Lab4

Andreas Stasinakis(andst745) & Mim Kemal Tekin(mimte666)

May 17, 2019

Contents

Time series models in Stan

a) Fit a AR model, with all the parameters known. Plots for different values of paramaters

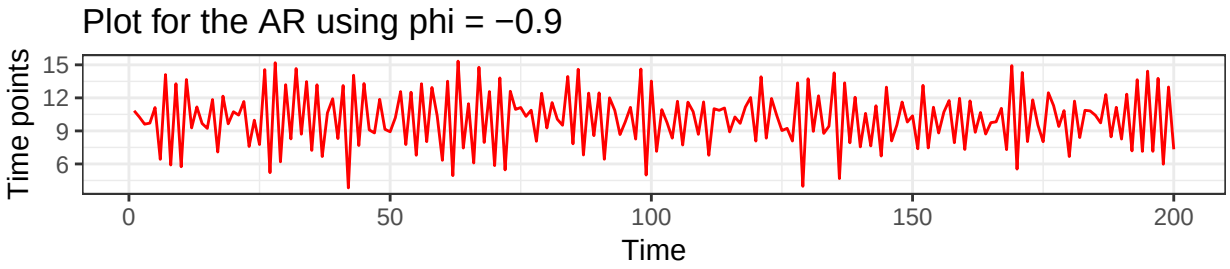
Write a function in R that simulates data from the AR(1)-process

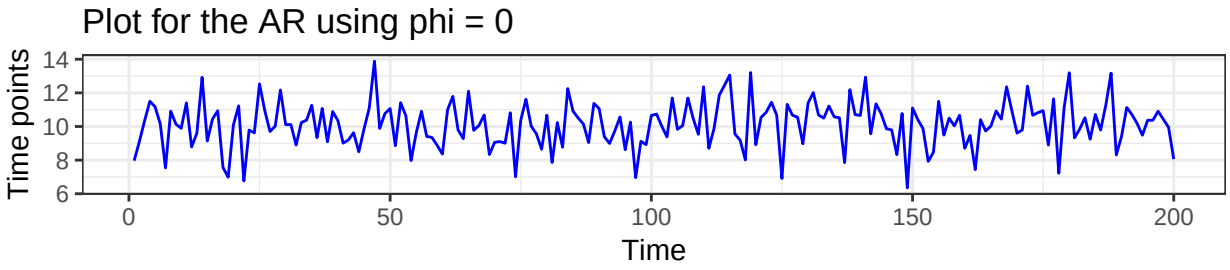
$$x_t = \mu + \phi(x_{t-1} - \mu) + \epsilon_t, \epsilon_t \overset{\text{i.i.d}}{\sim} N(0, \sigma^2),$$

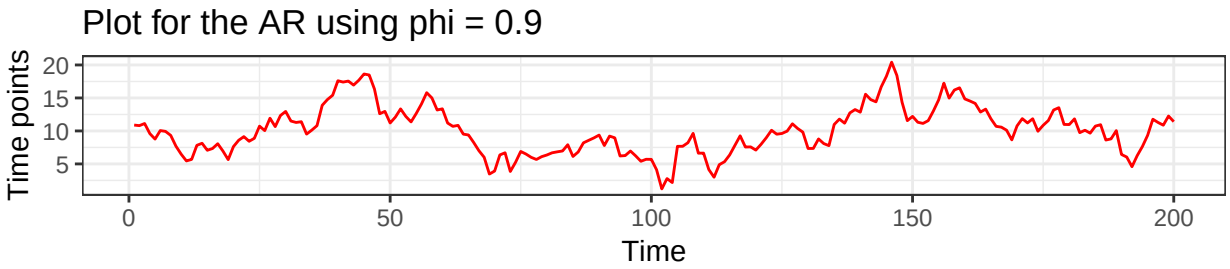
for given values of μ, ϕ and σ^2. Start the process at $x_1 = \mu$ and then simulate values for x_t for $t = 2, 3, \ldots,$ T and return the vector x_{1:T} containing all time points. Use $\mu = 10$, $\sigma^2 = 2$ and $T = 200$ and look at some different realizations (simulations) of $x_{1:T}$ for values 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 $x_{1:T}$?

```
library(ggplot2)
library(gridExtra)
set.seed(12345)

#function for AR model
AR = function(mu,phi,sigma,N){

    #store the data generated
    Xt = rep(0,N)
    #the first xt is the mu so i just add the error for the result
    Xt[1] = mu + rnorm(1,0,sd = sqrt(sigma))  #this is given from the instructions

    #for loop in order to generate the sample
    #each Xt depends on the precious one
    for (i in 2:N){

      Xt[i] = mu + phi*(Xt[i-1] - mu) + rnorm(1,0,sd = sqrt(sigma))
    }

    return(Xt)
}

#User's input
mu = 10 #it is givenn
sigma = 2 #it is given
N = 200 #also this
#We need different values of phi in order to test the role of phi in the model
phi = c(-0.9,0,0.9) #we have to try different values

Xt_samples = matrix(0,nrow = N,ncol = length(phi))

#for loop in order to test all the different phi values
for (f in 1:length(phi)) {

  #each column of the matrix is a different phi
  Xt_samples[,f] = as.vector(AR(mu = mu,phi[f],sigma,N))
}

#plot all of them in the same plt
```

```r
plot_df_mu = as.data.frame(Xt_samples)
colnames(plot_df_mu) = c("phi = -0.9","phi = 0", "phi = 0.9")
plot_df_mu$x = 1:N
#plot_df_mu = tidyr::gather(plot_df_mu, key = "all_Phi",
                           #value ="Time_points", -x)


# we plot them all together but it is really complicated to commend anything.
# plot_mu = ggplot(plot_df_mu) +
#   geom_line(aes(x=x, y = Time_points , color=all_Phi)) +
#   labs(title="AR with different phi",
#        x = "Time", y = "Time points") +
#   theme_bw()
# plot_mu

#plot for phi = -0.9
p1 = ggplot(plot_df_mu) +
  geom_line(mapping = aes(x = x,y = plot_df_mu$`phi = -0.9`), color = "red")+
  labs(title = "Plot for the AR using phi = -0.9",y = "Time points", x = "Time") +
  theme_bw()

#plot for phi = 0
p2 = ggplot(plot_df_mu) +
  geom_line(mapping = aes(x = x,y = plot_df_mu$`phi = 0`), color = "blue")+
  labs(title = "Plot for the AR using phi = 0",y = "Time points", x = "Time") +
  theme_bw()

#plot for phi = 0.9
p3 = ggplot(plot_df_mu) +
  geom_line(mapping = aes(x = x,y = plot_df_mu$`phi = 0.9`), color = "red")+
  labs(title = "Plot for the AR using phi = 0.9",y = "Time points", x = "Time") +
  theme_bw()


plots = list(p1,p2,p3)
grid.arrange(grobs = plots)
```








```
#
```

In this task we have to simulate 200 time values for a time series model,AR, with parameters $\mu = 10$, $\sigma^2 = 2$ and different values of ϕ. We run the simulation for 3 different values of $\phi = -0.9, 0,$ and 0.9. In order to commend the role of ϕ in the model, we plot the three different models. As we can see from the plots, while ϕ is close to zero, the correlation between the time points is really small. On the other hand, when ϕ is close to the limits(-1 or 1), we can observe a really significant correlation between the points. More specific, for values close to 1 we have positive correlation, while for values close to -1 negative. The truth is that we do not need the plots, in order to comment the importance of ϕ in this model. if we take a look in the formula, it is obvious that while $\phi \to 0$, the multiplication $\phi(x_{t-1} - \mu)$ will also go to zero. So the value x_t will not highly depend on the previous value.

b) Using stan with 3 unknown par and non informative priors(ESS, conv plots)

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 the values of μ, ϕ and σ^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: Look at the time-series models examples in the Stan reference manual, and note the different parameterization used here.]

i) Posterior mean, 95% CI and analysis for the Efficient sample size

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?

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

#implement the code using stan package in R
rstanSeedModel ='
  //here we have  the data and every other KNOWN parameter we will use
  data {
    int<lower=0> N; //number of observations
    vector[N] y; //data
  }

  parameters {
  //here are the UNKNOWN parameters we want to estimate
  real mu; //the mean of the model
  real phi; //We do not add the constrain here for stationarity: it gets unstable
  real<lower=0> sigma;
  }

  model {

  //PRIOR selection: Here we choose non formative priors
  mu ~ normal(0,100); // Normal with mean 0, st.dev. 100, not accurate
  //we use df = 1 which means that we are not sure about the prior
  sigma ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2
  //phi ~ normal(0,1)

  //MODEL
  //remember that this model is actual a normal distribution
  y[2:N] ~ normal(mu + phi * (y[1:(N - 1)] - mu), sqrt(sigma)); // Efficient implementation
    //for (n in 2:N)
      //y[n] ~ normal( mu + phi*(y[n-1] - mu), sigma);
}
'
#################3 USER's INPUT ########################################3
# We use the function in order to generate time points for 2 different phi
N =200 #number of observations
mu = 10 #mean of the model
sigma = 2 #variance
Xt = AR(mu = mu,phi = 0.3,sigma,N) #phi = 0.3
Yt = AR(mu = mu,phi = 0.95,sigma,N) #phi = 0.95

nBurnin <- 1000 #burn in period
nIter <- 2000 #number of iterations


#the input data SHOULD be a list includes response var and all known parameters
data1 = list(N = N,y = Xt) #for the first dataset
data2 = list(N = N,y = Yt) #for the second dataset

#input in the stan function
#model_code: all the stan code above, data: list of all data and known pars
#warmup:burnin period we want
```

```
#iter:Number of iteratios
#chains: number of chains we want

#first dataset Xt
fit1 = stan(model_code=rstanSeedModel,
            data=data1,
            warmup=nBurnin,
            iter=(nBurnin+nIter),
            chains=4,
             refresh = 0)

#Second dataset Yt
fit2 = stan(model_code=rstanSeedModel,
            data=data2,
            warmup=nBurnin,
            iter=(nBurnin+nIter),
            chains=4,
            refresh = 0)


################ Interpret the 2 results #################
############### USING BUILD IN PLOTS ###############
# Print the fitted model
#print(fit1,digits_summary=3)
# Do traceplots of the first chain
#par(mfrow = c(1,1))
#plot(postDraws$mu[1:(nIter-1000)],type="l",ylab="mu",main="Traceplot")
# Do automatic traceplots of all chains
#traceplot(fit1)
# Bivariate posterior plots
#pairs(fit1)

#For the first dataset Xt
# Extract posterior samples
postDraws1 <- extract(fit1)

# #for the second dataset Yt
# # Extract posterior samples
 postDraws2 <- extract(fit2)

###### We can take all the information we want from the summmary of the models
# for the first model Xt
summary_xt = summary(fit1)$summary
df1 = as.data.frame(summary_xt[-4,c(1,4,8,9)])
colnames(df1) = c("Posterior_mean", "lower CI", "Upper CI", "EFF Sample")
df1$real_values = c(10, 0.3,sqrt(2))
#for the second model Yt
summary_yt = summary(fit2)$summary
df2 = as.data.frame(summary_yt[-4,c(1,4,8,9)])
colnames(df2) = c("Posterior_mean", "lower CI", "Upper CI", "EFF Sample")
df2$real_values = c(10, 0.95,sqrt(2))
```

In this task we use two different datasets, generated by an AR model with different parameters of value ϕ. One dataset with $\phi = 0.35$ and one with $\phi = 0.95$. We take the three parameters(μ, ϕ, σ^2) as unknown and

we will use MCMC in order to estimate them. For each dataset, we implement Stan code which samples from the posterior of the three parameters using non informative priors. We print the results below in two different tables, one for each dataset.

For the first dataset we can see that all the three parameters are close to the real values, but there is an error occur. On the other hand though, all the true values are lying inside the credible intervals. For the efficient sample size, we have that

$$\text{ESS} = N/\text{IF}$$

, where IF is the ineffiecient factor. The ineffiecient factor, could be translated as the number of observation that they are correlated each other. For example, for the mean, the efficient sample is close to 7000 which means that only a percentage close to 15% gives correlated information. We can say the same for the other two parameters, which they have though a higher inefficient factor. The above can be easily explained, because for this dataset we used as $\phi = 0.3$. So, there is a correlation between the time points, but it is not that significant. For that reason, the algorithm is able to approximate the parameters close to the real values.

```
#print the tables
knitr::kable(x = df1,caption = "Statistics for the first dataset")
```

Table 1: Statistics for the first dataset

	Posterior_mean	lower CI	Upper CI	EFF Sample	real_values
mu	10.2275337	9.9535113	10.5106859	6035.510	10.000000
phi	0.2749939	0.1396384	0.4086501	7298.019	0.300000
sigma	2.0880126	1.7031285	2.5512513	6657.717	1.414214

On the other hand, for the the second dataset, the approximation for the mean is really inaccurate. The value for the estimated mean is really far away from the true value. Despite the fact that the true value lies in the credible interval of the mean, the witdth of the interval is really big. Therefore, we can not use it as a information for the converging. Moreover, the efficient sample size is really small, which makes the inefficient factor really high. The reason for the above output, as before, is the value of ϕ. We choose a value equal to 0.95, which means that every observation is really correlated to the previous one. For the other two parameters, despite the fact that the efficient sample size is really small, the estimation is close to the real values.

```
knitr::kable(x = df2,caption = "Statistics for the second dataset")
```

Table 2: Statistics for the second dataset

	Posterior_mean	lower CI	Upper CI	EFF Sample	real_values
mu	8.0716996	-21.3634573	48.569180	128.0901	10.000000
phi	0.9767486	0.9336553	1.006142	819.9077	0.950000
sigma	1.8282016	1.5060175	2.216403	2044.7872	1.414214

ii) convergence plots and joint posterior using STAN.

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

```
#evaluate the convergence of the chains for both datasets

#function for plotting different parameters
my_plots <- function(col, data){
  x = nrow(data)
```

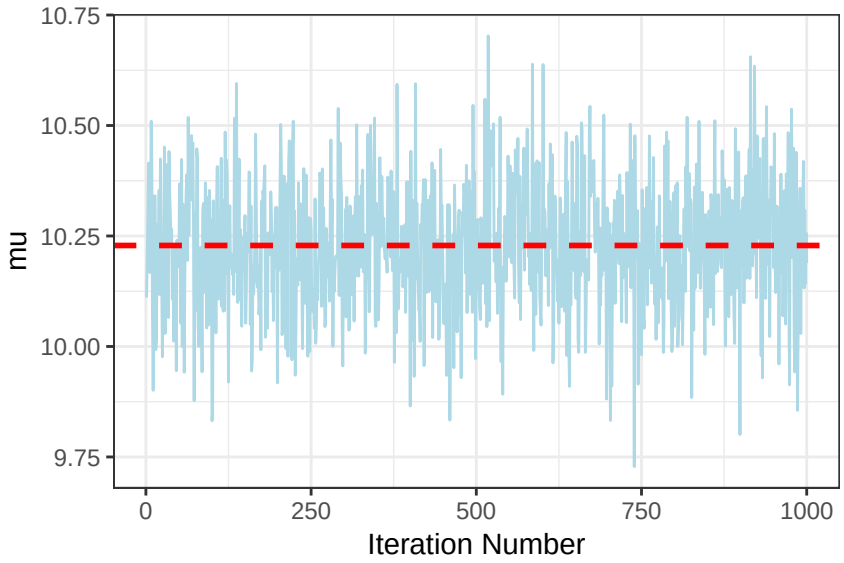
```
  ggplot(data = data, mapping = aes(x = 1:x)) +
    geom_line(mapping = aes(y = data[,col]), col = "lightblue") +
    geom_hline(yintercept = mean(data[,col]), size = 1, col = "red", lty =2) +
    labs(y = col, x = "Iteration Number") +
    theme_bw()
}



#for the first dataset
#we will evaluate only the first chain, all of them have the same behavior
df_conv1 = as.data.frame(postDraws1)[,-4] #we do not want the final column
names <- colnames(df_conv1)
conv_plot1 <- lapply(X = names, FUN = my_plots, data = df_conv1[1:1000,])


#for the second dataset
df_conv2= as.data.frame(postDraws2)[,-4] #we do not want the final column
names <- colnames(df_conv2)
conv_plot2 <- lapply(X = names, FUN = my_plots, data = df_conv2[1:1000,])

#We also have to plot the joint posterior of $\mu$ and $\phi$
#so we will use again the samples for both dataset
#for the first dataset
joint_post_Xt = ggplot(df_conv1)+
  geom_point(mapping = aes(x = df_conv1$mu, y = df_conv1$phi),
             alpha =0.3, color = "red")+
  labs(title = "Joint posterior using the Xt model",
       x = expression(mu),y  = expression(phi))

#for the second dataset Yt
joint_post_Yt = ggplot(df_conv2)+
  geom_point(mapping = aes(x = df_conv2$mu, y = df_conv2$phi),
             alpha =0.3, color = "yellow")+
  labs(title = "Joint posterior using the Yt model",
       x = expression(mu),y  = expression(phi))
```

We also plot the sample of each parameter vs the iterations in order to evaluate their convergence. It is pretty obvious that for the first dataset, all the paratemers converging quite fast in the stationery distribution. The read line is the mean of the sample and as we can observe it is cutting the plot in the middle, showing a symmetric behavior of the chain.

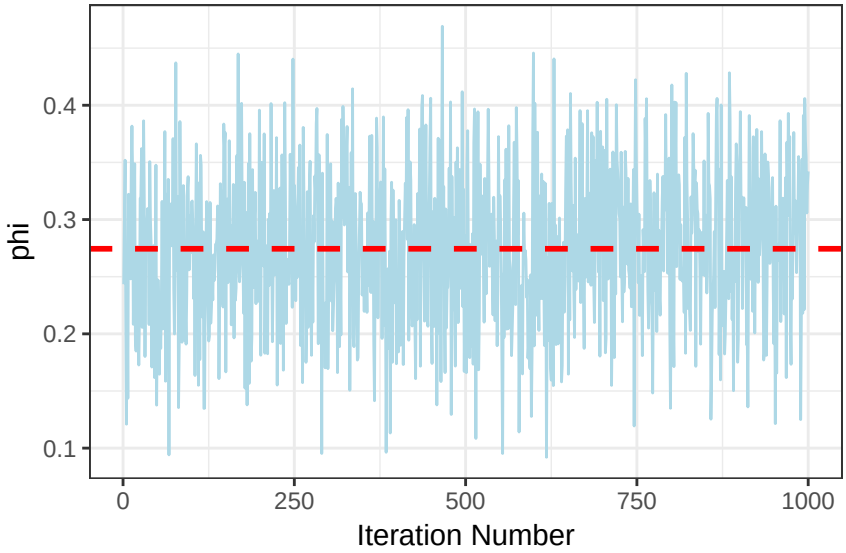
For the X_t dataset

```
conv_plot1
```

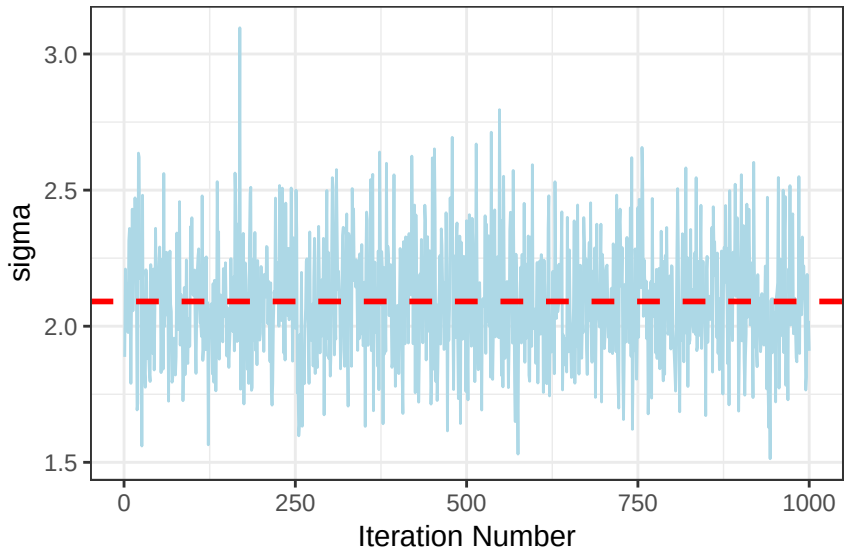
```
## [[1]]
```


10.75
10.50
10.25
10.00
9.75
mu
0
250
500
750
1000
Iteration Number

```
##
## [[2]]
```


0.4
0.3
0.2
0.1
phi
0
250
500
750
1000
Iteration Number

```
##
## [[3]]
```

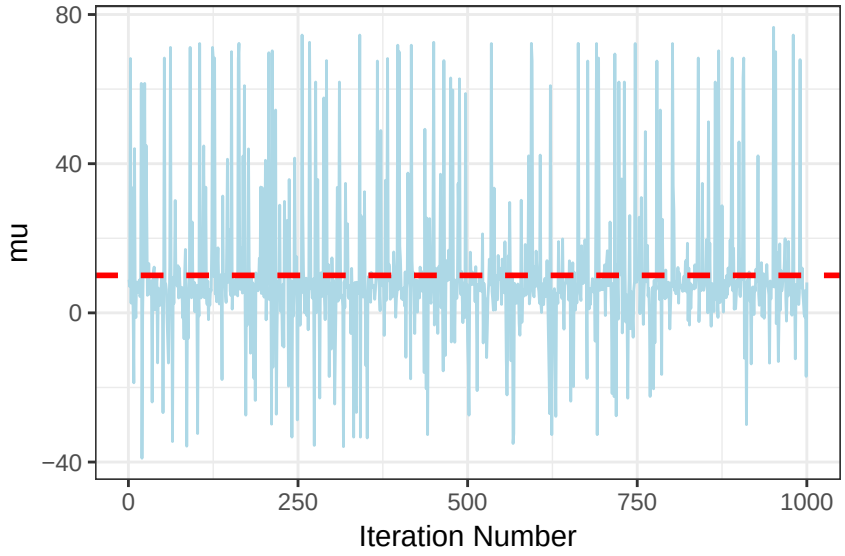


On the other hand, we can see that this is not the case for the second dataset Y_t. The parameter μ does not seem to converge at all. The variance of the points it is huge and their is not a clear pattern. That was also obvious from the tables above. The reason why this is happening is obviously the high value of ϕ which cause highy dependency bettween the points. The other two parameters, despite some outliers, seem that they have a pattern and the mean is close to the real value.

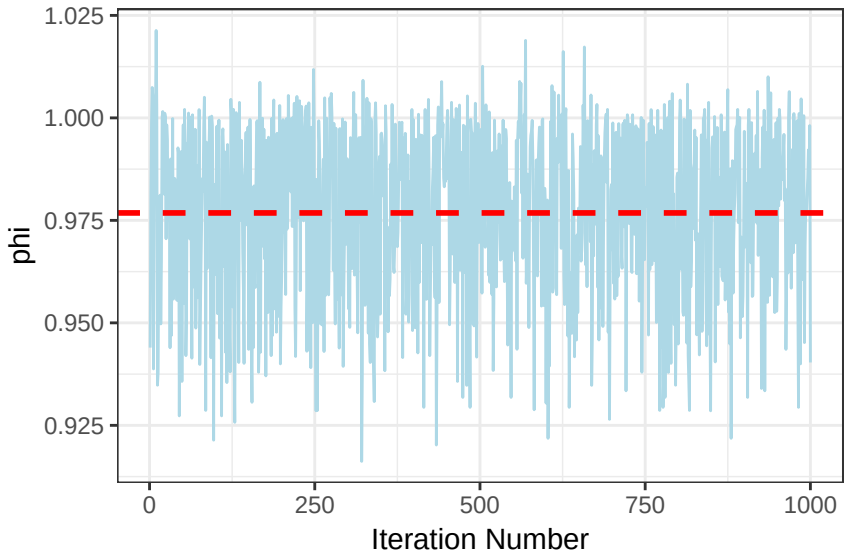
For the Y_t dataset

```
conv_plot2
```

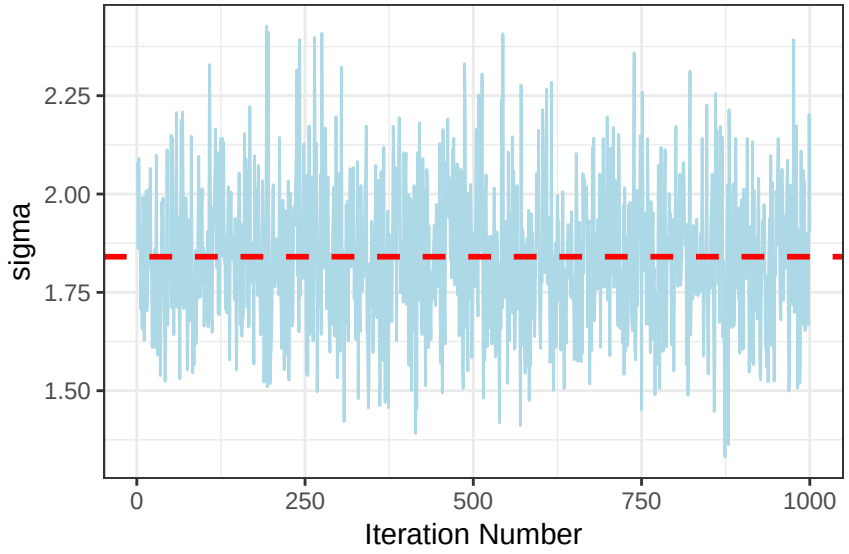
```
## [[1]]
```




```
##
## [[2]]
```



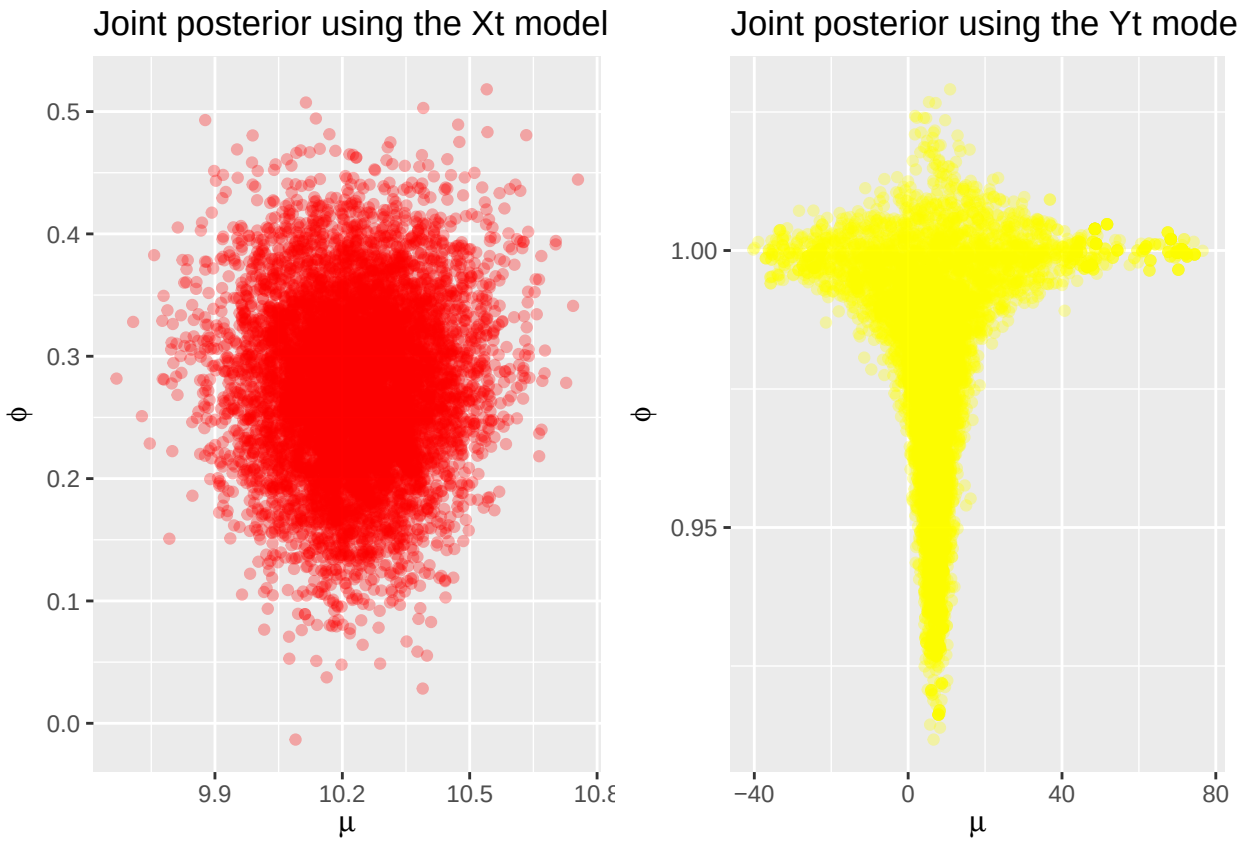

```
##
## [[3]]
```



We also plot the joint distrubution of the parameters μ, ϕ for both datasets. For the fist dataset, we can say that the plot seems like an ellipsoid. Of course there are some outliers but in general we can say that maybe the joint posterior could be a multivariate normal distribution.

On the other hand, for the joint posterior of the second dataset no conclusions for the posterior distribution can be done. The dependence is so high, especially for values really close to 1, as a result the variation of the μ is also huge.

```
plots_joint = list(joint_post_Xt,joint_post_Yt)
grid.arrange(grobs = plots_joint, ncol = 2)
```



c) Stan-code: Poisson model with rate as AR distributed

The data campy.dat contain the number of cases of campylobacter infections in the north of the province Quebec (Canada) in four week intervals from January 1990 to the end of October 2000. It has 13 observations per year and 140 observations in total. Assume that the number of infections ct at each time point follows an independent Poisson distribution when conditioned on a latent AR(1)-process xt, that is:

$$c_t/x_t \sim \text{Poisson}(\exp(x_t)),$$

where x_t is an AR(1)-process as in a). Implement and estimate the model in Stan, using suitable priors of your choice. Produce a plot that contains both the data and the posterior mean and 95% credible intervals for the latent intensity $\theta_t = exp(xt)$ over time. [Hint: Should x_t be seen as data or parameters?]

In this task we have to estimate a poisson model with parameter $\lambda = \exp(X_t)$, where Xt is given by the autoregressive model of step 1. We use Stan in order to implement the model. We have again to estimate the parameters for X_t, which are ϕ, μ, σ. We do not have any prior believe so we wil give non formative priors to stan package.

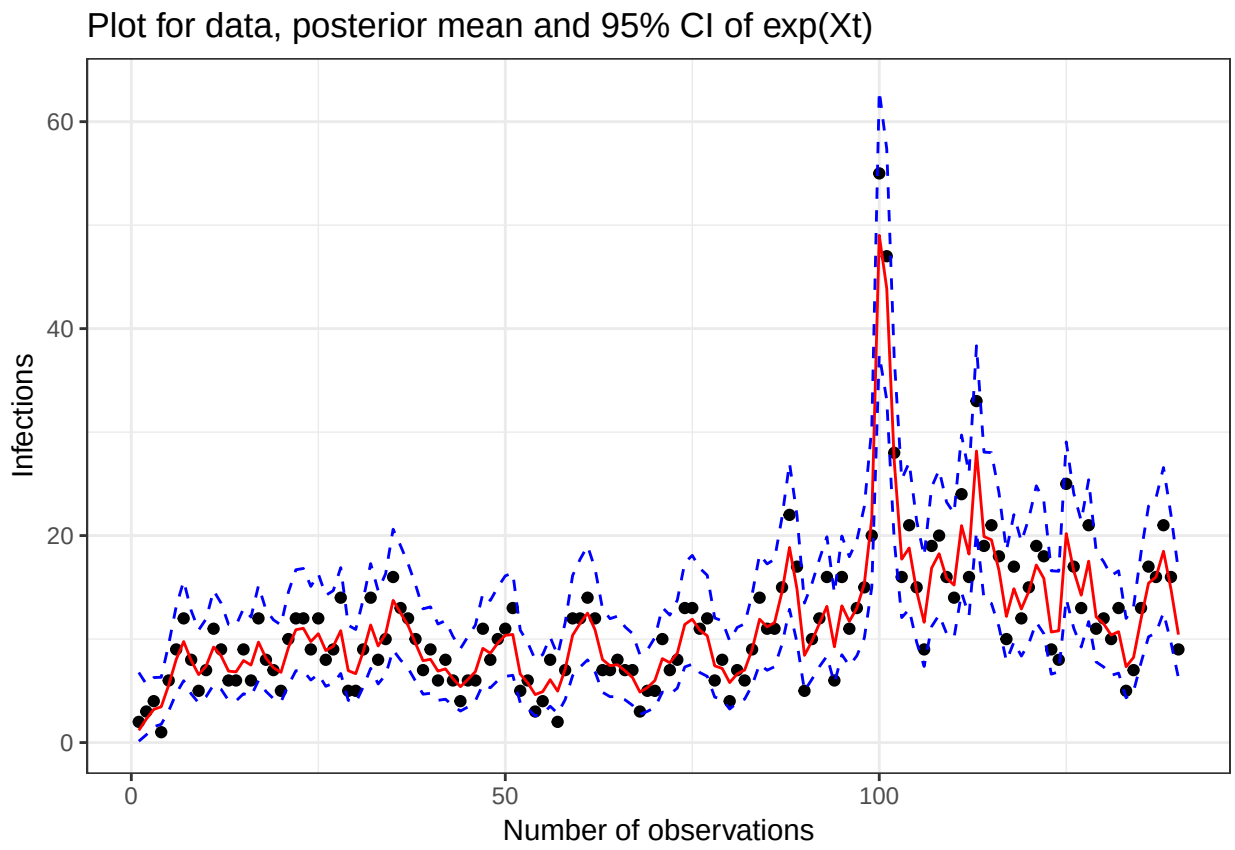
```
library(rstan)
data = read.table("../dataset/campy.dat",stringsAsFactors = F,header = T)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

rstanSeedModel ='
  //here we have  the data and every other KNOWN parameter we will use
  data {
```

```
    int<lower=0> N;
    int y[N, 1]; //this is a vector of integers length N
  }

  parameters {
    //here are the UNKNOWN parameters we want to estimate;

    real mu;
    real phi; // ????  constrains for stationarity
    real<lower=0> sigma;
    vector[N] xt;
  }

  model {

    // priors for the paratemeters
    phi ~ normal(0,1);
    mu ~ normal(0,100); // Normal with mean 0, st.dev. 100
    sigma ~ scaled_inv_chi_square(1,2); // Scaled-inv-chi2 with nu 1, sigma 2

    //here we have the model
    //more efficient way
    xt[2:N] ~ normal(mu + phi * (xt[1:(N - 1)] - mu), sqrt(sigma));
    for (n in 2:N){
      //xt[n] ~ normal(mu + phi*(xt[n-1] - mu), sigma);
      y[n] ~ poisson(exp(xt[n]));
    }
}'

#####User input#####

N = length(data$c)
nBurnin = 1000
nIter = 2000

#As before we need a list for the data
data_poisson = list(N = N, y = data)

fit = stan(model_code=rstanSeedModel,
           data=data_poisson,
           warmup=nBurnin,
           iter=(nBurnin+nIter),
           chains=4)

posteriors = extract(fit)

#We can have access to all information needed  from the summary
poisson_summary = summary(fit)$summary

df = data.frame(x = 1:nrow(data), real = data$c,
                predicted = exp(poisson_summary[4:(nrow(poisson_summary)-1),1]),
                lower = exp(poisson_summary[4:(nrow(poisson_summary)-1),4]),
                upper = exp(poisson_summary[4:(nrow(poisson_summary)-1),8]))
```

```
plot1 = ggplot(df) +
  geom_point(mapping = aes(x = x, y = real), color = "black")+
  geom_line(mapping = aes(x = x, y = predicted), color = "red")+
  geom_line(mapping = aes(x = x, y = lower), color = "blue", lty = 2)+
  geom_line(mapping = aes(x = x, y = upper), color = "blue", lty = 2)+
  labs(title = "Plot for data, posterior mean and 95% CI of exp(Xt)",
       x= "Number of observations", y = "Infections")+
  theme_bw()
```

We plot the real data(black points), the posterior sample mean(red) and the 95% credible interval(blue). As we can see from the plot, the model is able to capture the trend of the data in a really decent level. This is not good because we may have overfiting. The problem is that it follows the data too much, as a result, it also capture the outliers. Therefore, if we use this model for future predictions, we will probably not have accurate results.

```
plot1
```



d) Same as before but with better prior knowledge

Now, assume that we have a prior belief that the true underlying intensity θ_t varies more smoothly than the data suggests. Change the prior for σ^2 so that it becomes informative about that the AR(1)-process increments "ϵ_t should be small. Re-estimate the model using Stan with the new prior and produce the same plot as in c). Has the posterior for θ_t changed?

We now, have an idea about the data, so we can use a more informative prior for the model. We change the prior of the variance. We want a smoother model, for that reason we use the inverse chi square distribution

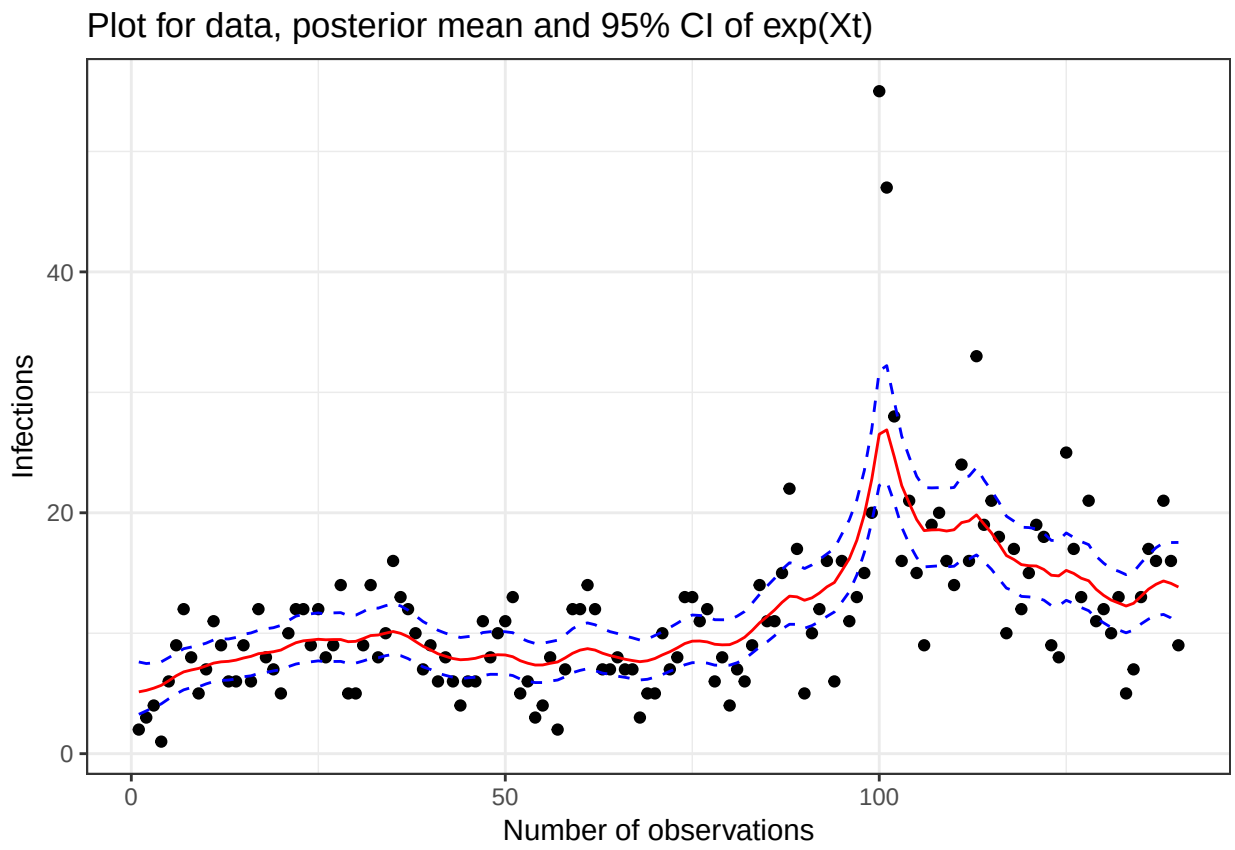
as before but with different parameters. More specific,

$$\sigma^2 \sim \text{Inv- X}(100, 0.6)$$

The first parameters, which is 100, is the degrees of freedom. Setting the degrees equal to zero means that we are really confident about our prior knownledge. The second parameter is the variance and giving a low value will give us less fluctuation in our model.

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

rstanSeedModel2 ='
  //here we have  the data and every other KNOWN parameter we will use
  data {
    int<lower=0> N;
    int y[N, 1];
  }

  parameters {
    //here are the UNKNOWN parameters we want to estimate;

    real mu;
    real phi;
    real<lower=0> sigma;
    vector[N] xt;
  }

  model {

    // priors for the paratemeters
    phi ~ normal(0,1);
    mu ~ normal(0,100);
    //Now we have a knownledge about the prior, that it is really small
    // so we use inv chi with par: df = high value(how sure we are)
    // and variance really small
    sigma ~ scaled_inv_chi_square(100,0.06);

    //here we have the model
    xt[2:N] ~ normal(mu + phi * (xt[1:(N - 1)] - mu), sqrt(sigma));
    for (n in 2:N){
      //xt[n] ~ normal(mu + phi*(xt[n-1] - mu), sigma);
      y[n] ~ poisson(exp(xt[n]));
    }
}'

#####User input#####


#As before we need a list for the data
data_poisson = list(N = nrow(data), y = data)

fit_prior = stan(model_code=rstanSeedModel2,
           data=data_poisson,
           warmup=nBurnin,
```

```
            iter=(nBurnin+nIter),
            chains=4)



#We can have access to all information needed  from the summary
poisson_prior_sum = summary(fit_prior)$summary

df = data.frame(x = 1:nrow(data), real = data$c,
             predicted = exp(poisson_prior_sum[4:(nrow(poisson_prior_sum)-1),1]),
             lower = exp(poisson_prior_sum[4:(nrow(poisson_prior_sum)-1),4]),
             upper = exp(poisson_prior_sum[4:(nrow(poisson_prior_sum)-1),8]))

plot2 = ggplot(df) +
  geom_point(mapping = aes(x = x, y = real), color = "black")+
  geom_line(mapping = aes(x = x, y = predicted), color = "red")+
  geom_line(mapping = aes(x = x, y = lower), color = "blue", lty = 2)+
  geom_line(mapping = aes(x = x, y = upper), color = "blue", lty = 2)+
  labs(title = "Plot for data, posterior mean and 95% CI of exp(Xt)",
       x= "Number of observations", y = "Infections")+
  theme_bw()

plot2
```



As we can see from the plot, using a more informative prior, we have a really smooth curve. The model did not capture all the points and the outliers as the model before, but this model will make more accurate predictions. So probably the training error will be higher in this case, but the generalized error will be smaller

which is want we want.