E3

```r
rm(list = ls())
library(rjags)
library(coda)
library(pander)
setwd("c:/e/brucebcampbell-git/bayesian-learning-with-R/E3")
load("heatwaves.RData")
n.chains = 2
nSamples = 20000
load("HWD2.RData")
```

Fit JAGS Poisson Random Effects

```r
######################################## Fit JAGS Poisson
model_pois = '
model
{
    ## Likelihood
    for(i in 1:N){
      for(j in 1:9){
        Y[i,j] ~ dpois(lambda[i,j])
        log(lambda[i,j]) <- mu[i,j]
        mu[i,j] <- alpha[j] + beta[j]*t[i]
      }
    }

  ## Priors
  for(i in 1:9){
    alpha[i] ~ dnorm(0,taus[i])
    taus[i] ~ dgamma(0.1,0.1)
  }

  # Slopes
  for(i in 1:9){
    beta[i] ~ dnorm(mu.beta,taus.beta[i])
    taus.beta[i] ~ dgamma(0.1,0.1)
  }

  ## Posterior Predictive Checks
  for(i in 1:N){
    for(j in 1:9){
        Y2[i,j] ~ dpois(lambda[i,j])
    }
  }

  for(j in 1:9){
    Dm[j] <- mean(Y2[,j])
    Dsd[j] <- sd(Y2[,j])
  }
```

```
}
'

  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num,mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercept=.(
  # Choose the parameters to watch
  model_parameters =  c("beta", "alpha","Dm", "Dsd")
  model_pois <- jags.model(textConnection(model_pois),data = model_data,n.chains = n.chains)#Compile Mo
```

Compiling model graph Resolving undeclared variables Allocating nodes Graph information: Observed stochastic nodes: 369 Unobserved stochastic nodes: 405 Total graph size: 1953

Initializing model

```
  update(model_pois, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model_pois, variable.names=model_parameters,n.iter=2*nSamples)
  #plot(out.coda)
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
```

[[1]]

```
Stationarity start     p-value
test         iteration
```

Dm[1] passed 1 0.3191 Dm[2] passed 1 0.8361 Dm[3] passed 1 0.3140 Dm[4] passed 1 0.9060 Dm[5] passed 1 0.4657 Dm[6] passed 1 0.1854 Dm[7] passed 4001 0.0649 Dm[8] passed 1 0.3492 Dm[9] passed 1 0.3951 Dsd[1] passed 1 0.4946 Dsd[2] passed 1 0.1119 Dsd[3] passed 1 0.1336 Dsd[4] passed 1 0.7212 Dsd[5] passed 1 0.5530 Dsd[6] passed 1 0.9464 Dsd[7] passed 4001 0.1591 Dsd[8] passed 1 0.8394 Dsd[9] passed 1 0.6465 alpha[1] passed 1 0.5220 alpha[2] passed 1 0.5954 alpha[3] passed 1 0.1904 alpha[4] passed 1 0.7480 alpha[5] passed 1 0.9070 alpha[6] passed 1 0.4452 alpha[7] passed 1 0.0864 alpha[8] passed 1 0.6833 alpha[9] passed 1 0.8371 beta[1] passed 1 0.3744 beta[2] passed 1 0.5687 beta[3] passed 1 0.2256 beta[4] passed 1 0.8755 beta[5] passed 1 0.8629 beta[6] passed 1 0.4901 beta[7] passed 1 0.1490 beta[8] passed 1 0.8115 beta[9] passed 1 0.9164

```
Halfwidth Mean      Halfwidth
test
```

Dm[1] passed 0.732653 0.001905 Dm[2] passed 1.926753 0.003103 Dm[3] passed 1.061794 0.002296 Dm[4] passed 0.859227 0.002100 Dm[5] passed 0.925205 0.002442 Dm[6] passed 1.691604 0.002934 Dm[7] passed 0.635030 0.001846 Dm[8] passed 0.513565 0.001808 Dm[9] passed 0.875134 0.002172 Dsd[1] passed 0.860919 0.001657 Dsd[2] passed 1.525817 0.003387 Dsd[3] passed 1.034865 0.001640 Dsd[4] passed 0.951524 0.002199 Dsd[5] passed 1.072960 0.002797 Dsd[6] passed 1.331543 0.002013 Dsd[7] passed 0.812528 0.001835 Dsd[8] passed 0.713208 0.001424 Dsd[9] passed 0.941503 0.001587 alpha[1] failed -0.031156 0.008327 alpha[2] failed 0.002360 0.009028 alpha[3] passed -0.154889 0.009478 alpha[4] passed 0.273975 0.008183 alpha[5] passed -1.202588 0.022373 alpha[6] passed 0.204909 0.008365 alpha[7] failed 0.000168 0.008193 alpha[8] passed -0.849688 0.017594 alpha[9] passed -0.424558 0.012417 beta[1] passed -0.015475 0.000384 beta[2] passed 0.027990 0.000342 beta[3] passed 0.008956 0.000386 beta[4] passed -0.023380 0.000395 beta[5] passed 0.045467 0.000758 beta[6] passed 0.014003 0.000328 beta[7] passed -0.025437 0.000407 beta[8] failed 0.006316 0.000707 beta[9] passed 0.012149 0.000485

[[2]]

```
Stationarity start     p-value
test         iteration
```

Dm[1] passed 1 0.6040 Dm[2] passed 1 0.5346 Dm[3] passed 1 0.3115 Dm[4] passed 1 0.4400 Dm[5] passed 1 0.0792 Dm[6] passed 1 0.1024 Dm[7] passed 1 0.6596 Dm[8] passed 1 0.8707 Dm[9] passed 1 0.1474 Dsd[1] passed 1 0.8468 Dsd[2] passed 1 0.0912 Dsd[3] passed 1 0.2677 Dsd[4] passed 1 0.3489 Dsd[5] passed 1 0.4536

Dsd[6] passed 1 0.4766 Dsd[7] passed 1 0.7985 Dsd[8] passed 1 0.8432 Dsd[9] passed 1 0.5829 alpha[1] passed 1 0.2041 alpha[2] passed 1 0.9589 alpha[3] passed 1 0.5741 alpha[4] passed 1 0.2625 alpha[5] passed 1 0.2996 alpha[6] passed 1 0.2101 alpha[7] passed 1 0.9793 alpha[8] passed 1 0.5703 alpha[9] passed 1 0.7940 beta[1] passed 1 0.2481 beta[2] passed 1 0.9236 beta[3] passed 1 0.4904 beta[4] passed 1 0.3476 beta[5] passed 1 0.2970 beta[6] passed 1 0.2261 beta[7] passed 1 0.9894 beta[8] passed 1 0.5222 beta[9] passed 1 0.6972

```
Halfwidth Mean      Halfwidth
test
```

Dm[1] passed 0.734699 0.001891 Dm[2] passed 1.926162 0.003100 Dm[3] passed 1.060812 0.002283 Dm[4] passed 0.861309 0.002074 Dm[5] passed 0.926251 0.002454 Dm[6] passed 1.690026 0.002927 Dm[7] passed 0.632301 0.001724 Dm[8] passed 0.512541 0.001896 Dm[9] passed 0.876562 0.002104 Dsd[1] passed 0.863944 0.001780 Dsd[2] passed 1.523306 0.003195 Dsd[3] passed 1.034772 0.001670 Dsd[4] passed 0.953034 0.002199 Dsd[5] passed 1.068634 0.002562 Dsd[6] passed 1.330278 0.002052 Dsd[7] passed 0.810871 0.001740 Dsd[8] passed 0.712640 0.001467 Dsd[9] passed 0.942767 0.001597 alpha[1] failed -0.027345 0.008202 alpha[2] failed 0.004483 0.008479 alpha[3] passed -0.161193 0.009577 alpha[4] passed 0.272307 0.007905 alpha[5] passed -1.181156 0.021498 alpha[6] passed 0.206314 0.008090 alpha[7] failed 0.000622 0.008068 alpha[8] passed -0.857457 0.018159 alpha[9] passed -0.418150 0.012225 beta[1] passed -0.015577 0.000383 beta[2] passed 0.027892 0.000318 beta[3] passed 0.009210 0.000390 beta[4] passed -0.023214 0.000383 beta[5] passed 0.044776 0.000721 beta[6] passed 0.013901 0.000317 beta[7] passed -0.025507 0.000403 beta[8] failed 0.006497 0.000701 beta[9] passed 0.011905 0.000480

```r
# check that our chain???s length is satisfactory.
raftery.diag(out.coda)
```

[[1]]

Quantile (q) = 0.025 Accuracy (r) = +/- 0.005 Probability (s) = 0.95

```
Burn-in  Total Lower bound  Dependence
(M)      (N)   (Nmin)       factor (I)
```

Dm[1] 2 4052 3746 1.08
Dm[2] 2 4394 3746 1.17
Dm[3] 2 4760 3746 1.27
Dm[4] 2 3854 3746 1.03
Dm[5] 2 4054 3746 1.08
Dm[6] 2 4214 3746 1.12
Dm[7] 2 5817 3746 1.55
Dm[8] 2 5235 3746 1.40
Dm[9] 2 5016 3746 1.34
Dsd[1] 2 4134 3746 1.10
Dsd[2] 2 3816 3746 1.02
Dsd[3] 2 3928 3746 1.05
Dsd[4] 2 3888 3746 1.04
Dsd[5] 2 3815 3746 1.02
Dsd[6] 2 3773 3746 1.01
Dsd[7] 2 3898 3746 1.04
Dsd[8] 2 4198 3746 1.12
Dsd[9] 2 3938 3746 1.05
alpha[1] 14 15324 3746 4.09
alpha[2] 24 27660 3746 7.38
alpha[3] 20 23084 3746 6.16
alpha[4] 12 14798 3746 3.95
alpha[5] 42 50029 3746 13.40
alpha[6] 18 20445 3746 5.46
alpha[7] 15 18525 3746 4.95

alpha[8] 24 29448 3746 7.86
alpha[9] 25 28665 3746 7.65
beta[1] 10 11680 3746 3.12
beta[2] 20 22568 3746 6.02
beta[3] 12 16065 3746 4.29
beta[4] 12 16086 3746 4.29
beta[5] 30 35004 3746 9.34
beta[6] 15 18363 3746 4.90
beta[7] 15 16419 3746 4.38
beta[8] 15 16098 3746 4.30
beta[9] 16 22260 3746 5.94

[[2]]

Quantile (q) = 0.025 Accuracy (r) = +/- 0.005 Probability (s) = 0.95

```
      Burn-in  Total Lower bound  Dependence
      (M)      (N)   (Nmin)       factor (I)
```

Dm[1] 3 5345 3746 1.43
Dm[2] 2 4486 3746 1.20
Dm[3] 2 4677 3746 1.25
Dm[4] 2 5374 3746 1.43
Dm[5] 2 4194 3746 1.12
Dm[6] 2 4354 3746 1.16
Dm[7] 2 5674 3746 1.51
Dm[8] 2 5654 3746 1.51
Dm[9] 2 5028 3746 1.34
Dsd[1] 2 3888 3746 1.04
Dsd[2] 1 3761 3746 1.00
Dsd[3] 2 3893 3746 1.04
Dsd[4] 2 4051 3746 1.08
Dsd[5] 1 3791 3746 1.01
Dsd[6] 2 3747 3746 1.00
Dsd[7] 2 3924 3746 1.05
Dsd[8] 2 4302 3746 1.15
Dsd[9] 2 3854 3746 1.03
alpha[1] 15 17184 3746 4.59
alpha[2] 18 20568 3746 5.49
alpha[3] 24 27068 3746 7.23
alpha[4] 15 16842 3746 4.50
alpha[5] 30 34908 3746 9.32
alpha[6] 18 18696 3746 4.99
alpha[7] 15 17790 3746 4.75
alpha[8] 30 31370 3746 8.37
alpha[9] 30 31680 3746 8.46
beta[1] 12 15483 3746 4.13
beta[2] 20 22916 3746 6.12
beta[3] 15 17046 3746 4.55
beta[4] 15 16533 3746 4.41
beta[5] 24 28008 3746 7.48
beta[6] 15 18225 3746 4.87
beta[7] 12 16311 3746 4.35
beta[8] 12 15213 3746 4.06
beta[9] 15 16920 3746 4.52

```r
geweke.diag(out.coda)
```

[[1]]

Fraction in 1st window = 0.1 Fraction in 2nd window = 0.5

Dm[1] Dm[2] Dm[3] Dm[4] Dm[5] Dm[6] Dm[7] Dm[8] 1.12497 -0.30457 1.46001 0.56779 1.19886 0.45402 -1.51110 0.51946 Dm[9] Dsd[1] Dsd[2] Dsd[3] Dsd[4] Dsd[5] Dsd[6] Dsd[7] 0.83022 0.19884 -2.08110 0.10557 0.91174 0.69403 -0.53656 -2.40674 Dsd[8] Dsd[9] alpha[1] alpha[2] alpha[3] alpha[4] alpha[5] alpha[6] 0.17871 0.57754 -0.67445 1.60924 0.32298 0.10505 0.15256 0.27190 alpha[7] alpha[8] alpha[9] beta[1] beta[2] beta[3] beta[4] beta[5] -1.79060 0.82213 -0.02897 1.10208 -1.61424 -0.28067 0.05449 -0.16363 beta[6] beta[7] beta[8] beta[9] -0.21510 1.80072 -0.81810 0.25703

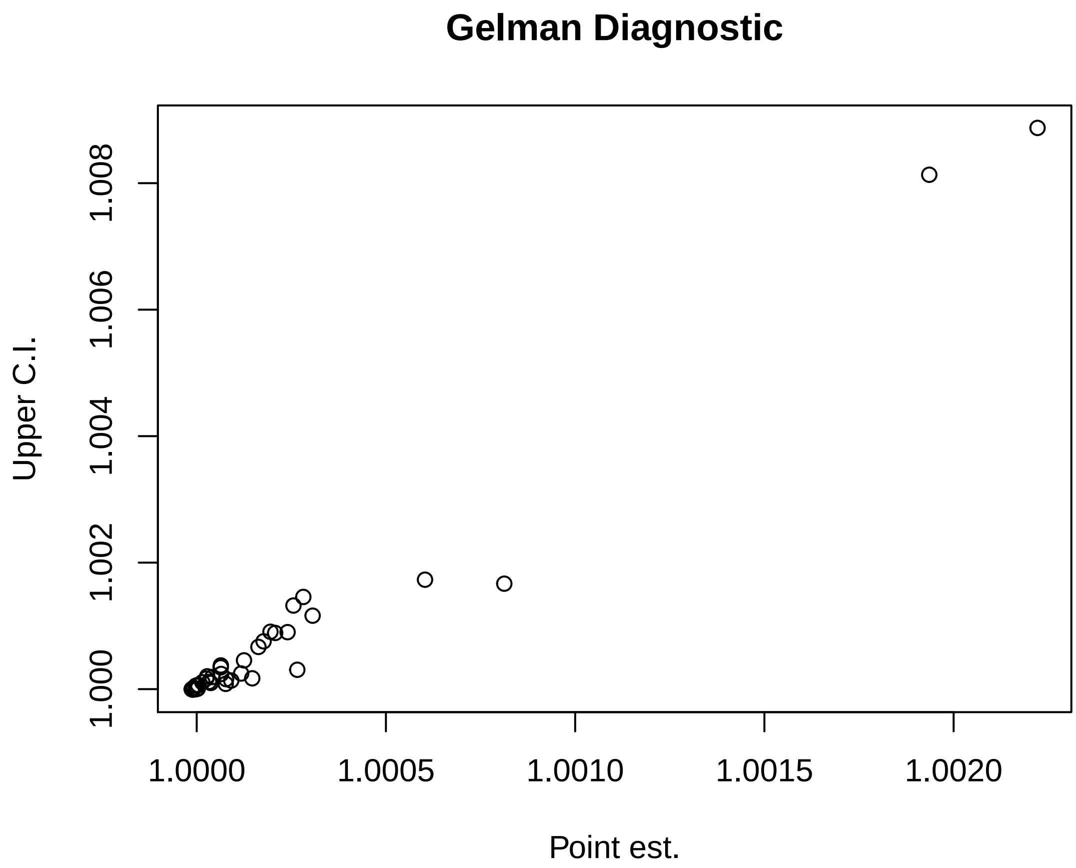
[[2]]

Fraction in 1st window = 0.1 Fraction in 2nd window = 0.5

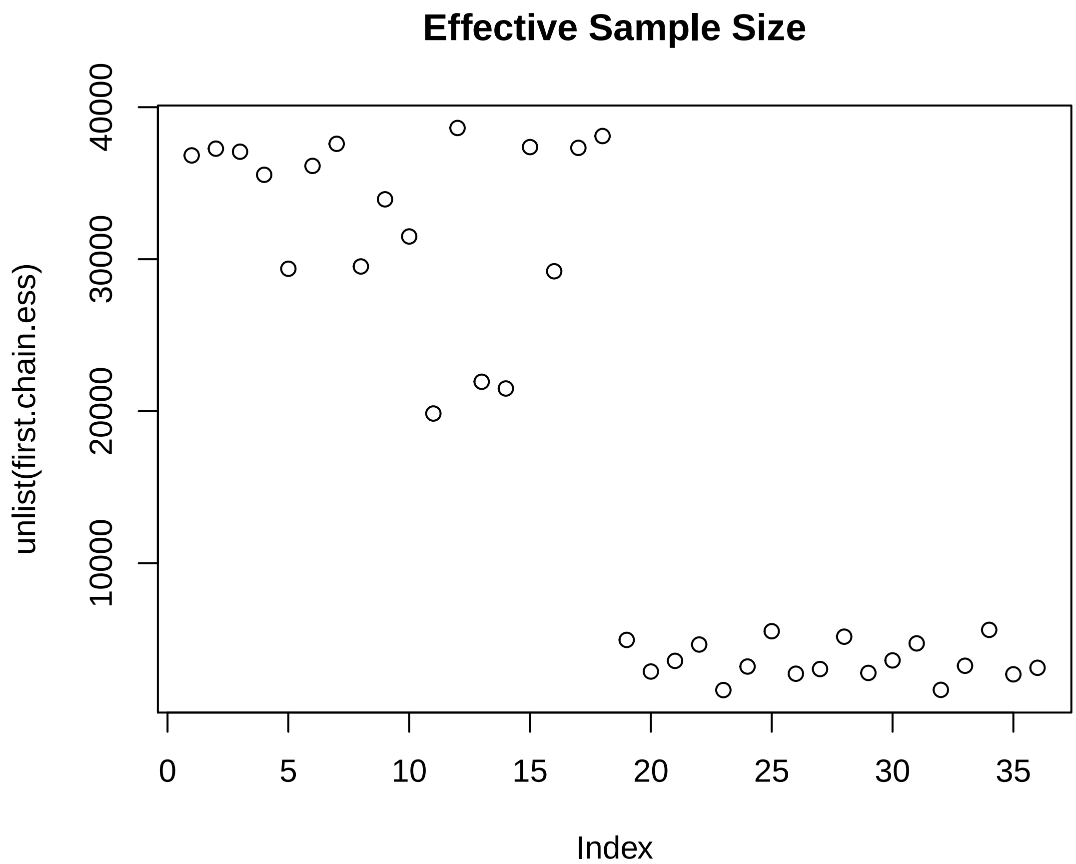
```
Dm[1]     Dm[2]     Dm[3]     Dm[4]     Dm[5]     Dm[6]     Dm[7]
```

0.912479 1.108245 -1.005375 0.793634 -1.158799 -0.199818 -0.752865 Dm[8] Dm[9] Dsd[1] Dsd[2] Dsd[3] Dsd[4] Dsd[5] -0.427206 2.723383 0.036788 -0.739673 -1.220140 0.848573 0.144092 Dsd[6] Dsd[7] Dsd[8] Dsd[9] alpha[1] alpha[2] alpha[3] -0.281158 -0.925917 -0.197469 0.455786 0.007614 0.558412 1.049502 alpha[4] alpha[5] alpha[6] alpha[7] alpha[8] alpha[9] beta[1] 0.976656 -0.828065 1.100394 -0.642340 -0.820338 1.483605 0.218536 beta[2] beta[3] beta[4] beta[5] beta[6] beta[7] beta[8] -0.332665 -1.124019 -0.775898 0.833748 -1.213919 0.556608 0.812515 beta[9] -1.261860

```r
if(n.chains > 1)
{
 gelman.srf <-gelman.diag(out.coda)
 plot(gelman.srf$psrf,main = "Gelman Diagnostic")
}
```




```r
chains.ess <- lapply(out.coda,effectiveSize)
first.chain.ess <- chains.ess[1]
plot(unlist(first.chain.ess), main="Effective Sample Size")
```




```
pval.m <- matrix(nrow = 9,ncol = 2)
for(k in 1:9){
  # Compute the test stats for the data
  D0    <- c(    mean(X.num[,k]),    sd(X.num[,k]))
  Dnames <- c("mean Y", "sd Y")
  # Compute the test stats for the models
  chain <- out.coda[[1]]
  D1    <- cbind(chain[,paste("Dm[",k,"]",sep='')],chain[,paste("Dsd[",k,"]",sep='')])
  pval1 <- rep(0,2)
  names(pval1)<-Dnames

  for(j in 1:2){
  pval1[j] <- mean(D1[,j]>D0[j])
  }
  pval.m[k,] <- pval1
}
colnames(pval.m)<-c("pval.mean","pval.sd")
pander(data.frame(pval.m), caption = "Baeysian p-values Poisson GLM")
```

Table 1: Baeysian p-values Poisson GLM

pval.mean	pval.sd
0.4505	0.2622

pval.mean	pval.sd
0.4672	0.4199
0.4787	0.3865
0.4144	0.2115
0.4981	0.6656
0.4442	0.5961
0.4471	0.0933
0.5015	0.2892
0.4963	0.2056

Fit JAGS Negative Binomial Random Effects

```
######################################## Fit JAGS Negative Binomial Random Effects
model_nb = '
model
{
    ## Likelihood
    for(i in 1:N){
      for(j in 1:9){
        Y[i,j] ~ dnegbin(p[i,j],r[j])
        p[i,j] <- r[j]/(r[j]+lambda[i,j])
        log(lambda[i,j]) <- mu[i,j]
        mu[i,j] <- alpha[j] + beta[j]*t[i]
      }
    }

  ## Priors
  for(i in 1:9){
    alpha[i] ~ dnorm(0,taus[i])
    taus[i] ~ dgamma(0.1,0.1)
  }

  # Slopes
  for(i in 1:9){
    beta[i] ~ dnorm(mu.beta,taus.beta[i])
    taus.beta[i] ~ dgamma(0.1,0.1)
  }

  # r
  for(i in 1:9){
    r[i] ~ dunif(0,10)
  }

  ## Posterior Predictive Checks
  for(i in 1:N){
    for(j in 1:9){
        Y2[i,j] ~ dnegbin(p[i,j],r[j])
    }
  }

  for(j in 1:9){
```

```
    Dm[j] <- mean(Y2[,j])
    Dsd[j] <- sd(Y2[,j])
  }
}
'
  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num,mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercept=.
  # Choose the parameters to watch
  model_parameters =  c("r","beta", "alpha","Dm","Dsd")# model_parameters =  c("r")
  model_nb <- jags.model(textConnection(model_nb),data = model_data,n.chains = n.chains)#Compile Model
```

Compiling model graph Resolving undeclared variables Allocating nodes Graph information: Observed stochastic nodes: 369 Unobserved stochastic nodes: 414 Total graph size: 2701

Initializing model

```
  update(model_nb, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model_nb, variable.names=model_parameters,n.iter=2*nSamples)
  #plot(out.coda)
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
```

[[1]]

```
Stationarity start     p-value
test         iteration
```

Dm[1] passed 1 0.06044 Dm[2] passed 1 0.89797 Dm[3] passed 1 0.73300 Dm[4] passed 1 0.57537 Dm[5] passed 1 0.84561 Dm[6] passed 1 0.43689 Dm[7] passed 1 0.84863 Dm[8] passed 4001 0.36851 Dm[9] passed 1 0.42823 Dsd[1] passed 4001 0.11036 Dsd[2] passed 1 0.50688 Dsd[3] passed 1 0.58560 Dsd[4] passed 1 0.50736 Dsd[5] passed 1 0.99351 Dsd[6] passed 1 0.66288 Dsd[7] passed 1 0.90764 Dsd[8] passed 1 0.10573 Dsd[9] passed 1 0.86975 alpha[1] passed 1 0.91402 alpha[2] passed 1 0.28984 alpha[3] passed 1 0.88735 alpha[4] passed 1 0.83356 alpha[5] passed 1 0.94319 alpha[6] passed 1 0.86402 alpha[7] passed 1 0.39444 alpha[8] passed 1 0.10790 alpha[9] passed 1 0.28735 beta[1] passed 1 0.95514 beta[2] passed 1 0.18310 beta[3] passed 1 0.83940 beta[4] passed 1 0.83268 beta[5] passed 1 0.95542 beta[6] passed 1 0.92194 beta[7] passed 1 0.29386 beta[8] passed 1 0.15317 beta[9] passed 1 0.24112 r[1] passed 1 0.18295 r[2] passed 1 0.88851 r[3] passed 1 0.46944 r[4] failed NA 0.00243 r[5] passed 1 0.77645 r[6] passed 1 0.14357 r[7] passed 1 0.68534 r[8] passed 1 0.87567 r[9] passed 1 0.56440

```
Halfwidth Mean     Halfwidth
test
```

Dm[1] passed 0.74185 0.002086 Dm[2] passed 1.94987 0.003627 Dm[3] passed 1.07384 0.002575 Dm[4] passed 0.86732 0.002540 Dm[5] passed 0.93870 0.002485 Dm[6] passed 1.70165 0.003316 Dm[7] passed 0.64291 0.002119 Dm[8] passed 0.52330 0.002156 Dm[9] passed 0.89412 0.002450 Dsd[1] passed 0.93898 0.002518 Dsd[2] passed 1.76611 0.004669 Dsd[3] passed 1.15855 0.002626 Dsd[4] passed 1.06752 0.003736 Dsd[5] passed 1.16968 0.004035 Dsd[6] passed 1.50400 0.002947 Dsd[7] passed 0.91843 0.003269 Dsd[8] passed 0.77943 0.002483 Dsd[9] passed 1.07183 0.002943 alpha[1] failed -0.01573 0.009561 alpha[2] failed -0.01606 0.009290 alpha[3] passed -0.15663 0.010077 alpha[4] passed 0.28578 0.009330 alpha[5] passed -1.20016 0.023175 alpha[6] passed 0.20650 0.008577 alpha[7] failed 0.00832 0.009612 alpha[8] passed -0.83735 0.019132 alpha[9] passed -0.40517 0.013431 beta[1] passed -0.01597 0.000433 beta[2] passed 0.02900 0.000368 beta[3] passed 0.00931 0.000416 beta[4] passed -0.02384 0.000449 beta[5] passed 0.04572 0.000822 beta[6] passed 0.01409 0.000345 beta[7] passed -0.02555 0.000468 beta[8] failed 0.00611 0.000746 beta[9] passed 0.01178 0.000536 r[1] passed 5.70855 0.030101 r[2] passed 6.55656 0.028352 r[3] passed 5.83041 0.030451 r[4] NA NA r[5] passed 6.65142 0.029973 r[6] passed 6.95652 0.028274 r[7] passed 4.23476 0.041261 r[8] passed 4.98277 0.035068 r[9] passed 4.97382 0.034904

[[2]]

```
Stationarity start     p-value
test         iteration
```

Dm[1] passed 1 0.0860 Dm[2] passed 1 0.3372 Dm[3] passed 1 0.5834 Dm[4] passed 1 0.7664 Dm[5] passed 1 0.9733 Dm[6] passed 1 0.9545 Dm[7] passed 1 0.7529 Dm[8] failed NA 0.0201 Dm[9] passed 1 0.1460 Dsd[1] passed 1 0.0683 Dsd[2] passed 1 0.0641 Dsd[3] passed 1 0.7962 Dsd[4] passed 1 0.8641 Dsd[5] passed 12001 0.1348 Dsd[6] passed 1 0.6945 Dsd[7] passed 1 0.8481 Dsd[8] passed 12001 0.5353 Dsd[9] passed 1 0.3877 alpha[1] passed 1 0.0822 alpha[2] passed 1 0.2629 alpha[3] passed 1 0.9251 alpha[4] passed 1 0.3958 alpha[5] passed 1 0.0792 alpha[6] passed 1 0.2104 alpha[7] passed 1 0.8366 alpha[8] passed 12001 0.1438 alpha[9] passed 1 0.4709 beta[1] passed 1 0.0928 beta[2] passed 1 0.1621 beta[3] passed 1 0.8621 beta[4] passed 1 0.3529 beta[5] passed 1 0.0603 beta[6] passed 1 0.1515 beta[7] passed 1 0.8304 beta[8] passed 4001 0.0731 beta[9] passed 1 0.5630 r[1] passed 1 0.1054 r[2] passed 1 0.5524 r[3] passed 1 0.7126 r[4] passed 1 0.2792 r[5] passed 1 0.2158 r[6] passed 1 0.1190 r[7] passed 1 0.9410 r[8] passed 8001 0.0589 r[9] passed 1 0.9418

```
Halfwidth Mean     Halfwidth
test
```

Dm[1] passed 0.74058 0.002077 Dm[2] passed 1.95009 0.003558 Dm[3] passed 1.07192 0.002600 Dm[4] passed 0.86701 0.002602 Dm[5] passed 0.93856 0.002482 Dm[6] passed 1.70340 0.003323 Dm[7] passed 0.64467 0.002032 Dm[8] NA NA Dm[9] passed 0.89141 0.002420 Dsd[1] passed 0.93647 0.002300 Dsd[2] passed 1.76688 0.005050 Dsd[3] passed 1.15808 0.002704 Dsd[4] passed 1.06701 0.003698 Dsd[5] passed 1.17173 0.005007 Dsd[6] passed 1.50393 0.002949 Dsd[7] passed 0.92025 0.003153 Dsd[8] passed 0.77873 0.002985 Dsd[9] passed 1.06759 0.002749 alpha[1] failed -0.02248 0.009239 alpha[2] failed -0.01548 0.009409 alpha[3] passed -0.17187 0.010410 alpha[4] passed 0.29337 0.009182 alpha[5] passed -1.20994 0.023236 alpha[6] passed 0.20295 0.009021 alpha[7] failed 0.00824 0.009143 alpha[8] passed -0.85467 0.023595 alpha[9] passed -0.40899 0.013156 beta[1] passed -0.01568 0.000427 beta[2] passed 0.02897 0.000361 beta[3] passed 0.00994 0.000429 beta[4] passed -0.02421 0.000438 beta[5] passed 0.04610 0.000803 beta[6] passed 0.01420 0.000363 beta[7] passed -0.02547 0.000448 beta[8] failed 0.00634 0.000822 beta[9] passed 0.01192 0.000524 r[1] passed 5.70944 0.029824 r[2] passed 6.57717 0.028853 r[3] passed 5.80898 0.030275 r[4] passed 5.16584 0.033711 r[5] passed 6.64076 0.029937 r[6] passed 6.96733 0.028614 r[7] passed 4.24975 0.041601 r[8] passed 4.98695 0.039295 r[9] passed 4.99895 0.034916

```
# check that our chain???s length is satisfactory.
raftery.diag(out.coda)
```

[[1]]

Quantile (q) = 0.025 Accuracy (r) = +/- 0.005 Probability (s) = 0.95

```
Burn-in  Total Lower bound  Dependence
(M)      (N)   (Nmin)       factor (I)
```

Dm[1] 2 4585 3746 1.22
Dm[2] 2 3839 3746 1.02
Dm[3] 2 4812 3746 1.28
Dm[4] 2 3965 3746 1.06
Dm[5] 2 5454 3746 1.46
Dm[6] 2 4281 3746 1.14
Dm[7] 2 5416 3746 1.45
Dm[8] 2 4306 3746 1.15
Dm[9] 2 4523 3746 1.21
Dsd[1] 2 3954 3746 1.06
Dsd[2] 2 3856 3746 1.03
Dsd[3] 2 3707 3746 0.99
Dsd[4] 2 3896 3746 1.04
Dsd[5] 1 3757 3746 1.00

Dsd[6] 2 3810 3746 1.02
Dsd[7] 2 3782 3746 1.01
Dsd[8] 2 4037 3746 1.08
Dsd[9] 2 3938 3746 1.05
alpha[1] 25 28670 3746 7.65
alpha[2] 20 24628 3746 6.57
alpha[3] 20 25484 3746 6.80
alpha[4] 15 16533 3746 4.41
alpha[5] 30 31955 3746 8.53
alpha[6] 15 16074 3746 4.29
alpha[7] 20 23680 3746 6.32
alpha[8] 24 26157 3746 6.98
alpha[9] 25 27760 3746 7.41
beta[1] 15 18489 3746 4.94
beta[2] 24 23488 3746 6.27
beta[3] 20 21780 3746 5.81
beta[4] 16 18244 3746 4.87
beta[5] 24 32268 3746 8.61
beta[6] 20 23268 3746 6.21
beta[7] 16 20692 3746 5.52
beta[8] 15 17601 3746 4.70
beta[9] 12 15813 3746 4.22
r[1] 3 4577 3746 1.22
r[2] 4 4954 3746 1.32
r[3] 3 4539 3746 1.21
r[4] 3 4501 3746 1.20
r[5] 4 5134 3746 1.37
r[6] 5 5459 3746 1.46
r[7] 3 4258 3746 1.14
r[8] 3 4501 3746 1.20
r[9] 3 4391 3746 1.17

[[2]]

Quantile (q) = 0.025 Accuracy (r) = +/- 0.005 Probability (s) = 0.95

```
      Burn-in  Total Lower bound  Dependence
      (M)      (N)   (Nmin)       factor (I)
```

Dm[1] 2 4530 3746 1.21
Dm[2] 2 3940 3746 1.05
Dm[3] 2 4992 3746 1.33
Dm[4] 2 4114 3746 1.10
Dm[5] 2 4073 3746 1.09
Dm[6] 2 4292 3746 1.15
Dm[7] 2 5102 3746 1.36
Dm[8] 2 4131 3746 1.10
Dm[9] 2 4743 3746 1.27
Dsd[1] 2 3808 3746 1.02
Dsd[2] 2 3804 3746 1.02
Dsd[3] 2 3818 3746 1.02
Dsd[4] 2 3895 3746 1.04
Dsd[5] 2 3885 3746 1.04
Dsd[6] 2 3798 3746 1.01
Dsd[7] 2 3843 3746 1.03
Dsd[8] 2 3897 3746 1.04

Dsd[9] 2 3796 3746 1.01
alpha[1] 20 23008 3746 6.14
alpha[2] 21 22839 3746 6.10
alpha[3] 24 25300 3746 6.75
alpha[4] 12 15020 3746 4.01
alpha[5] 40 44625 3746 11.90
alpha[6] 18 20457 3746 5.46
alpha[7] 20 27355 3746 7.30
alpha[8] 28 34364 3746 9.17
alpha[9] 28 30444 3746 8.13
beta[1] 12 16587 3746 4.43
beta[2] 18 19119 3746 5.10
beta[3] 20 22288 3746 5.95
beta[4] 15 18324 3746 4.89
beta[5] 24 26940 3746 7.19
beta[6] 20 25096 3746 6.70
beta[7] 12 14460 3746 3.86
beta[8] 16 22524 3746 6.01
beta[9] 15 15270 3746 4.08
r[1] 3 4567 3746 1.22
r[2] 4 5144 3746 1.37
r[3] 4 4644 3746 1.24
r[4] 3 4511 3746 1.20
r[5] 5 5505 3746 1.47
r[6] 5 5482 3746 1.46
r[7] 3 4223 3746 1.13
r[8] 3 4567 3746 1.22
r[9] 3 4302 3746 1.15

```
geweke.diag(out.coda)
```

[[1]]

Fraction in 1st window = 0.1 Fraction in 2nd window = 0.5

Dm[1] Dm[2] Dm[3] Dm[4] Dm[5] Dm[6] Dm[7] Dm[8] -2.19307 0.19756 -0.50604 -1.36781 -0.46078 1.58231 -0.30361 -2.76027 Dm[9] Dsd[1] Dsd[2] Dsd[3] Dsd[4] Dsd[5] Dsd[6] Dsd[7] 1.63638 -1.58910 -0.30538 -0.69502 0.11355 -0.89913 1.08212 -0.49997 Dsd[8] Dsd[9] alpha[1] alpha[2] alpha[3] alpha[4] alpha[5] alpha[6] -2.35424 -0.58791 -0.72102 0.05676 -0.38933 0.62017 0.38378 0.89995 alpha[7] alpha[8] alpha[9] beta[1] beta[2] beta[3] beta[4] beta[5] -0.64658 -2.25544 2.16693 0.34126 -0.09406 0.38783 -0.75674 -0.41896 beta[6] beta[7] beta[8] beta[9] r[1] r[2] r[3] r[4] -0.73594 0.33068 2.14722 -2.11263 -1.16250 0.57021 1.38264 -0.39900 r[5] r[6] r[7] r[8] r[9] 0.60534 -0.79131 -0.37866 0.79243 -0.32069

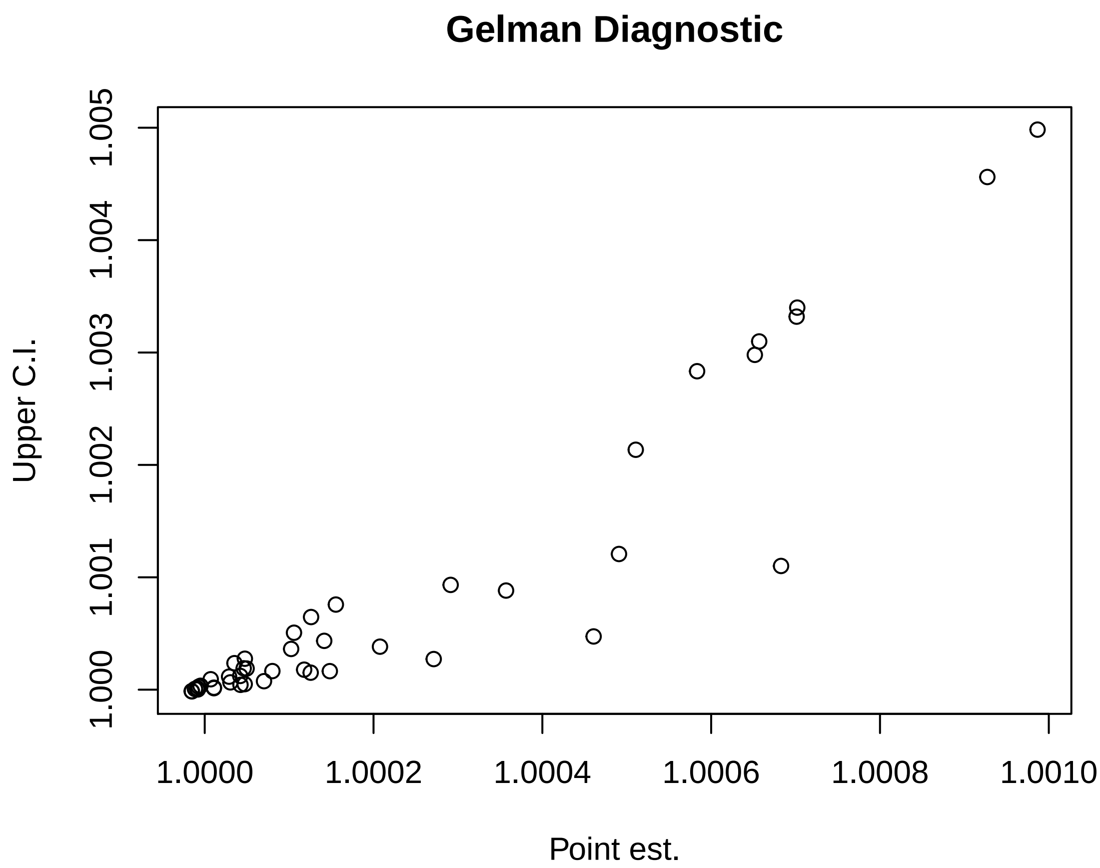
[[2]]

Fraction in 1st window = 0.1 Fraction in 2nd window = 0.5

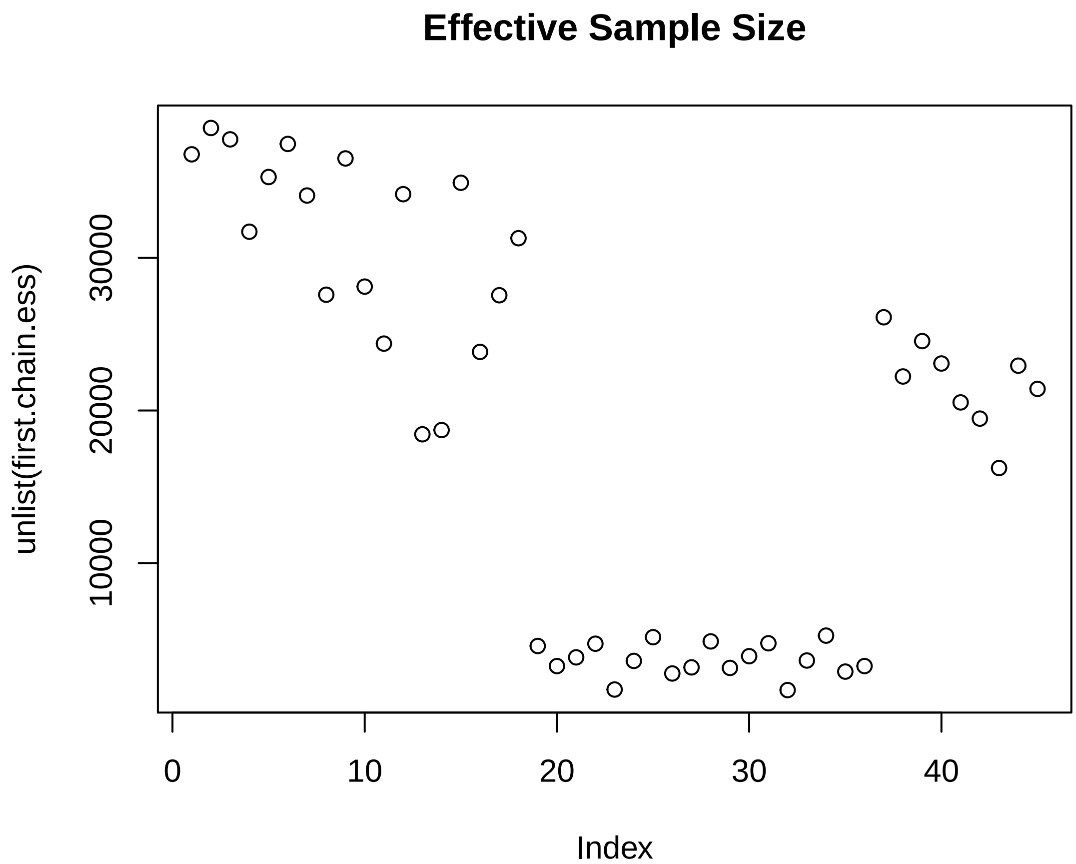
Dm[1] Dm[2] Dm[3] Dm[4] Dm[5] Dm[6] Dm[7] Dm[8] 1.46736 -1.11913 -0.62549 -0.20852 0.19278 -0.13775 1.11762 1.20441 Dm[9] Dsd[1] Dsd[2] Dsd[3] Dsd[4] Dsd[5] Dsd[6] Dsd[7] 1.54818 2.47010 -2.65782 0.32850 -0.11915 0.35179 0.06317 1.37696 Dsd[8] Dsd[9] alpha[1] alpha[2] alpha[3] alpha[4] alpha[5] alpha[6] 0.77397 0.86022 3.09953 2.01293 0.09599 -0.45858 0.38855 -1.04784 alpha[7] alpha[8] alpha[9] beta[1] beta[2] beta[3] beta[4] beta[5] 0.09371 1.99345 1.24963 -3.09719 -2.22718 -0.15136 0.38816 -0.44343 beta[6] beta[7] beta[8] beta[9] r[1] r[2] r[3] r[4] 1.18041 0.01071 -1.93048 -1.09873 -0.69572 1.67452 -0.51845 -0.37511 r[5] r[6] r[7] r[8] r[9] -1.37009 0.98095 -0.44984 -0.49855 -0.02268

```
if(n.chains > 1)
{
 gelman.srf <-gelman.diag(out.coda)
```

```
  plot(gelman.srf$psrf,main = "Gelman Diagnostic")
}
```




```
chains.ess <- lapply(out.coda,effectiveSize)
first.chain.ess <- chains.ess[1]
plot(unlist(first.chain.ess), main="Effective Sample Size")
```




```r
pval.m <- matrix(nrow = 9,ncol = 2)
 for(k in 1:9){
   # Compute the test stats for the data
   D0   <- c(   mean(X.num[,k]),   sd(X.num[,k]))
   Dnames <- c("mean Y", "sd Y")
   # Compute the test stats for the models
   chain <- out.coda[[1]]
   D1   <- cbind(chain[,paste("Dm[",k,"]",sep='')],chain[,paste("Dsd[",k,"]",sep='')])
   pval1 <- rep(0,2)
   names(pval1)<-Dnames

   for(j in 1:2){
   pval1[j] <- mean(D1[,j]>D0[j])
   }
   pval.m[k,] <- pval1
 }
 colnames(pval.m)<-c("pval.mean","pval.sd")
 pander(data.frame(pval.m), caption = "Baeysian p-values Poisson GLM")
```

Table 2: Baeysian p-values Poisson GLM

pval.mean	pval.sd
0.462	0.4286

pval.mean	pval.sd
0.486	0.6965
0.4913	0.6038
0.4282	0.4234
0.5135	0.7558
0.4551	0.7998
0.4553	0.2731
0.5118	0.4291
0.5134	0.4462

DIC Calculation

```
dic_pois <- dic.samples(model_pois, variable.names = c("beta",
    "alpha"), n.iter = nSamples, progress.bar = "none")
dic_pois
```

```
## Mean deviance:  957.6
## penalty 16.76
## Penalized deviance: 974.4
```

```
dic_nb <- dic.samples(model_nb, variable.names = c("beta",
    "alpha"), n.iter = nSamples, progress.bar = "none")
dic_nb
```

```
## Mean deviance:  957.7
## penalty 19.4
## Penalized deviance: 977.1
```