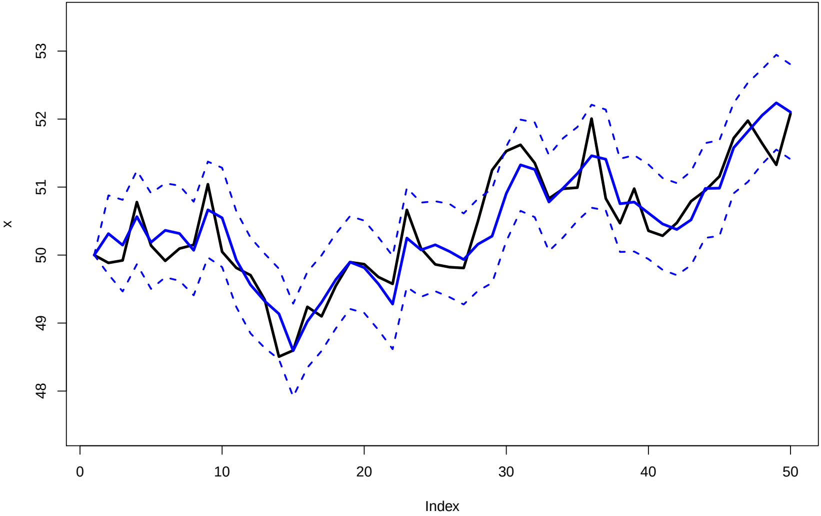


Example: Simple Linear Gaussian Model

- Now, let's use a Kalman Filter to estimate the states

```
## initialize states for kalman filter and smoother
states.kf <- matrix(NA, nrow = 1000, ncol = t)
states.kf[,1] <- x[1]
states.kf[1,] <- (y - beta) / alpha
## sample 1000 points with KF
for (i in 2:1000){
  for (j in 2:t){
    states.kf[i,j] <- rnorm(1, mean = (phi*(A*states.kf[i-1,j-1] + b)
                            + tau*(alpha*y[j] - alpha*beta))
                            / (phi + tau*alpha^2),
                            sd = sqrt(1 / (phi + tau*alpha^2)))
  }
}
```

Example: Simple Linear Gaussian Model



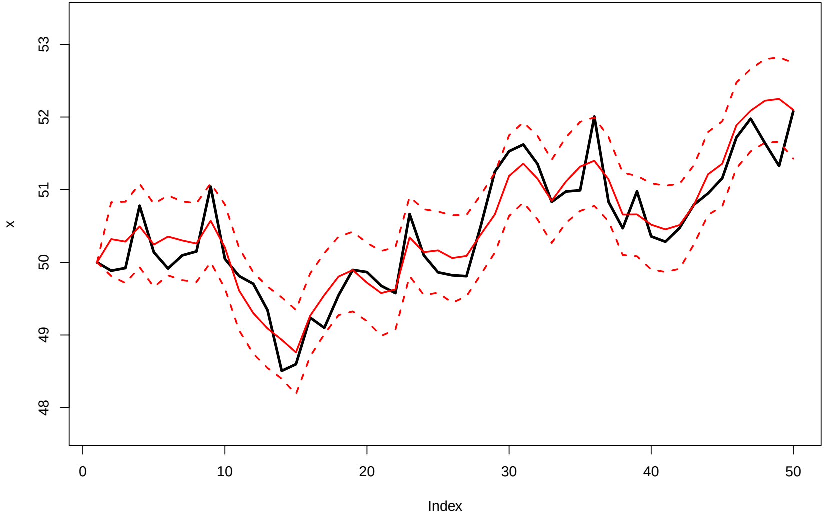
Example: Simple Linear Gaussian Model

- Now, let's estimate the states with the smoothing solution

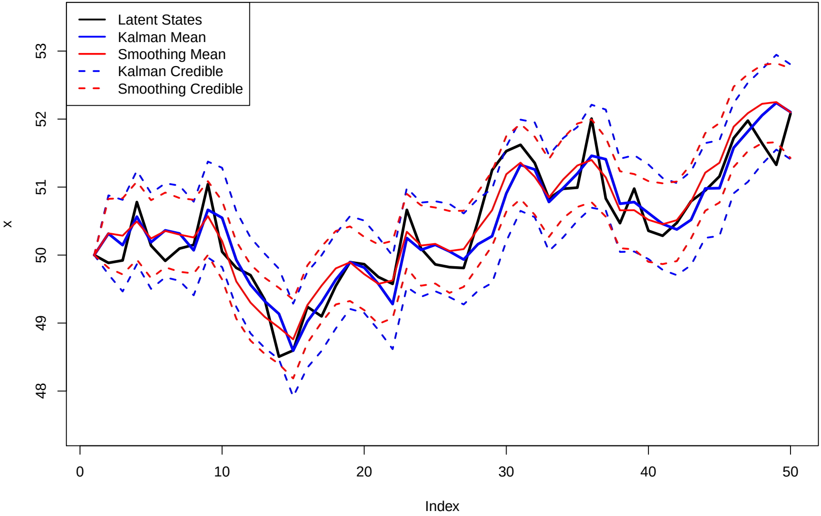
```
## initialize states for kalman filter and smoother
states.smooth <- matrix(NA, nrow = 1000, ncol = t)
states.smooth[,1] <- x[1]
states.smooth[1,] <- (y - beta) / alpha

##
for (i in 2:1000){
  for (j in 2:(t-1)){
    states.smooth[i,j] <- rnorm(1,
                        mean = (phi*(A*states.smooth[i-1,j-1] + b + A*
                        sd = sqrt(1 / (phi + tau*alpha^2 + A^2 *phi))
  }
  j = t
  states.smooth[i,j] <- rnorm(1,
                             mean = (phi*(A*states.smooth[i,j-1] + b)
                             / (phi + tau*alpha^2),
                             sd = sqrt(1 / (phi + tau*alpha^2)) )
}
```

Example: Simple Linear Gaussian Model



Example: Simple Linear Gaussian Model



JAGS Refresher

JAGS Refresher

- JAGS stands for Just Another Gibbs Sampler

JAGS Refresher

- JAGS stands for Just Another Gibbs Sampler
- Symbolic language makes it easily accessible

JAGS Refresher

- JAGS stands for Just Another Gibbs Sampler
- Symbolic language makes it easily accessible
- Written in C++ with easy parallelization across chains

JAGS Refresher

- JAGS stands for Just Another Gibbs Sampler
- Symbolic language makes it easily accessible
- Written in C++ with easy parallelization across chains
- Does univariate Metropolis-Hastings to sample parameters (usually)

NDLMs in JAGS

NDLMs in JAGS

- JAGS does quite a good job fitting NDLMs

NDLMs in JAGS

- JAGS does quite a good job fitting NDLMs
- The latent states have analytic Gibbs sampling updates

NDLMs in JAGS

- JAGS does quite a good job fitting NDLMs
- The latent states have analytic Gibbs sampling updates
- JAGS will use the same updates for the latent states that we used for the smoothing solution

Example: NDLM in JAGS

Suppose that we want to fit the following NDLM in JAGS:

$$\begin{aligned} x_t &= Ax_{t-1} + b + \epsilon_{proc} \\ y_t &= x_t + \epsilon_{obs} \\ A &= .99, b = .5, x_1 \sim N(50, 1) \\ \epsilon_{proc} &\sim N(0, \phi), \epsilon_{obs} \sim N(0, \tau), \end{aligned}$$

where A, b, τ are all known

Example: NDLM in JAGS

```
## set values for parameters
t <- 25
y <- x <- rep(0, t)
A <- .99
b <- .5
phi <- tau <- 4
alpha <- 1
beta <- 0
## set initial x, y values
set.seed(54321)
x[1] = 50
y[1] = alpha*x[1] + beta + rnorm(1, 0, sqrt(1/tau))
## generate latent states and observations
for (i in 2:t){
  x[i] = A*x[i-1] + b + rnorm(1, 0, sqrt(1/phi))
  y[i] = alpha*x[i] + beta + rnorm(1, 0, sqrt(1/tau))
}
```

Example: NDLM in JAGS

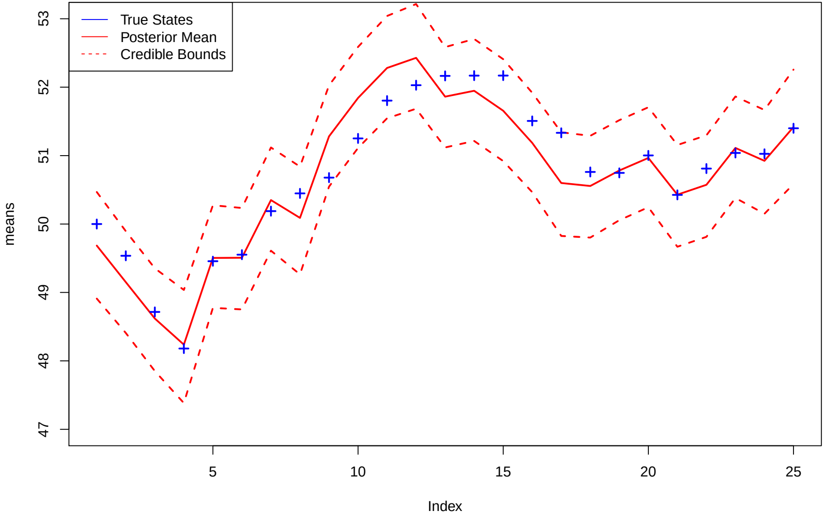
```
library(rjags)
## sink jags model
sink('jags_test.bug')
cat('model {
  for(i in 2:nday){
    x.pred[i] = A*x[i-1] + b
    x[i] ~ dnorm(x.pred[i], phi)
  }
  for(i in 1:nday){
    y[i] ~ dnorm(x[i], 4)
  }
  ## Initial conditions
  x[1] ~ dnorm(50, 1)

  ## Priors on process errors
  phi ~ dnorm(0, .01)T(0,100)
}'
)
sink()
```

Example: NDLM in JAGS

```
library(rjags, quietly= TRUE)
## make data list
model_data <- list('nday' = t,
                   'y' = y,
                   'A' = A,
                   'b' = b)
## compile model
jags_ex1 <- jags.model('jags_test.bug',
                   data = model_data,
                   n.chains=1,
                   n.adapt=1000)
## generate samples
samples_ex1 = coda.samples(model = jags_ex1,
                variable.names =
                c('phi', paste0(paste0('x[', 1:25), ']')),
                n.iter = 20000)
```

Example: NDLM in JAGS



Example: NDLMs in JAGS

We can extract summary statistics in JAGS using `summary()`

```
>                         phi      x[10]      x[11]      x[12]
> Mean            2.493203880 51.842341901 52.280849023 52.428670927
> SD              1.264050828  0.377807593  0.382022262  0.388409123
> Naive SE        0.008938189  0.002671503  0.002701305  0.002746467
> Time-series SE  0.024994267  0.003428142  0.003576455  0.003702672
```

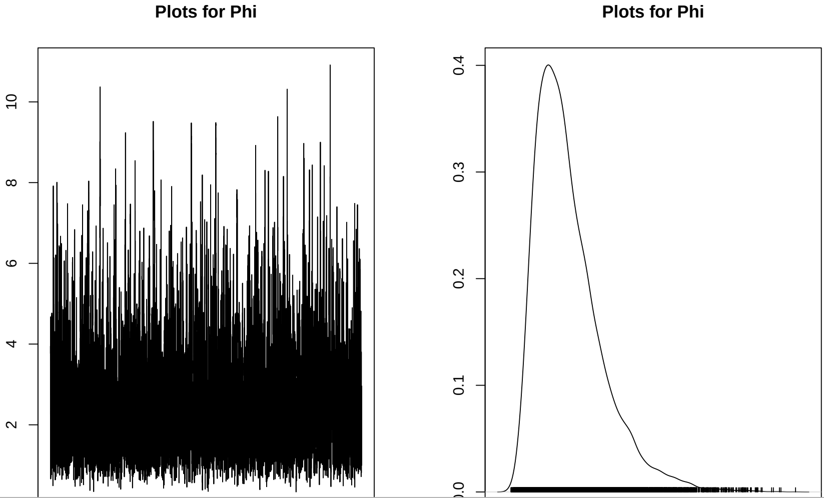
Example: NDLMs in JAGS

`summary()` can also be used to extract quantiles

```
>              phi    x[10]    x[11]    x[12]    x[13]
> 2.5%  0.8576493 51.11137 51.54536 51.68505 51.11910
> 25%   1.5893488 51.58633 52.02076 52.16526 51.61301
> 50%   2.2213025 51.83817 52.28233 52.42565 51.86433
> 75%   3.1045108 52.09853 52.53284 52.68484 52.11452
> 97.5% 5.7465298 52.58831 53.04071 53.21512 52.58584
```

Example: NDLMs in JAGS

Trace plots and density plots allow us to visualize our posterior distributions and help to assess problems with mixing



Example: NDLMs in JAGS

`effectiveSize()` in the `coda` package gives us an estimate of how many *unique* posterior samples we have generated with our MCMC. This is an important diagnostic in State Space Models, where states can be slow to mix and highly autocorrelated.

```
library(coda)
effectiveSize(samples_ex1)
>       phi     x[10]     x[11]     x[12]     x[13]     x[14]     x[15]
>  2557.692 12145.734 11409.634 11003.944 14481.395 13163.271 13692.753
>     x[17]     x[18]     x[19]      x[1]     x[20]     x[21]     x[22]
> 12124.621 13220.469 15361.542 13761.179 12800.519 11381.574 12398.019
>     x[24]     x[25]      x[2]      x[3]      x[4]      x[5]      x[6]
> 13283.061 13947.661 14155.159 11705.865  6000.615 11569.733 12949.963
>      x[8]      x[9]
>  8848.881 13755.663
```

Extension: Missing Data

In the example case, we had all of our observations available. It turns out that it's simple to incorporate missing observation data into our JAGS analysis!

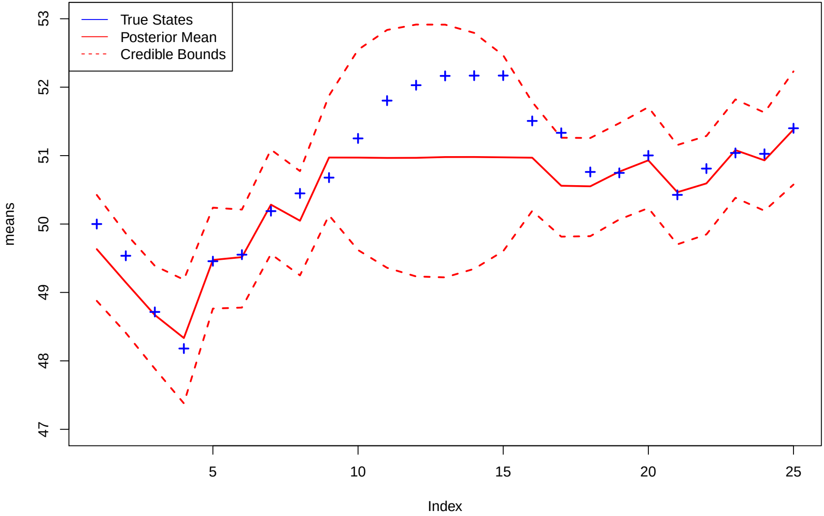
```
## make y_miss, an example with missing observation data
y_miss <- y

## set observations 10 through 15 to be missing
y_miss[10:15] <- NA
```

Extension: Missing Data

```r
## make list of data with observations missing
model_data_missing <- list('nday' = t,
                   'y' = y_miss,
                   'A' = A,
                   'b' = b)
## compile model with missing data
jags_ex1_missing <- jags.model('jags_test.bug',
                      data = model_data_missing,
                      n.chains=1,
                      n.adapt=1000)
## generate samples
samples_ex1_missing = coda.samples(model = jags_ex1_missing,
                       variable.names =
                       c('phi', paste0(paste0('x[', 1:25), ']')),
                       n.iter = 20000)
```

Extension: Missing Data



Extension: Forecasting

We have talked about how State Space Models are powerful tools for forecasting, but how can we forecast with them in JAGS? It turns out that we can forecast in JAGS by just tacking on `NA`s on the end of our observation list

```
## add 7 days with no observations onto the end of y_miss
y_miss <- c(y_miss, rep(NA, 7))

## change nday value to reflect the 7 new days added
t <- 32
```

Extension: Forecasting

```
## create mode data list with new values of t, y_miss
model_data_forecast <- list('nday' = t,
                             'y' = y_miss,
                             'A' = A,
                             'b' = b)
## compile model for forecast
jags_ex1_forecast <- jags.model('jags_test.bug',
                                 data = model_data_forecast,
                                 n.chains=1,
                                 n.adapt=1000)
## generate samples
samples_ex1_missing = coda.samples(model = jags_ex1_forecast,
                                    variable.names =
                                    c('phi', paste0(paste0('x[', 1:t), '
                                    n.iter = 20000)
```

Extension: Forecasting