732A91-Lab4-Report

Zhixuan Duan(zhidu838) Fengjuan Chen(fench417)

5/25/2020

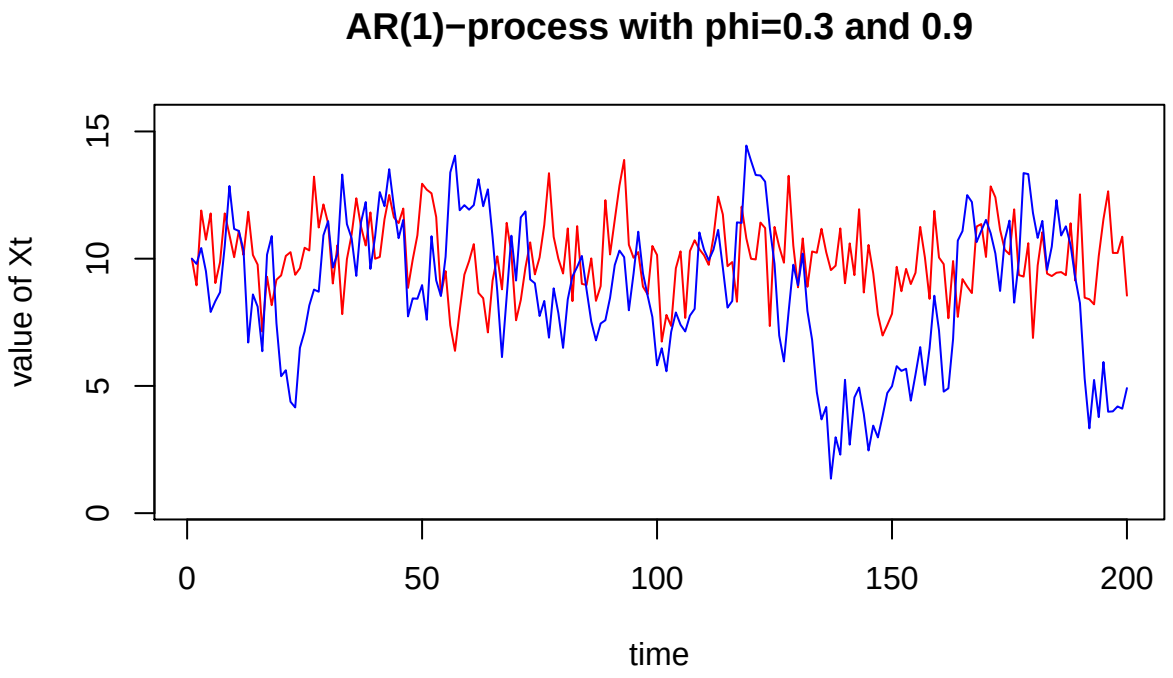
1. Time series models in Stan

(a) Function to simulate data from a given AR(1)-process

We have the AR(1)-process model $x_t = \mu + \phi \cdot (x_{t-1} - \mu) + \epsilon_t$, where $\epsilon_t \overset{iid}{\sim} N(0, \sigma^2)$.

We implement a function in R to simulate data from that AR(1)-process with the parmaters $\mu = 10$,$\sigma^2 = 2$, $T = 200$ and different values of ϕ between -1 and 1.

We plot three simulations with parameters above and $\phi = 0.3$ and 0.9 respectively. In the figure, the red line is the simulation of AR(1) with $\phi = 0.3$ while the blue line is the simulation of AR(1) with $\phi = 0.9$.



What effect does the value of phi have on x_t?

The AR(1)-process model, $x_t = \mu + \phi \cdot (x_{t-1} - \mu) + \epsilon_t$ is equivalent to the form $x_t = \alpha + \beta x_{t-1} + \epsilon_t$ if we set $\alpha = (1 - \phi) \cdot \mu$ and $\beta = \phi$.

In AR(1) model, the value of x at time t, x_t, is a linear function of the value of x at time t-1, x_{t-1}.

The parameter ϕ is the coefficient of x_{t-1}, the slope in the AR(1) model.

(b) Use Stan to implement MCMC sampless from the posterior

We first use the function from (a) to simulate two AR(1)-processes with $\phi = 0.3$ and $\phi = 0.9$ as synthetic data.

```
X=simu_AR1(mu=10,sigma2 = 2,T=200,phi = 0.3)
Y=simu_AR1(mu=10,sigma2 = 2,T=200,phi = 0.95)
```

Then we plan to use MCMC to estimate the unknown parameters of AR(1)-process, μ, ϕ, and σ^2

We use the code from the time-series models examples (AR(1)-models) in Stan user's guide manual as the template.

Because the template code implement the formula $x_t = \alpha + \beta x_{t-1} + \epsilon_t$, we set $\alpha = (1 - \phi) \cdot \mu$ and $\beta = \phi$ to obtain the same model by changing the parameters block and model block in original Stan code. The adapted code is show as follows.

```
data {
  int<lower=0> N;
  vector[N] y;
}

parameters {
  real mu;
  real phi;
  real<lower=0> sigma;
}

model {
  y[2:N] ~ normal(mu*(1-phi) + phi * y[1:(N - 1)], sigma);
}
```

Then we prepare the data and call stan() function to obtain the simulation of posterior distribution.

```
ar_data=list(N=200,y=X)
ar_data2=list(N=200,y=Y)
fit1=stan(file = 'lab4-2.stan',data = ar_data)
# reuse the model
fit2=stan(fit = fit1,data = ar_data2)
```

posterior mean and 95% credible intervals

We call print() method of stanfit object to obtain the posterior mean, 95% credible intervals and the number of effective posterior samplers for the three inferred parameters for two AR(1)-processes (phi=0.3 and 0.95).

```
print(fit1,pars = c("mu","phi","sigma"),probs = c(0.025,0.975))
print(fit2,pars = c("mu","phi","sigma"),probs = c(0.025,0.975))
```

```
## Inference for Stan model: lab4-2.
## 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% 97.5% n_eff Rhat
## mu    10.14       0 0.15 9.85 10.43  3705    1
## phi    0.29       0 0.07 0.16  0.43  3657    1
## sigma  1.45       0 0.08 1.31  1.61  3629    1
##
## Samples were drawn using NUTS(diag_e) at Tue May 26 05:21:58 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).
```

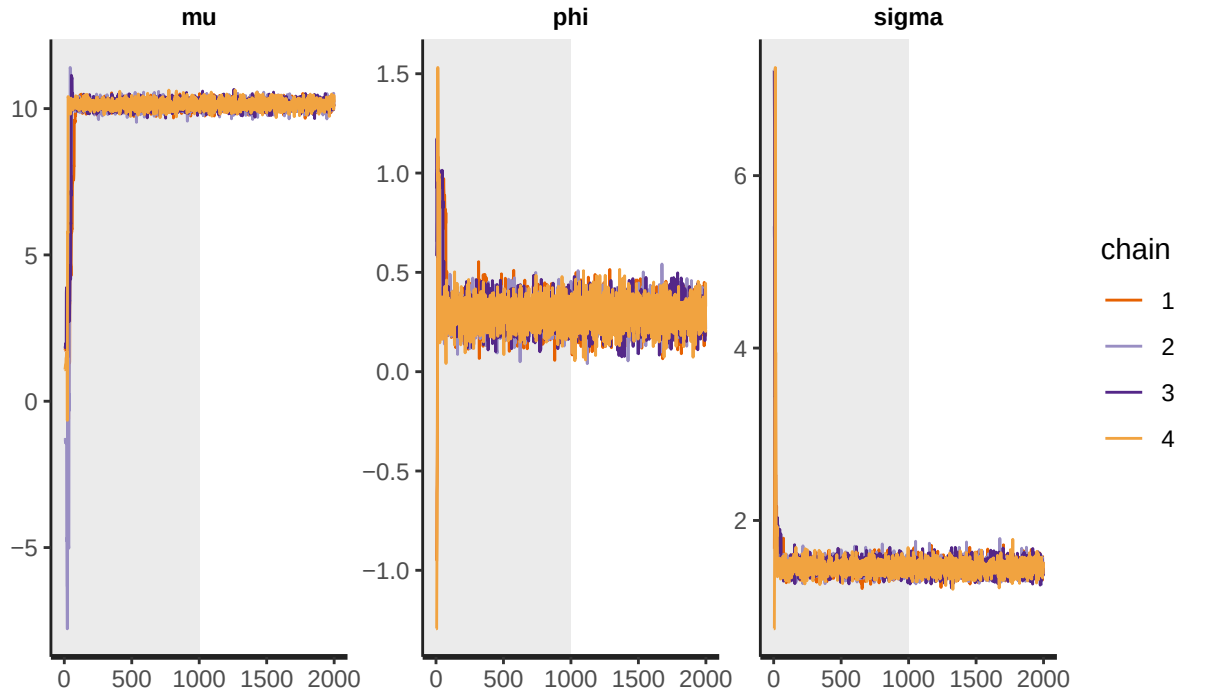
```
## Inference for Stan model: lab4-2.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean    sd   2.5% 97.5% n_eff Rhat
## mu    10.72    0.98 13.02 -19.06 39.69   175 1.01
## phi    0.95    0.00  0.03   0.89  1.00   292 1.01
## sigma  1.50    0.00  0.08   1.36  1.67   743 1.00
##
## Samples were drawn using NUTS(diag_e) at Tue May 26 05:22:02 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).
```

Because we use MCMC to simulate, the true values of these three parameters is the average of these simulations.

Evaluate the convergence of the samplers

We use traceplot() and get_sampler_params() functions to evaluate the convergence of the samplers. Then pairs() functions is used to plot the joint posterior of μ and phi.

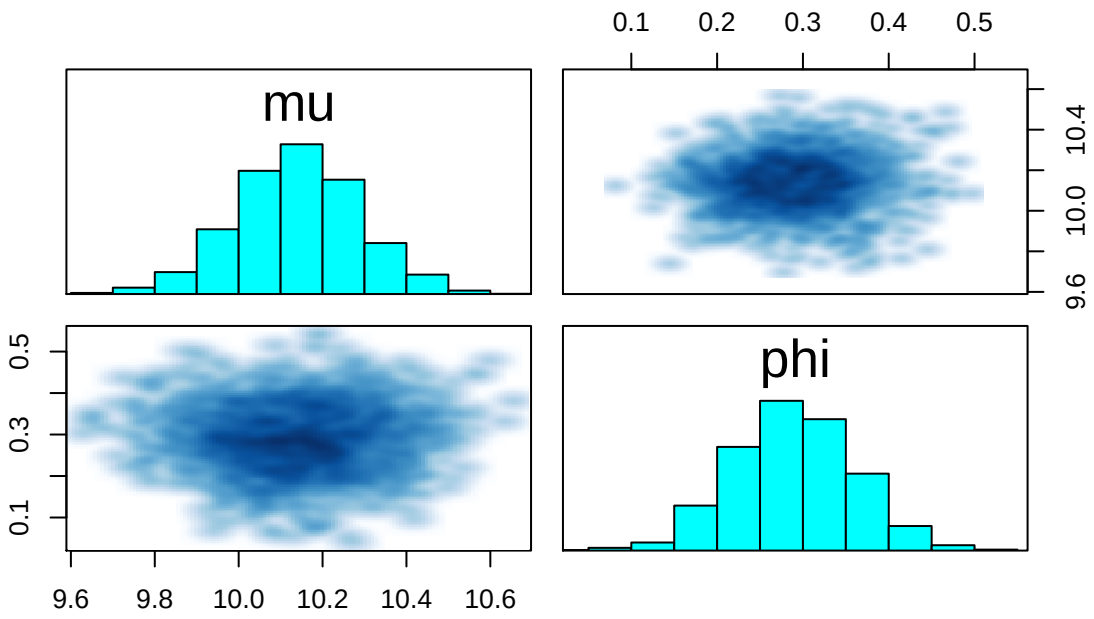
```
traceplot(fit1,inc_warmup=TRUE)
```



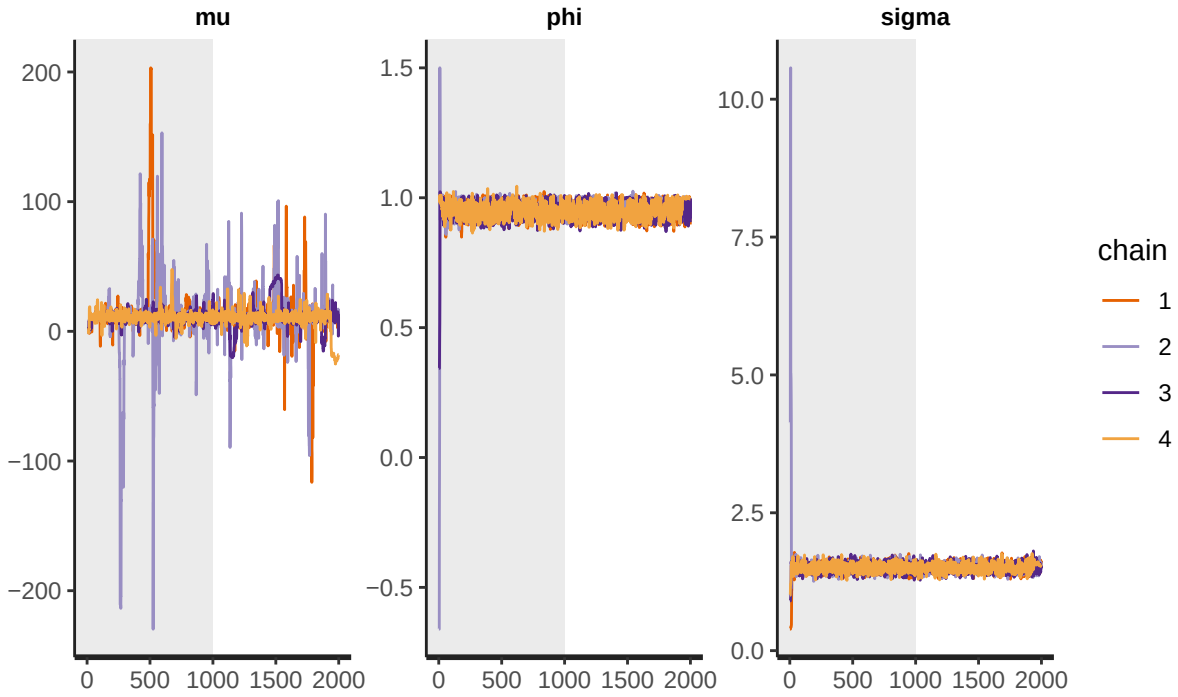

```
sampler_params1 <- get_sampler_params(fit1, inc_warmup = TRUE)
summary(do.call(rbind, sampler_params1), digits = 2)
```

```
##  accept_stat__    stepsize__        treedepth__    n_leapfrog__
##  Min.   :0.00   Min.   :9.8e-04   Min.   : 0.0   Min.   :   1.0
##  1st Qu.:0.84   1st Qu.:7.0e-01   1st Qu.: 2.0   1st Qu.:   3.0
##  Median :0.95   Median :7.5e-01   Median : 2.0   Median :   3.0
##  Mean   :0.86   Mean   :8.6e-01   Mean   : 2.3   Mean   :   6.6
##  3rd Qu.:0.99   3rd Qu.:9.1e-01   3rd Qu.: 3.0   3rd Qu.:   7.0
##  Max.   :1.00   Max.   :2.7e+01   Max.   :10.0   Max.   :1023.0
##   divergent__       energy__
##  Min.   :0.000   Min.   :    171
##  1st Qu.:0.000   1st Qu.:    173
##  Median :0.000   Median :    174
##  Mean   :0.006   Mean   :    371
##  3rd Qu.:0.000   3rd Qu.:    175
##  Max.   :1.000   Max.   :1047111
```

```
### each chain
#lapply(sampler_params1, summary, digits = 2)
pairs(fit1,pars = c("mu","phi"))
```



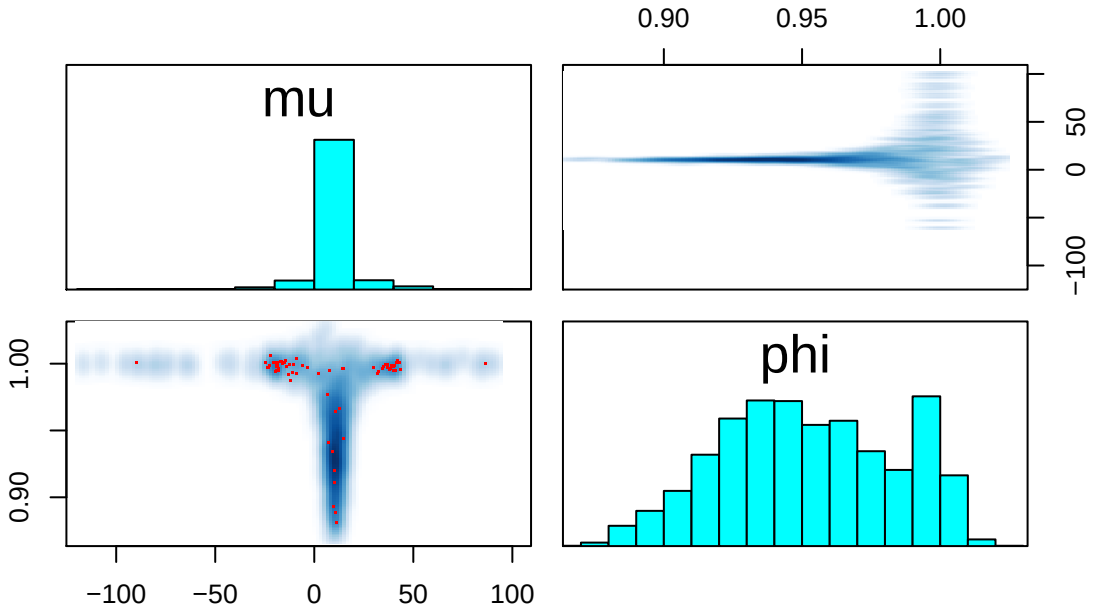

```
traceplot(fit2,inc_warmup=TRUE)
```



```
sampler_params2 <- get_sampler_params(fit2, inc_warmup = TRUE)
summary(do.call(rbind, sampler_params2), digits = 2)
```

```
##  accept_stat__    stepsize__        treedepth__    n_leapfrog__
##  Min.   :0.00   Min.   :7.6e-04   Min.   : 0.0   Min.   :   1
##  1st Qu.:0.77   1st Qu.:7.8e-02   1st Qu.: 2.0   1st Qu.:   5
##  Median :0.95   Median :3.3e-01   Median : 3.0   Median :   7
##  Mean   :0.82   Mean   :3.2e-01   Mean   : 3.2   Mean   :  23
##  3rd Qu.:0.99   3rd Qu.:3.8e-01   3rd Qu.: 4.0   3rd Qu.:  23
##  Max.   :1.00   Max.   :1.4e+01   Max.   :10.0   Max.   :1023
##   divergent__       energy__
##  Min.   :0.000   Min.   :  177
##  1st Qu.:0.000   1st Qu.:  179
##  Median :0.000   Median :  180
##  Mean   :0.034   Mean   :  206
##  3rd Qu.:0.000   3rd Qu.:  182
##  Max.   :1.000   Max.   :57025
```

```
### each chain
#lapply(sampler_params2, summary, digits = 2)
pairs(fit2,pars = c("mu","phi"))
```



From the plots and information about sampler, we can see the simulation is converged well for the first AR(1)-process data set while it is not converged well for the second data set if we set the iter=2000 and warm_up=1000.

(c) Poisson AR(1) in Stan with big value for Sigma

The number of infections c_t at each time point follows an independent Poisson distribution when conditioned on a latent AR(1)-process x_t, that is

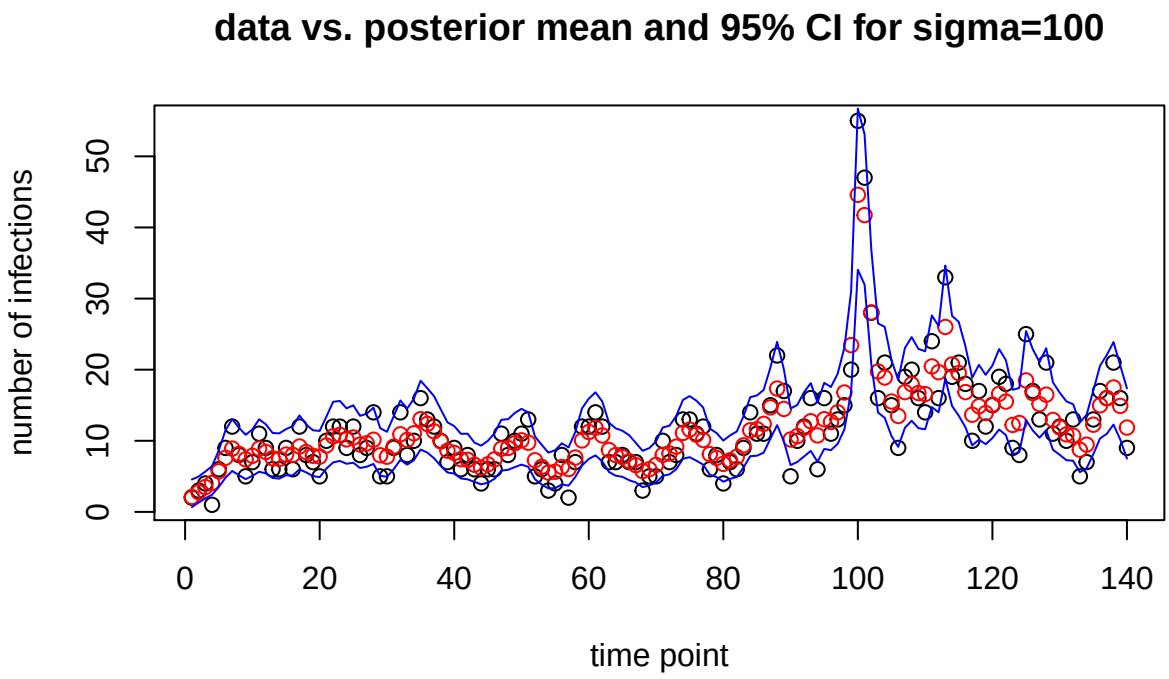
$c_t|x_t \sim Poisson(exp(x_t))$, where x_t is an AR(1)-process as in (a).

```
# The stan model code
data {
  int<lower=0> T;
  int y[T];
}

parameters {
  real alpha;           // intercept
  real<lower=0> rho;    // autoregression parameter
  real<lower=0> sigma;  // noise scale on latent state evolution
  vector[T] u;          // latent state ("error term")
}

model {
  // priors
  alpha ~ normal(0, 5);
  rho ~ lognormal(0, 2);
  sigma ~ lognormal(0, 100);

  // latent state
  u[1] ~ normal(0, 5);
  for (t in 2:T)
    u[t] ~ normal(rho * u[t - 1], sigma);

  // likelihood
  for (t in 1:T)
    y[t] ~ poisson_log(alpha + u[t]);
}
```

After implementing a MCMC by Stan, we obtain the posterior of all the parameters of AR(1)-process. Then we use $\theta_t = exp(x_t)$ to calculate the posterior simulation of θ. The $\theta_t = exp(x_t)$ can be obtained by adding 4000 simulations of alpha to 4000 simulations from u[1] to u[140] where $u[t] \sim normal(rho * u[t-1] + sigma)$.

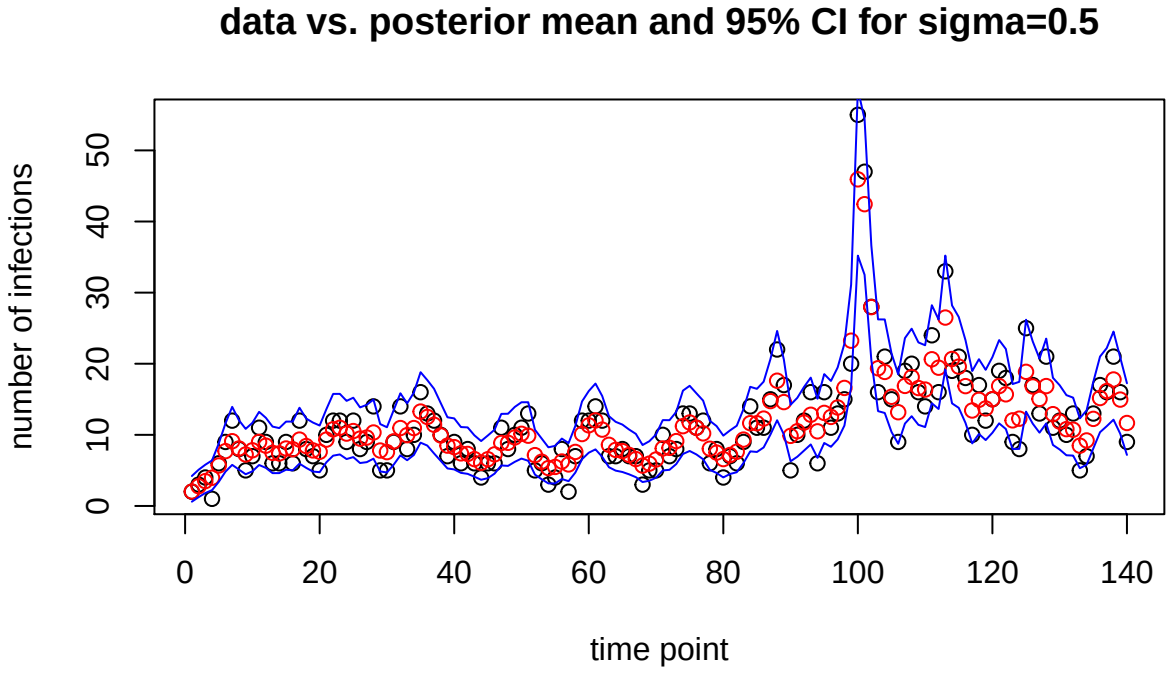
By using these 4000 simulations of posterior $\theta_t[1:140]$, we can calculatet the posterior mean and 95% equal tail credible interval for 140 time points.

In the plot, the black points are original data and the red points are the posterior mean of θ_t. The blue lines are lower bound and upper bound of 95% equal tail credible interval.



(d) Poisson AR(1) in Stan with small value for Sigma

We change the prior for σ from 100 to 0.5 and the posterior for θ_t has not changed.



Appendix

```
# 1. Time series models in Stan
## (a) function to simulate data from a given AR(1)-process
simu_AR1 <- function(mu=10,sigma2=2,T=200,phi=0.5){
  X=rep(0,T)
  X[1]=mu
  for (i in 2:T) {
    X[i]=mu+phi*(X[i-1]-mu)+rnorm(1,mean=0,sd=sqrt(sigma2))
  }
  return(X)
}

X2=simu_AR1(mu=10,sigma2 = 2,T=200,phi = 0.3)
X4=simu_AR1(mu=10,sigma2 = 2,T=200,phi = -0.9)

### show phi=0.3 and phi=0.9 in the same figure

plot(c(1:length(X2)),X2,type = "l", col="red",
     xlab = "time",ylab = "value of Xt",
     ylim = c(4,17),
     main = "AR(1)-process with phi=0.3 and 0.9")
lines(c(1:length(X4)),X4,col="blue")


## (b) Use Stan to implement MCMC sampless from the posterior
library(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)

X=simu_AR1(mu=10,sigma2 = 2,T=200,phi = 0.3)
Y=simu_AR1(mu=10,sigma2 = 2,T=200,phi = 0.95)
ar_data=list(N=200,y=X)
ar_data2=list(N=200,y=Y)
# use mu*(1-beta)+beta*y_(n-1) as AR(1) model, beta=phi
fit2=stan(file = 'lab4-2.stan',data = ar_data)
# reuse the model
fit3=stan(fit = fit2,data = ar_data2)
traceplot(fit2,inc_warmup=TRUE)
sampler_params <- get_sampler_params(fit2, inc_warmup = TRUE)
summary(do.call(rbind, sampler_params), digits = 2)
### each chain
lapply(sampler_params, summary, digits = 2)
pairs(fit2,pars = c("mu","phi"))

## (c) Poisson AR(1) in Stan with big value for Sigma
campy=read.table("campy.dat",header = TRUE)
plot(1:140,campy$c,type="l")

T <- 140
alpha <- 4.6  # log(alpha) approx. 100
sigma <- 1.25
rho <- 0.7
```

```r
y=campy$c
### big value of sigma (sigma=100)
fit_ar1 <- stan(file = "lab4-3.stan", data=c("T", "y"))

res_1=extract(fit_ar1)
pos_alpha1=res_1[[1]]
pos_u1=res_1[[4]] # 4000 rows and 140 columns,
# each row is a time series of normal(beta*x_(t-1),sigma)
# each row plus alpha is each time series point (140) x_t

## change alpha1 to matrix, copy each element to 140 columns
x_time=replicate(140,pos_alpha1)+pos_u1
pos_theta=exp(x_time)
plot(c(1:140),campy$c)
pos_mean_theta=colMeans(pos_theta)
points(c(1:140),pos_mean_theta,col="red")
low_bound=rep(0,140)
up_bound=rep(0,140)
for (i in 1:140) {
 low_bound[i]=quantile(pos_theta[,i],0.025)
 up_bound[i] =quantile(pos_theta[,i],0.975)
}
lines(c(1:140),low_bound,col="blue")
lines(c(1:140),up_bound,col="blue")

## (d) Poisson AR(1) in Stan with small value for Sigma
### small value of sigma(sigma=0.5)
fit_ar2 <- stan(file = "lab4-4.stan", data=c("T", "y"))
fit_ar3 <- stan(file = "lab4-5.stan", data=c("T", "y"))
res_2=extract(fit_ar2)
pos_alpha2=res_2[[1]]
pos_u2=res_2[[4]] # 4000 rows and 140 columns,
# each row is a time series of normal(beta*x_(t-1),sigma)
# each row plus alpha is each time series point (140) x_t

## change alpha1 to matrix, copy each element to 140 columns

x_time2=replicate(140,pos_alpha2)+pos_u2
pos_theta2=exp(x_time2)
plot(c(1:140),campy$c)
pos_mean_theta2=colMeans(pos_theta2)
points(c(1:140),pos_mean_theta2,col="red")
low_bound2=rep(0,140)
up_bound2=rep(0,140)
for (i in 1:140) {
  low_bound2[i]=quantile(pos_theta2[,i],0.025)
  up_bound2[i] =quantile(pos_theta2[,i],0.975)
}
lines(c(1:140),low_bound2,col="blue")
lines(c(1:140),up_bound2,col="blue")
```