Computer Lab 4

Elon Brange, Ludwig Thaung

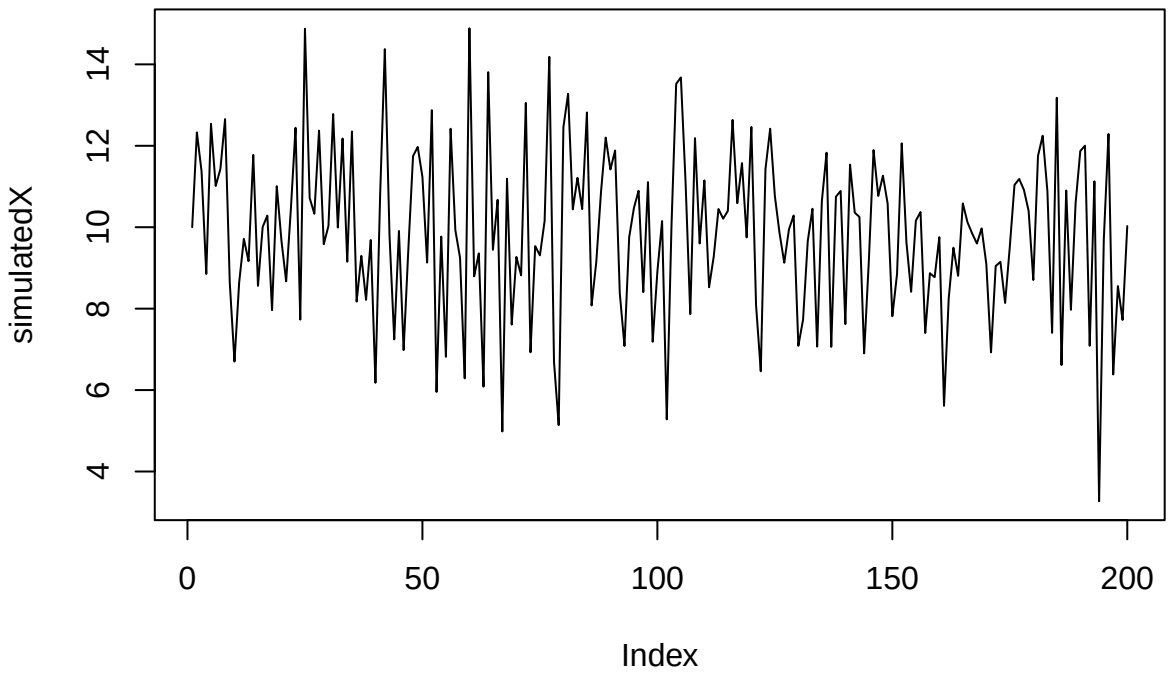
5/23/2019

Computer Lab 4

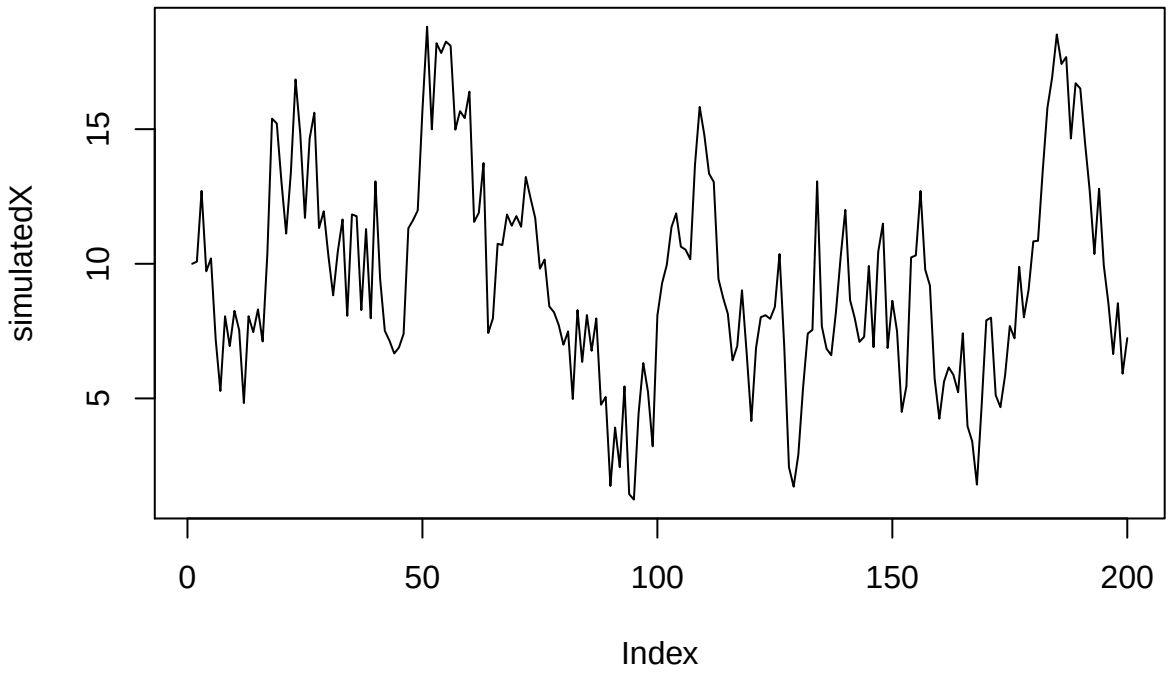
##1

1 a)

```
ARfunction <- function(n, mu, omega, sigma2) {
  x = matrix(0, n, 1)
  x[1, 1] = mu
  noise <- rnorm(n-1,0,sigma2)
  for(i in 1:(n-1)) {
    x[i+1, 1] = mu + omega*(x[i, 1] - mu) + noise[i]
  }
  return(x)
}

simulatedX <- ARfunction(200, 10, -11/10 + 1, 2)
plot(simulatedX, type='l')
```




```
simulatedX <- ARfunction(200, 10, -11/10 + 2, 2)
plot(simulatedX, type='l')
```



The effect of Phi on X1:T is that, depending on the value of Phi determines X1:T's autocorrelation with the previous value of the difference between X1:T-1 and the mean(mu). I.e. what omega does is how much the previous value affect the current value and in which direction.

1 b)

```
simulatedX <- c(ARfunction(1000, 10, 0.3, 1))
simulatedY <- c(ARfunction(1000, 10, 0.95, 1))

Nx = length(simulatedX)
Ny = length(simulatedY)

x_dat <- list(N = Nx,
              X = simulatedX)

y_dat <- list(N = Ny,
              X = simulatedY)

fitX <- stan(file = 'mystan.stan', data = x_dat)
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info (non-fatal): Comments beginning with # are deprecated.  Please use // in place of # for line co
```

```
print(fitX)
```

```
## Inference for Stan model: mystan.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## muRandom       9.98    0.00 0.05    9.89    9.95    9.98   10.01   10.07
## sigma2Random   1.00    0.00 0.02    0.96    0.98    1.00    1.01    1.05
```

```
## omegaRandom     0.32    0.00 0.03    0.26    0.30    0.32    0.34    0.38
## lp__         -496.97    0.03 1.24 -500.31 -497.50 -496.64 -496.07 -495.56
##              n_eff Rhat
## muRandom      2771    1
## sigma2Random  4347    1
## omegaRandom   4303    1
## lp__          2019    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 11:52:22 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
summaryFitX = summary(fitX)$summary
dataX = extract(fitX)

meanX = summaryFitX[,1]
highX = summaryFitX[,8]
lowX = summaryFitX[,4]
efficientX = summaryFitX[, 9]

print(paste0("The 95% interval for mu is: ", lowX[1], " - ", highX[1]))
```

```
## [1] "The 95% interval for mu is: 9.88810295021301 - 10.066740566065"
```

```
print(paste0(" and the mean for mu is: ", meanX[1]))
```

```
## [1] " and the mean for mu is: 9.9788840975677"
```

```
print(paste0("The 95% interval for sigma is: ", lowX[2], " - ", highX[2]))
```

```
## [1] "The 95% interval for sigma is: 0.955280658829604 - 1.04557304166958"
```

```
print(paste0(" and the mean for sigma is: ", meanX[2]))
```

```
## [1] " and the mean for sigma is: 0.998334194814575"
```

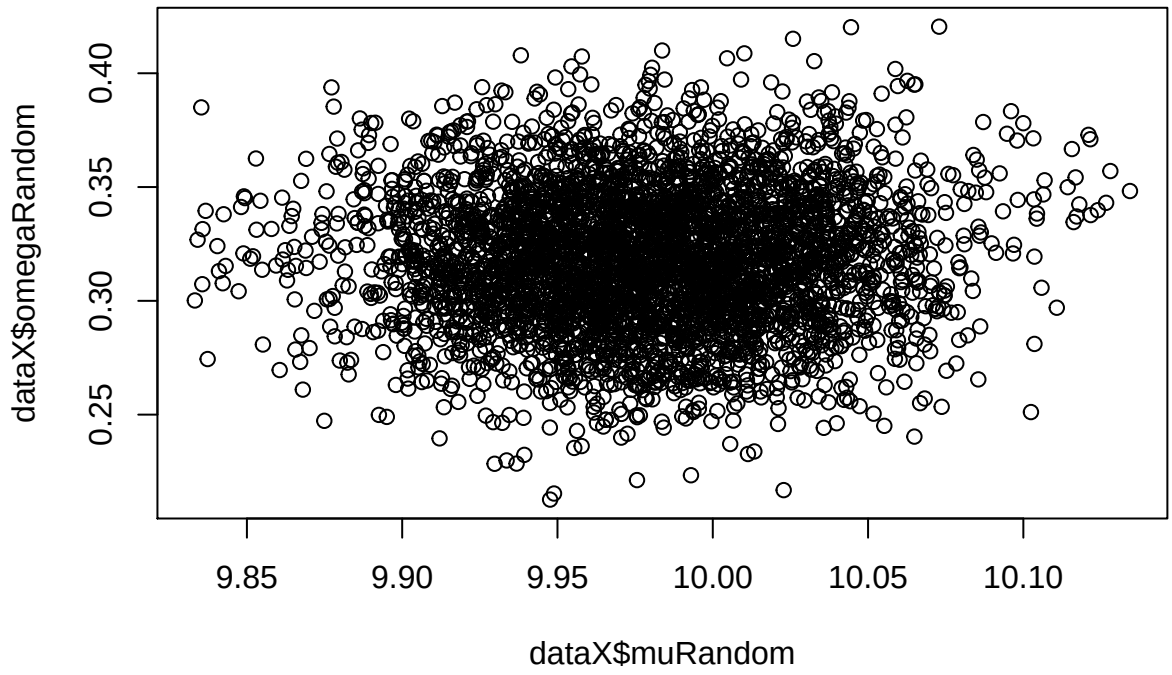
```
print(paste0("The 95% interval for omega is: ", lowX[3], " - ", highX[3]))
```

```
## [1] "The 95% interval for omega is: 0.26124444813333 - 0.37852625241563"
```

```
print(paste0(" and the mean for omega is: ", meanX[3]))
```

```
## [1] " and the mean for omega is: 0.318332952302275"
```

```
#print(meanX)
#print(highX)
#print(lowX)
#print(efficientX)

plot(dataX$muRandom, dataX$omegaRandom)
```




```
library(MASS)
den3d <- kde2d(dataX$muRandom, dataX$omegaRandom)
#library(plotly)
#plot_ly(x=den3d$x, y=den3d$y, z=den3d$z/length(dataX$muRandom)) %>% add_surface()

library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: dotCall64
```

```
## Loading required package: grid
```

```
## Spam version 2.2-2 (2019-03-07) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

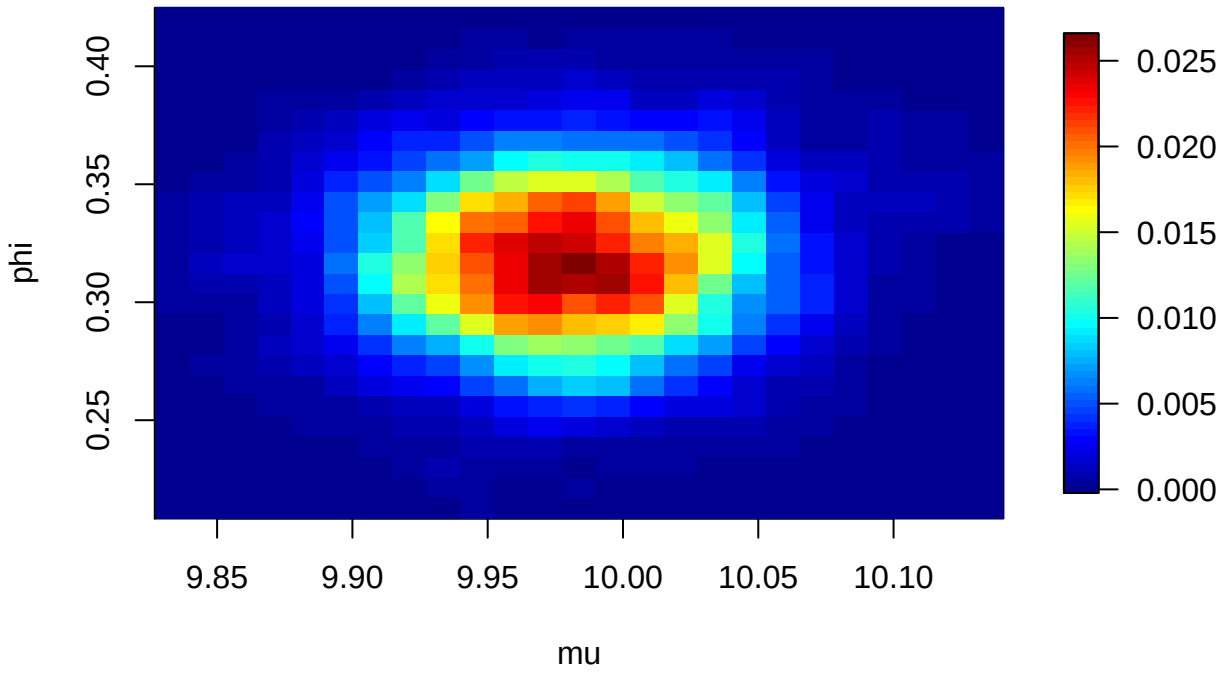
```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

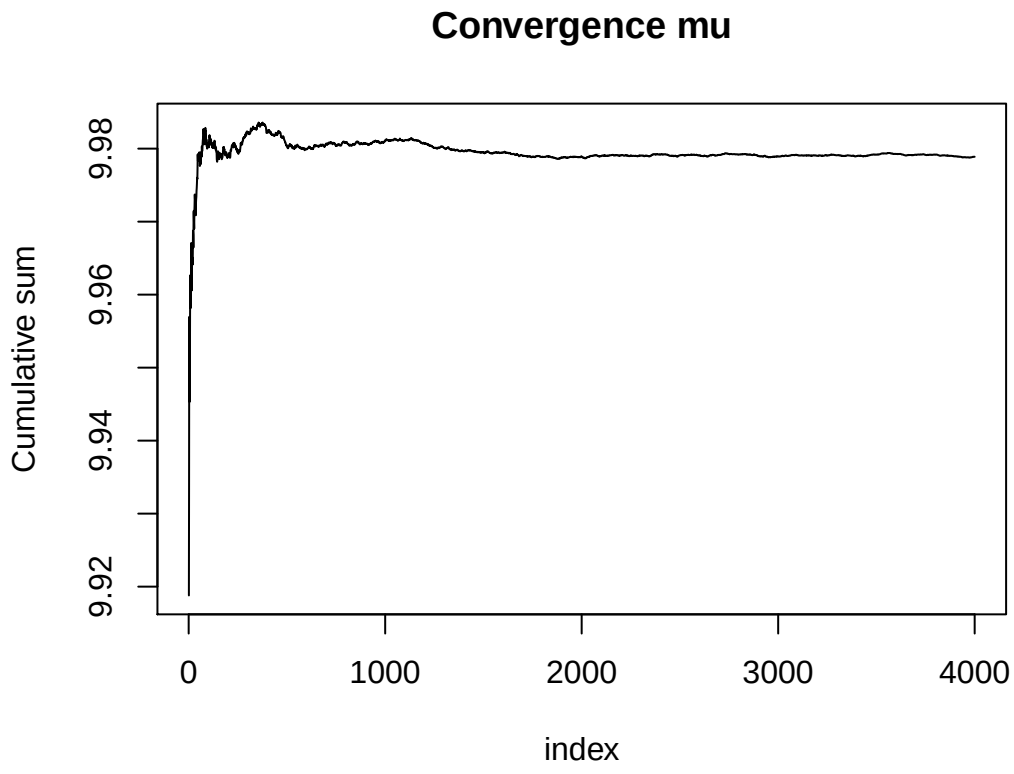
```
## Loading required package: maps
```

```
## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code
```

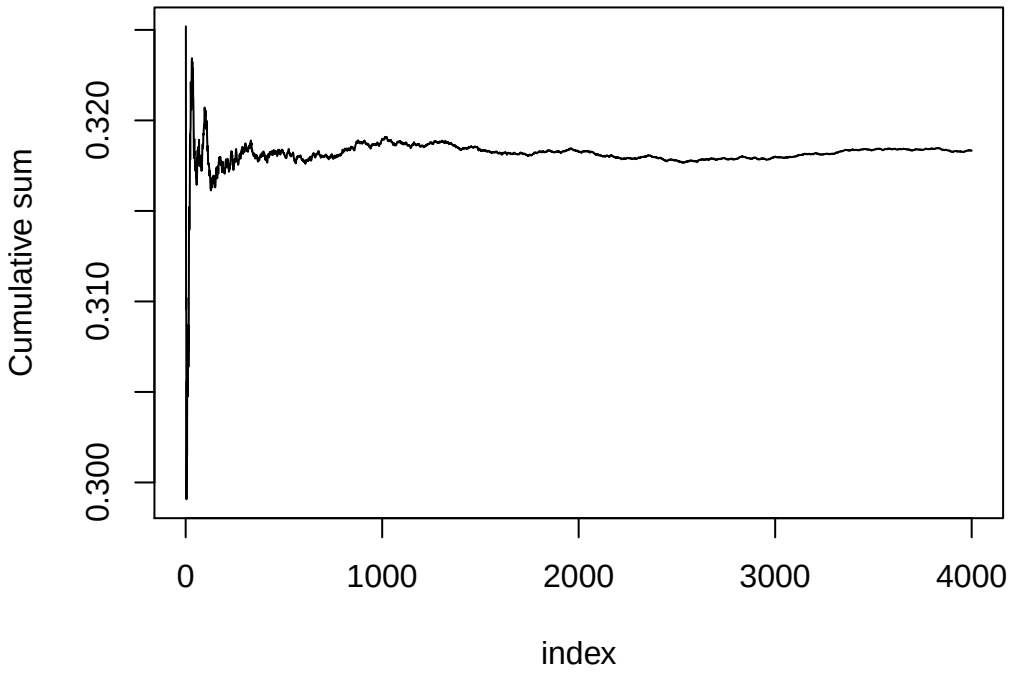
```
image.plot(den3d$x,den3d$y,den3d$z/length(dataX$muRandom),xlab="mu", ylab="phi")
```




```
plot(cumsum(dataX$muRandom)/seq(1,length(dataX$muRandom)), type='l', xlab ="index", ylab = "Cumulative s
```




```
plot(cumsum(dataX$omegaRandom)/seq(1,length(dataX$omegaRandom)), type='l', xlab = "index", ylab="Cumulat
```




```
fitY <- stan(file = 'mystan.stan', data = y_dat)
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info (non-fatal): Comments beginning with # are deprecated.  Please use // in place of # for line co
```

```
print(fitY)
```

```
## Inference for Stan model: mystan.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                 mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## muRandom        9.09    0.01 0.67    7.74    8.67    9.10    9.51   10.41
## sigma2Random    0.99    0.00 0.02    0.95    0.98    0.99    1.01    1.04
## omegaRandom     0.95    0.00 0.01    0.93    0.94    0.95    0.96    0.97
## lp__         -492.00    0.03 1.27 -495.36 -492.60 -491.66 -491.06 -490.54
##              n_eff Rhat
## muRandom      2625    1
## sigma2Random  3193    1
## omegaRandom   3159    1
## lp__          1801    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 11:52:28 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
summaryFitY = summary(fitY)$summary
dataY = extract(fitY)

meanY = summaryFitY[,1]
```

```r
highY = summaryFitY[,8]
lowY = summaryFitY[,4]
efficientY = summaryFitY[, 9]

print(paste0("The 95% interval for mu is: ", lowY[1], " - ", highY[1]))
```

```
## [1] "The 95% interval for mu is: 7.73562081703205 - 10.4123363175313"
```

```r
print(paste0(" and the mean for mu is: ", meanY[1]))
```

```
## [1] " and the mean for mu is: 9.0941361223012"
```

```r
print(paste0("The 95% interval for sigma is: ", lowY[2], " - ", highY[2]))
```

```
## [1] "The 95% interval for sigma is: 0.94886108395219 - 1.03711159445509"
```

```r
print(paste0(" and the mean for sigma is: ", meanY[2]))
```

```
## [1] " and the mean for sigma is: 0.992469917416263"
```

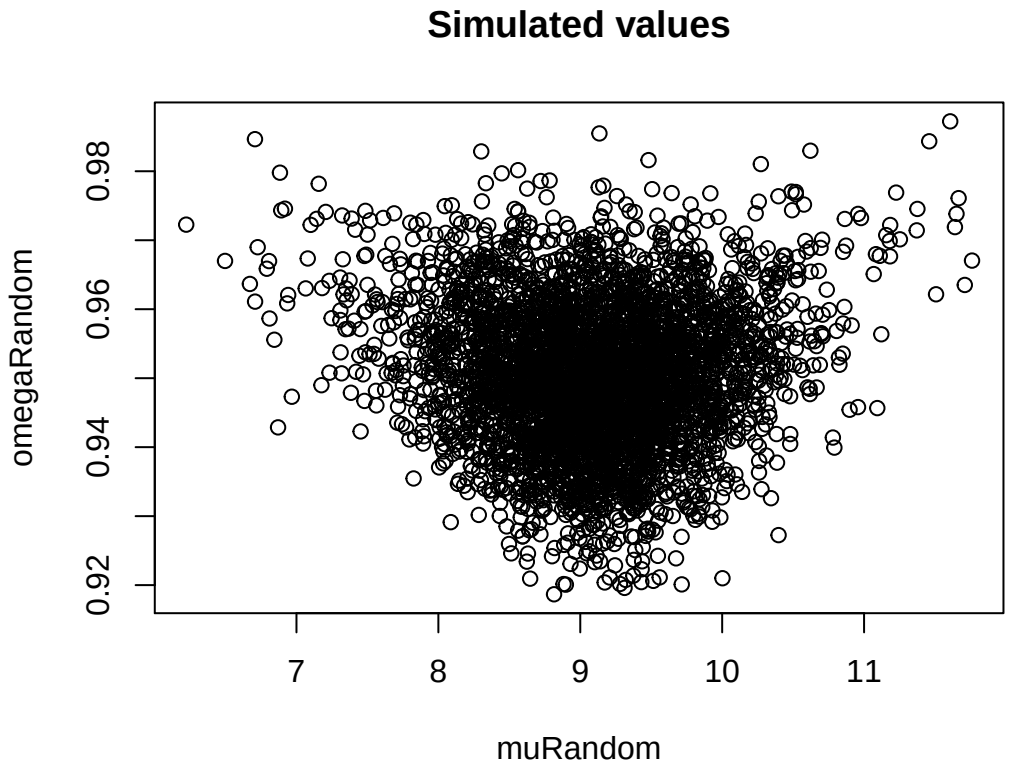
```r
print(paste0("The 95% interval for omega is: ", lowY[3], " - ", highY[3]))
```

```
## [1] "The 95% interval for omega is: 0.930474879736404 - 0.971066969694039"
```

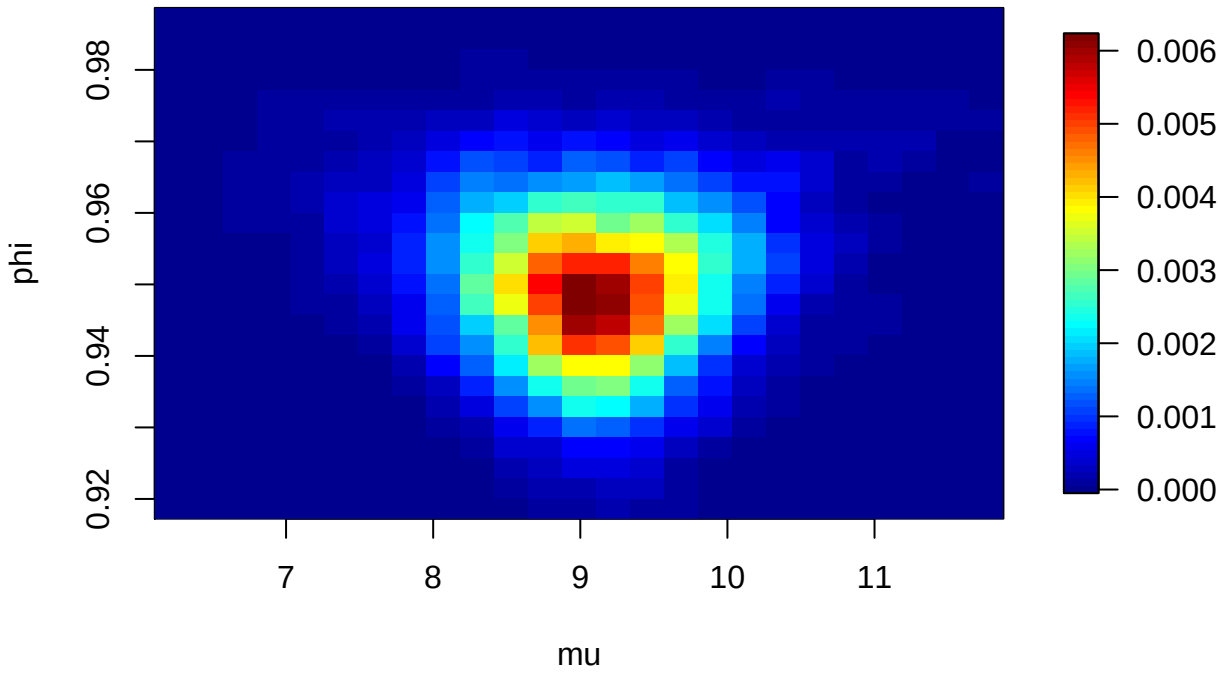
```r
print(paste0(" and the mean for omega is: ", meanY[3]))
```

```
## [1] " and the mean for omega is: 0.950358506193888"
```

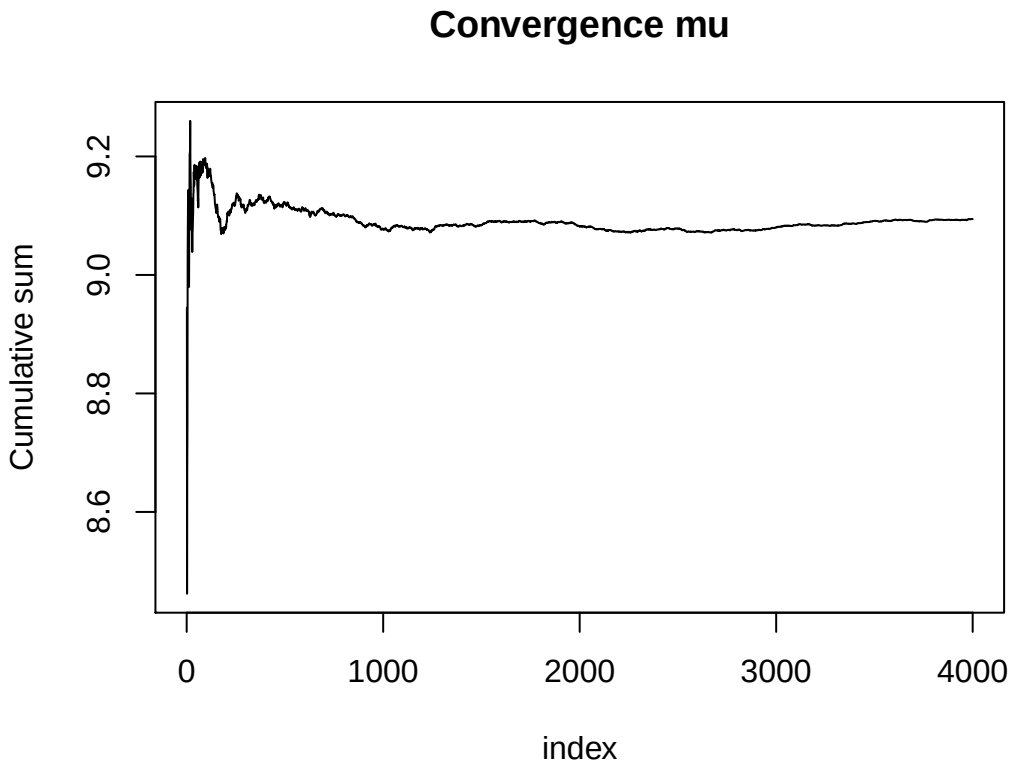
```r
#print(meanY)
#print(highY)
#print(lowY)
#print(efficientY)

plot(dataY$muRandom, dataY$omegaRandom, xlab ="muRandom", ylab ="omegaRandom", main = "Simulated values
```



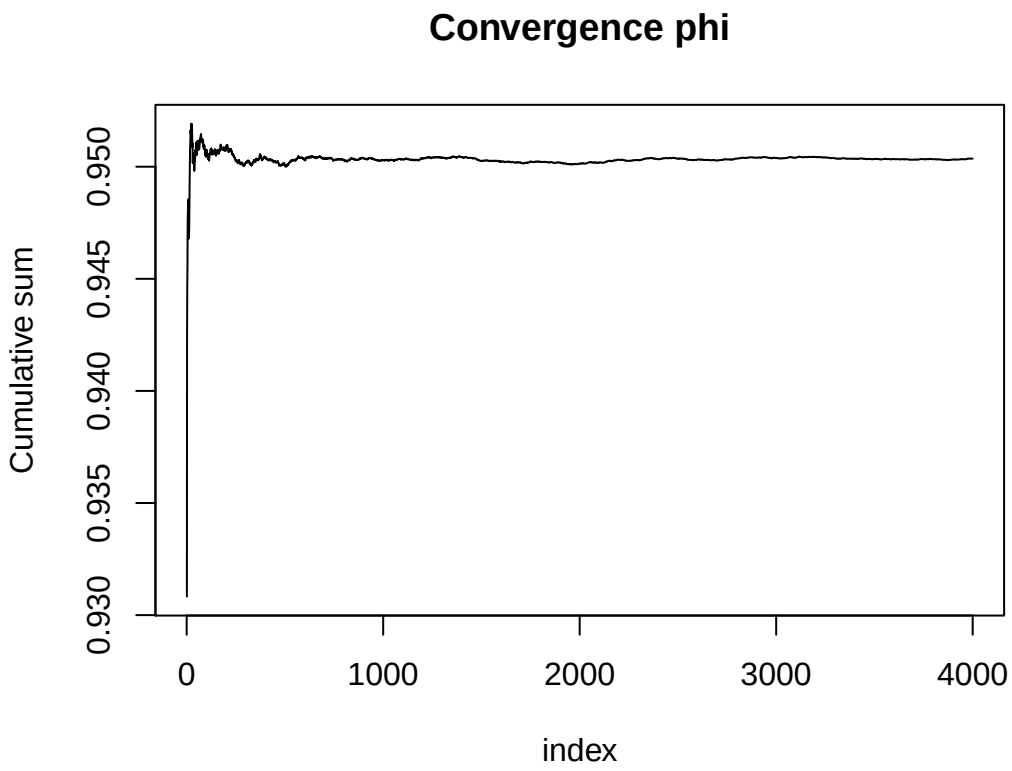

```
library(MASS)
den3d <- kde2d(dataY$muRandom, dataY$omegaRandom)
#library(plotly)
#plot_ly(x=den3d$x, y=den3d$y, z=den3d$z/length(dataY$muRandom)) %>% add_surface()

library(fields)
image.plot(den3d$x,den3d$y,den3d$z/length(dataX$muRandom), xlab="mu", ylab="phi")
```



```
plot(cumsum(dataY$muRandom)/seq(1,length(dataY$muRandom)), type='l', , xlab ="index", ylab ="Cumulative
```




```
plot(cumsum(dataY$omegaRandom)/seq(1,length(dataY$omegaRandom)), type='l', , xlab ="index", ylab ="Cumul
```



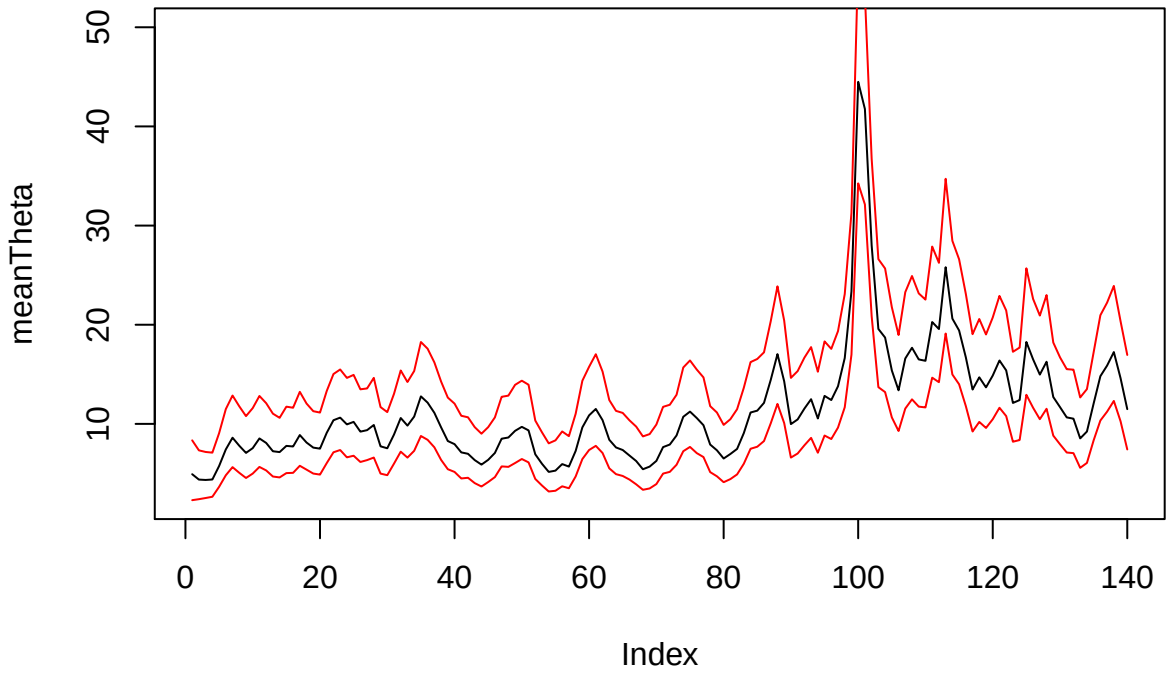
i) When compared to the values in SimulatedX & SimulatedY we can see that we are able to estimate the

true values.

ii) The two plots show that the parameters converges close to the real value quite quickly.

1 c)

```
campyData<-c(t(read.table("campy.dat",header=TRUE)))

Ny = length(campyData)

campy_dat <- list(N = Ny,
                  y = campyData)
fitP <- stan(file = 'poisson.stan', data = campy_dat)

summaryFitP = summary(fitP)$summary
rowsCols = dim(summaryFitP)
highTheta = c()
meanTheta = c()
lowTheta = c()
for(i in 1:(rowsCols[1]-4)) {
  highTheta = c(highTheta, exp(summaryFitP[i, 8]))
  meanTheta = c(meanTheta, exp(summaryFitP[i, 1]))
  lowTheta = c(lowTheta, exp(summaryFitP[i, 4]))
}

plot(meanTheta, type='l', ylim=c(min(lowTheta), 50))
lines(highTheta, col='red')
lines(lowTheta, col='red')
```



1

d)

```
campy_dat <- list(N = Ny,
                  y = campyData,
```

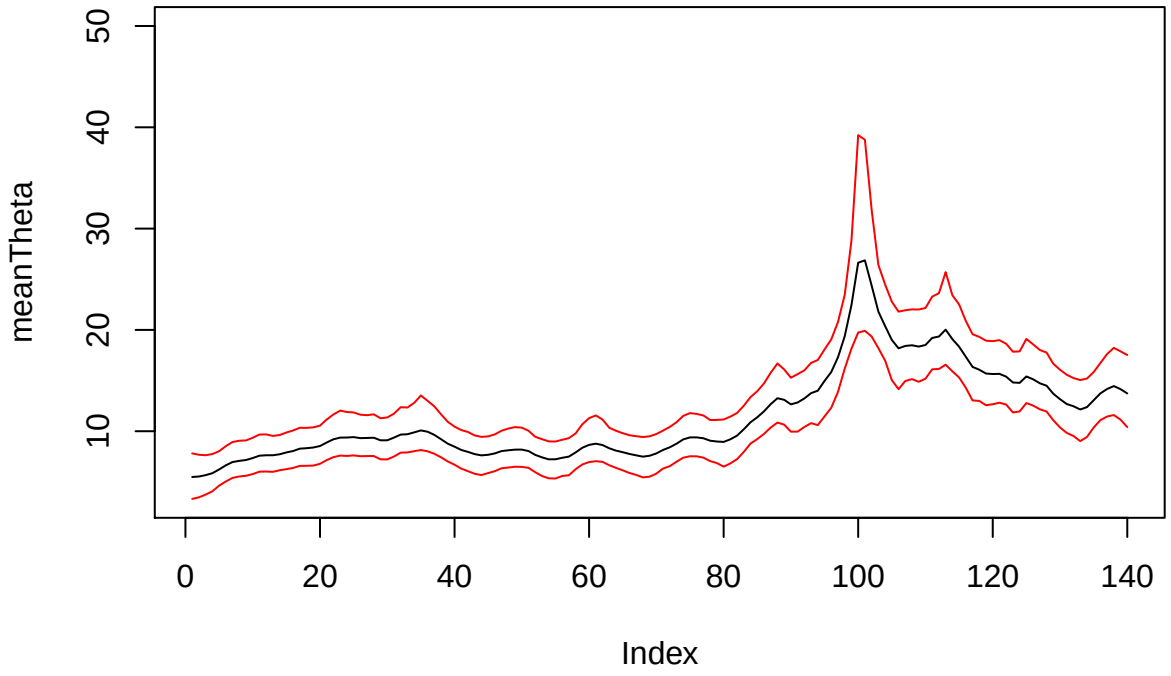
```
                  sigma2Nu = 50,
                  sigma2Sigma = 0.01)
fitP <- stan(file = 'poissonPrior.stan', data = campy_dat)
```

```
## Warning: There were 28 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was low. S
## http://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
summaryFitP = summary(fitP)$summary
rowsCols = dim(summaryFitP)
rowsCols[1] = rowsCols[1] - 4
highTheta = c()
meanTheta = c()
lowTheta = c()
for(i in 1:rowsCols[1]) {
  highTheta = c(highTheta, exp(summaryFitP[i, 8]))
  meanTheta = c(meanTheta, exp(summaryFitP[i, 1]))
  lowTheta = c(lowTheta, exp(summaryFitP[i, 4]))
}

plot(meanTheta, type='l', ylim=c(min(lowTheta), 50))
lines(highTheta, col='red')
lines(lowTheta, col='red')
```



The posterior has become less volatile.