8.5 Example: Learning from grouped data

- Let us assume that the heights of men from a local college are normally distributed with mean μ and standard deviation σ.
- A random sample of 211 men is taken and data about their heights is summarized in the Table below.

Height (inches)	Frequency
less than 66	14
between 66 and 68	30
between 68 and 70	49
between 70 and 72	70
between 72 and 74	33
over 74	15

8.5 Example: Learning from grouped data

- We are observing multinomial data with unknown bin probabilities $p_1, \ldots, p_6$, where the probabilities are functions of (μ, σ). For example,

  P(a male student's height is between 66 and 68 inches)

  $$= p_2 = \Phi\left(\frac{68-\mu}{\sigma}\right) - \Phi\left(\frac{66-\mu}{\sigma}\right),$$

  where $\Phi(\cdot)$ is the cdf a standard normal.

- The likelihood of the normal parameters given this grouped data is

$$L(\mu,\sigma) = p(\text{data}|\mu,\sigma) \propto \Phi\left(\frac{66-\mu}{\sigma}\right)^{14} \left[\Phi\left(\frac{68-\mu}{\sigma}\right) - \Phi\left(\frac{66-\mu}{\sigma}\right)\right]^{30} \left[\Phi\left(\frac{70-\mu}{\sigma}\right) - \Phi\left(\frac{68-\mu}{\sigma}\right)\right]^{49} \left[\Phi\left(\frac{72-\mu}{\sigma}\right) - \Phi\left(\frac{70-\mu}{\sigma}\right)\right]^{70} \left[\Phi\left(\frac{74-\mu}{\sigma}\right) - \Phi\left(\frac{72-\mu}{\sigma}\right)\right]^{33} \left[1 - \Phi\left(\frac{74-\mu}{\sigma}\right)\right]^{15}.$$

8.5 Example: Learning from grouped data

- Assume the noninformative prior $p(\mu, \sigma) \propto \frac{1}{\sigma}$. Then the posterior is

$$p(\mu, \sigma|\text{data}) \propto p(\text{data}|\mu, \sigma)\frac{1}{\sigma} = L(\mu, \sigma)\frac{1}{\sigma}.$$

- Let us transform the positive σ to the real line by $\lambda = \log(\sigma)$. Since $\sigma = \exp(\lambda)$ and $\frac{\mathrm{d}\sigma}{\mathrm{d}\lambda} = \exp(\lambda)$, the posterior of θ is

$$p(\theta|\text{data}) \propto L(\mu, \exp(\lambda)).$$

- Let us first create the matrix `data` containing the observations. Each row corresponds to a class; the first two columns contains the left and right limits respectively and the third column contains the frequency.

```
data <- cbind( c(-Inf,66,68,70,72,74),
               c(66,68,70,72,74,Inf),
               c(14,30,49,70,33,15) )
```

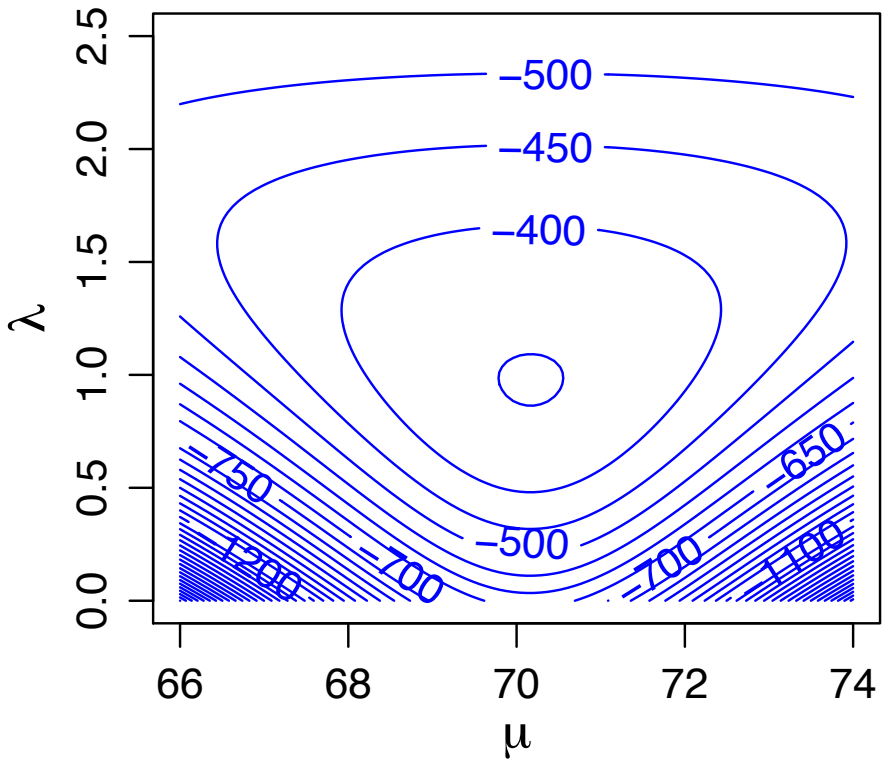
8.5 Example: Learning from grouped data

- Next, we write the function `h`, which computes $\log L(\mu, \exp(\lambda))$.

```
dpnorm <- function(a,b,mu,sigma){
 pnorm(b,mean=mu,sd=sigma) - pnorm(a,mean=mu,sd=sigma)
}
h <- function(theta, data){
 L <- 0
 G <- nrow(data)
 mu <- theta[1]
 sigma <- exp(theta[2])
 for (g in 1:G){
   L <- L + data[g,3]*
          log(dpnorm(data[g,1],data[g,2],mu,sigma))
 }
 return(L)
}
```

8.5 Example: Learning from grouped data

- We first find a normal approximation to the posterior and then use it to construct a proposal density for the Metropolis-Hastings algorithm.
- A contour plot of `h` reveals that the posterior mode is close to (70, 1). We will use this point as the initial value in the `optim` function.



8.5 Example: Learning from grouped data

- Using the `optim` function, we find the posterior mode and compute the covariance matrix of the normal approximation.

```
> start <- c(70,1)
> out <- optim(par=start,fn=h,hessian=TRUE,
+          control=list(fnscale=-1),data=data)
> (post.mode <- out$par)
[1] 70.169880  0.973644
> (post.cov <- -solve(out$hessian))
              [,1]          [,2]
[1,] 3.534713e-02 3.520776e-05
[2,] 3.520776e-05 3.146470e-03
```

- We use the normal approximation to design a Metropolis random walk algorithm. For the proposal density, we use the covariance matrix of the normal approximation and set the scale parameter as 2.

8.5 Example: Learning from grouped data

- We run 10,000 iterations of the random walk algorithm starting at `start`.
- The output `fit1` is a list with two components: `par` is a matrix of simulated values where each row corresponds to a single draw of θ, and `accept` gives the acceptance rate of the random walk chain.
- Here, the acceptance rate is 0.286, which is close to the desired acceptance rate for this Metropolis random walk algorithm.

```
> require(LearnBayes)
> proposal <- list(var=post.cov,scale=2)
> set.seed(1)
> fit1 <- rwmetrop(h,proposal,start,10000,data)
> fit1$accept
[1] 0.286
```

8.5 Example: Learning from grouped data

- We can summarize the parameters μ and λ by computing the posterior means and standard deviations using the last 5000 simulated draws.

```
> (post.means <- apply(fit1$par[5001:10000,],2,mean))
[1] 70.1767028  0.9809451
> (post.sds <- apply(fit1$par[5001:10000,],2,sd))
[1] 0.17922588 0.05626142
```

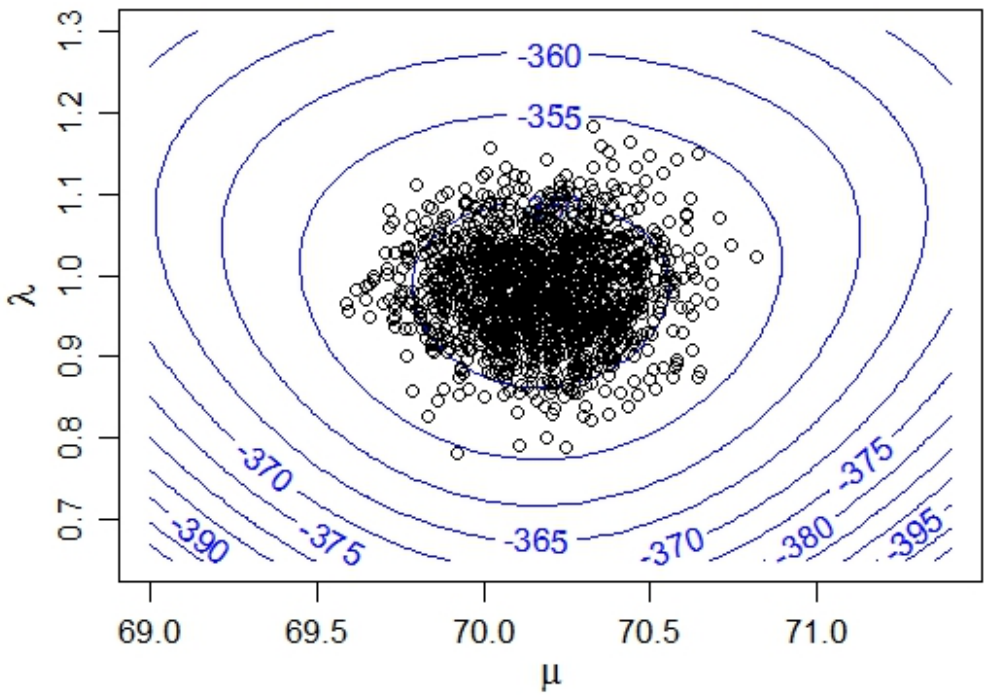
  The corresponding values in the normal approximation are

```
> post.mode
[1] 70.169880  0.973644
> sqrt(diag(post.cov))
[1] 0.18800834 0.05609341
```

- There is close agreement between the two sets of posterior moments which indicates that the normal approximation is reasonably accurate.

8.5 Example: Learning from grouped data

- The figure below shows a contour plot of the joint posterior of μ and λ with the last 5000 simulated draws from the random walk Metropolis algorithm drawn on top.
- The contour lines have an elliptical shape that confirms the accuracy of the normal approximation in this example.



8.5 Example: Learning from grouped data

- We illustrate MCMC output analysis using the R package coda.
- Suppose we rerun the Metropolis random walk algorithm with poor choices of starting value and proposal density.
- As a starting value, we choose $(\mu, \lambda) = (65, 1)$ (choice of μ is too small) and the small scale factor of 0.2 instead of 2. Hereafter, we refer to this modified algorithm as "Algorithm 2" and the original algorithm as "Algorithm 1".

```
> start <- c(65,1)
> proposal <- list(var=post.cov,scale=0.2)
> set.seed(1)
> fit2 <- rwmetrop(h,proposal,start,10000,data)
> fit2$accept
[1] 0.8874
```

8.5 Example: Learning from grouped data

- The acceptance rate of Algorithm 2 is 0.89, which is much larger than the 0.29 rate that we found using the scale factor 2.
- To analyze the draws using `coda`, we first convert the simulated values into MCMC objects. The function `mcmc` in `coda` takes as input a vector or a matrix where each column corresponds to a different variable.

```
require(coda)
colnames(fit1$par) <- c("mu","lambda")
colnames(fit2$par) <- c("mu","lambda")
mcmcobj1 <- mcmc(fit1$par)
mcmcobj2 <- mcmc(fit2$par)
```

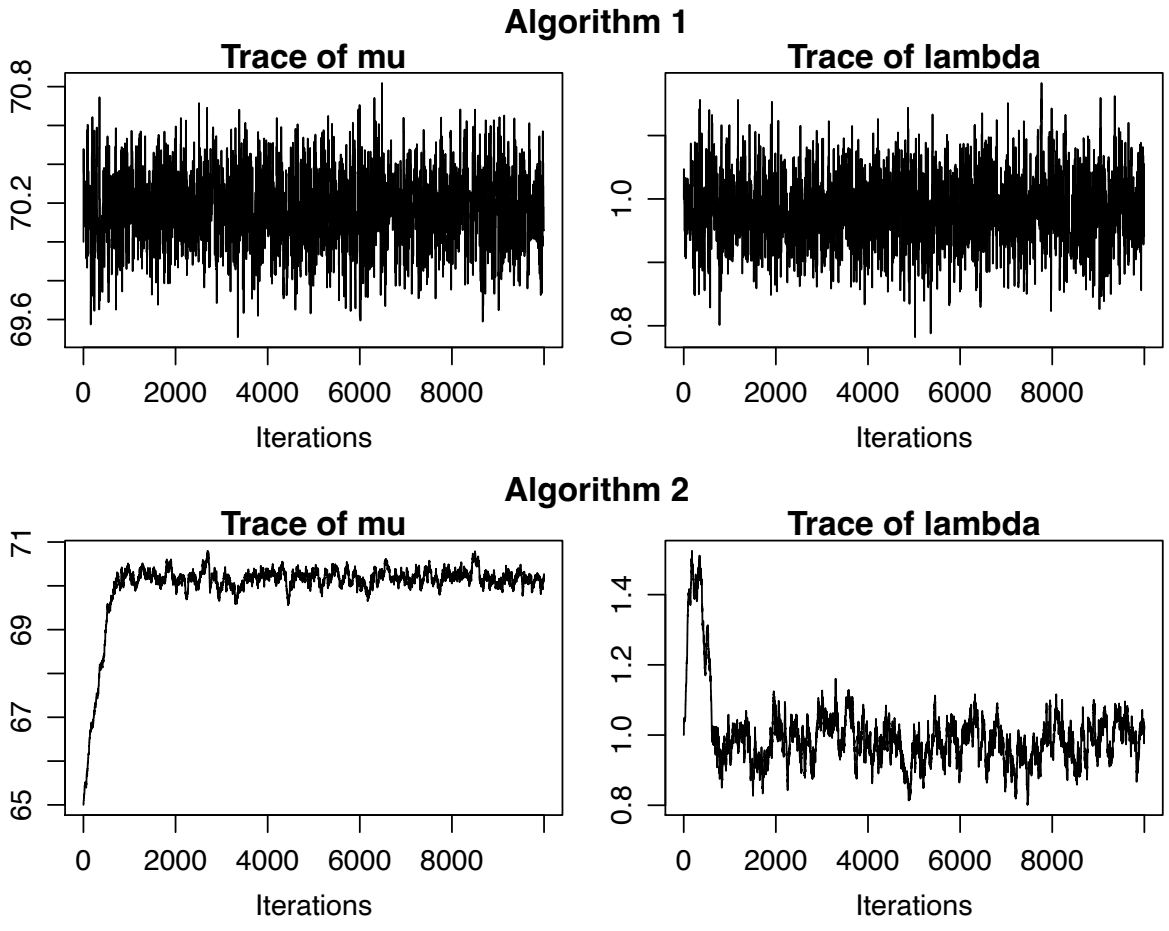
The MCMC objects `mcmcobj1` and `mcmcobj2` correspond to Algorithms 1 and 2 respectively.

8.5 Example: Learning from grouped data

- Trace plots of the simulated draws of μ and λ can be obtained using the `traceplot` function in `coda`.

```
par(mfrow=c(1,2),
  oma=c(0,0,1,0))
traceplot(mcmcobj1)
title("Algorithm 1",
  outer=TRUE)

par(mfrow=c(1,2),
  oma=c(0,0,1,0))
traceplot(mcmcobj2)
title("Algorithm 2",
  outer=TRUE)
```

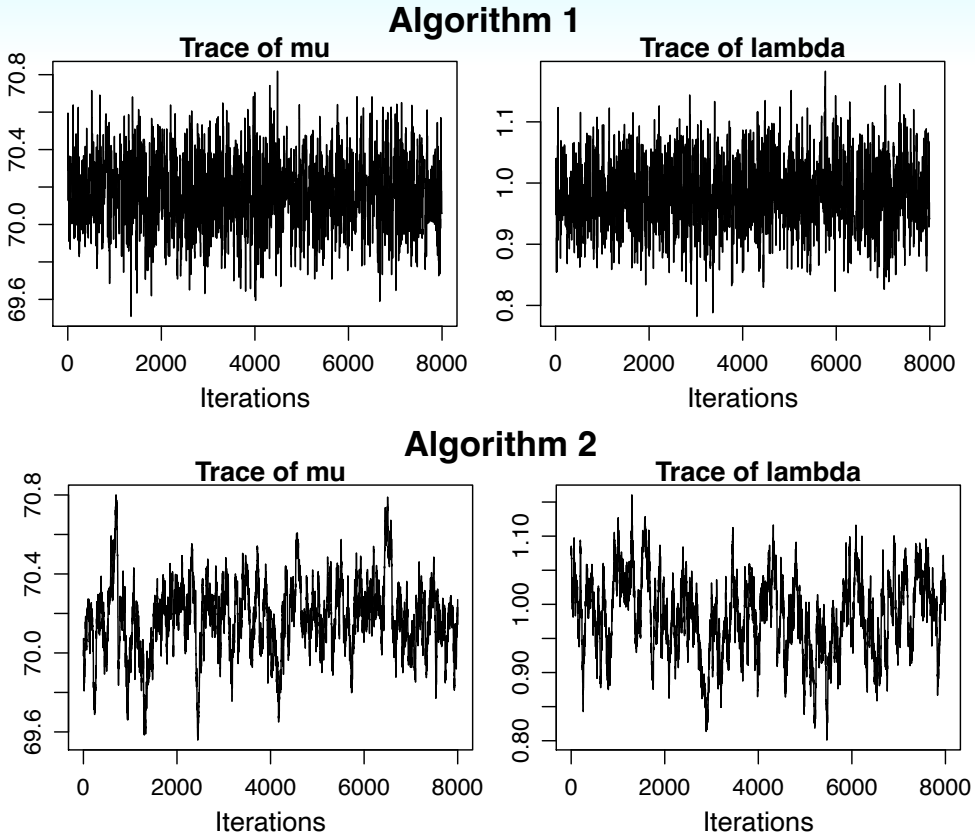


8.5 Example: Learning from grouped data

- The original algorithm converges quickly while convergence is slow for the modified algorithm due to the poor starting values.
- Let us discard the first 2000 iterations as burn-in for both algorithms and look at the trace plots again.

```
mcmcobj1 <- mcmc(fit1$par[2001:10000,])
mcmcobj2 <- mcmc(fit2$par[2001:10000,])
```

8.5 Example: Learning from grouped data

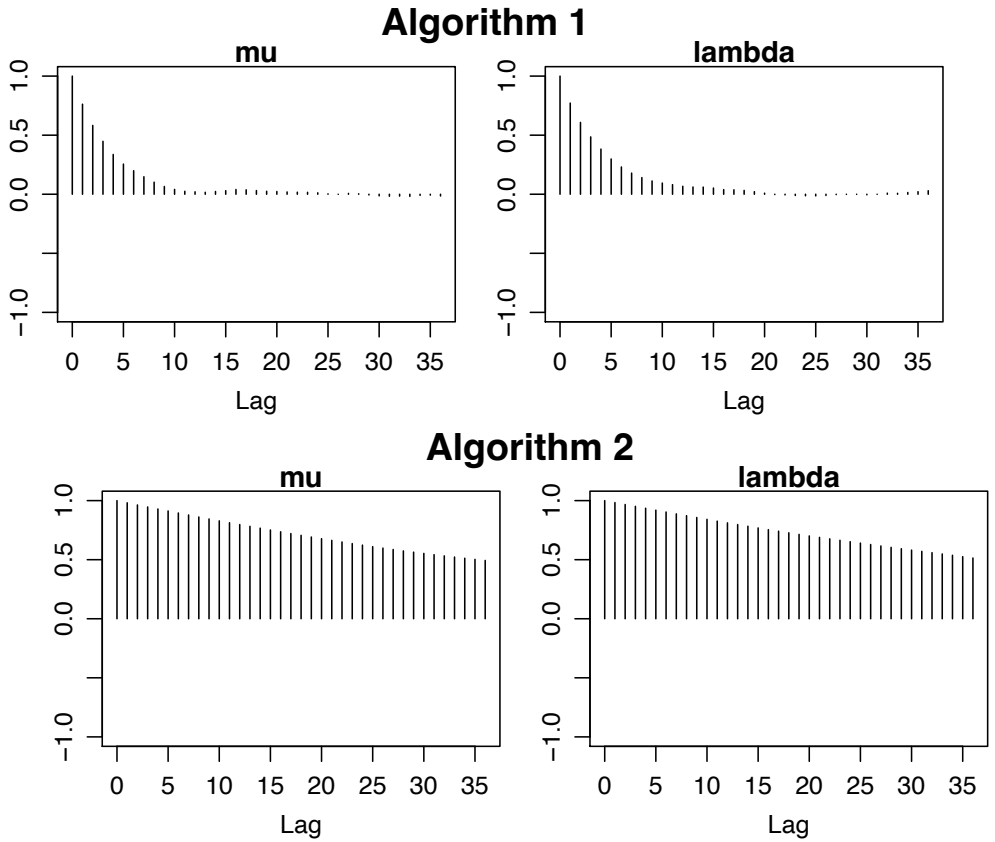


- The trace plot of the simulated streams of μ and λ look more like random noise for Algorithm 1.
- For Algorithm 2, the simulated draws appear to have reached the stationary distribution. However the simulated sequence appears irregular; e.g. the iterates will explore the region $\mu > 70.5$ for a while before returning to the center of the distribution.

8.5 Example: Learning from grouped data

- Autocorrelation plots can be produced by the `autocorr.plot` function in `coda`.

```
autocorr.plot(mcmcobj1)
autocorr.plot(mcmcobj2)
```



- The simulated sequences in Algorithm 2 has strong correlation structure; the autocorrelations are close to one for lag one and reduce very slowly as a function of the lag. For Algorithm 1, the lag one

8.5 Example: Learning from grouped data

- Summary statistics of the simulated draws can be obtained by taking the `summary` of the MCMC objects.

```
> summary(mcmcobj1)
Iterations = 1:8000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 8000
1. Empirical mean and standard deviation for each
   variable, plus standard error of the mean:
          Mean      SD  Naive SE Time-series SE
mu     70.1739 0.18318 0.0020480       0.005575
lambda  0.9792 0.05494 0.0006143       0.001795

2. Quantiles for each variable:
          2.5%     25%     50%    75%  97.5%
mu     69.8144 70.0477 70.1746 70.297 70.524
lambda  0.8703  0.9423  0.9755  1.018  1.087
```

8.5 Example: Learning from grouped data

```
> summary(mcmcobj2)

Iterations = 1:8000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 8000

1. Empirical mean and standard deviation for each
   variable, plus standard error of the mean:
          Mean      SD  Naive SE Time-series SE
mu     70.1766 0.18106 0.0020243       0.021463
lambda  0.9828 0.05688 0.0006359       0.006988

2. Quantiles for each variable:
          2.5%     25%     50%    75%  97.5%
mu     69.7932 70.0661 70.1806 70.291 70.544
lambda  0.8688  0.9447  0.9851  1.023  1.089
```

8.5 Example: Learning from grouped data

- The `Naive SE` in the summary statistics assumes that the sampled values are independent and is computed by taking the standard deviation of the iterates (computed using `sd`) divided by the square root of the number of iterates.
- The `Time-series SE` take into account the autocorrelation among the iterates and is computed by taking the standard deviation of the iterates divided by the "effective sample size" (computed using `effectiveSize` in `coda`). The "effective sample size" can be interpreted as the number of independent Monte Carlo samples necessary to give the same precision as the MCMC samples.
- The graphs and the summary statistics confirm the better performance of the MCMC chain with a starting value $(\mu, \lambda) = (70, 1)$ and scale factor of 2.