Bayesian Learning - Computer Lab 4

Stefano Toffol (steto820) and Nahid Farazmand (nahfa911)

26 May, 2019

Question 1 - Time series models with Stan

(a) Function for AR(1)

We are first of all asked to write a function that simulates from an AutoRegressive process of order 1 (also called AR(1)). The process has the following form:

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

where the values for μ, ϕ and σ^2 are given. Since in such a type of model the observations are a function of the previous one, we also need to specify the value of the first observation. In this case we will set $x_1 = \mu$. We will generate a total of $T = 200$ observations, using the values $\mu = 10$ and $\sigma^2 = 2$ for the parameters of the mean of the process and the variance of the Normal error.
We are then asked to implement the generation function, testing it for different values of ϕ and displaying the results in a plot. We have to remember however that ϕ has to be included in the open interval $(-1, +1)$ in order to guarantee the process to be stationary.

The code that follows include both the generation function and the testing of different ϕ:

```r
# Generation function for the AR(1) process
my_AR <- function(phi, t = 200, mu = 10, sigma2 = 2) {

  # Starting point is the mean of the error
  x_start <- mu
  # Prepare the vector where to store the results
  results <- rep(NA, t)
  # Generate the first value
  results[1] <- mu + rnorm(1, 0, sqrt(sigma2))
  # The actual generation process
  for(i in 2:t) {
    results[i] <- mu + phi*(results[i-1] - mu) + rnorm(1, 0, sqrt(sigma2))
  }
  return( results )

}

# Test different values for the AR process
set.seed(12345)
tested_phi <- c(0.8, 0.3, 0, -0.5)
generated_series <- as.data.frame(sapply(tested_phi, my_AR))
generated_series$Time <- 1:nrow(generated_series)
```

In Figure 1 we can find the plots containing the four generated time series. From this set of plots we can understand how the parameter ϕ controls the general trend of the time series.
In time series theory an AutoRegressive model of order 1 is said to be *stationary* if $|\phi| < 1$: in this case the average of the stochastic process remains the same throughout the passing of time. This process in fact consists of a linear combination of random errors, where the effect of the first observation, even if mitigated by the successive realizations, is carried until the end of the series: since each observation depends on a certain weight (ϕ) from the previous one, even after $k > 1$ lags, the newly generated observation will partially

depend on the random error of the first observation, weighted by ϕ^k. In an AR process a one-time shock affects values of the evolving variable infinitely far into the future. What we will get is:

$$\begin{aligned}
&x_0 = \mu \\
&x_1 = \mu + \phi(x_0 - \mu) + \epsilon_1 = \mu + \epsilon_1 \\
&x_2 = \mu + \phi(x_1 - \mu) + \epsilon_2 = \mu + \phi\epsilon_1 + \epsilon_2 \\
&\dots \\
&x_T = \mu + \phi(x_{T-1} - \mu) + \epsilon_T = \mu + \sum_{i=1}^{T} \phi^{T-i}\epsilon_i
\end{aligned}$$

Having the parameter ϕ close to 1 therefore means that every point of the time series will be heavily dependent on the previous observations. In fact for $\phi = 0.8$ we get an heavily correlated graph, with an almost linear sum of the random errors. If we instead nullify the effect of the time lag with $\phi = 0$ we will get a simple succession of Normal random errors. A parameter somehow in the middle ($\phi = 0.3$) leads instead to a series somehow in the middle of the previous two. Finally, a negative parameter such as $\phi = -0.5$ leads to approximately the same oscillation of the positive counterpart but the trend of the series is much more “sawtooth” like: this is because the neighbouring values typically have opposite sign, therefore changing direction at almost every new value.

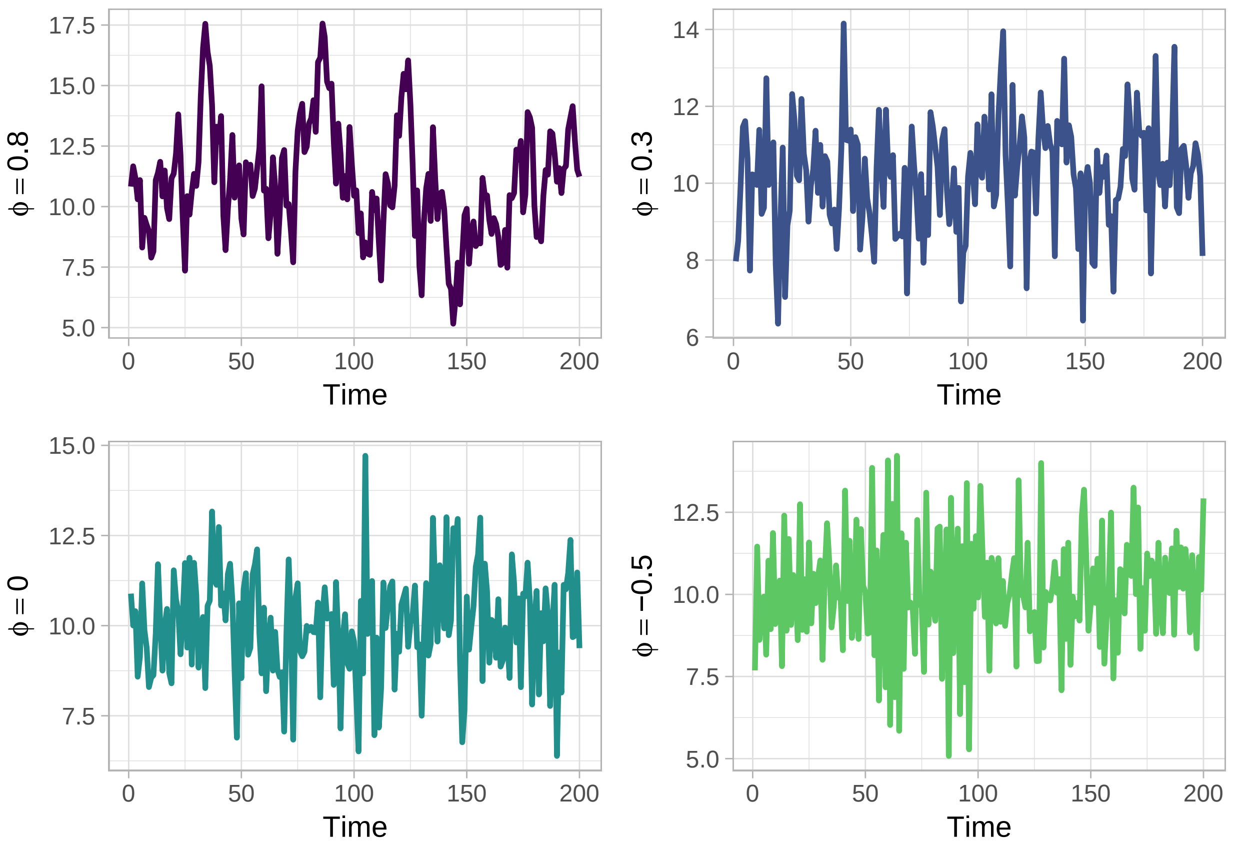

Figure 1: Generated time series for values of $\phi = (0.8, 0.3, 0, -0.5)$.

(b) AR(1) with MCMC and non-informative priors using *Stan*

We are now asked to generated two different time series: $x_{1:T}$ with $\phi = 0.3$ and $y_{1:T}$ with $\phi = 0.95$, while keeping the other parameters unchanged ($\mu_x = \mu_y = 10$ and $\mu_x = \mu_y = 10$). What we will do is then hypothesize that all the parameters (ϕ, μ and σ) are unknown and therefore need to be estimated using MCMC. In order to achieve it we will use Stan-code to sample from the posterior of the three parameters. We picked three non-informative priors which were similarly to the default ones of Stan:

$$\begin{aligned}\mu &\sim \mathcal{N}(0, 1000)\\ \phi &\sim \mathcal{N}(0, 1)\\ \sigma^2 &\sim \mathcal{N}(1, 10)\end{aligned}$$

We picked the distribution such that the mean would result in a practically flat distribution. Moreover we thought the AR parameter would be most likely included in the interval $[-1, 1]$, while the variance would have had a small value, so picked a distribution fluctuating around 1. We also needed to specify the constrain for sigma to be a positive value.
The code used to fit the models is the following:

```r
library(rstan)

# Generate the data
x <- my_AR(phi = 0.3)
y <- my_AR(phi = 0.95)

# Stan code for the model
Model_Stan <- "
data {
  int<lower=0> N;
  vector[N] d;
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  d[2:N] ~ normal(mu + phi * (d[1:(N - 1)] - mu), sigma);  // Efficient implementation
}"

# Prepare parameters of the model
data_x <- list(N = length(x), d = x)
data_y <- list(N = length(y), d = y)
burnin <- 1000
n_iter <- 2000

# Evaluate the models
fit_x <- stan(model_code = Model_Stan, data = data_x, warmup = burnin,
              iter = n_iter, chains = 4, refresh = 0)
fit_y <- stan(model_code = Model_Stan, data = data_y, warmup = burnin,
              iter = n_iter, chains = 4, refresh = 0)
```

```r
# Get the summary of the fitted model
s_x <- summary(fit_x)$summary
s_y <- summary(fit_y)$summary

# Extract posterior samples
postDraws_x <- extract(fit_x)
postDraws_y <- extract(fit_y)
```

In Table 1 is summarized the results of the fitted model for $x_{1:T}$ and $y_{1:T}$ respectively, together with the 95% credible intervals of the parameters.

For the first of the two time series the estimation was fairly successful. The point estimate of the mean μ_x was slightly underestimated (9.7852) but the real value 10 is included in the interval, despite being close to the upper extreme of its CI. The effective sample size, which measures the amount of independent samples leading to the same estimation power of our dependent sample, is almost equal to 3300. In other words only 17.5% of the generated values were not bringing useful informations for the estimation process: it suggests that the model is able to generate mainly uncorrelated values, providing high efficiency for our generator.
For the AR parameter ϕ_x the estimate, 0.3294, was quite close to the real value (0.3), however the uncertainty observed is remarkable, with a range of 0.2615. The effective sample size is smaller this time (2867), meaning that the choice of the priors may still be improved. Anyway we probably need an higher amount of generated points to get a more precise result.
Finally, the estimate of the variance σ has similar characteristics to the μ parameter: the point estimate in slightly underestimated and the real value ($\sqrt{2} = 1.4142$) is close to the upper margin of the interval. The effective sample size is almost 3200, meaning we obtain high efficiency for almost all the parameters of this model.

In the time series $y_{1:T}$ things are different, since the proximity of ϕ_y to 1 leads to heavily correlated points, which may generate local stochastic trends in the data (despite the stationary of the process). The parameter μ_y is therefore the one particularly affected by this change in the AR parameter: the point estimate, equal to 8.3386, is quite distant from the real value, but what changes the most is the precision of the estimation. In fact the range of the CI is 43.278, including negative values as well. The effective sample size is merely of 99 and this is due to the high dependency of the generated values.
ϕ_y on the other hand remains accurate, even more than the its counterpart from the time series $x_{1:T}$. The point estimate is 0.9634, differing of only 0.0134 from the real value. The precision of the estimate is quite high (the range of the interval is just 0.0911) despite the low number of estimated effective size. The interval however contains value above 1, the threshold under which the stationary of the process is guaranteed.
Lastly, the CI of the parameter σ_y has approximately the same width of the one from σ_x but is almost perfectly centred on the real value. The effective sample size is the highest of the three, although being still under 40% of the generated values.

Table 1: Summary of the fitted Stan model and credible intervals for the parameters of the time series x & y.

	Std error	Lower margin	Mean	Upper margin	Effective size
Time Seriers x					
μ_x	0.1454676	9.4983942	9.7851526	10.0660354	3318.34758
ϕ_x	0.0668383	0.1982639	0.3293639	0.4597881	2866.68245
σ_x	0.0688713	1.2242150	1.3450519	1.4934933	3148.48527
Time Seriers y					
μ_y	8.4977754	-18.0150585	8.3386133	25.2629270	98.82853
ϕ_y	0.0256796	0.9132667	0.9633695	1.0043420	482.64776
σ_y	0.0715950	1.2856192	1.4199153	1.5675986	1522.24285

In Figure 2 and 3 we can find the traceplots of the three parameters. Only the first chain was included in order to avoid confusions, since the behaviour of each one of them is pretty similar to one another, even with $\phi = 0.95$. The red dashed line represents the general mean of all the chains for that specific parameter.
For the time series $x_{1:T}$ there is no doubt that all the parameters have reached convergence: there are no visible trends in the generated values, nor many wide jumps between consecutive observations. The mean line is cutting through the middle of the points and in general the oscillations are contained within a symmetric area around the mean.
On the other hand for the time series $y_{1:T}$ there is more room for debate, especially after having analysed the results in Table 1. The parameter μ_y seems to not have reached convergence. In fact, despite the values oscillating around the general mean, each point is quite dependent from the previous, creating big jumps between successive observations and local trends. The other two parameters, although not having evident sequences of heavily dependent numbers, present some outliers among the generates values. Therefore the convergence for ϕ_y and σ_y, despite still questionable, is more likely than the one for the mean μ_y.

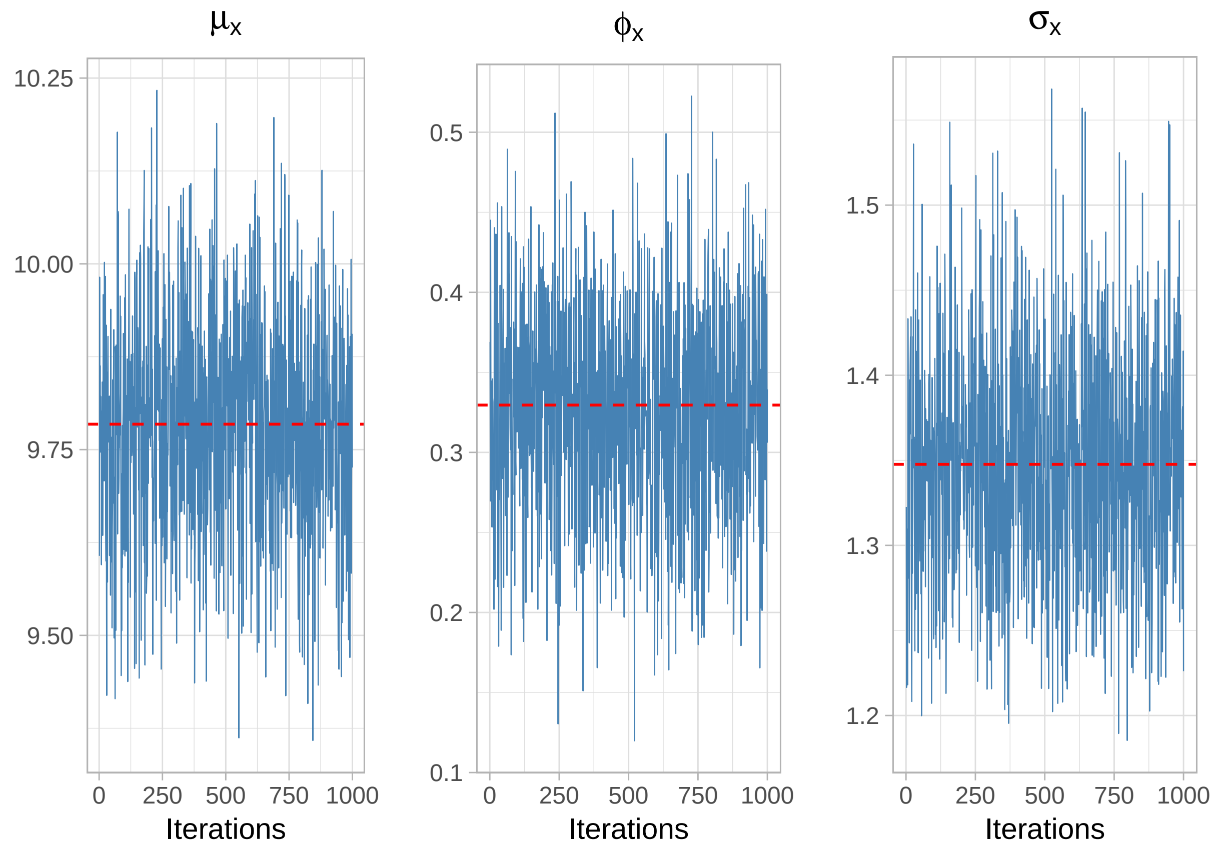

Figure 2: Traceplots of the first chain for the three paramaters (μ, ϕ, σ) **for the time series x**.

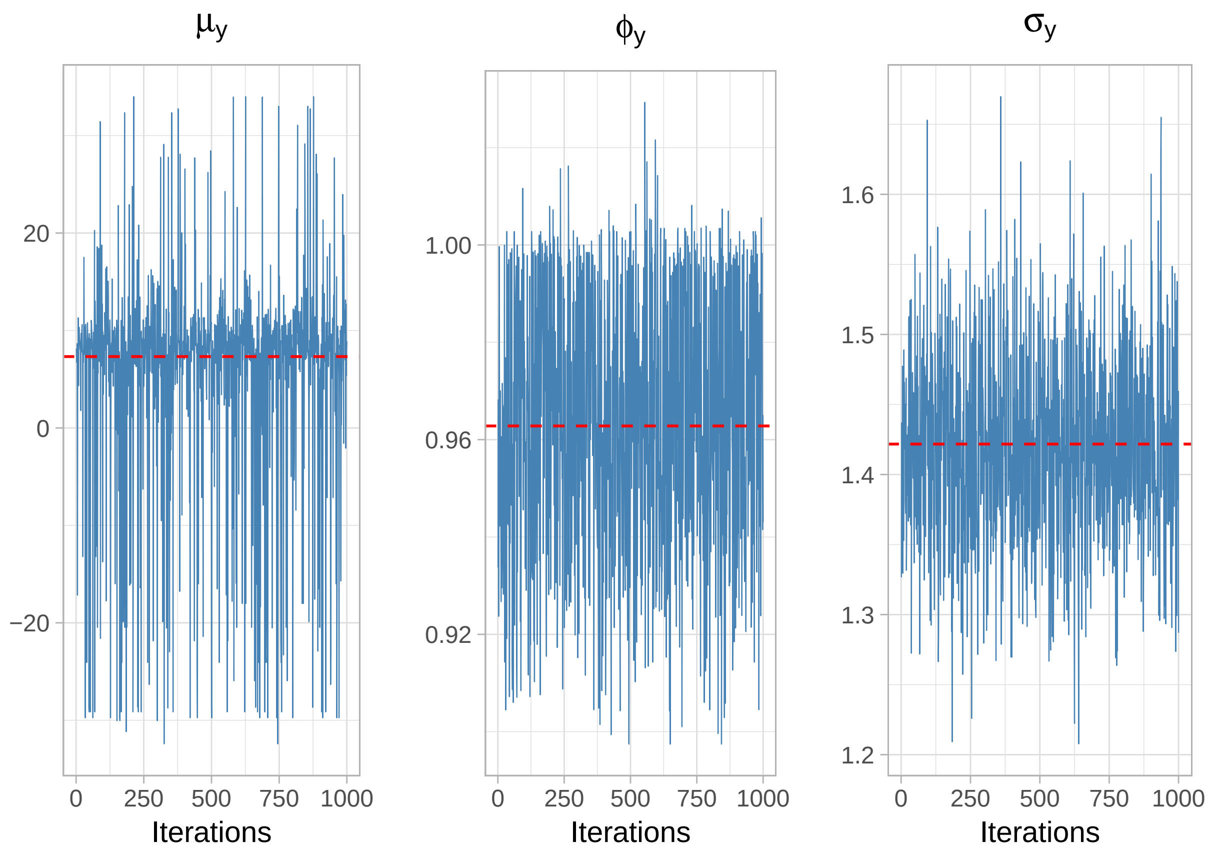

Figure 3: Traceplots of the first chain for the three paramaters (μ, ϕ, σ) **for the time series y**.

Finally, in Figure 4 and 5 we plotted the joint distribution of the parameters (μ, ϕ) for both time series $x_{1:T}$ and $y_{1:T}$.
For the data generated with $\phi = 0.3$ the shape of the contour plot closely resemble an ellipsoid, giving us the idea the posterior could at least be approximated by a multivariate normal distribution. The distribution is more stretched along the x-axis, meaning that the variability on μ is predominant over the one of ϕ. There are some outliers, but nothing that systematically deviates from normality.
On the contrary the contour plot of (μ_y, ϕ_y) does not look normal at all: the dots create the shape of a funnel, with small variation along the x-axis for values of $\phi_y < 0.95$ and an exponential increase of the span when ϕ_y approaches 1. In fact when the MCMC process leads to chains of a non-stationary process, the parameter for the mean gets unstable and unpredictable, explaining the big variation observed in its credible interval (Table 1).

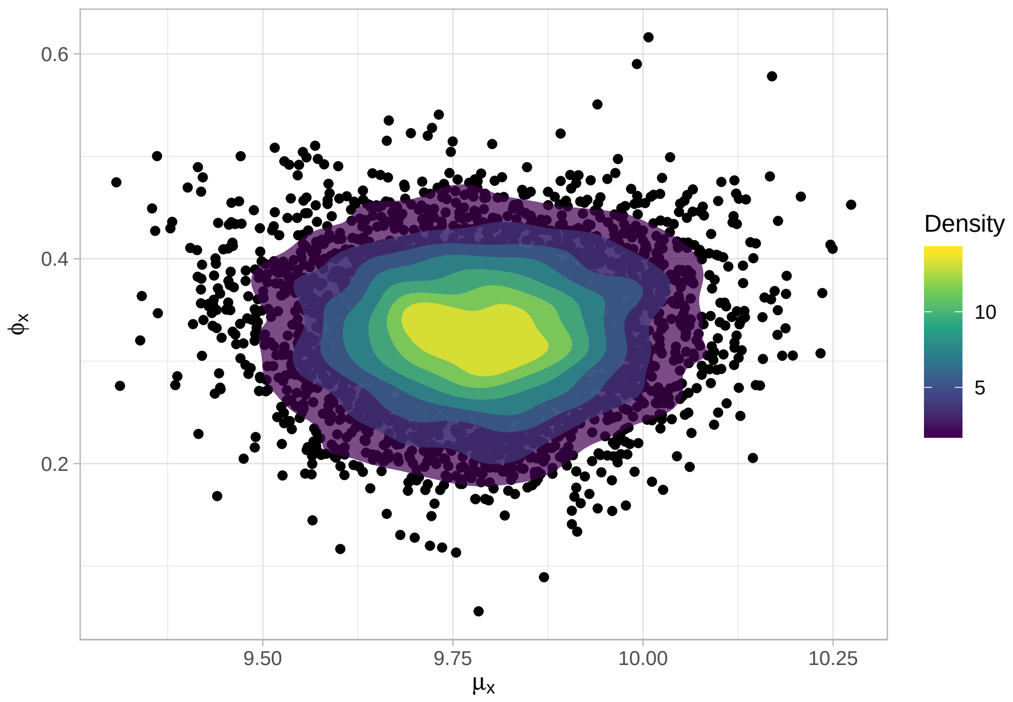

Figure 4: Joint posterior density of μ an ϕ **for the time series x**.

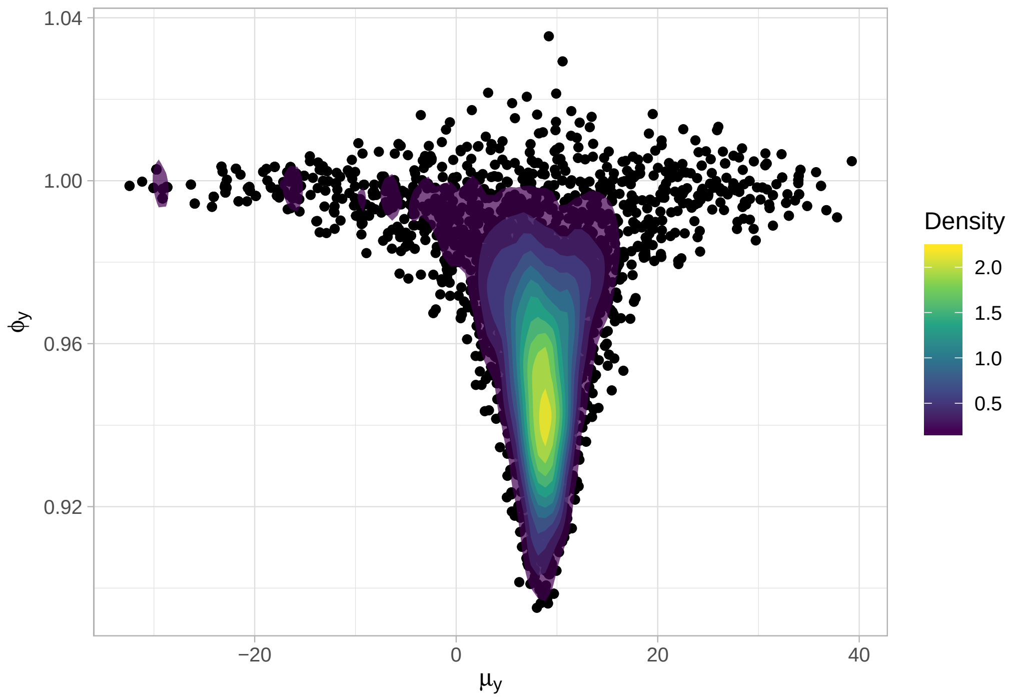

Figure 5: Joint posterior density of μ an ϕ **for the time series y**.

(C) Poisson model over a time series using *Stan*

For this task we will abandon the generated data and instead using a dataset called `campy.dat`. It contains the cases of campylobacter infections in the north of the Quebec province (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. In order to model the phenomena we will assume a independent *Poisson* distribution for each moment in time, where the intensity of the random variable is given by a AR(1)-process, that is:

$$c_t \mid x_t \sim Poisson(\exp(x_t))$$

In other words we are conditioning the number of infections c_t to the time series x_t, given by the autoregressive process of order 1 created in step a). Because of that, we can rewrite the model as follows:

$$c_t \mid x_t \sim Poisson\left(\exp\left[\mu + \sum_{i=1}^{t} \phi^{t-i}\epsilon_i\right]\right) \quad \text{where:} \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

We can therefore reuse the priors set in the previous exercise and try to fit the *Poisson* model using *Stan*.

The code that follows implement the above described model. As we can see, we needed to specify the hidden AR process as a parameter, even though strictly speaking it is not. In fact the process, despite being unknown and therefore needed to be estimated, solely depends on the parameters (μ, ϕ, σ), which in statistical terms is a hidden process, NOT a parameter. However we do not have any knowledge of the time series, hence we are unable to provide it as a `data` inside the `Stan` structure. Being random (stochastic process), it also needs to be generated in the MCMC process, and this is why it figures as a parameter.

```r
# Stan code for the model
Model_Stan <- "
data {
  int<lower=0> N;
  int d[N, 1];
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
  vector[N] x;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  x[2:N] ~ normal(mu + phi * (x[1:(N - 1)] - mu), sigma);  // Efficient implementation
  for (n in 2:N) {
    d[n] ~ poisson(exp(x[n]));
  }
}"

# Prepare parameters of the model
data_stan <- list(N = nrow(data), d = data)
burnin <- 1000
n_iter <- 2000
```

```
# Evaluate the models
fit_data <- stan(model_code = Model_Stan, data = data_stan, warmup = burnin,
                 iter = n_iter, chains = 4, refresh = 0)

# Get the summary of the fitted model
s_data <- summary(fit_data)$summary

# Extract posterior samples
postDraws_data <- extract(fit_data)
```

In Figure 6 we can observe the result of our predictions over the actual dataset. We can see how the model, despite capturing the actual trend, gets heavily influenced by the random noise present, resulting in a heavily bumping line. This exact same behaviour is reflected in its CI as well. This characteristic takes an extreme when, around $x = 100$, the distribution register two outliers, for a peak of more than 50 cases: in that occasion the posterior follows blindly the data. This type of model may be good for the description of the phenomena, but will almost surely perform extremely poorly over future observations to predict.
It's interesting to look into the estimations of the parameters (μ, ϕ, σ) as well. The mean of the AR process is about 2.3946, which means that the average number of observed cases across the whole time window is $\exp(x_t) = 10.9634$. The autoregressive factor ϕ has a quite high posterior mean (0.8149), which suggests correlated values. However the uncertainty over this parameter is considerable: its credible interval spans form 0.6809 up to 0.9296. Nonetheless the value of the parameter hints for dependent values. Finally the variance σ has a posterior mean of 0.2563. It does not vary that much, having a range of its CI equal to 0.1305. All the parameters registered a low number of effective samples (2911, 1208 and 778 respectively), suggesting a weak efficiency in the generation process.

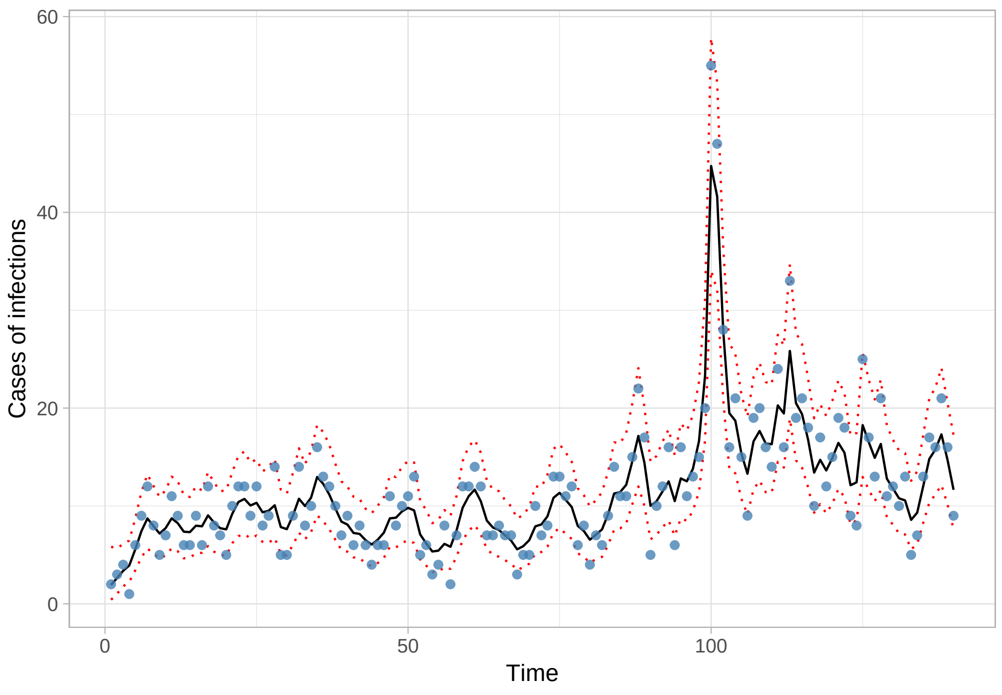

Figure 6: Time series of the dataset ‘campy‘ with the posterior mean (continuous black line) and its 95% credible interval (red dashed lines) **using an un-informative prior**.

(D) AR(1) with informative prior using *Stan*

For our last task we are asked to change the prior for σ with the aim of generating a more smooth change for the true underlying intensity $\theta_t = \exp(x_t)$. We would like a more informative prior suggesting small changes in the AR process caused by the random error ϵ_t.
In order to do so, we changed exclusively the prior for the variance, and taking inspiration from the slides we ended up with:

$$\text{Inv} - \chi^2(100, 0.0625)$$

This choice was partially driven by the previous results: we saw how an already small variance ($\tilde{\sigma} = 0.2563$) was generating too bumpy posterior means. Therefore, to be sure we would nullify its effect almost completely, we scaled it down to approximately 2.5% and input a high degree of belief in our prior ($\nu_0 = 100$). Here is the code for the task, which is almost the same of the previous one:

```r
# Stan code for the model
Model_Stan_prior <- "
data {
  int<lower=0> N;
  int d[N, 1];
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
  vector[N] x;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ scaled_inv_chi_square(100, 0.0625); // Scaled-inv-chi2 with nu 1, sigma 2
  x[2:N] ~ normal(mu + phi * (x[1:(N - 1)] - mu), sqrt(sigma));  // Efficient implementation
  for (n in 2:N) {
    d[n] ~ poisson(exp(x[n]));
  }
}"

# Evaluate the models
fit_prior <- stan(model_code = Model_Stan_prior, data = data_stan, warmup = burnin,
                  iter = n_iter, chains = 4, refresh = 0)

# Get the summary of the fitted model
s_prior <- summary(fit_prior)$summary

# Extract posterior samples
postDraws_prior <- extract(fit_prior)
```

We can finally see in Figure 7 the achieved result: the posterior mean and its credible interval are much smoother, showing us the trend of the data without almost any interference of the random error presence. As a consequence, the CI is almost tighter around the mean. The model does not fully describe the peak of cases at $time = 100$, but still catches a substantial increase in the response.
The estimate of the parameters presents its differences as well: while the estimates for μ and ϕ remained almost unchanged from the previous step, the estimate of σ naturally deviates. The posterior mean of the

variance is now 0.0052, therefore even smaller than our prior belief, and the interval, equal to $[0.0039, 0.0074]$, is quite narrow, suggesting a relevant degree of belief. However the effective sample size is almost reverse: for the mean is now barely equal to 400, for ϕ it reaches the thousands and for the variance is slightly higher than half of the sample dimension from the MCMC.

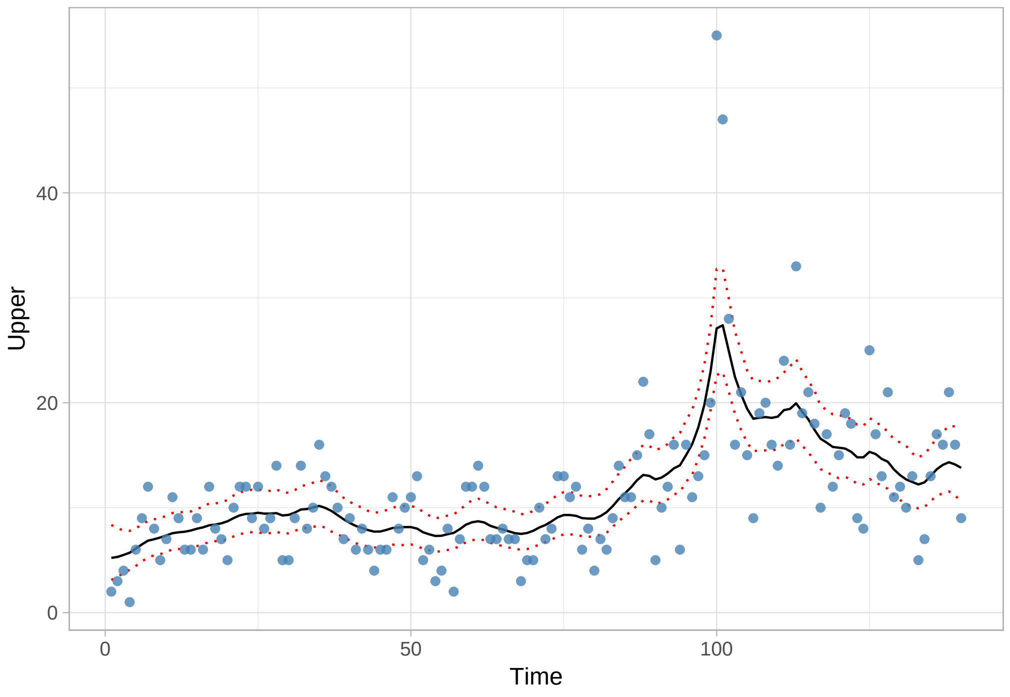

Figure 7: Time series of the dataset ‘campy‘ with the posterior mean (continuous black line) and its 95% credible interval (red dashed lines) **using an informative prior**.

Appendix

```r
knitr::opts_chunk$set(echo = F, message = F, error = F, warning = F,
                      fig.align='center', out.width="70%")


# ------------------------------------------------------------------------------
# Q1 - Intro
# ------------------------------------------------------------------------------

# Read the data
data <- read.delim("campy.dat", header = T)
colnames(data) <- "campy"
n <- nrow(data)
p <- ncol(data)


# ------------------------------------------------------------------------------
# Q1 - A
# ------------------------------------------------------------------------------


# Generation function for the AR(1) process
my_AR <- function(phi, t = 200, mu = 10, sigma2 = 2) {

  # Starting point is the mean of the error
  x_start <- mu
  # Prepare the vector where to store the results
  results <- rep(NA, t)
  # Generate the first value
  results[1] <- mu + rnorm(1, 0, sqrt(sigma2))
  # The actual generation process
  for(i in 2:t) {
    results[i] <- mu + phi*(results[i-1] - mu) + rnorm(1, 0, sqrt(sigma2))
  }
  return( results )

}

# Test different values for the AR process
set.seed(12345)
tested_phi <- c(0.8, 0.3, 0, -0.5)
generated_series <- as.data.frame(sapply(tested_phi, my_AR))
generated_series$Time <- 1:nrow(generated_series)


library(ggplot2)
library(viridis)
# For labels in the legend
library(latex2exp)

plot_fun <- function(col) {
  index <- as.numeric(substr(col, 2, 2))
  ggplot(aes(x = Time), data = generated_series) +
    geom_line(aes(y = generated_series[,col]), col = viridis(5)[index], size = 1) +
    labs(y = TeX(paste("$\\phi =", tested_phi[index], "$"))) +
    theme_light()
}

names <- colnames(generated_series)[-5]

grbs <- lapply(X = names, FUN = plot_fun)

library(gridExtra)
grid.arrange(grobs = grbs, ncol = 2)


# ------------------------------------------------------------------------------
# Q1 - B
# ------------------------------------------------------------------------------


library(rstan)

# Generate the data
x <- my_AR(phi = 0.3)
y <- my_AR(phi = 0.95)

# Stan code for the model
Model_Stan <- "
data {
  int<lower=0> N;
  vector[N] d;
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  d[2:N] ~ normal(mu + phi * (d[1:(N - 1)] - mu), sigma);  // Efficient implementation
}"

# Prepare parameters of the model
data_x <- list(N = length(x), d = x)
data_y <- list(N = length(y), d = y)
burnin <- 1000
n_iter <- 2000

# Evaluate the models
fit_x <- stan(model_code = Model_Stan, data = data_x, warmup = burnin,
              iter = n_iter, chains = 4, refresh = 0)
fit_y <- stan(model_code = Model_Stan, data = data_y, warmup = burnin,
               iter = n_iter, chains = 4, refresh = 0)

# Get the summary of the fitted model
s_x <- summary(fit_x)$summary
s_y <- summary(fit_y)$summary

# Extract posterior samples
postDraws_x <- extract(fit_x)
postDraws_y <- extract(fit_y)


library(kableExtra)

df_table <- as.data.frame(rbind(s_x[-4, c(3,4,6,8,9)], s_y[-4, c(3,4,6,8,9)]))
row.names(df_table) <- c("$\\mu_x$", "$\\phi_x$", "$\\sigma_x$",
                         "$\\mu_y$", "$\\phi_y$", "$\\sigma_y$")
names_table <- c("Std error", "Lower margin", "Mean", "Upper margin", "Effective size")

kable(df_table, "latex", booktabs = T, align = "c", col.names = names_table,
      linesep = "", escape = F,
      caption = "Summary of the fitted Stan model and credible intervals for the
      parameters of the time series x \\& y.") %>%
  column_spec(c(2,5), border_right = T) %>%
  row_spec(0, bold = T) %>%
  row_spec(3, hline_after = T) %>%
  group_rows("Time Seriers x", 1, 3) %>%
  group_rows("Time Seriers y", 4, 6) %>%
  kable_styling(latex_options = "hold_position")


# NOTE: one could have simply used the following to make a combined traceplot
# of all the chains.
# traceplot(fit_y)

# Do traceplots of all tha chains and parameters for x
l <- (n_iter-burnin)
df_plt_mu <- data.frame(V = postDraws_x$mu[1:l], mean = mean(postDraws_x$mu),
                        Iterations = 1:l)
df_plt_phi <- data.frame(V = postDraws_x$phi[1:l], mean = mean(postDraws_x$phi),
                         Iterations = 1:l)
df_plt_sigma <- data.frame(V = postDraws_x$sigma[1:l], mean = mean(postDraws_x$sigma),
                           Iterations = 1:l)

plot_fun <- function(d, title) {
  ggplot(aes(x = Iterations), data = d) +
    geom_line(aes(y = V), col = "steelblue", size = 0.2) +
    geom_hline(aes(yintercept = mean(d$mean[1])), size = 0.5, lty = 2, col = "red") +
    labs(y = "") +
    ggtitle(title) +
    theme_light() +
    theme(plot.title = element_text(hjust = 0.5))
}
```

```r
p1 <- plot_fun(df_plt_mu, TeX("$\\mu_x$"))
p2 <- plot_fun(df_plt_phi, TeX("$\\phi_x$"))
p3 <- plot_fun(df_plt_sigma, TeX("$\\sigma_x$"))

grid.arrange(p1, p2, p3, ncol=3)


# Do traceplots of all tha chains and parameters for y
df_plt_mu <- data.frame(V = postDraws_y$mu[1:l], mean = mean(postDraws_y$mu),
                        Iterations = 1:l)
df_plt_phi <- data.frame(V = postDraws_y$phi[1:l], mean = mean(postDraws_y$phi),
                         Iterations = 1:l)
df_plt_sigma <- data.frame(V = postDraws_y$sigma[1:l], mean = mean(postDraws_y$sigma),
                           Iterations = 1:l)

p1 <- plot_fun(df_plt_mu, TeX("$\\mu_y$"))
p2 <- plot_fun(df_plt_phi, TeX("$\\phi_y$"))
p3 <- plot_fun(df_plt_sigma, TeX("$\\sigma_y$"))

grid.arrange(p1, p2, p3, ncol=3)


df_plot <- data.frame(mu = postDraws_x$mu,
                      phi = postDraws_x$phi)

ggplot(df_plot, aes(x = mu, y = phi)) +
  geom_point() +
  stat_density_2d(aes(fill = stat(level)), geom = "polygon", alpha = 0.7) +
  labs(y = TeX("$\\phi_x"), x = TeX("$\\mu_x")) +
  scale_fill_viridis_c(name = "Density") +
  theme_light()


df_plot <- data.frame(mu = postDraws_y$mu,
                      phi = postDraws_y$phi)

ggplot(df_plot, aes(x = mu, y = phi)) +
  geom_point() +
  stat_density_2d(aes(fill = stat(level)), geom = "polygon", alpha = 0.7) +
  labs(y = TeX("$\\phi_y"), x = TeX("$\\mu_y")) +
  scale_fill_viridis_c(name = "Density") +
  theme_light()


# ------------------------------------------------------------------------------
# Q1 - C
# ------------------------------------------------------------------------------


# Stan code for the model
Model_Stan <- "
data {
  int<lower=0> N;
  int d[N, 1];
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
  vector[N] x;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ normal(1, 10);
  x[2:N] ~ normal(mu + phi * (x[1:(N - 1)] - mu), sigma);  // Efficient implementation
  for (n in 2:N) {
    d[n] ~ poisson(exp(x[n]));
  }
}"

# Prepare parameters of the model
data_stan <- list(N = nrow(data), d = data)
burnin <- 1000
n_iter <- 2000

# Evaluate the models
fit_data <- stan(model_code = Model_Stan, data = data_stan, warmup = burnin,
                 iter = n_iter, chains = 4, refresh = 0)

# Get the summary of the fitted model
s_data <- summary(fit_data)$summary

# Extract posterior samples
postDraws_data <- extract(fit_data)


df_plot <- data.frame(Time = 1:nrow(data),
                      Data = data$campy,
                      Lower = exp(s_data[4:(3+nrow(data)),4]),
                      Mean = exp(s_data[4:(3+nrow(data)),6]),
                      Upper = exp(s_data[4:(3+nrow(data)),8]))

ggplot(df_plot, aes(x = Time)) +
  geom_line(aes(y = Upper), lty = 3, col = "red") +
  geom_line(aes(y = Lower), lty = 3, col = "red") +
  geom_line(aes(y = Mean), col = "black") +
  geom_point(aes(y = Data), col = "steelblue", alpha = 0.8) +
  labs(y = "Cases of infections") +
  theme_light()

# Stan code for the model
Model_Stan_prior <- "
data {
  int<lower=0> N;
  int d[N, 1];
```

```
}
parameters {
  real mu;
  real phi; // DO NOT add the constrains for stationarity: it gets unstable!
  real<lower=0> sigma;
  vector[N] x;
}
model {
  mu ~ normal(0, 1000);
  phi ~ normal(0, 1);
  sigma ~ scaled_inv_chi_square(100, 0.0625); // Scaled-inv-chi2 with nu 1, sigma 2
  x[2:N] ~ normal(mu + phi * (x[1:(N - 1)] - mu), sqrt(sigma));  // Efficient implementation
  for (n in 2:N) {
    d[n] ~ poisson(exp(x[n]));
  }
}"

# Evaluate the models
fit_prior <- stan(model_code = Model_Stan_prior, data = data_stan, warmup = burnin,
                  iter = n_iter, chains = 4, refresh = 0)

# Get the summary of the fitted model
s_prior <- summary(fit_prior)$summary

# Extract posterior samples
postDraws_prior <- extract(fit_prior)


df_plot <- data.frame(Time = 1:nrow(data),
                      Data = data$campy,
                      Lower = exp(s_prior[4:(3+nrow(data)),4]),
                      Mean = exp(s_prior[4:(3+nrow(data)),6]),
                      Upper = exp(s_prior[4:(3+nrow(data)),8]))

ggplot(df_plot, aes(x = Time)) +
  geom_line(aes(y = Upper), lty = 3, col = "red") +
  geom_line(aes(y = Lower), lty = 3, col = "red") +
  geom_line(aes(y = Mean), col = "black") +
  geom_point(aes(y = Data), col = "steelblue", alpha = 0.8) +
  theme_light()
```