Bayesian Learning - lab4

Joris van Doorn , Weng Hang Wong

4/15/2020

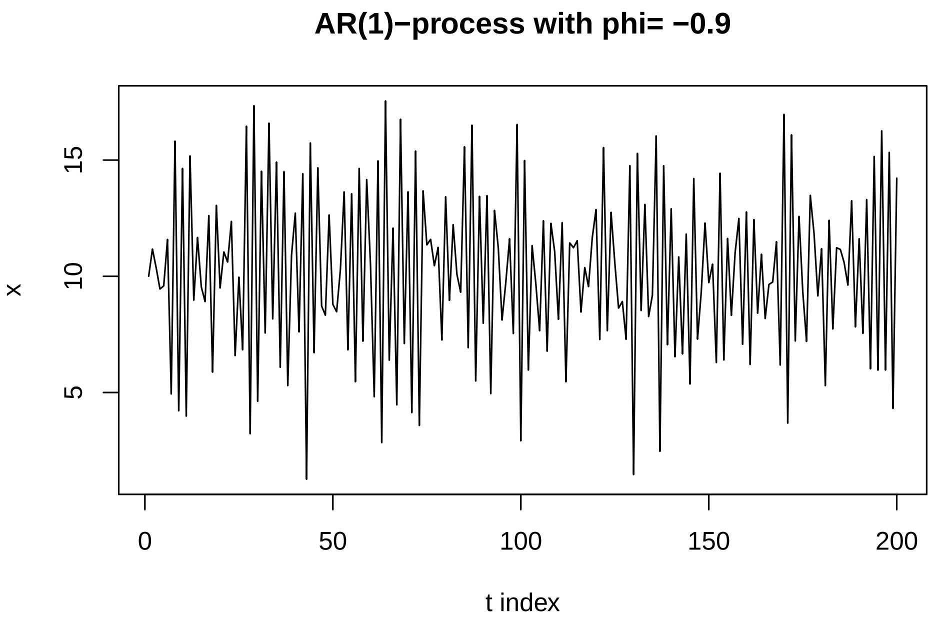
1. Time series models in Stan

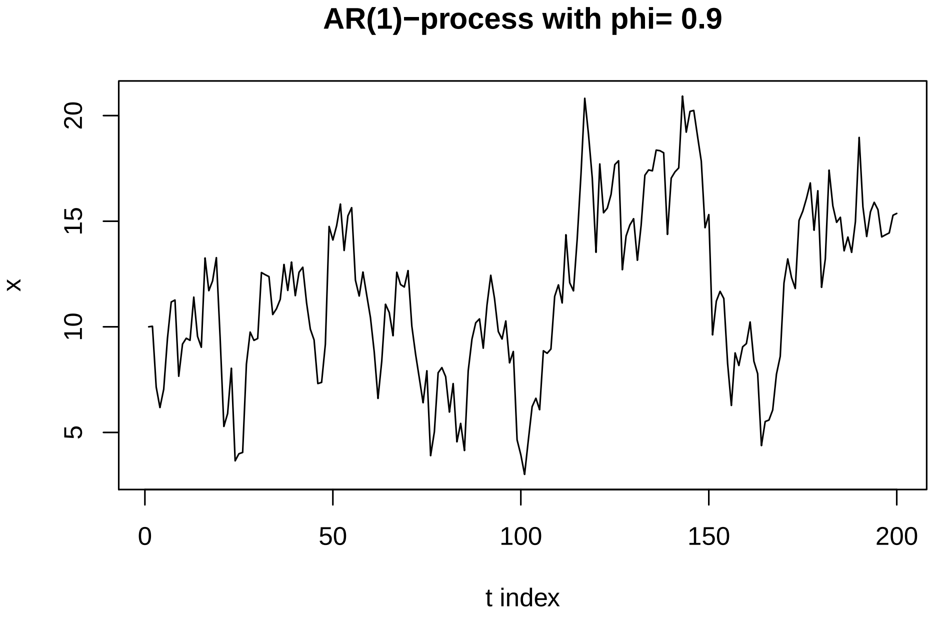
(a) Write a function in R that simulates data from the AR(1)-process

$$x_t = \mu + \phi(x_{t-1} - \mu) + \varepsilon_t, \varepsilon_t \sim N(0, \sigma^2),$$

for given values of $\mu, \phi \ and \ \sigma^2$ Start the process at $x_1 = \mu$ and then simulate values for x_t for t=2, 3. . . ,T and return the vector $x_{1:T}$ containing all time points. Use $\mu = 10, \sigma^2 = 2 \ and \ T = 2000$ and look at some different realizations (simulations) of $x_{1:T}$ for values of ϕ between -1 and 1 (this is the interval of ϕ where the AR(1)-process is stable). Include a plot of at least one realization in the report. What effect does the value of ϕ have on $x_{1:T}$?

Here, we using the given value to simulate the AR(1)-process, where the values of ϕ is between -1 and 1, so we take the ϕ as -0.9 and 0.9, so that to observe the difference between two plots.ϕ is a momentum parameters that can be used to ajust the algorithm, According to the below two plots, it's very obviously to see that with a low value of $\phi = -0.9$ is moving on the plot very intensively. On the other hand, with a higher value $\phi = 0.9$, it comes with a less difference in each iteration so it moves more slowly.





(b) Use your function from a) to simulate two AR(1)-processes, $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.95$. Now, treat your simulated vectors as synthetic data, and treat the values of $\mu, \phi \; and \; \sigma^2$ as unknown and estimate them using MCMC. Implement Stan-code that samples from the posterior of the three parameters, using suitable non-informative priors of your choice. [Hint: Lookat the time-series models examples in the Stan user's guide/reference manual, and note the different parameterization used here.]

i. Report the posterior mean, 95% credible intervals and the number of effective posterior samples for the three inferred parameters for each of the simulated AR(1)-process. Are you able to estimate the true values?

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.19.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Running /usr/lib/R/bin/R CMD SHLIB foo.c
## gcc -std=gnu99 -I"/usr/share/R/include" -DNDEBUG   -I"/home/mecon/R/x86_64-pc-linux-gnu-library/3.6/
## In file included from /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/Core:88:
##                  from /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/Dense:1,
##                  from /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/StanHeaders/include/stan/math/pr
##                  from <command-line>:0:
## /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:1:
```

```
##  namespace Eigen {
##  ^~~~~~~~~
## /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:17:
##  namespace Eigen {
##                  ^
## In file included from /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/Dense:1:
##                  from /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/StanHeaders/include/stan/math/pr
##                  from <command-line>:0:
## /home/mecon/R/x86_64-pc-linux-gnu-library/3.6/RcppEigen/include/Eigen/Core:96:10: fatal error: compl
##  #include <complex>
##           ^~~~~~~~~
## compilation terminated.
## /usr/lib/R/etc/Makeconf:168: recipe for target 'foo.o' failed
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 5.2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.52 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: 0.106478 seconds (Warm-up)
## Chain 1:                0.128561 seconds (Sampling)
## Chain 1:                0.235039 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.19 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: 0.10539 seconds (Warm-up)
## Chain 2:                0.186725 seconds (Sampling)
## Chain 2:                0.292115 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.2e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.22 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: 0.133024 seconds (Warm-up)
## Chain 3:                0.27337 seconds (Sampling)
## Chain 3:                0.406394 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.44 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: 0.174921 seconds (Warm-up)
## Chain 4:                0.098746 seconds (Sampling)
## Chain 4:                0.273667 seconds (Total)
## Chain 4:

##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.22 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: 0.583708 seconds (Warm-up)
## Chain 1:                0.241106 seconds (Sampling)
## Chain 1:                0.824814 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.8e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.18 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: 0.498637 seconds (Warm-up)
## Chain 2:                0.230207 seconds (Sampling)
## Chain 2:                0.728844 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.36 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: 0.443923 seconds (Warm-up)
## Chain 3:                0.343158 seconds (Sampling)
## Chain 3:                0.787081 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '4fd8f8e983e46e9858faca2f63abcb18' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.7e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.17 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: 0.378854 seconds (Warm-up)
## Chain 4:                0.407856 seconds (Sampling)
## Chain 4:                0.78671 seconds (Total)
## Chain 4:
```

```
## Inference for Stan model: 4fd8f8e983e46e9858faca2f63abcb18.
## 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% n_eff
## mu       10.00    0.00 0.24    9.52    9.84    9.99   10.16   10.48  3412
## phi       0.41    0.00 0.07    0.27    0.36    0.41    0.45    0.53  3511
## sigma2    3.98    0.01 0.40    3.29    3.70    3.95    4.22    4.86  3844
## lp__   -238.57    0.03 1.24 -241.77 -239.18 -238.25 -237.65 -237.15  1929
##        Rhat
## mu        1
## phi       1
## sigma2    1
## lp__      1
##
## Samples were drawn using NUTS(diag_e) at Mon May 25 18:31:47 2020.
## 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).
```

Table 1: Posterior of x

parameters	mean	X95CI	EffectSamples	TrueValue
mu	10.43	(9.98,10.89)	3609	9.996674
phi	0.35	(0.22,0.48)	3575	0.300000
sigma2	4.26	(3.52,5.16)	3550	4.637813

```
## Inference for Stan model: 4fd8f8e983e46e9858faca2f63abcb18.
## 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% n_eff
## mu       17.79    1.19 22.08   -5.68   11.90   14.26   17.65   73.92   347
## phi       0.96    0.00  0.03    0.90    0.94    0.96    0.98    1.00   421
## sigma2    4.17    0.01  0.43    3.40    3.87    4.14    4.42    5.11   835
## lp__   -245.38    0.14  2.23 -251.03 -246.49 -244.71 -243.71 -242.88   259
##        Rhat
## mu     1.01
## phi    1.01
## sigma2 1.00
## lp__   1.03
##
## Samples were drawn using NUTS(diag_e) at Mon May 25 18:31:50 2020.
## 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).
```

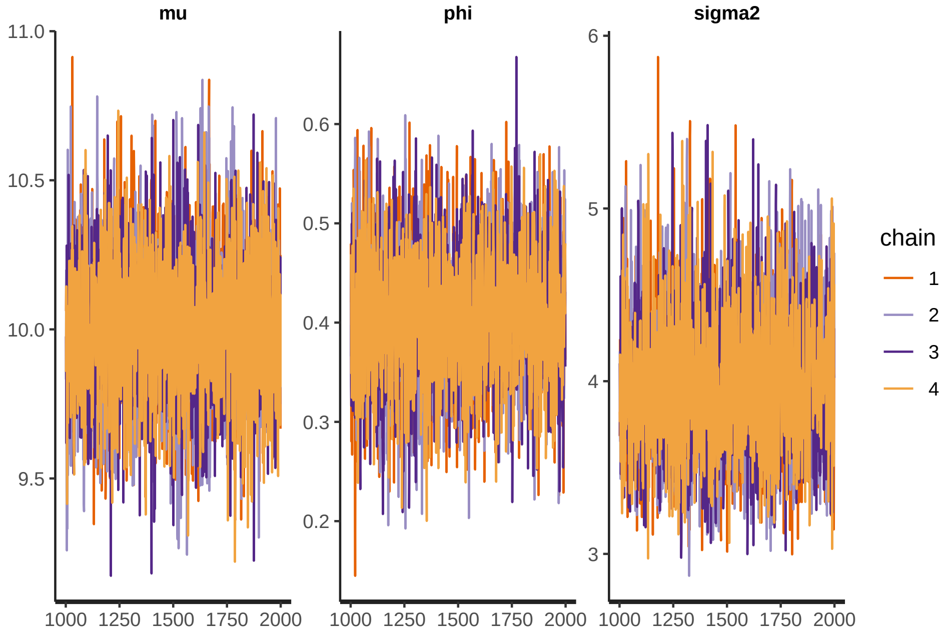
Table 2: Posterior of y

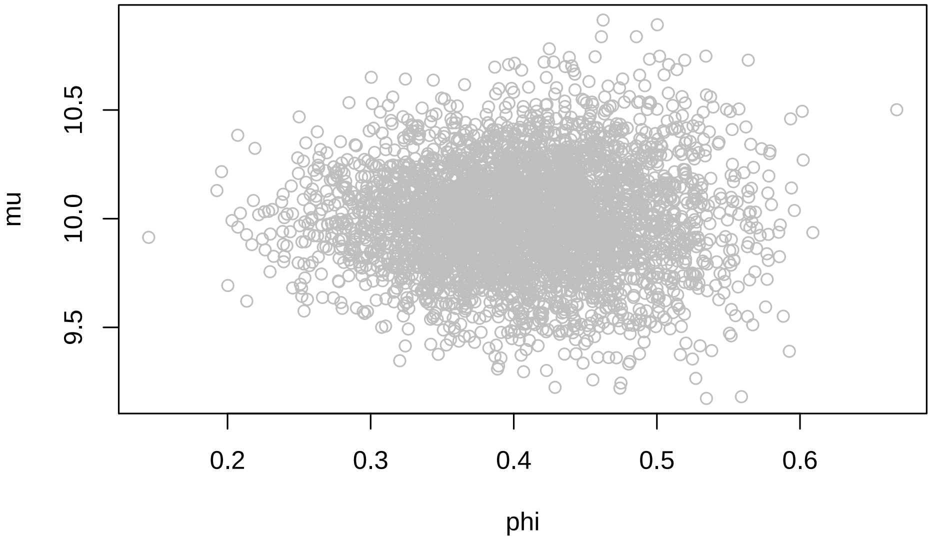
parameters	mean	CI	EffectSamples	TrueValue
mu	15.67	(-41.45,81.54)	588	12.47552

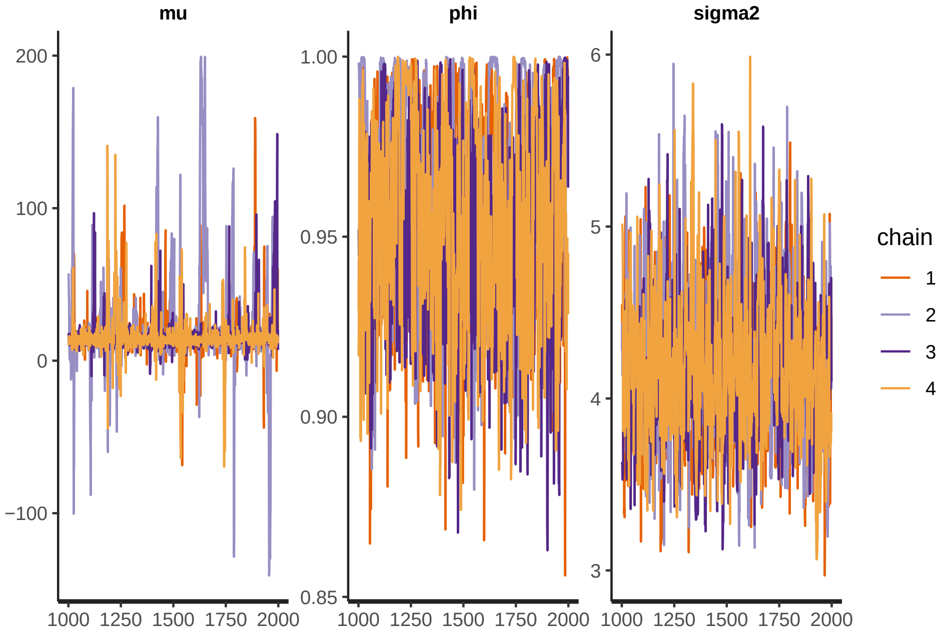
parameters	mean	CI	EffectSamples	TrueValue
phi	0.98	(0.93,1.00)	430	0.95000
sigma2	4.89	(3.99,6.07)	342	29.43775

In 2000 iterations, we use rtan to simulate the posterior parameters $\mu, \phi\ and\ \sigma^2$. Look at the table above we can see the posterior mean, 95% CI and effective samples from the simulations. Comparing the posterior mean with the true value from (a), we can see that the simulation from $x_{1:T}$ has the better estimation then from $y_{1:T}$, especially the σ^2 of y true value is 64.196, compare to the simulation which is 4.89, it is because we set the prior of $\sigma^2 \sim$ Normal(0,10). That's the reason why it lower down the accurracy.

ii. For each of the two data sets, evaluate the convergence of the samplers and plot the joint posterior of μ and ϕ. Comments?







Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#1.a
```

```
set.seed(12345)

#given values
mu=10
sigma2=2
t=200

AR_1_process = function(mu, sigma2, t, phi){
  x = c()
  x[1] = mu
  for(i in 2:t){
    eps = rnorm(1,0,sigma2)
    x[i] = mu +phi*(x[i-1]-mu)+eps
  }
  return(x)
}


# plot and compare with different phi
for(i in c(-0.9,0.9)){
  AR_1 = AR_1_process(mu,sigma2,t,phi=i)
  plot(AR_1,type="l", main=paste("AR(1)-process with phi=",i), xlab="t index", ylab="x"  )

}



#1.b
library(rstan)

AR_x = AR_1_process(mu, sigma2,t,phi = 0.3)
AR_y = AR_1_process(mu,sigma2,t,phi = 0.95)

#plot(AR_x,type="l")
#plot(AR_y,type="l")


#Stan to simulate mu, sigam2,phi
#AR(1)model
#https://mc-stan.org/docs/2_23/stan-users-guide/autoregressive-section.html

StanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=-1,upper=1> phi;
  real<lower=0> sigma2;
}
model {
  mu ~ normal(10,100);  //non-informative, larger sigma2
  phi ~ normal(0,10); //-1 1
  sigma2 ~ scaled_inv_chi_square(1,2);
  for (n in 2:N)
    y[n] ~ normal(mu + phi * (y[n-1]-mu), sqrt(sigma2));
}'
# i

## fit the AR x samples
N_x = length(AR_x)
dataX = list(N=N_x, y=AR_x)
burnin=1000
niter=2000
fit_x=stan(model_code = StanModel,data=dataX,warmup=burnin, iter=niter, chains = 4)

## fit the AR y samples
N_y = length(AR_y)
dataY = list(N=N_y,y=AR_y)
fit_y = stan(model_code = StanModel,data=dataY,warmup=burnin, iter=niter, chains = 4)

#print(fit, digits_summary=3)
# Extract posterior samples
postDraws_x = extract(fit_x)
postDraws_y = extract(fit_y)

print(fit_x)

library(knitr)
post_x=data.frame(parameters=c("mu","phi","sigma2"),mean=c(10.43,0.35,4.26),"95CI"=c("(9.98,10.89)","(0
kable(post_x,caption="Posterior of x")


print(fit_y)
post_y = data.frame(parameters=c("mu","phi","sigma2"),mean=c(15.67,0.98,4.89),"CI"=c("(-41.45,81.54)","
kable(post_y,caption="Posterior of y")

# ii
traceplot(fit_x, main="Posterior of x")

plot(mu~phi,data=fit_x, col="grey")


traceplot(fit_y, main="posterior of y")
```

“‘