Bayesian Learning (732A91) Lab3 Report

Christoforos Spyretos (chrsp415) & Marketos Damigos (marda352)

2022-05-06

Assignment 1 *Gibbs sampler for a normal model*

Task 1a

The normal model with conditionally conjugate prior is:

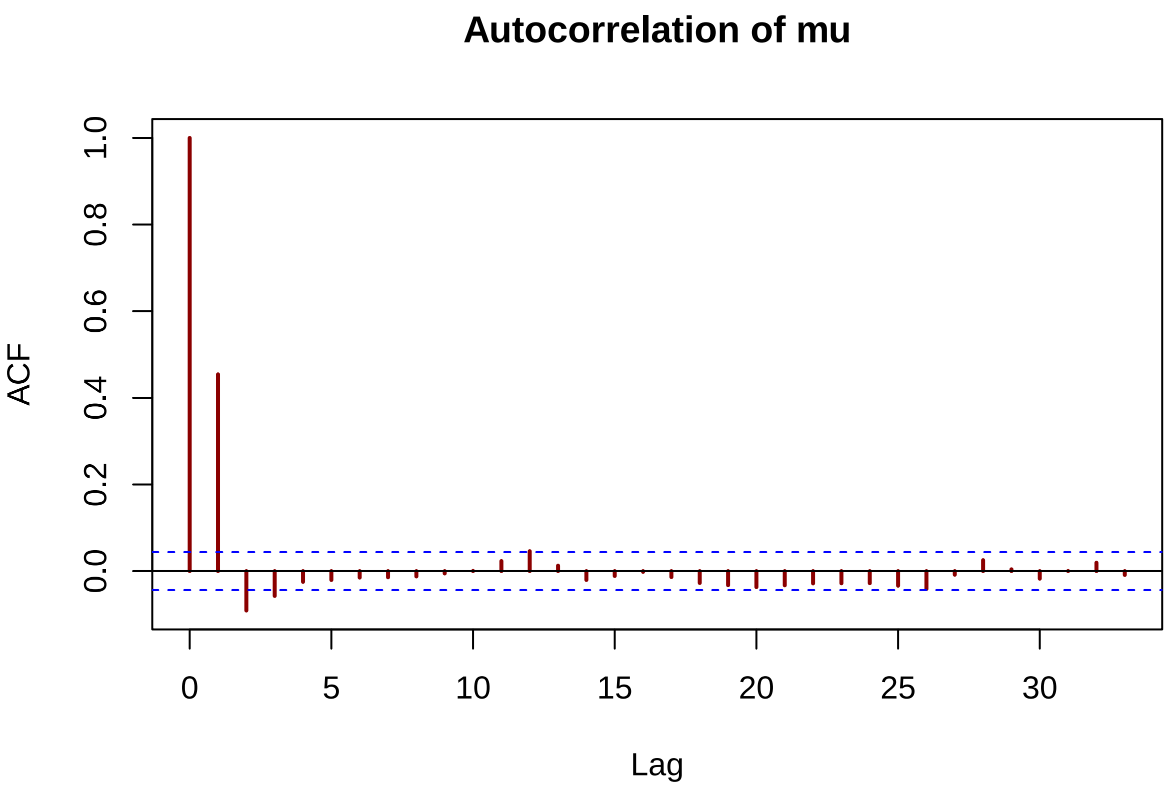
$$\mu \sim N(\mu_0, \tau_0^2)$$
$$\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$$

The full conditional posteriors are:

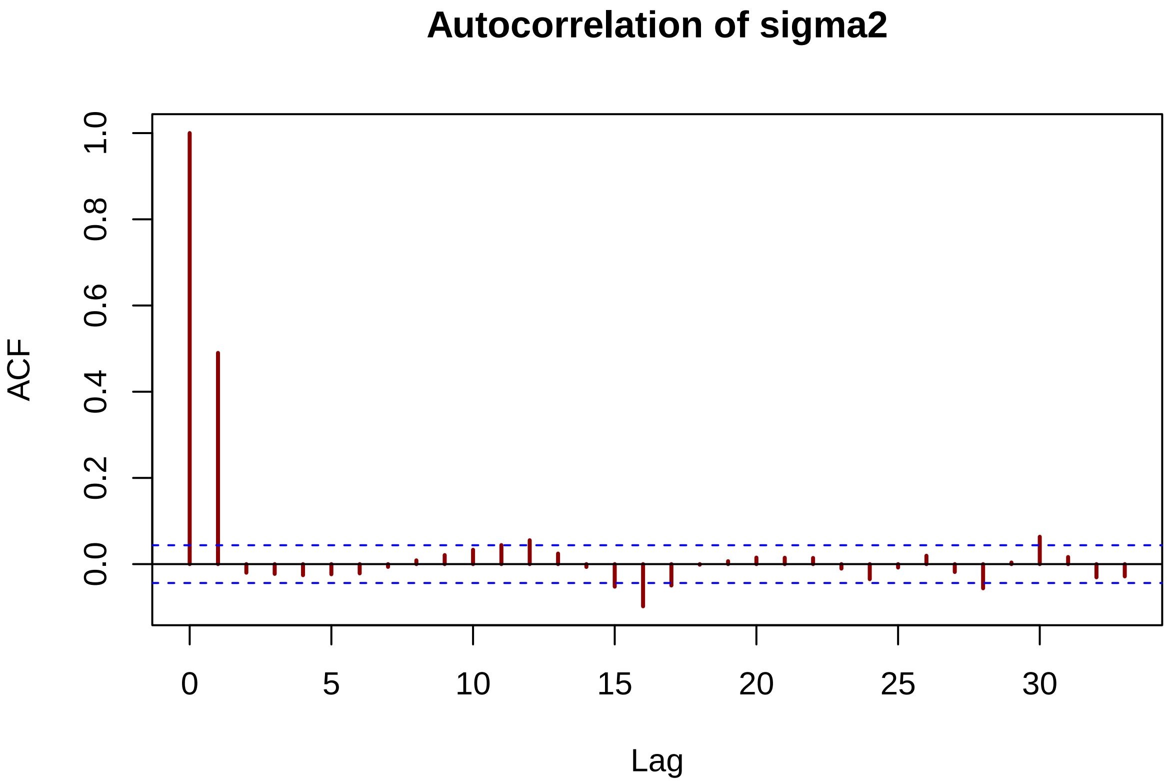
$$\mu|\sigma^2, x \sim N(\mu_n, \tau_n^2)$$
$$\sigma^2|\mu, x \sim Inv - \chi^2\Big(\nu_n, \frac{\nu_0\sigma_0^2 + \sum_{i=1}^n (x_i - \mu)^2}{n + \nu_0}\Big)$$

Where:

$$w = \frac{\frac{n}{\sigma^2}}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$
$$\mu_n = w\overline{x} + (1 - w)\mu_0$$
$$\tau_n^2 = \frac{1}{\frac{n}{\sigma^2} + \frac{1}{\tau_0^2}}$$
$$\nu_n = \nu_0 + n$$



The above graph illustrates the correlation coefficient of μ against the lag. The red bars illustrate the correlation coefficient of μ of each lag, and the blue lines represent the statistically significant boundaries. The ACF equals 1 for a lag 0, which is expected because the "first step" is always perfectly correlated with itself (the lag 0 autocorrelation is fixed at one by convention). Furthermore, as lag increases, the correlation gets closer to 0, which means the variables are unrelated between them.



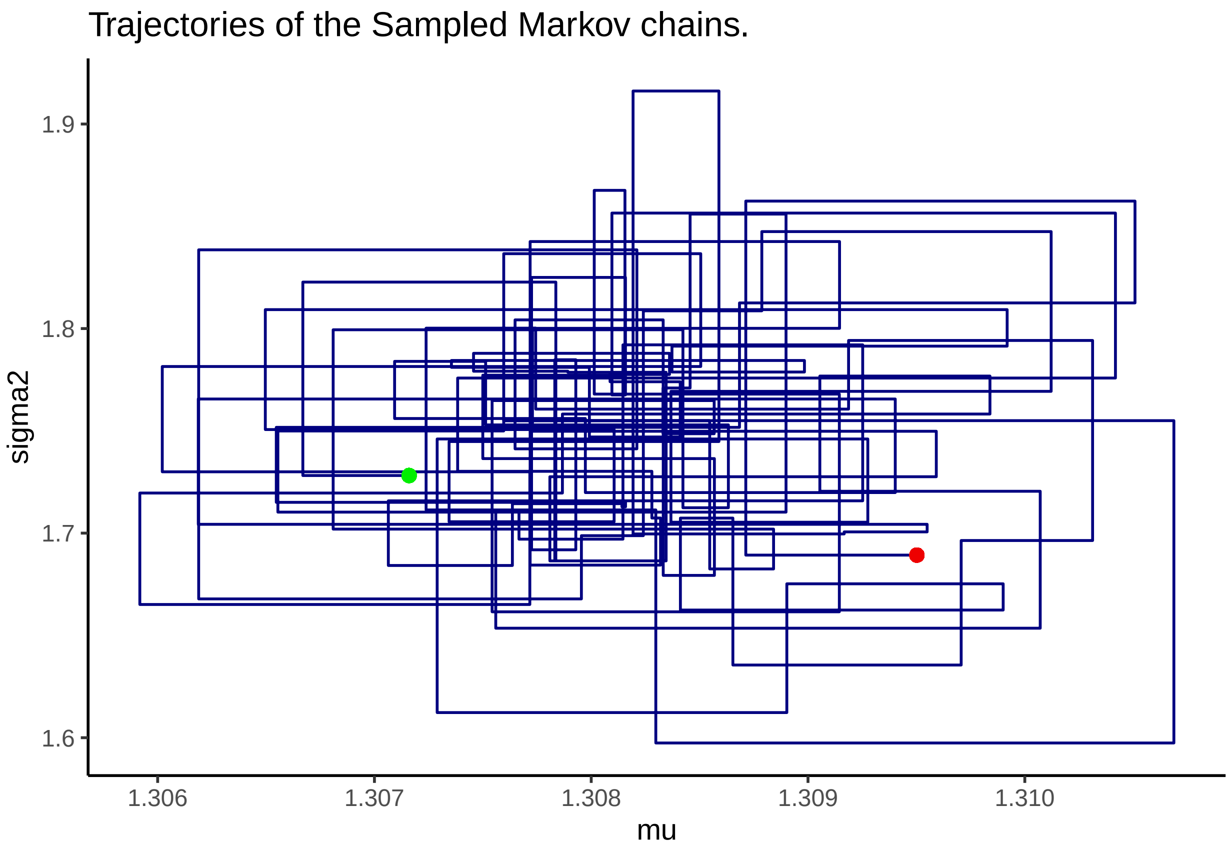
The above graph illustrates the correlation coefficient of σ^2 against the lag. The red bars illustrate the correlation coefficient of σ^2 of each lag, and the blue lines represent the statistically significant boundaries. The ACF equals 1 for a lag 0, which is expected because the "first step" is always perfectly correlated with itself (the lag zero autocorrelation is fixed at one by convention). Furthermore, as lag increases, the correlation gets closer to 0, which means the variables are unrelated between them. Finally, however, the ACF value crosses the "boundaries", which means that the variables are related when lag equals 12, 16 and 30.

https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling
https://en.wikipedia.org/wiki/Autocorrelation

The below table illustrates the inefficiency factors of μ and σ^2.

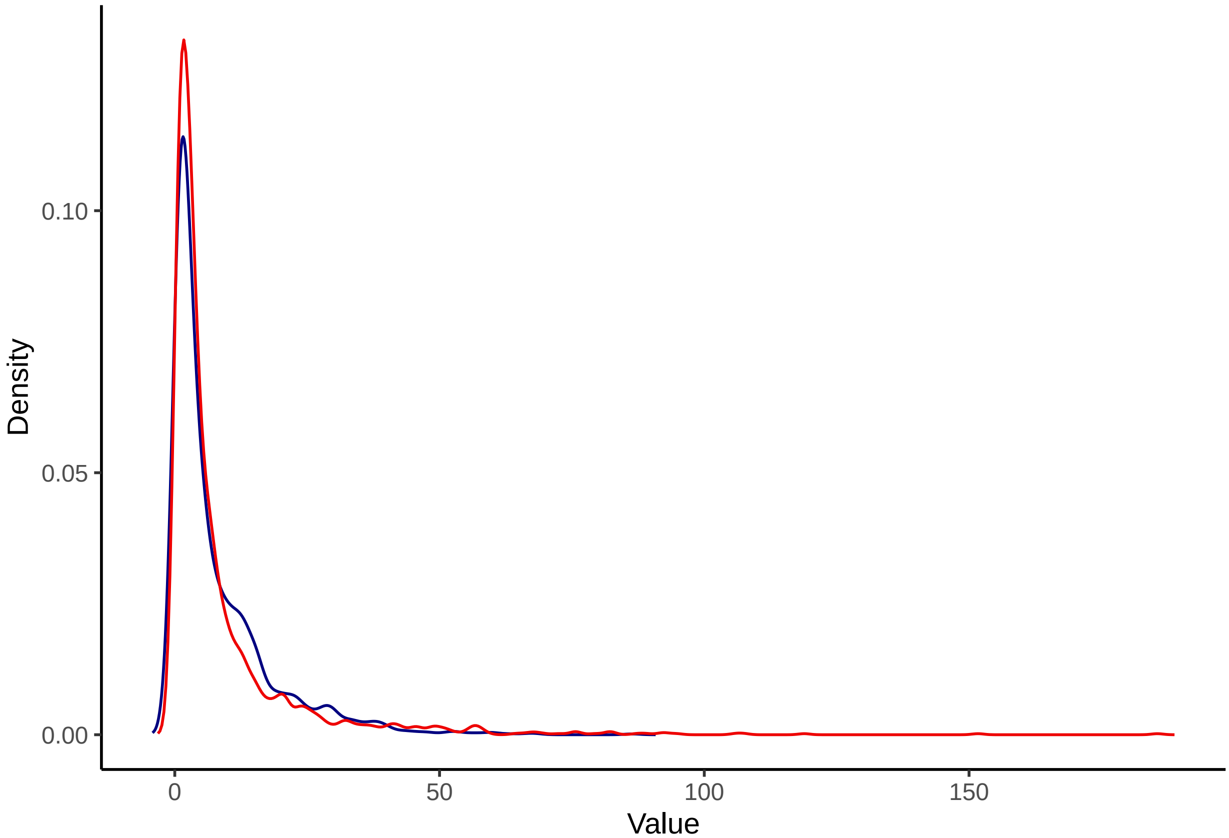
	mu	sigma2
Inefficiency Factors	0.9468888	1.635177

The inefficiency factor equals 1 if there is not any autocorrelation. The inefficiency factor of μ is close to 1, which means this is an efficient sample. However, the inefficiency factor of σ^2 equals approximately 1.63, which means that the sample is efficient enough.



The above plot illustrates the trajectories of the first 200 point of the sample, the red dot is the starting point and the green dot is the ending point.

Task b



Assignment 2 *Metropolis Random Walk for Poisson regression*

Task a

```
##
## Call:
## glm(formula = nBids ~ 0 + ., family = poisson, data = ebay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## Const        1.07244    0.03077  34.848  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## Minblem     -0.05220    0.06020  -0.867   0.3859
## MajBlem     -0.22087    0.09144  -2.416   0.0157 *
## LargNeg      0.07067    0.05633   1.255   0.2096
## LogBook     -0.12068    0.02896  -4.166 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6264.01  on 1000  degrees of freedom
## Residual deviance:  867.47  on  991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

Task b

Poisson regression model: $y_i|beta \sim Poisson[exp(x_i^T\beta)]$, $i = 1, 2, ..., n$

The likelihood function of the Poisson regression model is:

$$L(\theta|X,Y) = \prod_{i}^{n} \frac{e^{Y_i\theta^T X_i} e^{-\theta^T X_i}}{Y_i!}$$

The log-likelihood of the Poisson regression model is:

$$l(\theta|X,Y) = log\Big(L(\theta|X,Y)\Big) = \sum_{i}^{n} \Big(Y_i\theta^T X_i - e^{\theta^T Xi} - log(Y_i)\Big)$$

In the above formula, θ appears in the first two terms; therefore, given that we are only interested in the finding of the best value of θ, we drop the $log(Y_i)$.

The final log-likelihood of the Poisson regression model is:

$$l(\theta|X,Y) = \sum_{i}^{n} \Big(Y_i\theta^T X_i - e^{\theta^T Xi}\Big)$$

https://en.wikipedia.org/wiki/Poisson_regression

The below table illustrates the $J_y^{-1}(\widetilde{\beta})$.

Constant	PowerSeller	VerifyID	Sealed	MinBlem	MajBlem	LargNeg	LogBook	MinBidShare
0.0009455	-0.0007139	-0.0002742	-0.0002709	-0.0004455	-0.0002772	-0.0005128	0.0000644	0.0011099
-0.0007139	0.0013531	0.0000402	-0.0002949	0.0001143	-0.0002083	0.0002802	0.0001182	-0.0005686
-0.0002742	0.0000402	0.0085154	-0.0007825	-0.0001014	0.0002283	0.0003314	-0.0003192	-0.0004293
-0.0002709	-0.0002949	-0.0007825	0.0025578	0.0003577	0.0004532	0.0003376	-0.0001311	-0.0000576
-0.0004455	0.0001143	-0.0001014	0.0003577	0.0036246	0.0003492	0.0000584	0.0000585	-0.0000644
-0.0002772	-0.0002083	0.0002283	0.0004532	0.0003492	0.0083651	0.0004049	-0.0000898	0.0002622
-0.0005128	0.0002802	0.0003314	0.0003376	0.0000584	0.0004049	0.0031751	-0.0002542	-0.0001063
0.0000644	0.0001182	-0.0003192	-0.0001311	0.0000585	-0.0000898	-0.0002542	0.0008385	0.0010374

Constant	PowerSeller	VerifyID	Sealed	MinBlem	MajBlem	LargNeg	LogBook	MinBidShare
0.0011099	-0.0005686	-0.0004293	-0.0000576	-0.0000644	0.0002622	-0.0001063	0.0010374	0.0050548

The below table illustrates the posterior mode values of every feature of the dataset.

	Value
Constant	1.0698412
PowerSeller	-0.0205125
VerifyID	-0.3930060
Sealed	0.4435555
MinBlem	-0.0524663
MajBlem	-0.2212384
LargNeg	0.0706968
LogBook	-0.1202177
MinBidShare	-1.8919850

The below table illustrates the approximate posterior standard deviation values of every feature of the dataset.

	Value
Constant	0.0307484
PowerSeller	0.0367842
VerifyID	0.0922787
Sealed	0.0505745
MinBlem	0.0602047
MajBlem	0.0914607
LargNeg	0.0563477
LogBook	0.0289564
MinBidShare	0.0710968

Task c

For the Random walk Metropolis algorithm:

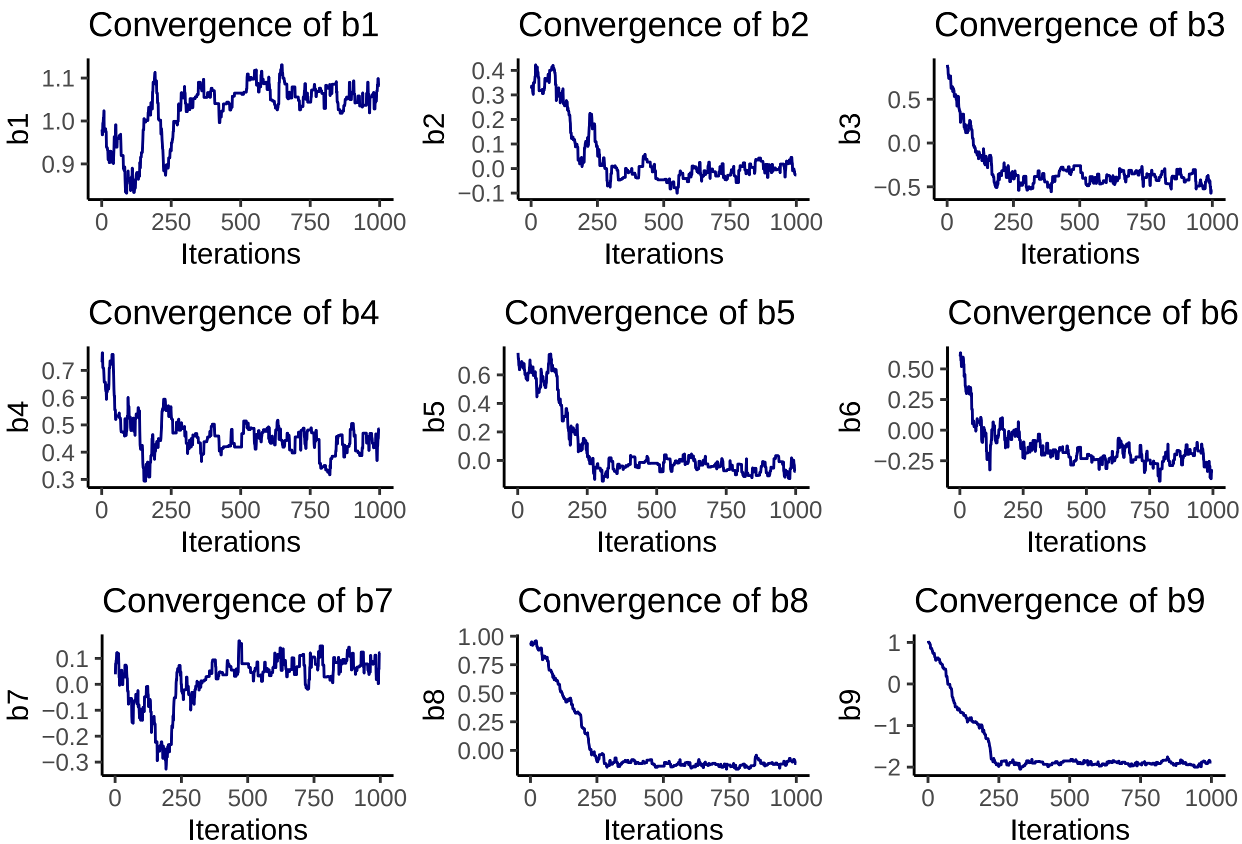
1) Initialize θ^0 and iterate for i = 1, 2, . . .

2) Sample proposal: $\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c \cdot \Sigma)$, where $\Sigma = J^{-1}_{\hat{\theta},y}$

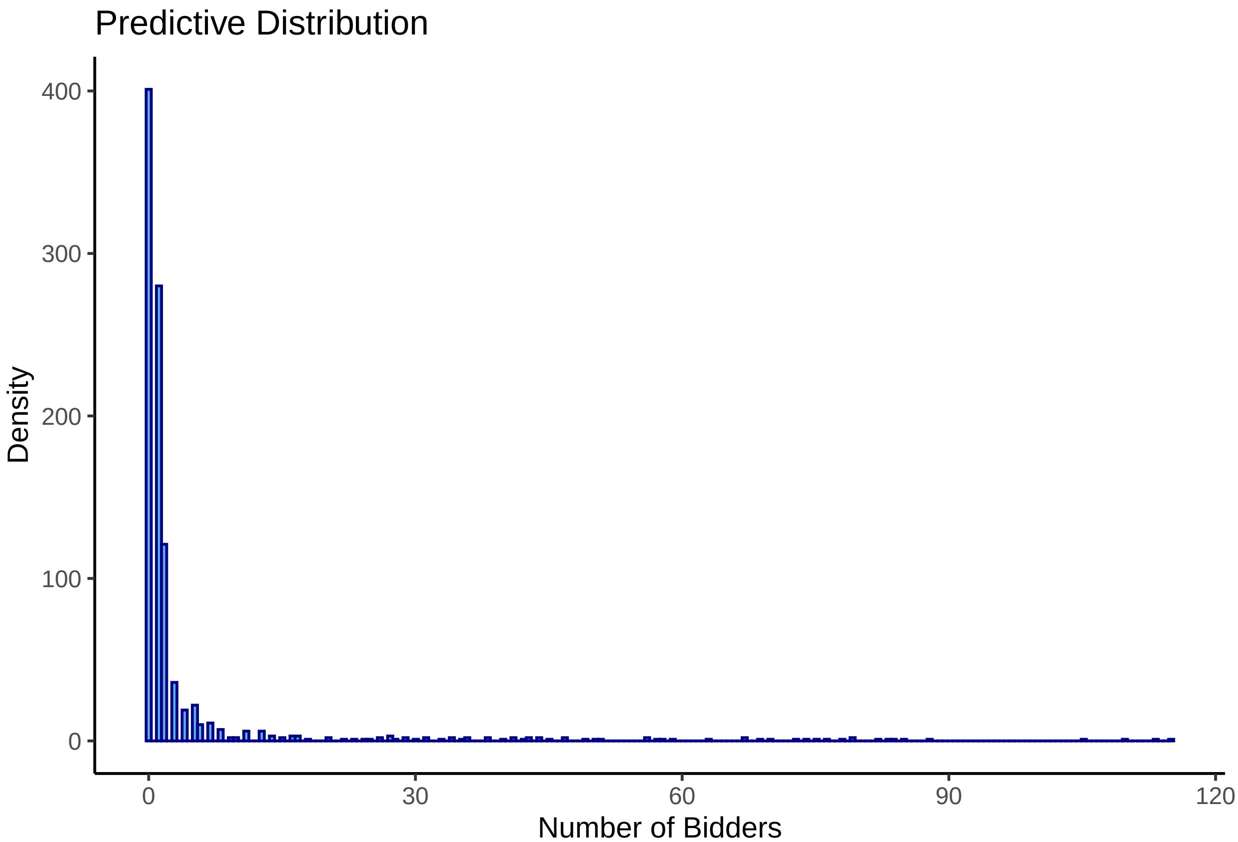
3) Compute the acceptance probability: $\alpha = min\Big(1, \frac{p(\theta_p|y)}{p(\theta^{(i-1)}|y)}$, where :

$\frac{p(\theta_p|y)}{p(\theta^{(i-1)}|y)} = exp\Big[log(p(\theta_p|y)) - log(p(\theta^{(i-1)}|y))$

4) With probability α set $\theta^{(i)} = \theta_p$ and $\theta^{(i)} = \theta^{(i-1)}$

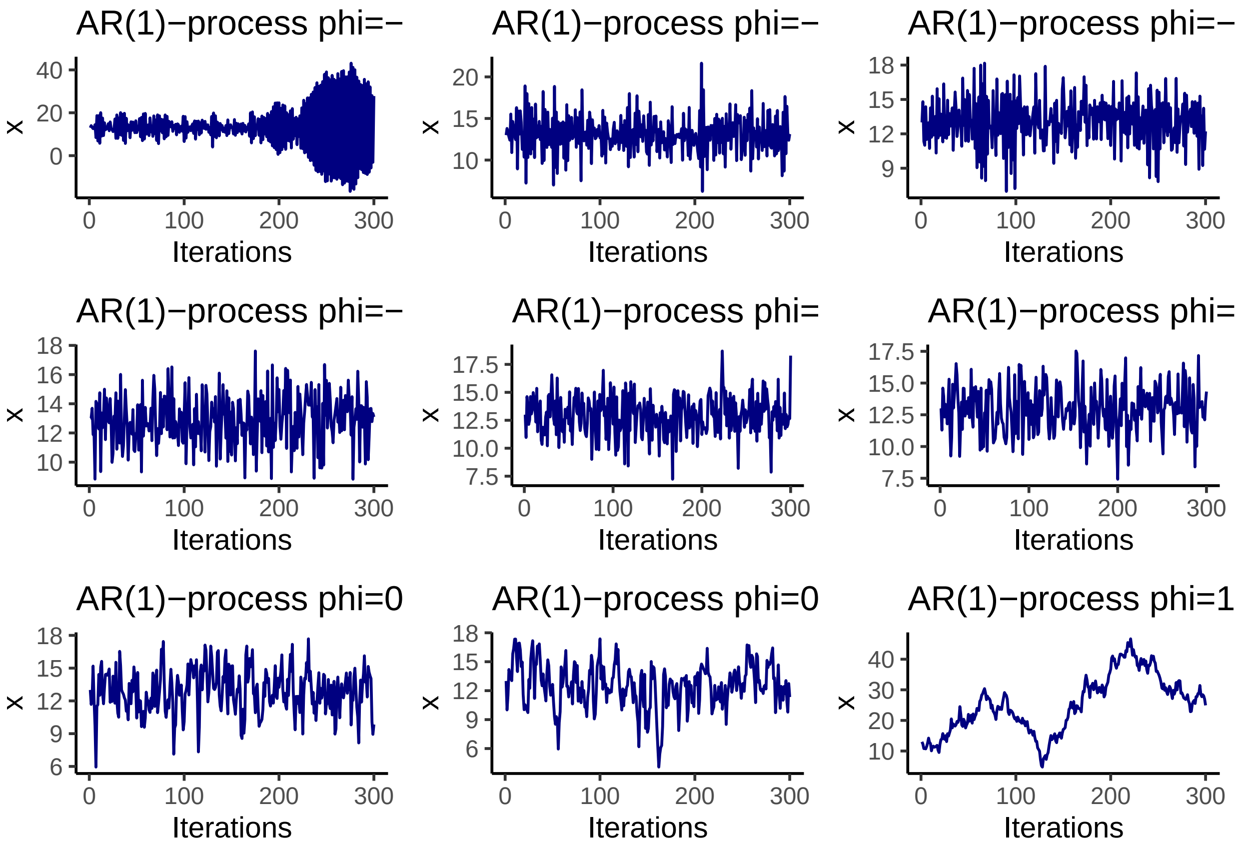

Convergence of b1
b1
1.1
1.0
0.9
0 250 500 750 1000
Iterations
Convergence of b2
b2
0.4
0.3
0.2
0.1
0.0
−0.1
0 250 500 750 1000
Iterations
Convergence of b3
b3
0.5
0.0
−0.5
0 250 500 750 1000
Iterations
Convergence of b4
b4
0.7
0.6
0.5
0.4
0.3
0 250 500 750 1000
Iterations
Convergence of b5
b5
0.6
0.4
0.2
0.0
0 250 500 750 1000
Iterations
Convergence of b6
b6
0.50
0.25
0.00
−0.25
0 250 500 750 1000
Iterations
Convergence of b7
b7
0.1
0.0
−0.1
−0.2
−0.3
0 250 500 750 1000
Iterations
Convergence of b8
b8
1.00
0.75
0.50
0.25
0.00
0 250 500 750 1000
Iterations
Convergence of b9
b9
1
0
−1
−2
0 250 500 750 1000
Iterations

Task d



Assignment 3 *Time series models in Stan*

Task a



Task b

```
## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/incluc
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/incluc
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/incluc
## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
```

```
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.67 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.979774 seconds (Warm-up)
## Chain 1:                0.143125 seconds (Sampling)
## Chain 1:                1.1229 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.7e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.27 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.408295 seconds (Warm-up)
## Chain 2:                0.139528 seconds (Sampling)
## Chain 2:                0.547823 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.31 seconds.
```

```
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.257146 seconds (Warm-up)
## Chain 3:                0.149365 seconds (Sampling)
## Chain 3:                0.406511 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.5e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.25 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.422331 seconds (Warm-up)
## Chain 4:                0.147544 seconds (Sampling)
## Chain 4:                0.569875 seconds (Total)
## Chain 4:

##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
```

```
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.217095 seconds (Warm-up)
## Chain 1:                0.155412 seconds (Sampling)
## Chain 1:                0.372507 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.8e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.28 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.152602 seconds (Warm-up)
## Chain 2:                0.153951 seconds (Sampling)
## Chain 2:                0.306553 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.34 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
```

```
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.239068 seconds (Warm-up)
## Chain 3:                0.136203 seconds (Sampling)
## Chain 3:                0.375271 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '7a86c0d268cad593288653352f880fd9' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.6e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.26 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.340496 seconds (Warm-up)
## Chain 4:                0.131948 seconds (Sampling)
## Chain 4:                0.472444 seconds (Total)
## Chain 4:
```

i)

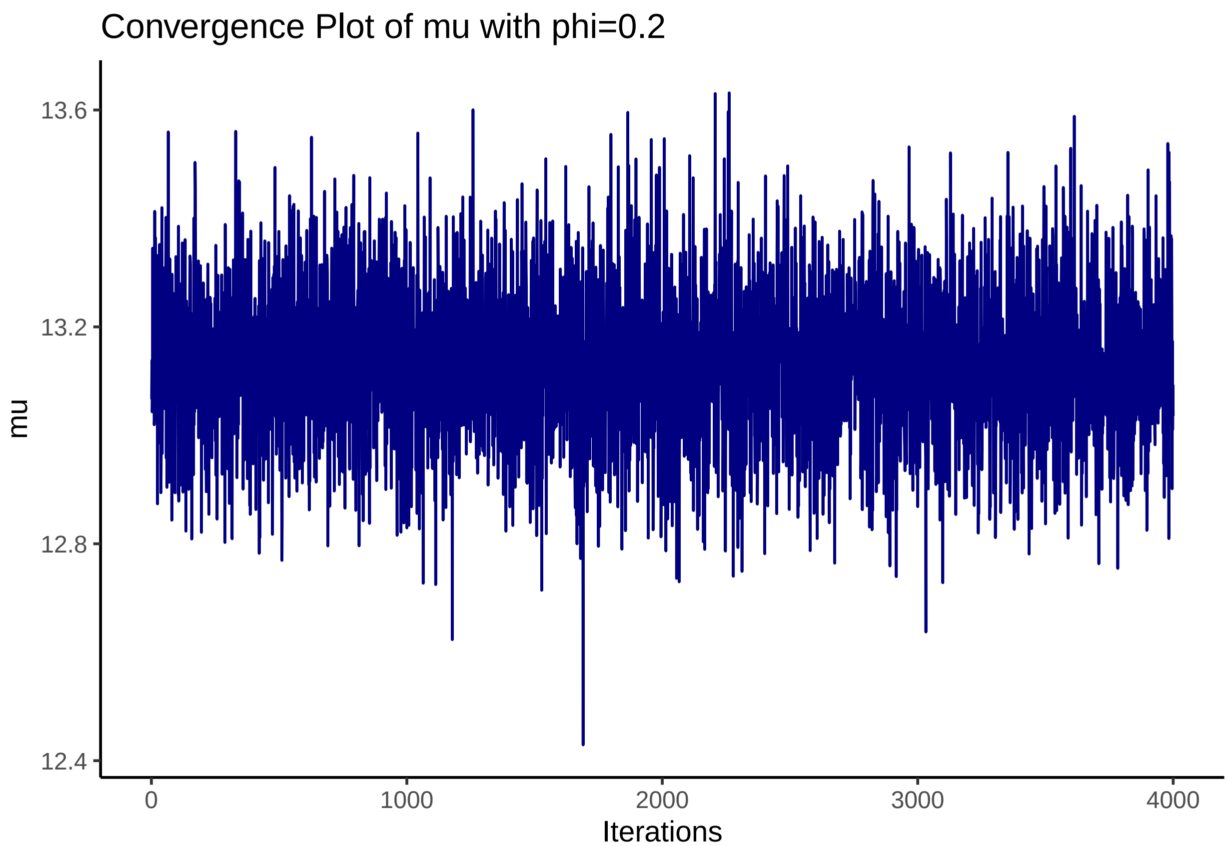
Table 5: Table for phi=0.2

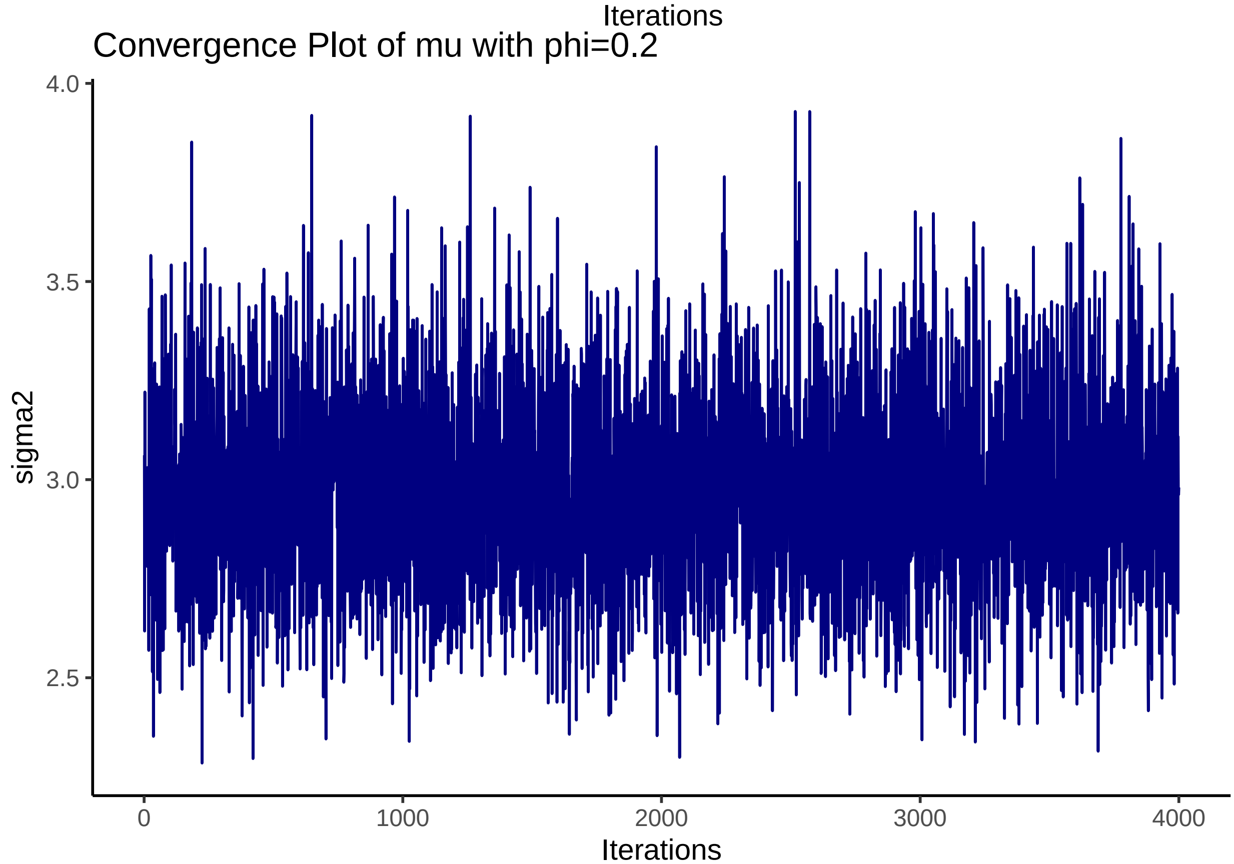
	mean	sd	2.5%	97.5%	n_eff
mu	13.1329660	0.1412896	12.8561068	13.4045697	3796.636
sigma2	2.9675904	0.2436688	2.5199519	3.4647268	3652.744
phi	0.2854719	0.0566617	0.1707683	0.3984766	3886.856

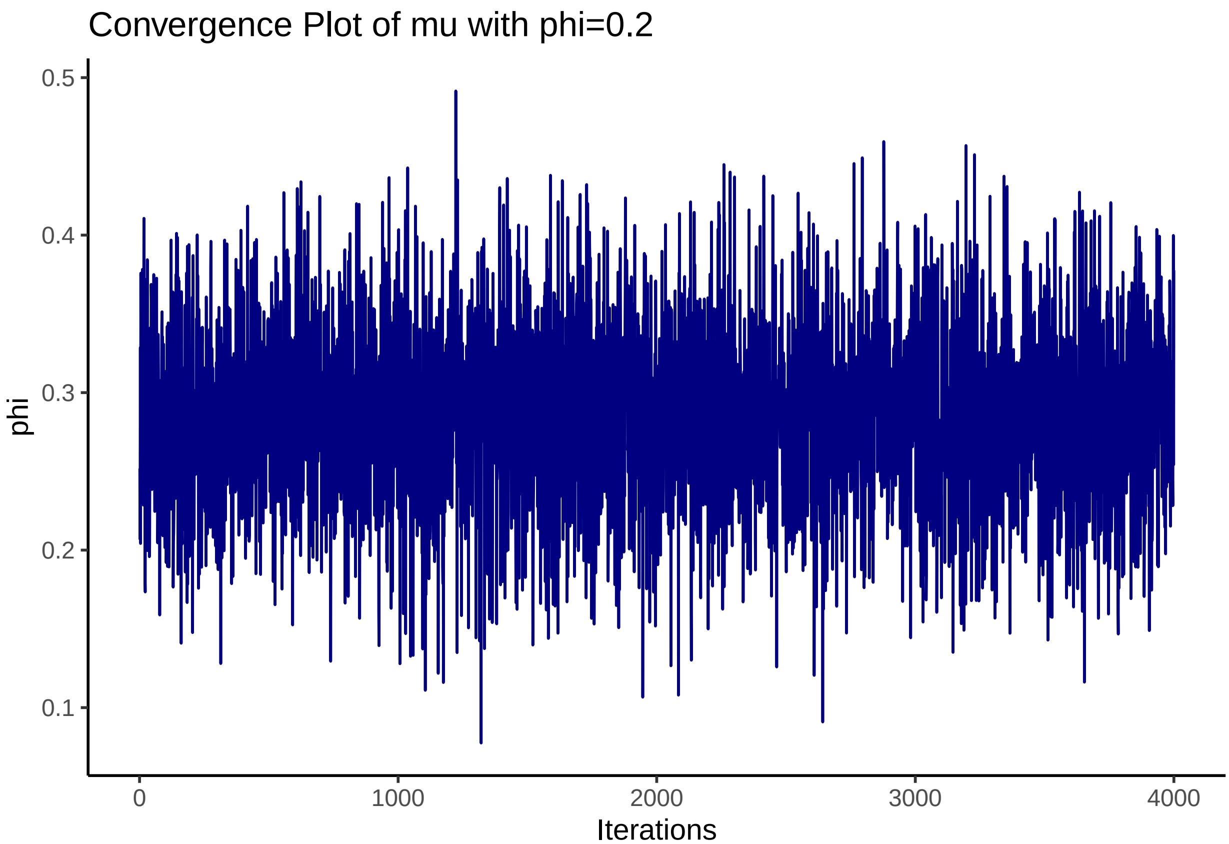
Table 6: Table for phi=0.95

	mean	sd	2.5%	97.5%	n_eff
mu	12.9642423	0.1374885	12.6838053	13.236513	3587.043
sigma2	3.3359148	0.2713306	2.8430332	3.913049	3730.681
phi	0.2298855	0.0575086	0.1169206	0.341547	4297.623

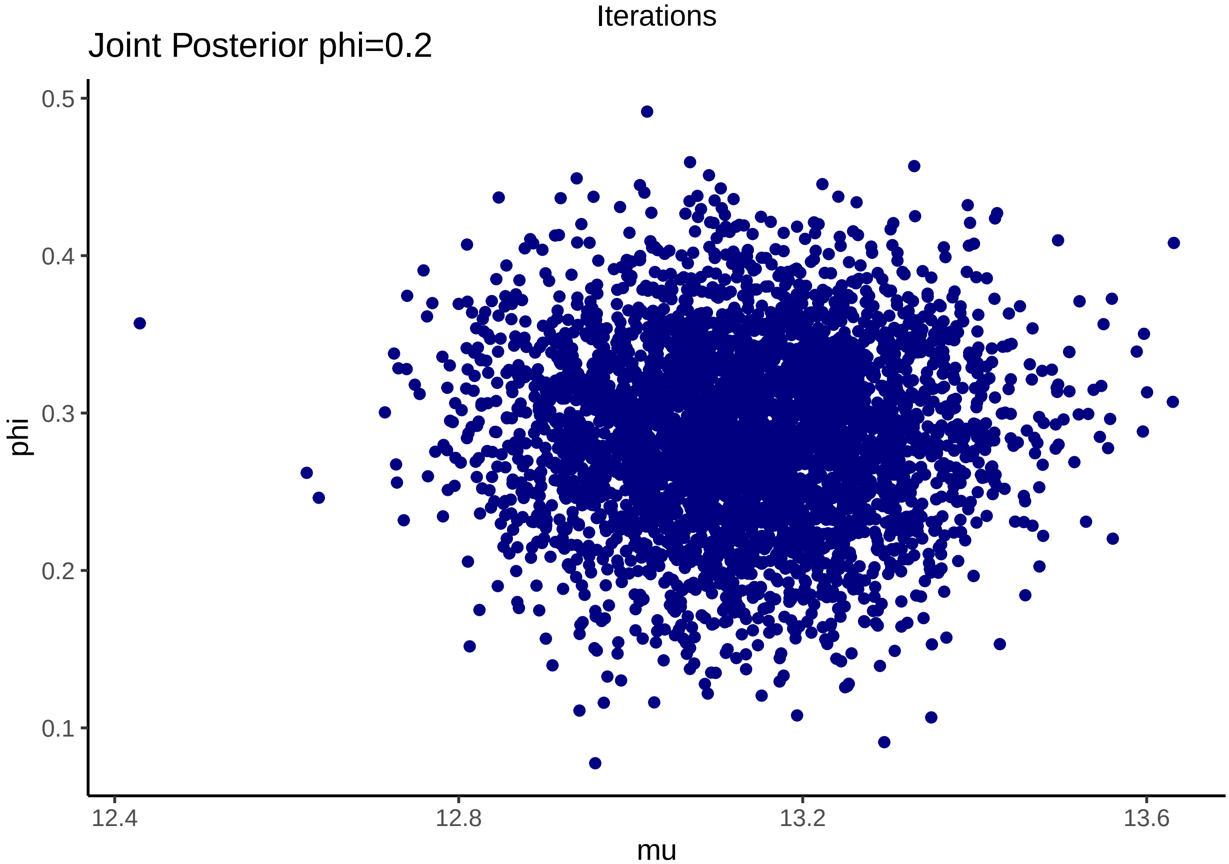
ii)


Convergence Plot of mu with phi=0.2
mu
13.6
13.2
12.8
12.4
0
1000
2000
3000
4000
Iterations

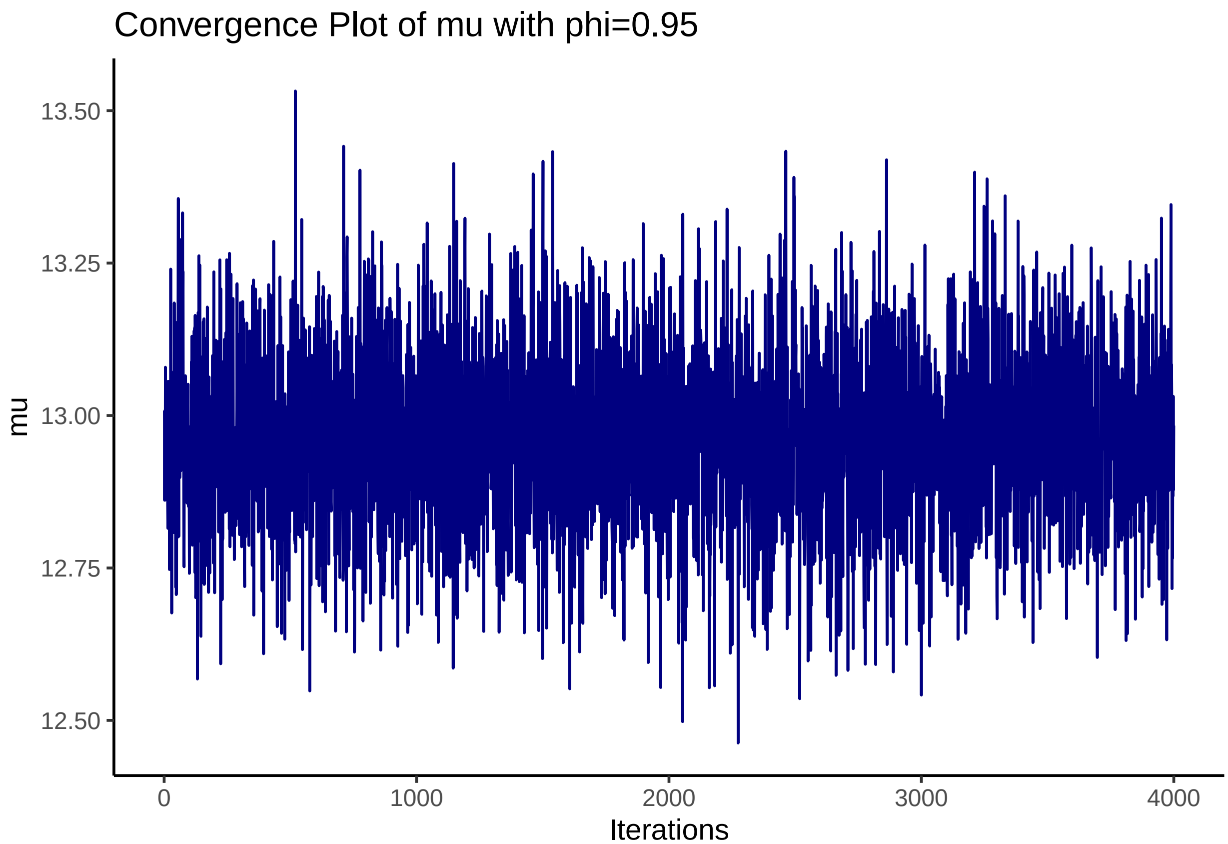

Convergence Plot of mu with phi=0.2
sigma2
4.0
3.5
3.0
2.5
0
1000
2000
3000
4000
Iterations

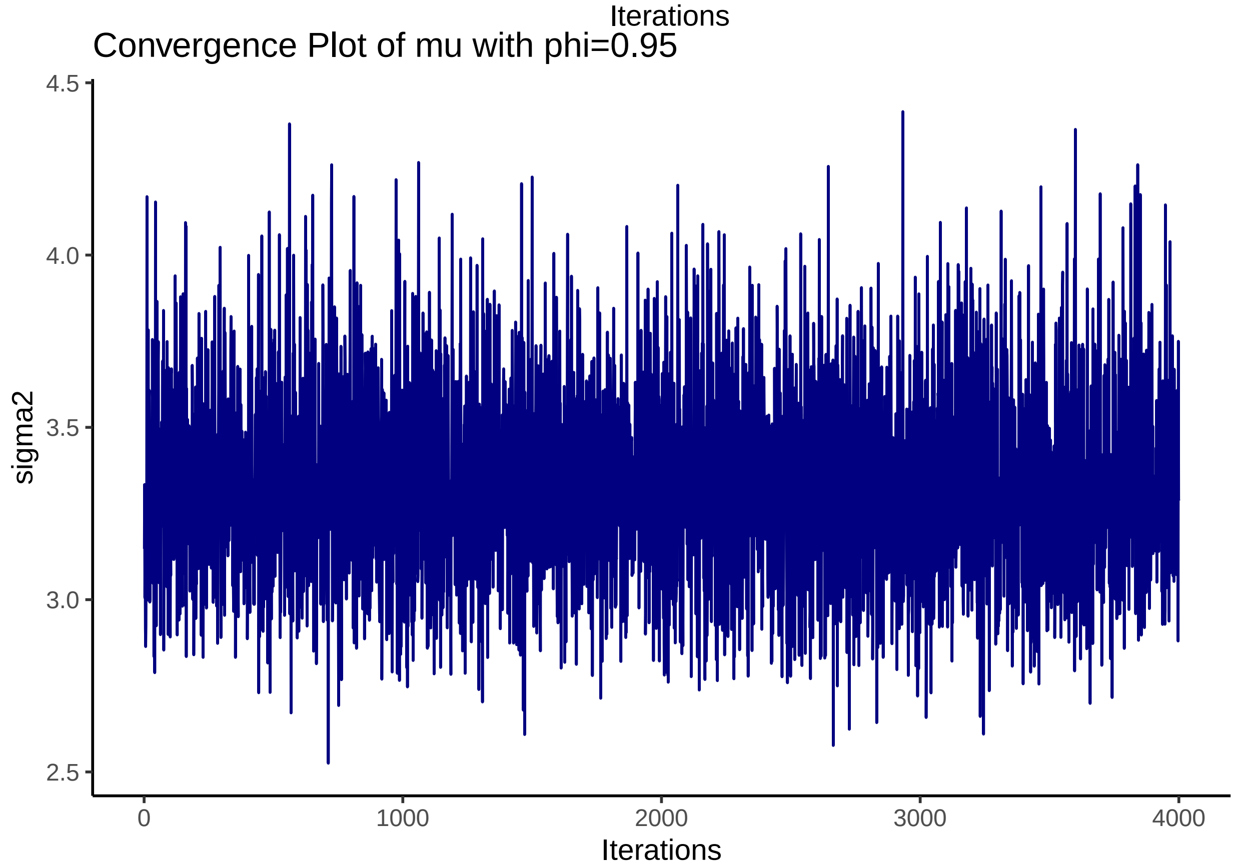


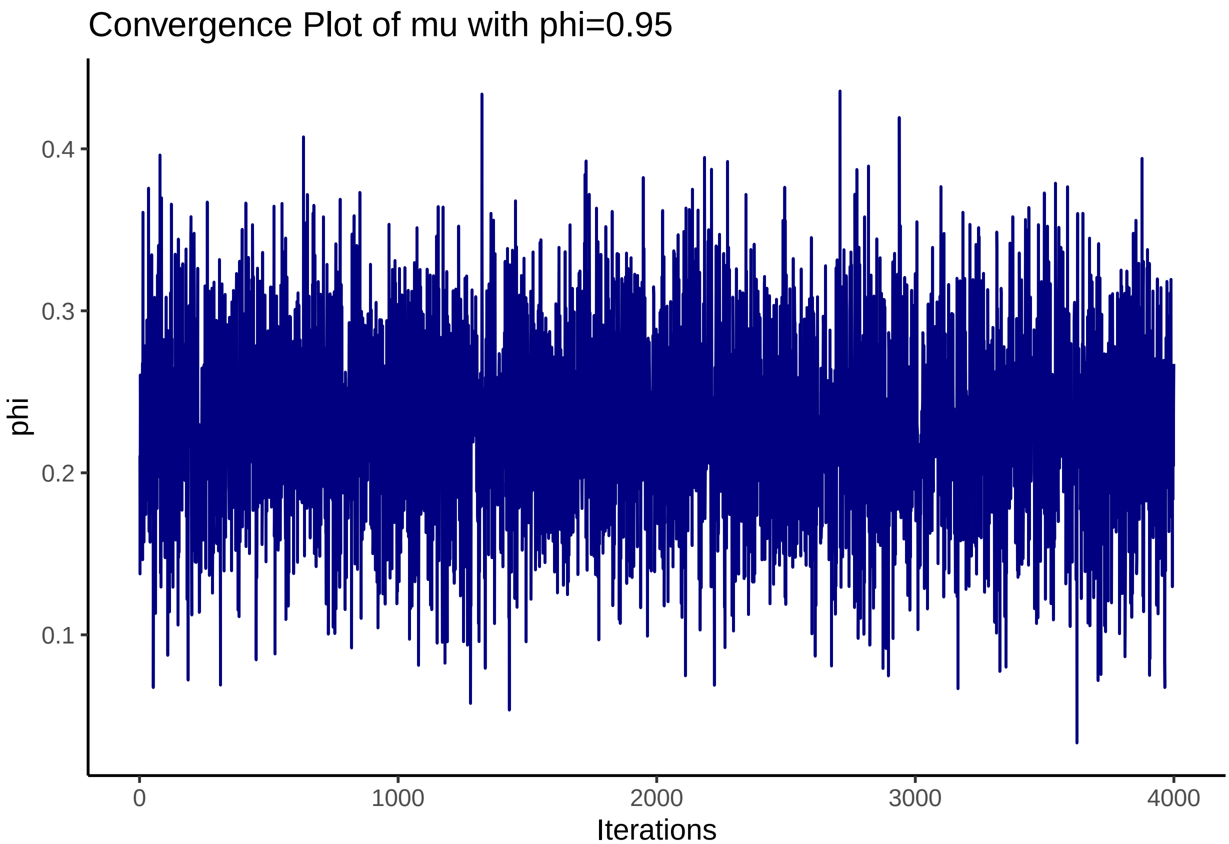
Convergence Plot of mu with phi=0.2
phi
0.1
0.2
0.3
0.4
0.5
0
1000
2000
3000
4000
Iterations

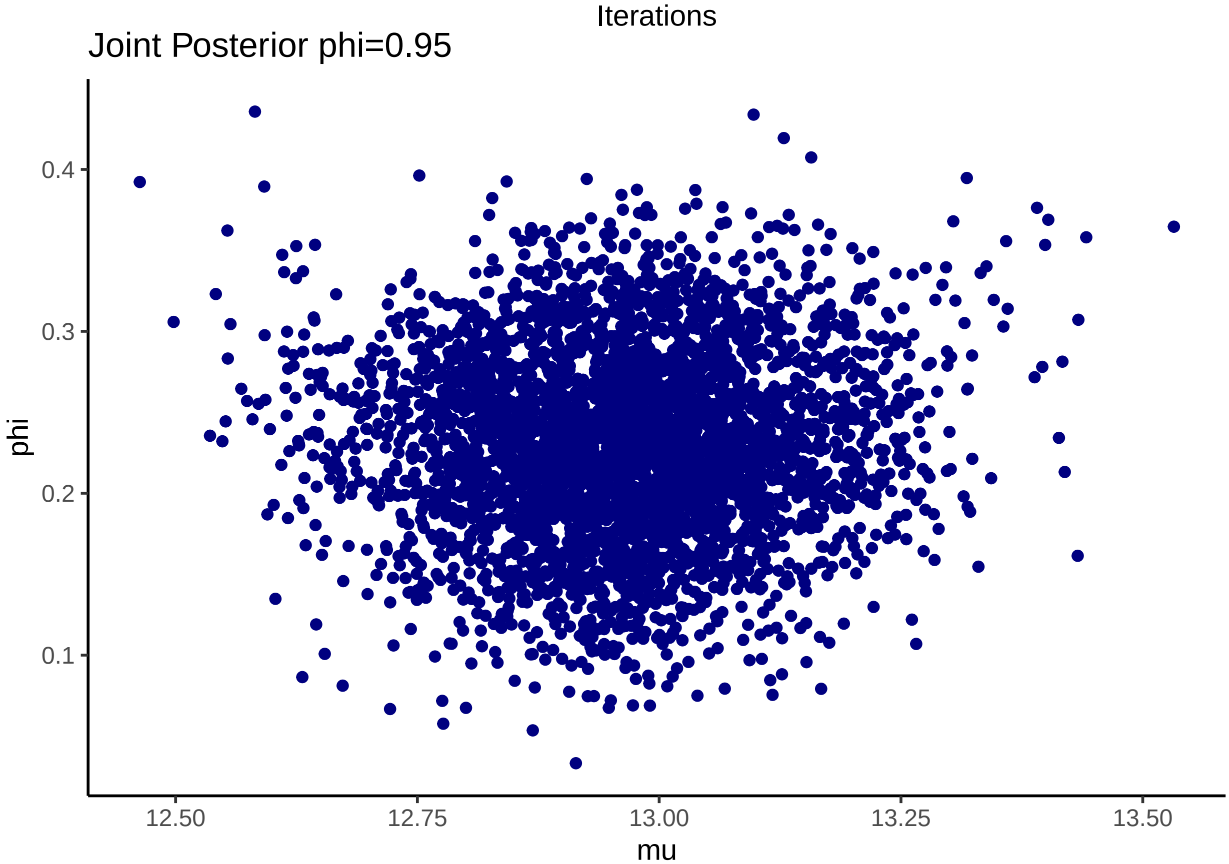


Joint Posterior phi=0.2
phi
0.1
0.2
0.3
0.4
0.5
12.4
12.8
13.2
13.6
mu


Convergence Plot of mu with phi=0.95
13.50
13.25
13.00
12.75
12.50
mu
0
1000
2000
3000
4000
Iterations


Convergence Plot of mu with phi=0.95
4.5
4.0
3.5
3.0
2.5
sigma2
0
1000
2000
3000
4000
Iterations


Convergence Plot of mu with phi=0.95
phi
0.1
0.2
0.3
0.4
0
1000
2000
3000
4000
Iterations


Joint Posterior phi=0.95
phi
0.1
0.2
0.3
0.4
12.50
12.75
13.00
13.25
13.50
mu

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
library(ggplot2)
library(mvtnorm)
library(gridExtra)
library(rstan)
# ____________________________Assignment
# 1____________________________#

# Reading & preparing data
precipitation <- as.data.frame(readRDS("Precipitation.rds"))
colnames(precipitation) <- "records"
log_records <- log(precipitation$records)
# Task 1a
set.seed(1234567890)

# setting up prior parameters
n <- length(log_records)
n_0 <- 1
mu_0 <- mean(log_records)
sigma2_0 <- var(log_records)
tau2_0 <- 1

# normal model with conditionally conjugate prior mu_prop
# <- rnorm(n = 1, mean = mu_0, sqrt(tau2_0)) sigma2_prop <-
# n_0*sigma2_0 / rchisq(1,n_0)

# sigma2 starting value
sigma2 <- sigma2_0

# vectors to store the samples
mu_gibbs <- c()
sigma2_gibbs <- c(sigma2)

N <- 1000
for (i in 1:N) {

    # calculating parameters
    w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
    mu_n <- w * mean(log_records) + (1 - w) * mu_0
    tau2_n <- 1/(n/sigma2 + 1/tau2_0)

    # calculating mu
    mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)

    # temporaly store the previous sigma2
    previous_sigma2 <- sigma2

    # calculating parameter
    n_n <- n_0 + n
```

```
    # calculating sigma2
    sigma2 <- (n_n * ((n_0 * sigma2_0 + sum((log_records - mu)^2))/n_n))/rchisq(1,
        n_n)

    # conditions of how to store the samples
    if (i == 1) {
        mu_gibbs <- c(mu)
        sigma2_gibbs <- c(sigma2)
    } else {
        mu_gibbs <- append(mu_gibbs, c(mu, mu))
        sigma2_gibbs <- append(sigma2_gibbs, c(previous_sigma2,
            sigma2))
    }
}

# calculating autocorrelation for mu
mu_rho <- acf(mu_gibbs, main = "Autocorrelation of mu", col = "red4",
    lwd = 2)

# callculating the inefficiency factor of mu
IF_mu <- 1 + 2 * sum(mu_rho$acf[-1])

# calculating autocorrelation for sigma2
sigma2_rho <- acf(sigma2_gibbs, main = "Autocorrelation of sigma2",
    col = "red4", lwd = 2)

# callculating the inefficiency factor of sigma2
IF_sigma2 <- 1 + 2 * sum(sigma2_rho$acf[-1])

# df with the inefficiency factors of mu & sigma
IFs_df <- data.frame(mu = IF_mu, sigma2 = IF_sigma2)
rownames(IFs_df) <- c("Inefficiency Factors")
knitr::kable(IFs_df)

# df for plot
plot_df_traj <- data.frame(mu = mu_gibbs, sigma2 = sigma2_gibbs)

# plotting some amount of points
plot_df_traj <- plot_df_traj[1:200, ]

# trajectories of the sampled Markov chains.
ggplot(plot_df_traj) + geom_path(aes(x = mu, y = sigma2), color = "navy") +
    geom_point(aes(x = mu[1], y = sigma2[1]), color = "red2",
        size = 2) + geom_point(aes(x = mu[200], y = sigma2[200]),
    color = "green2", size = 2) + ggtitle("Trajectories of the Sampled Markov chains.") +
    ylab("sigma2") + xlab("mu") + theme_classic()

# Task b

set.seed(1234567890)

# predicted draws
predictions <- rnorm(nrow(precipitation), mu_gibbs, sqrt(sigma2_gibbs))
```

```
# kernel density estimation
density_actual <- density(precipitation$records)
density_pred <- density(exp(predictions))

# df for plot
plot_df_dens <- data.frame(actual_x = density_actual$x, actual_y = density_actual$y,
    pred_x = density_pred$x, pred_y = density_pred$y)

ggplot(plot_df_dens) + geom_line(aes(x = actual_x, y = actual_y),
    color = "navy") + geom_line(aes(x = pred_x, y = pred_y),
    color = "red2") + theme(legend.position = "right") + scale_color_identity(guide = "legend",
    name = "", breaks = c("navy", "red2"), labels = c("Actual Values",
        "Predicted Values")) + xlab("Value") + ylab("Density") +
    theme_classic()

# __________________Assignment 2_________________#

# Reading data
ebay <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
# Task a

# glm function, + 0 in order to do not input the covariate
# Const
model <- glm(formula = nBids ~ 0 + ., data = ebay, family = poisson)

# print the summary of the model
summary(model)
# the below code snippets from a demo of logistic
# regression in Lecture 6

# target value
y <- ebay$nBids

# prior inputs Select which covariates/features to include
X <- as.matrix(ebay[2:10])
Xnames <- colnames(X)

Npar <- dim(X)[2]

# Setting up the prior
mu <- as.matrix(rep(0, Npar))  # Prior mean vector
Sigma <- 100 * solve(t(X) %*% X)  # Prior covariance matrix

# Functions that returns the log posterior for the logistic
# and probit regression.  First input argument of this
# function must be the parameters we optimize on, i.e. the
# regression coefficients beta.

LogPostPoisson <- function(betas, y, X, mu, Sigma) {
    linPred <- X %*% betas
    # loglikelihood of Poisson regression
    logLik <- sum(linPred * y - exp(linPred))
    if (abs(logLik) == Inf)
```

```r
        logLik = -20000  # Likelihood is not finite, stear the optimizer away from here!
    logPrior <- dmvnorm(betas, mu, Sigma, log = TRUE)

    return(logLik + logPrior)
}

# Select the initial values for beta
initVal <- matrix(0, Npar, 1)

# The argument control is a list of options to the
# optimizer optim, where fnscale=-1 means that we minimize
# the negative log posterior. Hence, we maximize the log
# posterior.
OptimRes <- optim(initVal, LogPostPoisson, gr = NULL, y, X, mu,
    Sigma, method = c("BFGS"), control = list(fnscale = -1),
    hessian = TRUE)

names(OptimRes$par) <- Xnames  # Naming the coefficient by covariates
approxPostStd <- sqrt(diag(-solve(OptimRes$hessian)))  # Computing approximate standard deviations.
names(approxPostStd) <- Xnames  # Naming the coefficient by covariates
# print('The posterior mode is:') print(OptimRes$par)
# print('The approximate posterior standard deviation is:')
approxPostStd <- sqrt(diag(-solve(OptimRes$hessian)))
# print(approxPostStd)

invHessian <- data.frame(invhes = -solve(OptimRes$hessian))
colnames(invHessian) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(invHessian)

df_post_mode <- data.frame(post_mode = OptimRes$par)
colnames(df_post_mode) <- c("Value")
rownames(df_post_mode) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(df_post_mode)

df_approxPostStd <- data.frame(approxPostStd = sqrt(diag(-solve(OptimRes$hessian))))
colnames(df_approxPostStd) <- c("Value")
rownames(df_approxPostStd) <- c("Constant", "PowerSeller", "VerifyID",
    "Sealed", "MinBlem", "MajBlem", "LargNeg", "LogBook", "MinBidShare")
knitr::kable(df_approxPostStd)
set.seed(1234567890)

RWMSampler <- function(N, logPostFunc, init_beta, constant, cov_mat,
    target_val, features, mu) {

    init_sample = rmvnorm(1, init_beta, cov_mat)

    # matrix to store the betas
    betas <- matrix(nrow = N, ncol = length(init_sample))

    betas[1, ] <- init_sample
```

```r
    for (i in 2:N) {
        # sample proposal
        sample_proposal <- as.vector(rmvnorm(1, betas[i - 1,
            ], constant * cov_mat))

        # LogPostPoisson <- function(betas,y,X,mu,Sigma)
        # calculating the new posterior
        post_new <- logPostFunc(sample_proposal, target_val,
            features, mu, cov_mat)

        # calculating the old posterior
        post_old <- logPostFunc(betas[i - 1, ], target_val, features,
            mu, cov_mat)

        # acceptance probability
        a <- min(1, exp(post_new - post_old))

        u <- runif(1, 0, 1)

        if (u < a) {
            betas[i, ] <- sample_proposal
        } else {
            betas[i, ] <- betas[i - 1, ]
        }
    }
    return(betas)
}


RWMbetas <- RWMSampler(1000, LogPostPoisson, runif(9, 0, 1),
    0.35, -solve(OptimRes$hessian), ebay$nBids, as.matrix(ebay[2:10]),
    OptimRes$par)
plot_df_RWM <- data.frame(RWMbetas)
colnames(plot_df_RWM) <- c("b1", "b2", "b3", "b4", "b5", "b6",
    "b7", "b8", "b9")
plot_df_RWM$iterations <- 1:1000

p1 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b1),
    color = "navy") + ggtitle("Convergence of b1") + xlab("Iterations") +
    ylab("b1") + theme_classic()

p2 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b2),
    color = "navy") + ggtitle("Convergence of b2") + xlab("Iterations") +
    ylab("b2") + theme_classic()

p3 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b3),
    color = "navy") + ggtitle("Convergence of b3") + xlab("Iterations") +
    ylab("b3") + theme_classic()

p4 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b4),
    color = "navy") + ggtitle("Convergence of b4") + xlab("Iterations") +
    ylab("b4") + theme_classic()
```

```
p5 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b5),
    color = "navy") + ggtitle("Convergence of b5") + xlab("Iterations") +
    ylab("b5") + theme_classic()

p6 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b6),
    color = "navy") + ggtitle("Convergence of b6") + xlab("Iterations") +
    ylab("b6") + theme_classic()

p7 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b7),
    color = "navy") + ggtitle("Convergence of b7") + xlab("Iterations") +
    ylab("b7") + theme_classic()

p8 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b8),
    color = "navy") + ggtitle("Convergence of b8") + xlab("Iterations") +
    ylab("b8") + theme_classic()

p9 <- ggplot(plot_df_RWM) + geom_line(aes(x = iterations, y = b9),
    color = "navy") + ggtitle("Convergence of b9") + xlab("Iterations") +
    ylab("b9") + theme_classic()

grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, ncol = 3)
set.seed(1234567890)

auction <- c(1, 1, 0, 1, 0, 1, 0, 1.2, 0.8)

nbidders = c()
for (i in 1:nrow(RWMbetas)) {
    nbidders[i] <- rpois(1, exp(RWMbetas[i, ] %*% auction))
}

nbidders <- data.frame(nbidders)

prob <- sum(nbidders$nbidders == 0)/(nrow(nbidders))

ggplot(nbidders, aes(x = nbidders)) + geom_histogram(bins = 200,
    color = "navy", fill = "steelblue2") + ggtitle("Predictive Distribution") +
    xlab("Number of Bidders") + ylab("Density") + theme_classic()
# __________________Assignment 3__________________#

# Task 3a

set.seed(12345)

# AR(1)-process
ar_process <- function(t, mu, phi, sigma2) {
    x <- c()
    x[1] <- mu
    for (i in 2:t) {
        x[i] <- mu + phi * (x[i - 1] - mu) + rnorm(1, 0, sqrt(sigma2))
    }
    return(x)
}
```

```r
# given parameters
mu <- 13
sigma2 <- 3
t <- 300
phi <- seq(-1, 1, 0.25)

res <- as.data.frame(sapply(phi, function(phi) ar_process(t,
    mu, phi, sigma2)))

res$iterations <- 1:t

# Time series plots of different phis

p1 <- ggplot(res) + geom_line(aes(x = iterations, y = V1), color = "navy") +
    ggtitle("AR(1)-process phi=-1") + xlab("Iterations") + ylab("x") +
    theme_classic()

p2 <- ggplot(res) + geom_line(aes(x = iterations, y = V2), color = "navy") +
    ggtitle("AR(1)-process phi=-0.75") + xlab("Iterations") +
    ylab("x") + theme_classic()

p3 <- ggplot(res) + geom_line(aes(x = iterations, y = V3), color = "navy") +
    ggtitle("AR(1)-process phi=-0.5") + xlab("Iterations") +
    ylab("x") + theme_classic()

p4 <- ggplot(res) + geom_line(aes(x = iterations, y = V4), color = "navy") +
    ggtitle("AR(1)-process phi=-0.25") + xlab("Iterations") +
    ylab("x") + theme_classic()

p5 <- ggplot(res) + geom_line(aes(x = iterations, y = V5), color = "navy") +
    ggtitle("AR(1)-process phi=0") + xlab("Iterations") + ylab("x") +
    theme_classic()

p6 <- ggplot(res) + geom_line(aes(x = iterations, y = V6), color = "navy") +
    ggtitle("AR(1)-process phi=0.25") + xlab("Iterations") +
    ylab("x") + theme_classic()

p7 <- ggplot(res) + geom_line(aes(x = iterations, y = V7), color = "navy") +
    ggtitle("AR(1)-process phi=0.5") + xlab("Iterations") + ylab("x") +
    theme_classic()

p8 <- ggplot(res) + geom_line(aes(x = iterations, y = V8), color = "navy") +
    ggtitle("AR(1)-process phi=0.75") + xlab("Iterations") +
    ylab("x") + theme_classic()

p9 <- ggplot(res) + geom_line(aes(x = iterations, y = V9), color = "navy") +
    ggtitle("AR(1)-process phi=1") + xlab("Iterations") + ylab("x") +
    theme_classic()

grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, ncol = 3)

# Task 3b
```

```r
set.seed(1234567890)

# stan_model
StanModel = "
data {
  int<lower=0> N;
  vector[N] x;
}
parameters {
  real mu;
  real<lower=0> sigma2;
  real phi;
}
model {
  mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
  sigma2 ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1,sigma 2
  for(i in 2:N){
    x[i] ~ normal( mu + phi*(x[i-1]-mu), sqrt(sigma2));
  }
}"

# given parameters
phi1 <- 0.2
phi2 <- 0.95

# ar with given parameters
res1 <- ar_process(t, mu, phi1, sigma2)
res2 <- ar_process(t, mu, phi1, sigma2)

data1 <- list(N = t, x = res1)
data2 <- list(N = t, x = res2)

warmup <- 1000
niter <- 2000


fit1 <- stan(model_code = StanModel, data = data1, warmup = warmup,
    iter = niter, chains = 4)
fit2 <- stan(model_code = StanModel, data = data2, warmup = warmup,
    iter = niter, chains = 4)

fit1_summary <- summary(fit1)
phi1_stats <- fit1_summary$summary[1:3, c(1, 3, 4, 8, 9)]

knitr::kable(phi1_stats, caption = "Table for phi=0.2")

fit2_summary <- summary(fit2)
phi2_stats <- fit2_summary$summary[1:3, c(1, 3, 4, 8, 9)]

knitr::kable(phi2_stats, caption = "Table for phi=0.95")

postDraws1 <- data.frame(extract(fit1))
postDraws1$iterations <- 1:4000
```

```
ggplot(postDraws1) + geom_line(aes(x = iterations, y = mu), color = "navy") +
    ggtitle("Convergence Plot of mu with phi=0.2") + xlab("Iterations") +
    ylab("mu") + theme_classic()

ggplot(postDraws1) + geom_line(aes(x = iterations, y = sigma2),
    color = "navy") + ggtitle("Convergence Plot of mu with phi=0.2") +
    xlab("Iterations") + ylab("sigma2") + theme_classic()

ggplot(postDraws1) + geom_line(aes(x = iterations, y = phi),
    color = "navy") + ggtitle("Convergence Plot of mu with phi=0.2") +
    xlab("Iterations") + ylab("phi") + theme_classic()

ggplot(postDraws1) + geom_point(aes(x = mu, y = phi), color = "navy") +
    ggtitle("Joint Posterior phi=0.2") + theme_classic()
postDraws2 <- data.frame(extract(fit2))
postDraws2$iterations <- 1:4000

ggplot(postDraws2) + geom_line(aes(x = iterations, y = mu), color = "navy") +
    ggtitle("Convergence Plot of mu with phi=0.95") + xlab("Iterations") +
    ylab("mu") + theme_classic()

ggplot(postDraws2) + geom_line(aes(x = iterations, y = sigma2),
    color = "navy") + ggtitle("Convergence Plot of mu with phi=0.95") +
    xlab("Iterations") + ylab("sigma2") + theme_classic()

ggplot(postDraws2) + geom_line(aes(x = iterations, y = phi),
    color = "navy") + ggtitle("Convergence Plot of mu with phi=0.95") +
    xlab("Iterations") + ylab("phi") + theme_classic()

ggplot(postDraws2) + geom_point(aes(x = mu, y = phi), color = "navy") +
    ggtitle("Joint Posterior phi=0.95") + theme_classic()
```