Bayesian Learning - Lab 04

Lakshidaa Saigiridharan (laksa656) and Maximilian Pfundstein (maxpf364)

2019-05-23

Contents

1 Time series models in Stan

Exercise:

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

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

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 . . . , 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 $x_{1:T}$?

(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 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) 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?

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

(c) 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 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). 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 $\theta_t = exp(x_t)$ over time. [Hint: Should x_t be seen as data or parameters?]

(d) 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?

1.1 Samples from an AR(1)-Process

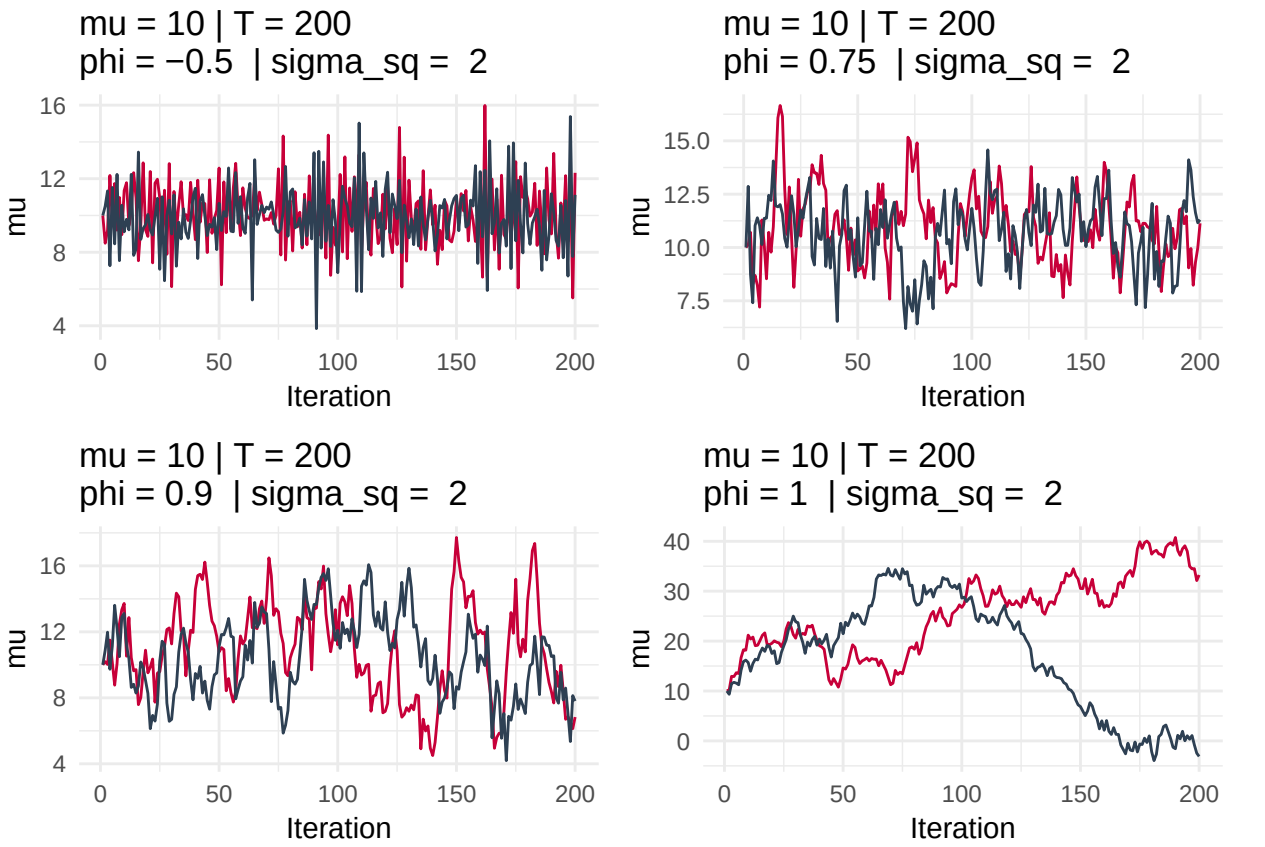
First we define the function for simulating from the AR(1)-process.

```
################################################################################
# Exercise 1.a)
################################################################################

ar_process = function(mu, tau, phi, sigma_sq) {

  # storing the values
  X = rep(NaN, tau)

  X[1] = mu

  for (i in 1:(tau - 1)) {
    X[i+1] = mu + phi *(X[i] - mu) + rnorm(n = 1, mean = 0, sd = sqrt(sigma_sq))
  }

  return(X)
}
```

The following is a helper function to take to samples from the same parameters and to return a plot of those.

```
simulate_ar_process = function(mu, tau, phi, sigma_sq) {

  res_1 = ar_process(mu, tau, phi, sigma_sq)
  res_2 = ar_process(mu, tau, phi, sigma_sq)

  df = data.frame(x = 1:tau,
                  y1 = res_1,
                  y2 = res_2)

  p = ggplot(df) +
    geom_line(aes(x = x, y = y1), color = "#C70039") +
    geom_line(aes(x = x, y = y2), color = "#2E4053") +
    labs(title = paste("mu =", mu, "| T =", tau, "\nphi =",
                       phi, " | sigma_sq = ", sigma_sq) , y = "mu",
    x = "Iteration", color = "Legend") +
    theme_minimal()

  return(p)
}

p1 = simulate_ar_process(mu = 10, tau = 200, phi = -0.5, sigma_sq = 2)
p2 = simulate_ar_process(mu = 10, tau = 200, phi = 0.75, sigma_sq = 2)
p3 = simulate_ar_process(mu = 10, tau = 200, phi = 0.9, sigma_sq = 2)
p4 = simulate_ar_process(mu = 10, tau = 200, phi = 1, sigma_sq = 2)
```

```
grid.arrange(p1, p2, p3, p4, nrow = 2)
```



The plots shows two simulations from each parameter set. If $\phi = 1$, we basically simulate a random walk, if $|\phi| < 1$ it's wide-sense stationary and if $|\phi| > 1$ the process is not stationary.

1.2 Using Stan for Finding the Posterior

We simulate from the AR(1)-process with the given parameters. And create the given Stan models.

```
################################################################################
# Exercise 1.b)
################################################################################

X = ar_process(mu = 10, tau = 200, phi = 0.3, sigma_sq = 2)
Y = ar_process(mu = 10, tau = 200, phi = 0.95, sigma_sq = 2)

stanModel = '
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
}
model {
```

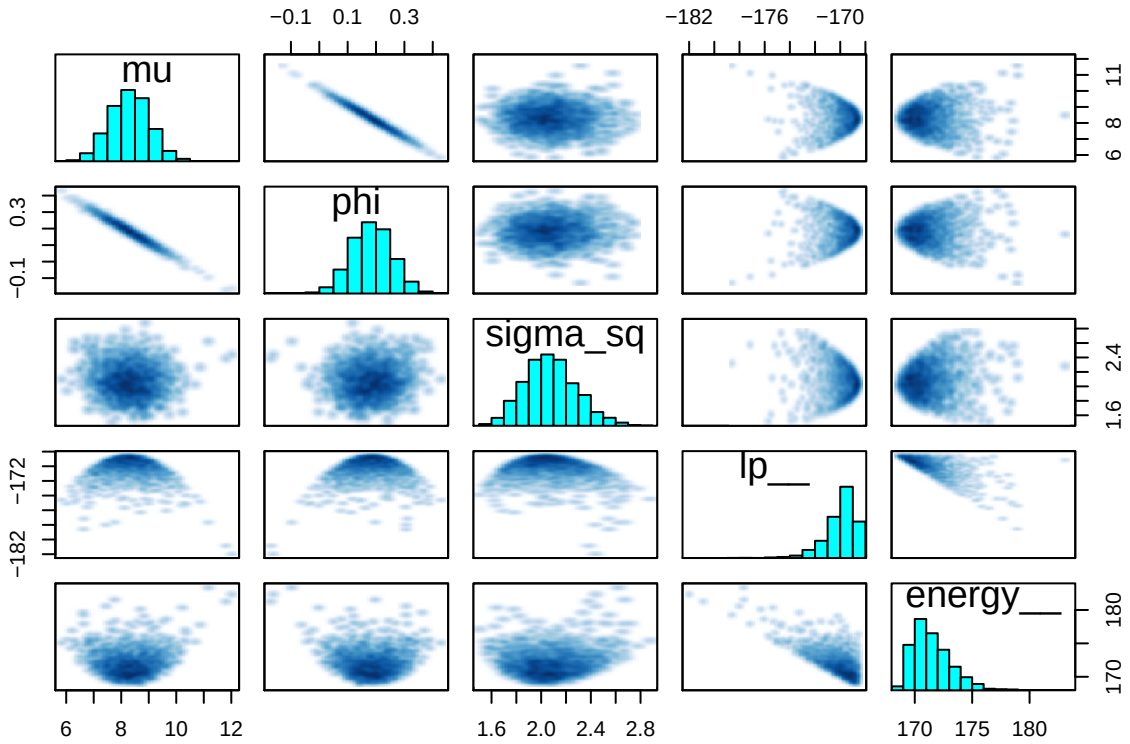
```
  y[2:N] ~ normal(mu + phi * y[1:(N - 1)], sqrt(sigma_sq));
}
'

stanModelX = stan(model_code = stanModel,
                  model_name = "AR_X",
                  data = list(N = length(X), y = X),
                  warmup = 1000,
                  iter = 2000)

stanModelY = stan(model_code = stanModel,
                  model_name = "AR_Y",
                  data = list(N = length(Y), y = Y),
                  warmup = 1000,
                  iter = 2000)

posteriorX = extract(stanModelX)
posterior_paramsX = As.mcmc.list(stanModelX)

posteriorY = extract(stanModelY)
posterior_paramsY = As.mcmc.list(stanModelY)
```

This is the summary from the first draws. It includes the 95 percent credible interval for all parameters.

```
stanModelX
```

```
## Inference for Stan model: AR_X.
## 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%
## mu          8.31    0.02 0.73    6.89    7.81    8.30    8.81    9.71
## phi         0.18    0.00 0.07    0.05    0.14    0.18    0.23    0.32
## sigma_sq    2.07    0.00 0.21    1.70    1.93    2.06    2.21    2.52
## lp__     -170.00    0.04 1.25 -173.04 -170.53 -169.71 -169.11 -168.56
##          n_eff Rhat
## mu        1222    1
## phi       1229    1
## sigma_sq  1773    1
## lp__      1139    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 17:48:59 2019.
## 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).
```

```
#summary(stanModelX,
#        pars = c("mu", "phi", "sigma_sq"), probs = c(0.025, 0.975))$summary

#plot(posterior_paramsX)
pairs(stanModelX)
```

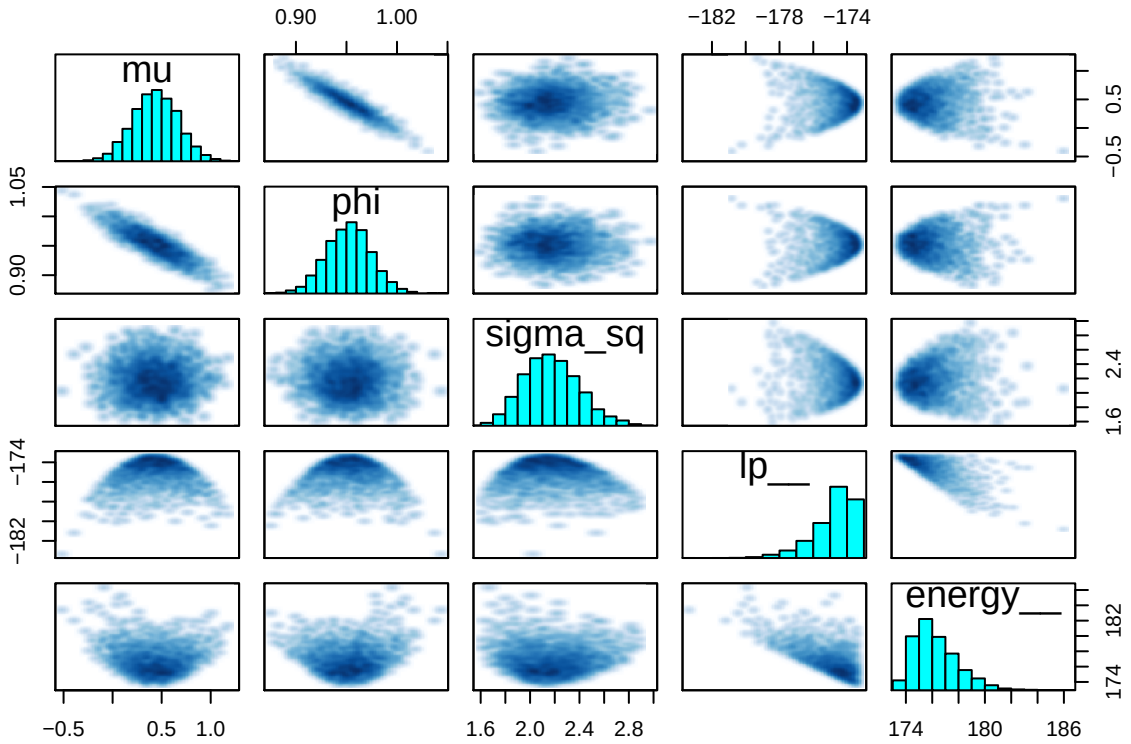


And these is the summary from the second draws, also including the 95 percent credible interval for all parameters.

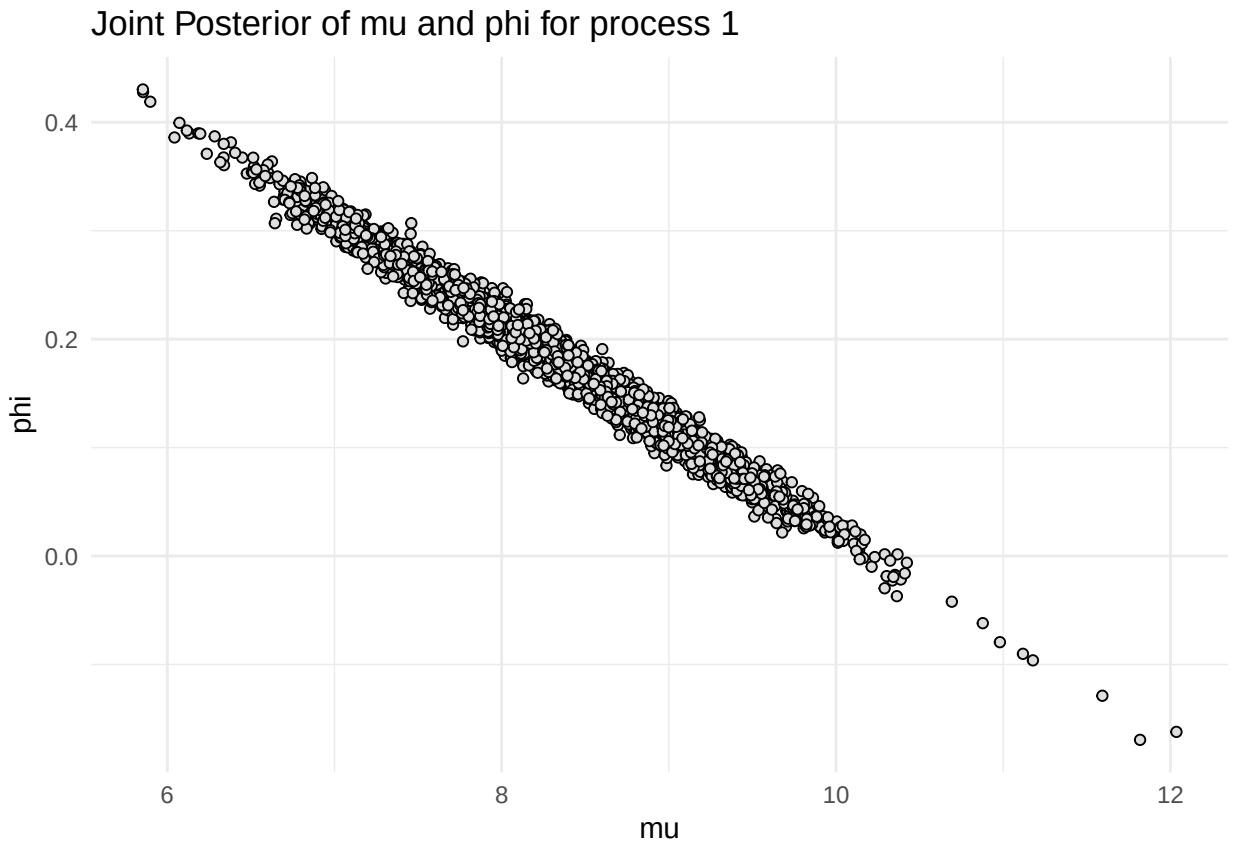
```
stanModelY
```

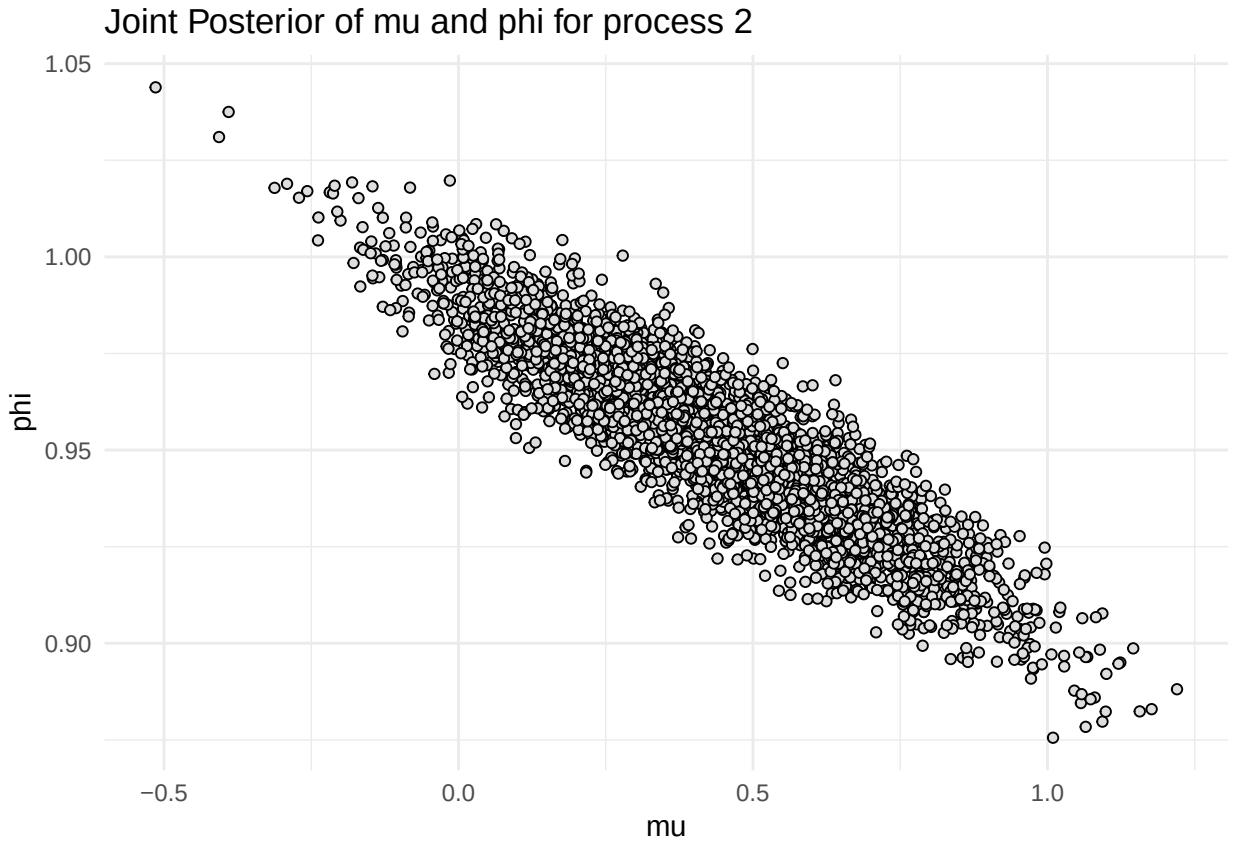
```
## Inference for Stan model: AR_X.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## mu          0.44    0.01 0.23   -0.01    0.28    0.44    0.60    0.88
## phi         0.95    0.00 0.02    0.91    0.94    0.95    0.97    1.00
## sigma_sq    2.18    0.00 0.22    1.78    2.01    2.16    2.32    2.65
## lp__     -174.78    0.03 1.20 -177.91 -175.35 -174.45 -173.89 -173.37
##          n_eff Rhat
## mu        1445    1
## phi       1455    1
## sigma_sq  2262    1
## lp__      1524    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 17:49:00 2019.
## 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).
```

```
#summary(stanModelY,
#          pars = c("mu", "phi", "sigma_sq"), probs = c(0.025, 0.975))$summary

#plot(posterior_paramsY)
pairs(stanModelY)
```



In the following plot the joint posterior between μ and ϕ can be seen. The variance for the second joint posterior is higher.





1.3 Campylobacter Infection: Modeling with Stan

First we import the data.

```
################################################################################
# Exercise 1.c)
################################################################################

campy_data = read.table("data/campy.dat", header=TRUE)
```

We define the following Stan model. Note that we basically have the same paramters as before. One new parameter is x, which represents the AR(1) process and it dependends on the previous parameters. We define the first x as a normal around μ with the prior for σ as the variance. It will therefore heavily depend on the given data but as that's the best we know about the underlying process, we will go for it. For σ^2 we chose a chi squared prior as the variance has to be positve and a normal would not suit this. As we assume to not know that much about the prior we went for 1 degree of freedom and $\nu = 10$.

```
stanModel = '
data {
  int<lower=0> N;
  int<lower=0> y[N];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
  vector[N] x;
```

```
}
model {

  // Priors
  mu ~ normal(5, 1);
  phi ~ normal(0, 10);
  sigma_sq ~ scaled_inv_chi_square(1, 10);

  // Model
  x[1] ~ normal(mu, sigma_sq);
  x[2:N] ~ normal(mu + phi * x[1:(N-1)], sqrt(sigma_sq));
  y[1:N] ~ poisson(exp(x[1:N]));
}
'

stanModelPoisson = stan(model_code = stanModel,
                  model_name = "AR_Poisson",
                  data = list(N = length(campy_data$c), y = campy_data$c),
                  warmup = 1000,
                  iter = 2000)

posteriorPoisson = extract(stanModelPoisson)
```

```
stanModelPoisson
```

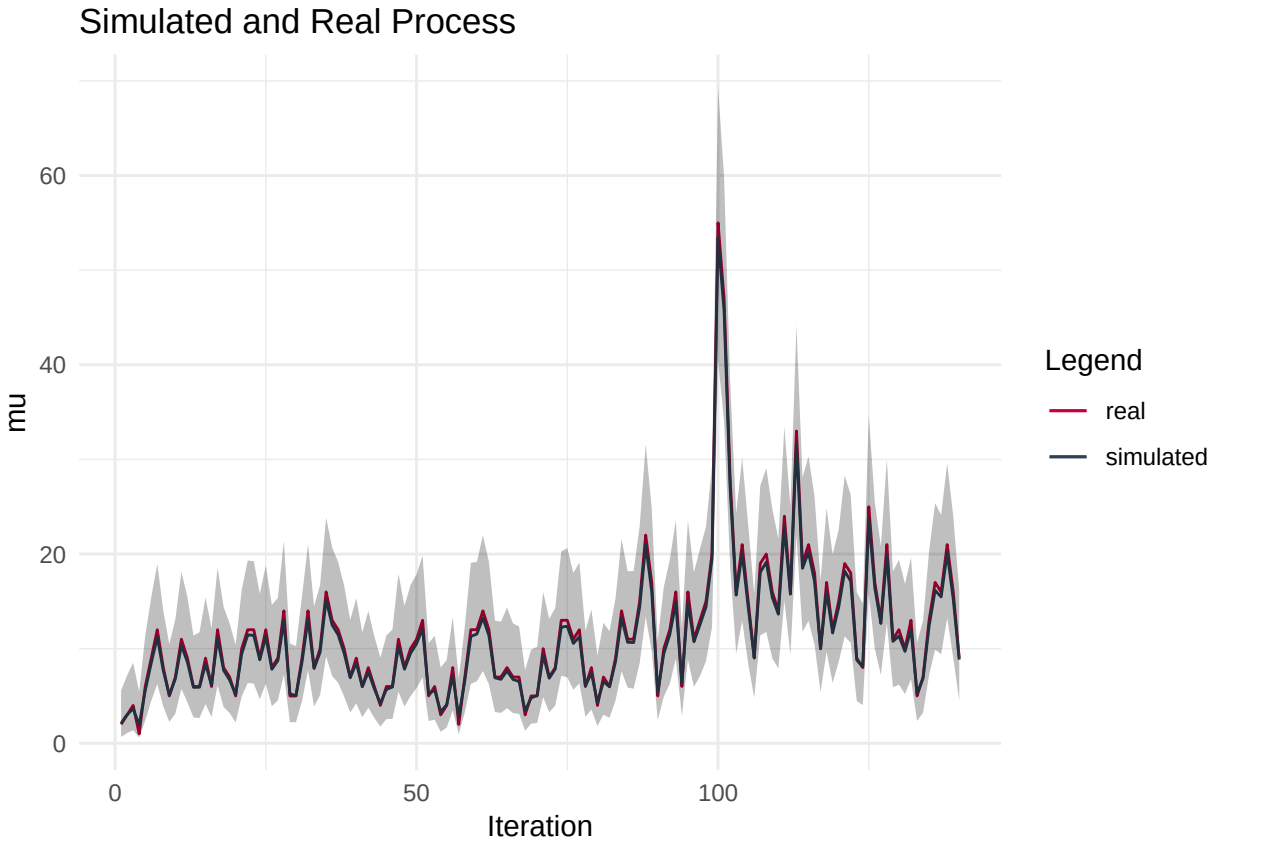
```
## Inference for Stan model: AR_Poisson.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean   sd    2.5%     25%     50%     75%   97.5%
## mu          1.42    0.00 0.33    0.81    1.20    1.41    1.63    2.10
## phi         0.39    0.00 0.14    0.10    0.29    0.39    0.48    0.65
## sigma_sq    1.02    0.00 0.13    0.79    0.93    1.01    1.10    1.30
## x[1]        0.76    0.01 0.55   -0.39    0.40    0.79    1.16    1.72
## x[2]        1.11    0.01 0.49    0.07    0.78    1.13    1.45    1.97
##          n_eff Rhat
## mu        4784    1
## phi       4634    1
## sigma_sq  5995    1
## x[1]      6210    1
## x[2]      6318    1
##  [ reached getOption("max.print") -- omitted 139 rows ]
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 17:50:20 2019.
## 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).
```

We will plot the mean of our samples. As they respresent x_t which is equal to λ which is in turn the mean of the Possion distribution. We can therefore directly take it as our mean and plot it. We use the `quantile()` function for obtaining the credible interval.

```
ci = apply(exp(posteriorPoisson$x), 2, quantile, probs = c(0.025, 0.975))

df = data.frame(x = 1:length(campy_data$c),
```

```
                real = campy_data$c,
                simulated = exp(colMeans(posteriorPoisson$x)),
                lower_ci = ci[1,],
                upper_ci = ci[2,])
```

The plot looks as follows. The darker area indicates the 95% credible interval.



1.4 Using an Informative Prior for Generalizability

We choose a chi squared with 300 degress of freedom and a very small ν to make the prior for σ^2 informative. We observe that the prediction curve is more flattened out and therefore generalizes better if we'd use it for making future predictions.

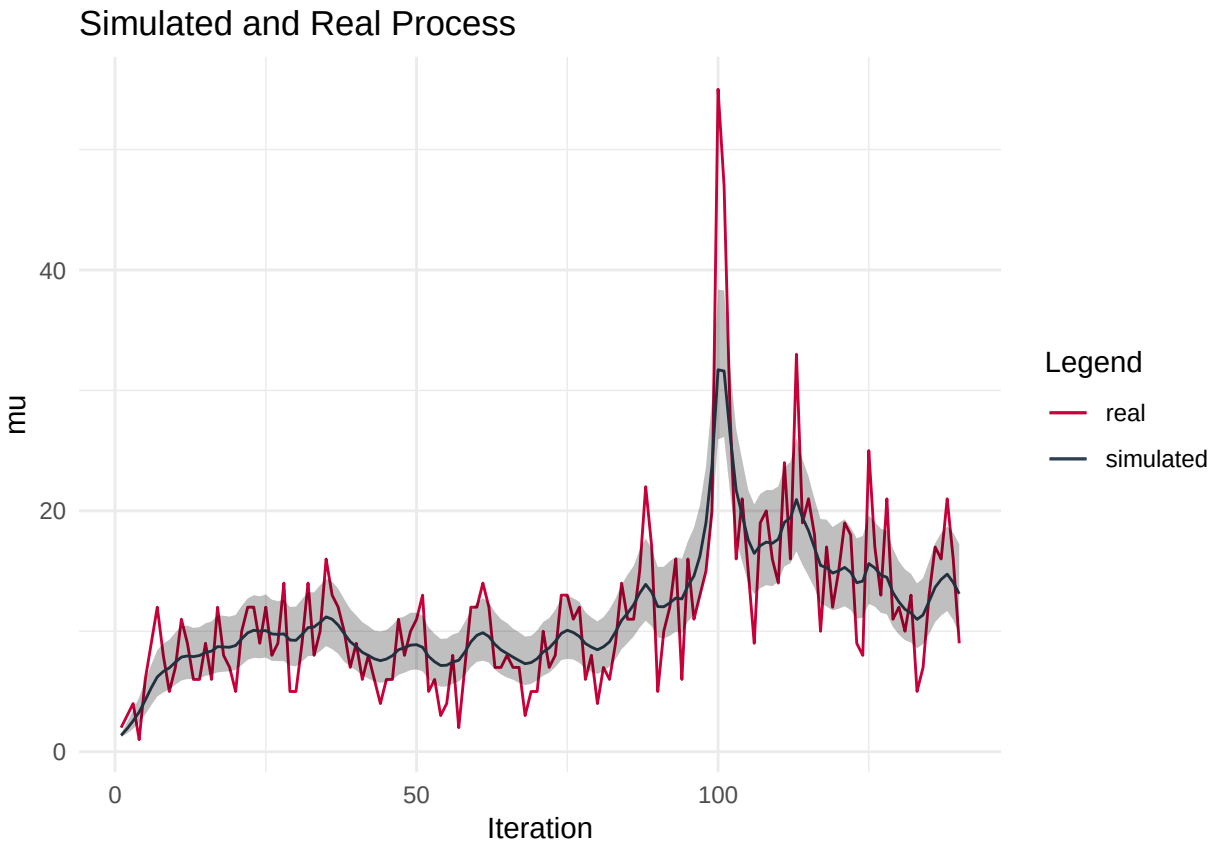
```
################################################################################
# Exercise 1.d)
################################################################################

stanModel = '
data {
  int<lower=0> N;
  int<lower=0> y[N];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
```

```
  vector[N] x;
}
model {

  // Priors
  mu ~ normal(5, 1);
  phi ~ normal(0, 10);
  sigma_sq ~ scaled_inv_chi_square(300, 0.1);

  // Model
  x[1] ~ normal(mu, sigma_sq);
  x[2:N] ~ normal(mu + phi * x[1:(N-1)], sqrt(sigma_sq));
  y[1:N] ~ poisson(exp(x[1:N]));
}
'

stanModelPoisson = stan(model_code = stanModel,
                  model_name = "AR_Poisson2",
                  data = list(N = length(campy_data$c), y = campy_data$c),
                  warmup = 1000,
                  iter = 2000)

posteriorPoisson = extract(stanModelPoisson)

ci = apply(exp(posteriorPoisson$x), 2, quantile, probs = c(0.025, 0.975))

df = data.frame(x = 1:length(campy_data$c),
                  real = campy_data$c,
                  simulated = exp(colMeans(posteriorPoisson$x)),
                  lower_ci = ci[1,],
                  upper_ci = ci[2,])
```

```
stanModelPoisson
```

```
## Inference for Stan model: AR_Poisson2.
## 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%
## mu          0.30     0.0  0.05    0.21    0.27    0.30    0.34    0.42
## phi         0.88     0.0  0.02    0.83    0.86    0.88    0.89    0.92
## sigma_sq    0.01     0.0  0.00    0.01    0.01    0.01    0.01    0.01
## x[1]        0.31     0.0  0.06    0.20    0.27    0.30    0.34    0.42
## x[2]        0.65     0.0  0.12    0.41    0.56    0.65    0.73    0.89
##          n_eff Rhat
## mu        2573    1
## phi       2634    1
## sigma_sq  2469    1
## x[1]      2670    1
## x[2]      3043    1
##  [ reached getOption("max.print") -- omitted 139 rows ]
##
## Samples were drawn using NUTS(diag_e) at Thu May 23 17:51:33 2019.
## 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).
```



2 Source Code

```
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(ggplot2)
library(gridExtra)
library(coda)
library(rstan)

options(max.print=50)

# RStan Setup
## Use multi-cores
options(mc.cores = parallel::detectCores())
## Reuse compiled binary
rstan_options(auto_write = TRUE)

################################################################################
# Exercise 1.a)
################################################################################

ar_process = function(mu, tau, phi, sigma_sq) {
```

```r
  # storing the values
  X = rep(NaN, tau)

  X[1] = mu

  for (i in 1:(tau - 1)) {
    X[i+1] = mu + phi *(X[i] - mu) + rnorm(n = 1, mean = 0, sd = sqrt(sigma_sq))
  }

  return(X)
}


simulate_ar_process = function(mu, tau, phi, sigma_sq) {

  res_1 = ar_process(mu, tau, phi, sigma_sq)
  res_2 = ar_process(mu, tau, phi, sigma_sq)

  df = data.frame(x = 1:tau,
                  y1 = res_1,
                  y2 = res_2)

  p = ggplot(df) +
    geom_line(aes(x = x, y = y1), color = "#C70039") +
    geom_line(aes(x = x, y = y2), color = "#2E4053") +
    labs(title = paste("mu =", mu, "| T =", tau, "\nphi =",
                       phi, " | sigma_sq = ", sigma_sq) , y = "mu",
    x = "Iteration", color = "Legend") +
    theme_minimal()

  return(p)
}

p1 = simulate_ar_process(mu = 10, tau = 200, phi = -0.5, sigma_sq = 2)
p2 = simulate_ar_process(mu = 10, tau = 200, phi = 0.75, sigma_sq = 2)
p3 = simulate_ar_process(mu = 10, tau = 200, phi = 0.9, sigma_sq = 2)
p4 = simulate_ar_process(mu = 10, tau = 200, phi = 1, sigma_sq = 2)

grid.arrange(p1, p2, p3, p4, nrow = 2)


##############################################################################
# Exercise 1.b)
##############################################################################

X = ar_process(mu = 10, tau = 200, phi = 0.3, sigma_sq = 2)
Y = ar_process(mu = 10, tau = 200, phi = 0.95, sigma_sq = 2)

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

```
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
}
model {
  y[2:N] ~ normal(mu + phi * y[1:(N - 1)], sqrt(sigma_sq));
}
'

stanModelX = stan(model_code = stanModel,
                  model_name = "AR_X",
                  data = list(N = length(X), y = X),
                  warmup = 1000,
                  iter = 2000)

stanModelY = stan(model_code = stanModel,
                  model_name = "AR_Y",
                  data = list(N = length(Y), y = Y),
                  warmup = 1000,
                  iter = 2000)

posteriorX = extract(stanModelX)
posterior_paramsX = As.mcmc.list(stanModelX)

posteriorY = extract(stanModelY)
posterior_paramsY = As.mcmc.list(stanModelY)


stanModelX
#summary(stanModelX,
#        pars = c("mu", "phi", "sigma_sq"), probs = c(0.025, 0.975))$summary

#plot(posterior_paramsX)
pairs(stanModelX)


stanModelY
#summary(stanModelY,
#          pars = c("mu", "phi", "sigma_sq"), probs = c(0.025, 0.975))$summary

#plot(posterior_paramsY)
pairs(stanModelY)


plotdf = data.frame(muX = posteriorX$mu,
                    phiX = posteriorX$phi,
                    muY = posteriorY$mu,
                    phiY = posteriorY$phi)

ggplot(plotdf)+
  geom_point(aes(x = muX, y = phiX), color = "black",
             fill = "#dedede", shape = 21) +
```

```
  labs(title = "Joint Posterior of mu and phi for process 1",
       y = "phi", x = "mu", color = "Legend") +
  theme_minimal()

ggplot(plotdf)+
  geom_point(aes(x = muY, y = phiY), color = "black",
             fill = "#dedede", shape = 21) +
  labs(title = "Joint Posterior of mu and phi for process 2",
       y = "phi", x = "mu", color = "Legend") +
  theme_minimal()


################################################################################
# Exercise 1.c)
################################################################################

campy_data = read.table("data/campy.dat", header=TRUE)


stanModel = '
data {
  int<lower=0> N;
  int<lower=0> y[N];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
  vector[N] x;
}
model {

  // Priors
  mu ~ normal(5, 1);
  phi ~ normal(0, 10);
  sigma_sq ~ scaled_inv_chi_square(1, 10);

  // Model
  x[1] ~ normal(mu, sigma_sq);
  x[2:N] ~ normal(mu + phi * x[1:(N-1)], sqrt(sigma_sq));
  y[1:N] ~ poisson(exp(x[1:N]));
}
'

stanModelPoisson = stan(model_code = stanModel,
                  model_name = "AR_Poisson",
                  data = list(N = length(campy_data$c), y = campy_data$c),
                  warmup = 1000,
                  iter = 2000)

posteriorPoisson = extract(stanModelPoisson)
```

```
stanModelPoisson


ci = apply(exp(posteriorPoisson$x), 2, quantile, probs = c(0.025, 0.975))

df = data.frame(x = 1:length(campy_data$c),
                  real = campy_data$c,
                  simulated = exp(colMeans(posteriorPoisson$x)),
                  lower_ci = ci[1,],
                  upper_ci = ci[2,])


ggplot(df) +
    geom_line(aes(x = x, y = real, color = "real")) +
    geom_line(aes(x = x, y = simulated, color = "simulated")) +
    geom_ribbon(aes(x = x, ymin = lower_ci, ymax = upper_ci), fill = "black",
              alpha = 0.25) +
    labs(title = "Simulated and Real Process" , y = "mu",
    x = "Iteration", color = "Legend") +
    scale_color_manual(values = c("#C70039", "#2E4053")) +
    theme_minimal()


##############################################################################
# Exercise 1.d)
##############################################################################

stanModel = '
data {
  int<lower=0> N;
  int<lower=0> y[N];
}
parameters {
  real mu;
  real phi;
  real<lower=0> sigma_sq;
  vector[N] x;
}
model {

  // Priors
  mu ~ normal(5, 1);
  phi ~ normal(0, 10);
  sigma_sq ~ scaled_inv_chi_square(300, 0.1);

  // Model
  x[1] ~ normal(mu, sigma_sq);
  x[2:N] ~ normal(mu + phi * x[1:(N-1)], sqrt(sigma_sq));
  y[1:N] ~ poisson(exp(x[1:N]));
}
'

stanModelPoisson = stan(model_code = stanModel,
```

```
                  model_name = "AR_Poisson2",
                  data = list(N = length(campy_data$c), y = campy_data$c),
                  warmup = 1000,
                  iter = 2000)

posteriorPoisson = extract(stanModelPoisson)

ci = apply(exp(posteriorPoisson$x), 2, quantile, probs = c(0.025, 0.975))

df = data.frame(x = 1:length(campy_data$c),
                  real = campy_data$c,
                  simulated = exp(colMeans(posteriorPoisson$x)),
                  lower_ci = ci[1,],
                  upper_ci = ci[2,])


stanModelPoisson


ggplot(df) +
    geom_line(aes(x = x, y = real, color = "real")) +
    geom_line(aes(x = x, y = simulated, color = "simulated")) +
    geom_ribbon(aes(x = x, ymin = lower_ci, ymax = upper_ci), fill = "black",
              alpha = 0.25) +
    labs(title = "Simulated and Real Process" , y = "mu",
    x = "Iteration", color = "Legend") +
    scale_color_manual(values = c("#C70039", "#2E4053")) +
    theme_minimal()
```