S1: Fitting GAMs to medical monitoring data

Johannes Enevoldsen & Simon Tilma Vistisen

Table of contents

The following are a few examples on how to fit and work with generalized additive models (GAMs) in R using *mgcv*.

Packages

Packages used in this demonstration can be installed from CRAN using `install.packages("package name")`.

```
library(mgcv) # fitting GAMs
library(gratia) # visualising and working with GAMs
library(dplyr) # working with data-frames
library(ggplot2) # plotting
library(patchwork) # combining plots

# only used for heart beat detection (dependencies of `find_abp_beats()`)
# install.packages("accelerometry")
```

```
# install.packages("purrr")

theme_set(theme_minimal()) # Change the default plotting theme
```

Pulse pressure model

Load data

The first example corresponds to the model shown in Figure 2 in the paper.

We use a 30 second recording of arterial blood pressure (ABP). The sample data is structured as a list. It includes ABP `sample_pp$abp` and the start time of each inspiration `sample_pp$insp_start`.

```
sample_pp <- readRDS("sample_PP.RDS")
```

The ABP is stored as a data frame with a time column and a pressure column. The time is in seconds, the sample rate is 125 Hz, and the pressure unit is mmHg.

```
head(sample_pp$abp)
```

```
## # A tibble: 6 x 2
##      time   ABP
##     <dbl> <dbl>
## 1 0.00588  60.5
## 2 0.0139   60.8
## 3 0.0219   61.1
## 4 0.0299   61.2
## 5 0.0379   61.4
## 6 0.0459   61.6
```

The timing of inspirations is a data frame with a time column that indicates when a new inspiration starts.

```
head(sample_pp$insp_start)
```

```
## # A tibble: 6 x 2
##    time label
##   <dbl> <chr>
## 1  2.49 Insp start
## 2  6.25 Insp start
## 3  9.98 Insp start
## 4 13.7  Insp start
## 5 17.5  Insp start
## 6 21.2  Insp start
```

We can visualize the two data frames together.

```
abp_plot <- ggplot(sample_pp$abp, aes(time, ABP)) +
  geom_line() +
  geom_vline(aes(xintercept = time), color = "red",
             data = sample_pp$insp_start)

abp_plot
```

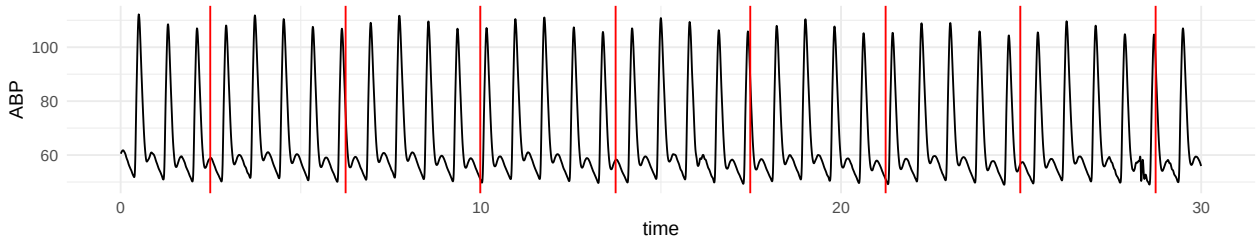


Figure 1: Arterial blood pressure (ABP). Red lines indicate inspiration start.

Detect heart beats

We now detect individual heart beats from the ABP waveform. A function, `find_abp_beats()`, is shared in the file `functions.R`. It takes an ABP waveform and returns a data frame of individual beats. The data frame contains the following columns (plus a few more):

- `time`: timing of the diastole (negative peak). This marks the beginning of a beat.
- `dia`: diastolic pressure.
- `time_systole`: timing of the systole (positive peak).
- `sys`: the following systolic pressure.
- `PP`: pulse pressure (`sys - dia`)
- `beat_len`: length of the beat (`time - lead(time)`)

```
source("functions.R")

beats <- find_abp_beats(sample_pp$abp)
head(beats)
```

```
## # A tibble: 6 x 10
##    time   dia time_systole   sys    PP beat_len dia_pos sys_pos .noise_wiggline~
##   <dbl> <dbl>        <dbl> <dbl> <dbl>    <dbl>   <dbl>   <dbl>            <dbl>
## 1 0.366  51.8        0.494  112.  60.4    0.808      46      62             17.1
## 2 1.17   50.9        1.31   108.  57.6    0.808     147     164             16.2
## 3 1.98   50.2        2.11   107   56.8    0.808     248     264             15.7
## 4 2.79   50.2        2.92   108.  57.9    0.8       349     365             15.6
## 5 3.59   51.5        3.72   112.  60.3    0.8       449     465             16.5
## 6 4.39   51.4        4.52   110.  59.1    0.808     549     565             16.6
## # ... with 1 more variable: .noise_pos_after_sys <dbl>
```

We can add this information to the previous plot.

```
abp_plot +
  geom_point(aes(x = time,
                 y = dia,
                 colour = "diastole"),
             data = beats) +
  geom_point(aes(x = time_systole,
                 y = sys,
                 colour = "systole"),
             data = beats)
```

For this model, we are only interested in the pulse pressure (PP) of each heart beat.

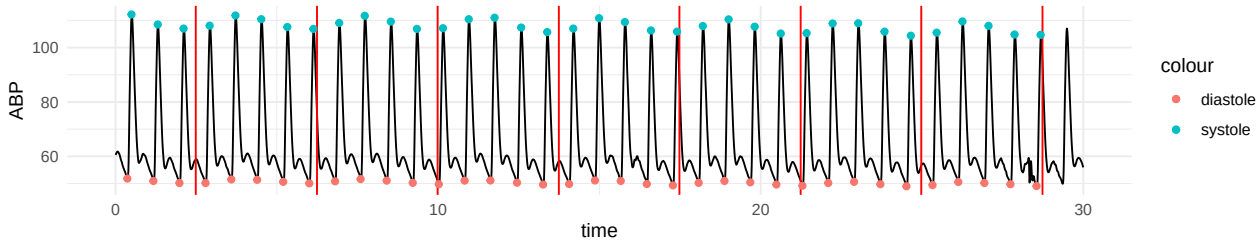


Figure 2: Beats are detected using 'find_abp_beats()'.

```
pp_plot <- ggplot(beats, aes(time, PP)) +
  geom_line() +
  geom_point() +
  geom_vline(aes(xintercept = time), color = "red",
             data = sample_pp$insp_start)

pp_plot
```

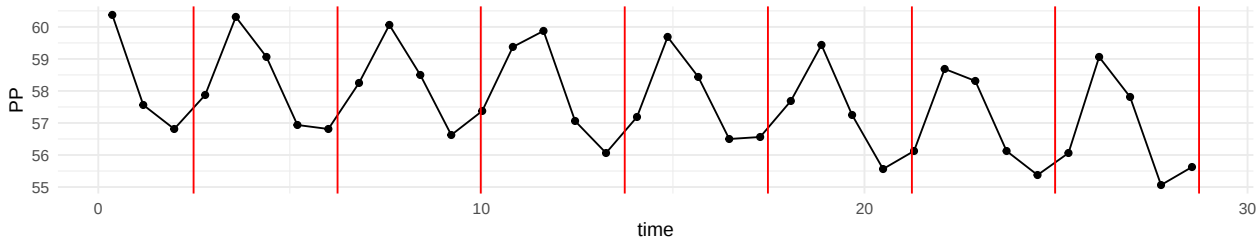


Figure 3: Pulse pressure (PP) of each heart beat.

Before we can fit the model, we need to calculate the position of each beat in the respiratory cycle. `functions.R` contain `add_time_since_event()`, that takes a data frame with a time column (here `beats`) and a vector of times corresponding to some event (here the timing of each inspiration start: `sample_pp$insp_start$time`) and returns the data frame with new columns indication the timing of each observation (beat) relative to the most recent event. The new columns are:

- `insp_index`: Time since the latest inspiration start.
- `insp_n`: Respiratory cycle number.
- `insp_cycle_len`: Length of the respiratory cycle.
- `insp_rel_index`: The relative position of the beat in the respiratory cycle (`insp_index` / `insp_cycle_len`).

```
beats_indexed <- add_time_since_event(beats,
                                      time_events = sample_pp$insp_start$time,
                                      prefix = "insp") %>%
  # the first beats are earlier than the first inspiration and
  # therefore have `insp_rel_index = NA`. We remove these.
  na.omit()

# Show only time and the four newly added columns. The other columns from `beats`
# are also in `beats_indexed`.
head(beats_indexed %>% select(-(dia:.noise_pos_after_sys)))
```

```
## # A tibble: 6 x 5
##    time insp_index insp_n insp_cycle_len insp_rel_index
##   <dbl>      <dbl>  <int>          <dbl>          <dbl>
## 1  2.79      0.303      1           3.76         0.0805
## 2  3.59      1.10       1           3.76         0.293
## 3  4.39      1.90       1           3.76         0.506
## 4  5.20      2.71       1           3.76         0.721
## 5  6.01      3.52       1           3.76         0.936
## 6  6.81      0.560      2           3.74         0.150
```

This lets us show each beat by its position in the respiratory cycle.

```
pp_plot_color <- ggplot(beats_indexed, aes(time, PP)) +
  geom_line() +
  # insp_n is a unique (consecutive) number for each respiratory cycle
  geom_point(aes(color = as.factor(insp_n)), show.legend = FALSE) +
  geom_vline(aes(xintercept = time), color = "red",
             data = sample_pp$insp_start) +
  labs(title = "Pulse pressure",
       subtitle = "by time. Color indicate respiratory cycle")

pp_insp_plot <- ggplot(beats_indexed,
                       aes(
                         insp_rel_index,
                         PP,
                         group = as.factor(insp_n),
                         color = as.factor(insp_n)
                         )
                       ) +
  geom_line(alpha = 0.3, show.legend = FALSE) +
  # insp_n is a unique (consecutive) number for each respiratory cycle
  geom_point(aes(color = as.factor(insp_n)), show.legend = FALSE) +
  labs(subtitle = "by position in the respiratory cycle")

pp_plot_color + pp_insp_plot + plot_layout(widths = c(2,1))
```

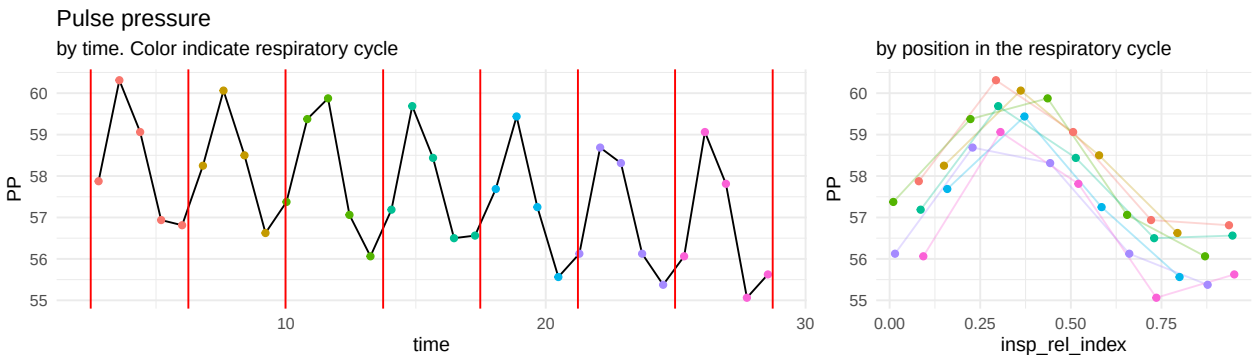


Figure 4: Pulse pressure indexed to the respiratory cycle.

We now have the variables we need to fit the model. For clarity, we select only the variables we need.

```r
PP_data <- select(beats_indexed, PP, time, insp_rel_index)

head(PP_data)
```

```
## # A tibble: 6 x 3
##      PP  time insp_rel_index
##   <dbl> <dbl>          <dbl>
## 1  57.9  2.79         0.0805
## 2  60.3  3.59         0.293
## 3  59.1  4.39         0.506
## 4  56.9  5.20         0.721
## 5  56.8  6.01         0.936
## 6  58.2  6.81         0.150
```

Fit and visualize the GAM

To fit the model, we use the `gam()` function from `mgcv`.

```r
PP_gam <- gam(
  # The first parameter to the gam() function is the model specification,
  # supplied using formula notation:
  PP ~ # Left of the tilde (~) is our dependent variable PP

    # Right of the tilde is our independent variables.
    # Define a smooth function of insp_rel_index.
    s(insp_rel_index,
      k = 15, # 15 knots.
      bs = "cc" # The basis is a cyclic cubic spline
      ) +
    # Define a smooth function of time
    s(time,
      bs = "cr" # The basis is a natural cubic spline.
      # default k is 10. This will be fine here.
      ),

  # We can specify the positions of the knots for each smooth.
  # If only two knots are specified for a cyclic spline, these will
  # set the positions of the limiting knot(s). The remaining knots will
  # be positioned automatically (at quantiles).
  knots = list(insp_rel_index = c(0,1)),

  # We use restricted maximum likelihood (REML) to fit the optimal smoothing parameter.
  # This is often the best choice, but not the default.
  method = "REML",

  data = PP_data
)
```

Now, plot the model using `gratia::draw()` (or `plot()`). Adding partial residuals is a simple way to visualize how the well model fits observed data, and to identify systematic errors. Partial residuals are the model residuals + the specific smooth effect in the plot.

```
draw(PP_gam,
     residuals = TRUE)
```

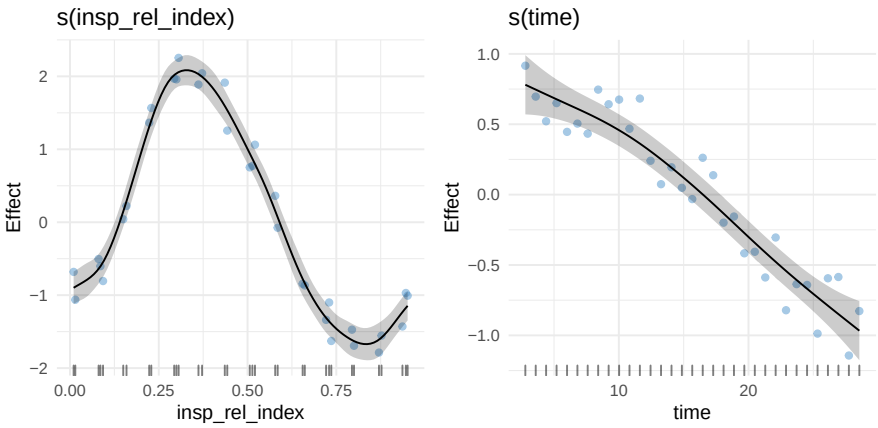


Figure 5: Smooth effects of the pulse pressure GAM.

In addition to the two smooth effects, the model also has a constant/intercept, which is not visualized in the plots above. The intercept is the mean PP.

```
coef(PP_gam)[1]
```

```
## (Intercept)
##    57.59848
```

Predictions (or fit) can be calculated as

$$\hat{PP} = Intercept + s(\text{insp\_rel\_index}) + s(\text{time}).$$

We can use `predict()` to calculate model predictions. If we only pass the GAM, it will make predictions using the observed independent variables we used to fit the model. We can add these predictions as a new column to our original dataset.

> When `predict()` is called on a GAM (an object of class "gam"), the specific method `predict.gam()` is used. See `?predict.gam()` for help.

```
PP_pred <- mutate(PP_data, pred = predict(PP_gam))

ggplot(PP_pred, aes(x=time)) +
  geom_line(aes(y=PP, color = "Observed")) +
  geom_point(aes(y=PP, color = "Observed")) +
  geom_point(aes(y=pred, color = "Predicted"))
```

We can use the `newdata` parameter in `predict()` to interpolate prediction between our observations (it rarely makes sense to extrapolate a spline fit).

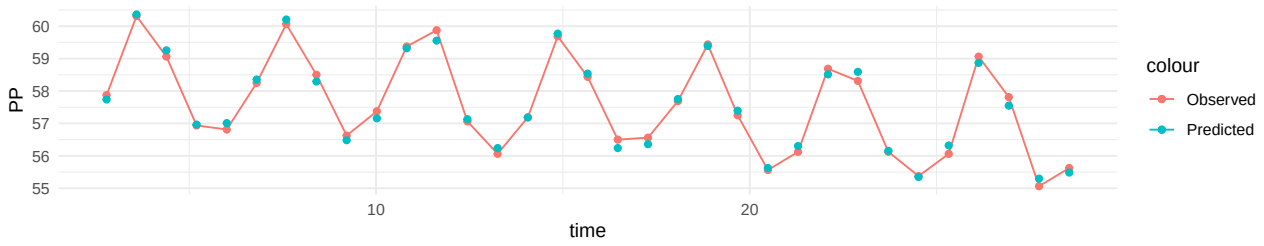


Figure 6: Observed and predicted pulse pressure.

```
PP_newdata <- tibble(
  # create 200 points from 0 to 30 to get a smooth line
  time = seq(0, 30, length.out = 200)) %>%
  # index each new time to our existing vector of inspiration times
  add_time_since_event(sample_pp$insp_start$time, prefix = "insp") %>%
  na.omit()

PP_interpolate <- bind_cols(
  PP_newdata,
  predict(PP_gam,
          newdata = PP_newdata,
          # in addition to the predictions (fit) also return the standard error
          # (se.fit) for each prediction. This makes predict return a named list,
          # that we can simply bind to our data frame.
          se.fit = TRUE)
)

ggplot(PP_interpolate, aes(x=time)) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit,
                  ymax = fit + 1.96*se.fit,
                  fill = "Predicted (95% CI)")) +
  geom_line(aes(y = fit)) +
  geom_point(aes(y=PP, color = "Observed"), data = PP_data) +
  scale_fill_manual(values = "skyblue")
```

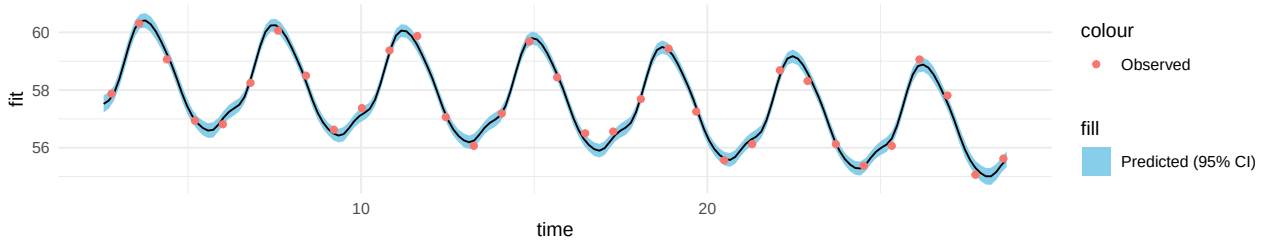


Figure 7: Observed and predicted pulse pressure, including predictions between observations.

Calculate pulse pressure variation

We can calculate pulse pressure variation (PPV) from this model using the formula

$$PPV = \frac{maximum(s(\text{insp\_rel\_index})) - minimum(s(\text{insp\_rel\_index}))}{Intercept}.$$

To find the the extreme of the smooth, we can generate a grid of predictions using only the one smooth term `s(insp_rel_index)`. We could use `predict(type="terms")` but `gratia::smooth_estimates()` conveniently returns the values of the smooth over the original range of the independent variable (here `insp_rel_index`).

```
insp_rel_index_smooth <- smooth_estimates(PP_gam,
                                          smooth = "s(insp_rel_index)",
                                          n=100)
min_PP <- min(insp_rel_index_smooth$est)
max_PP <- max(insp_rel_index_smooth$est)
intercept_PP <- coef(PP_gam)[1]
PPV_est <- (max_PP - min_PP) / intercept_PP

sprintf("PPV is %.1f%%", PPV_est*100)
```

```
## [1] "PPV is 6.5%"
```

This PPV is an estimate from the model. We should also report the uncertainty. `mgcv` lets us sample parameters from the posterior distribution of the model similarly to from a Bayesian model (see Gavin Simpson's (author of `gratia`) answer on StackOverflow for an in-depth example of how this can be done). Here we need posterior samples of a smooth, and, conveniently, `gratia::smooth_samples()` does just that. For each sampled smooth, we can calculate PPV, and use the samples of PPVs to calculate a confidence interval for PPV (this approach neglects that the model intercept is also an estimate, but since the standard error for the intercept is very small, this has negligible effect on the width of the confidence interval).

```
set.seed(1)
insp_smooth_samples <- smooth_samples(PP_gam, term = "s(insp_rel_index)", n = 5000)

ggplot(insp_smooth_samples %>% filter(draw <= 50), aes(.x1, value, group = draw)) +
  geom_line(alpha = 0.1) +
  labs(x="insp_rel_index")
```

```
# A function that returns PPV given a smooth and an intercept
calc_PPV <- function(smooth, intercept) {
  min_PP <- min(smooth)
  max_PP <- max(smooth)
  (max_PP - min_PP) / unname(intercept)
}

PPV_samples <- insp_smooth_samples$value %>%
  split(insp_smooth_samples$draw) %>%
  sapply(calc_PPV, intercept = coef(PP_gam)[1])

PPV_95 <- quantile(PPV_samples, probs = c(0.025, 0.975))

ggplot(data.frame(PPV = PPV_samples), aes(x=PPV)) +
  geom_histogram() +
  geom_vline(aes(xintercept = PPV_est, color = "Estimate"),
             data = data.frame()) +
  geom_vline(aes(xintercept = PPV_95, color = "95% confidence interval"),
             data = data.frame()) +
  scale_x_continuous(labels = scales::label_percent())
```

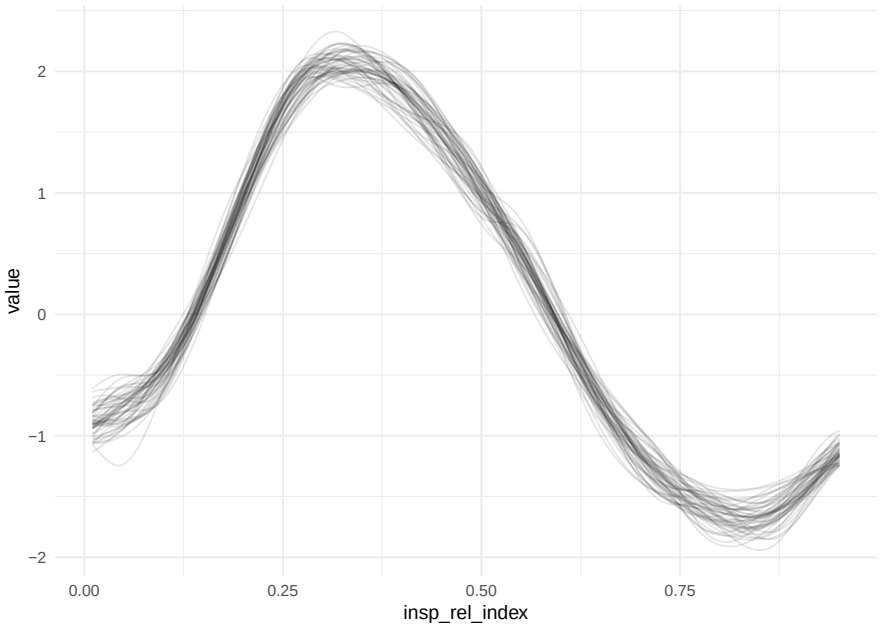


Figure 8: Plot of 50 of the 5000 sampled smooths from the posterior distribution of the respiratory cycle smooth ‘s(insp_rel_inded)‘.

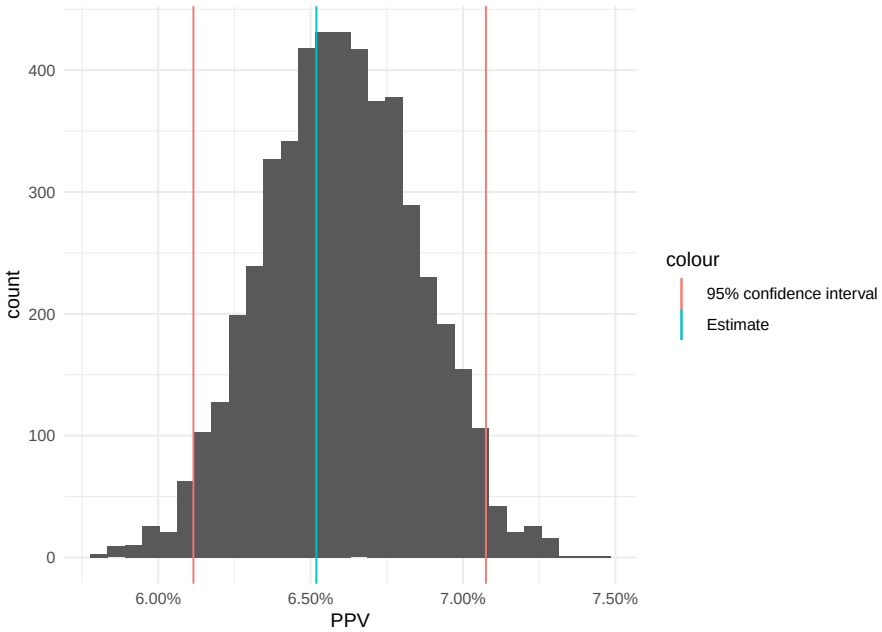


Figure 9: Histogram of PPVs calculated from each of the 5000 sampled smooths from the posterior distribution of the respiratory cycle smooth ‘s(insp_rel_inded)‘.

Central venous pressure

Load data

This example example corresponds to the model shown in Figure 5 in the paper.

We use a section of a recording of central venous pressure (CVP) `sample_cvp$cvp`. The sample data is structured as a list. In addition to CVP, it and also includes the start time of each inspiration `sample_cvp$insp_start` and of each QRS-complex `sample_cvp$qrs` and the interval in which 250 ml fluid is administered (`sample_cvp$fluid_start` to `sample_cvp$fluid_end`).

```
sample_cvp <- readRDS("sample_CVP.RDS")
```

The CVP is stored as a data frame with a time column and a pressure column. The time is in seconds, the sample rate is 125 Hz and the pressure unit is mmHg.

```
head(sample_cvp$cvp)
```

```
## # A tibble: 6 x 2
##      time   CVP
##     <dbl> <dbl>
## 1 0.00432  9.44
## 2 0.0123   9.44
## 3 0.0203   9.56
## 4 0.0283   9.62
## 5 0.0363   9.75
## 6 0.0443   9.88
```

```
plot_cvp_full <- ggplot(sample_cvp$cvp, aes(time, CVP)) +
  annotate("rect", xmin = sample_cvp$fluid_start, xmax = sample_cvp$fluid_end,
           ymin = -Inf, ymax = Inf,
           fill = alpha("blue", 0.4)) +
  annotate("rect", xmin = sample_cvp$fluid_start-30, xmax = sample_cvp$fluid_start,
           ymin = -Inf, ymax = Inf,
           fill = alpha("green", 0.4)) +
  geom_line() +
  labs(title = "Full sample",
       subtitle = "Blue area: administration of 250 ml fluid
Green area: section used to fit GAM")

plot_cvp_short <- ggplot(sample_cvp$cvp, aes(time, CVP)) +
  geom_line() +
  coord_cartesian(xlim = c(sample_cvp$fluid_start-30, sample_cvp$fluid_start)) +
  geom_vline(aes(xintercept = time), color = "red",
             data = sample_cvp$insp_start) +
  geom_vline(aes(xintercept = time), color = "blue",
             data = sample_cvp$qrs) +
  labs(title = "Green area",
       subtitle = "Blue lines: QRS-complexes
Red lines: inspiration start")

plot_cvp_full/plot_cvp_short
```

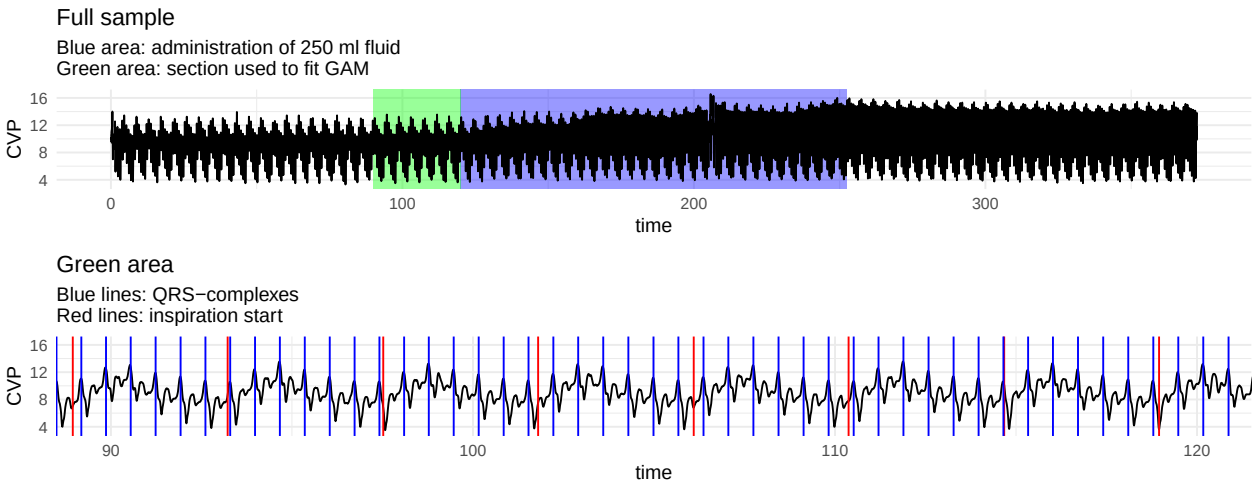


Figure 10: Sample data for CVP model.

We want to fit the model

$$CVP = f(pos_{cardiac}) + f(pos_{ventilation}) + f(pos_{cardiac},\ pos_{ventilation}) + f(t_{total}) + \epsilon.$$

First, we need to calculate each CVP sample's position in both the cardiac and respiratory cycles. For the respiratory cycle, we will fit a cyclic spline based on the relative position (as in the pulse pressure example above). For the cardiac cycle, we cannot simply use a cyclic spline, as the cycles vary in length. We could use a cyclic spline based on the relative position in the cardiac cycle of a CVP sample, but that would assume that the CVP waveform of a long cardiac cycle is simply a linearly stretched version of a short cycle. Instead we assume that the cardiac cycle effect depends on the time since the P-wave (initiation of atrial contraction). Instead of trying to detect P-waves from the ECG, we assume that they appear a constant interval before the QRS complex.

To find the PQ interval, we align 30 seconds of ECG recording by the detected QRS complexes.

```
sample_cvp$ecg %>%
  add_time_since_event(sample_cvp$qrs$time-0.3, prefix = "pre_qrs") %>%
  na.omit() %>%
  filter(time < 30) %>%
  ggplot(aes(pre_qrs_index-0.3, ECG_II, group = pre_qrs_n)) +
  geom_line(alpha = 0.3)
```

We can see that the P-wave starts ~150 ms before the QRS complex.

In this example we will fit a GAM to the last 30 seconds before fluid administration starts. We filter the CVP data to the relevant interval and add each sample's position in both the cardiac cycle (starting at the P-wave) and the respiratory cycle.

```
PQ_interval <- 0.150 #seconds

cvp_df <- sample_cvp$cvp %>%
  filter(between(time, sample_cvp$fluid_start-30, sample_cvp$fluid_start)) %>%
  # QRS time - PQ interval = P wave time.
  add_time_since_event(sample_cvp$qrs$time - PQ_interval, prefix = "cardiac") %>%
  add_time_since_event(sample_cvp$insp_start$time, prefix = "insp")
```

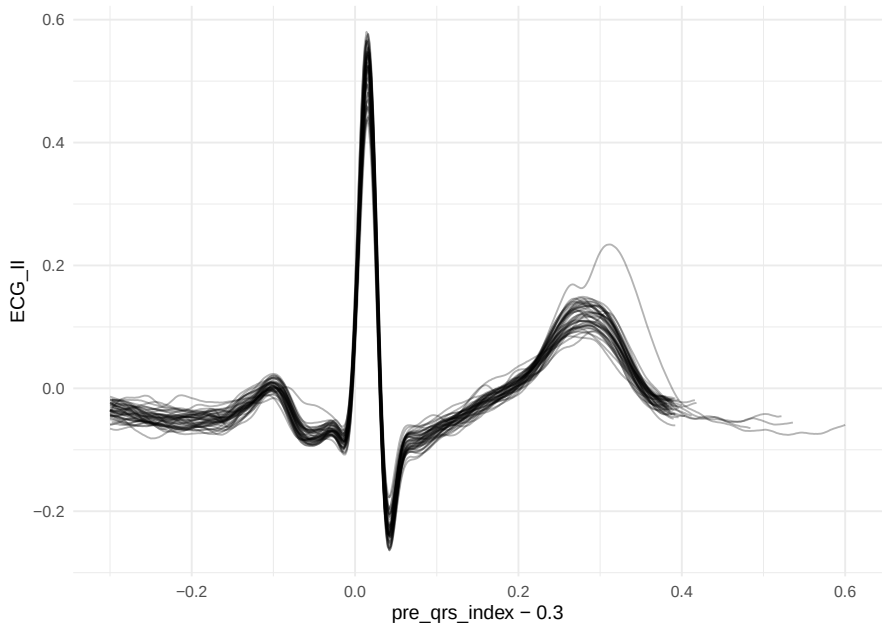


Figure 11: ECG recording (30 seconds) aligned by QRS complexes.

```
head(cvp_df)
```

```
## # A tibble: 6 x 10
##    time   CVP cardiac_index cardiac_n cardiac_cycle_len cardiac_rel_index
##   <dbl> <dbl>         <dbl>     <int>             <dbl>             <dbl>
## 1  90.0  8.56         0.290       130             0.682             0.425
## 2  90.0  8.12         0.298       130             0.682             0.437
## 3  90.0  7.81         0.306       130             0.682             0.449
## 4  90.0  7.75         0.314       130             0.682             0.460
## 5  90.0  7.88         0.322       130             0.682             0.472
## 6  90.0  8.12         0.330       130             0.682             0.484
## # ... with 4 more variables: insp_index <dbl>, insp_n <int>,
## #   insp_cycle_len <dbl>, insp_rel_index <dbl>
```

Before fitting the model, we can visualize (individually) the three effects we subsequently want to model (collectively): continuous time, respiratory cycle and heart cycle.

```
cvp_time <- ggplot(cvp_df, aes(time, CVP)) +
  geom_line() +

  labs(title="Continuous time")

cvp_insp <- ggplot(cvp_df, aes(insp_rel_index, CVP, group = insp_n)) +
  geom_line() +
  labs(title="Respiratory cycle (relative)")

cvp_p <- ggplot(cvp_df, aes(cardiac_index, CVP, group = cardiac_n)) +
  geom_line() +
  labs(title="Cardiac cycle (seconds since P wave")
```

```
cvp_time / (cvp_insp + cvp_p)
```

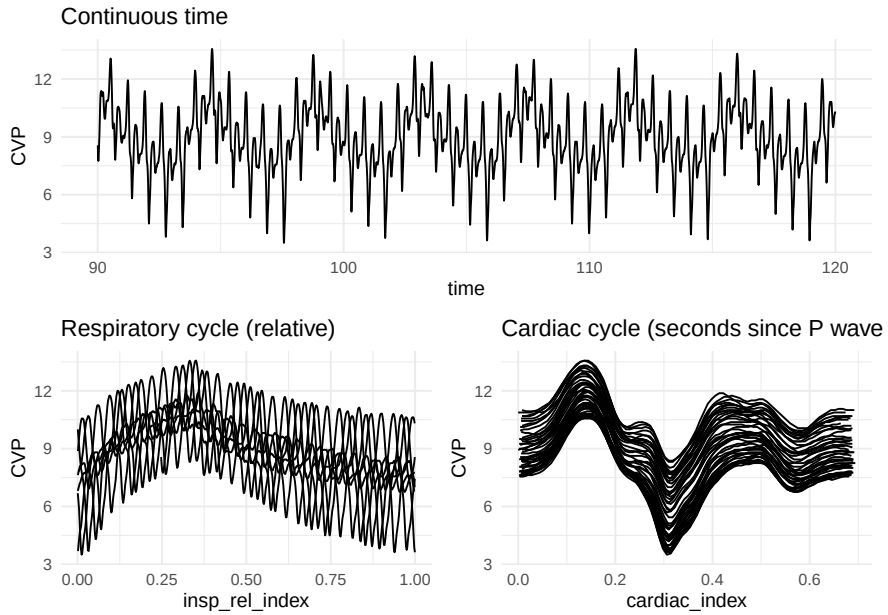


Figure 12: Different visualizations of the data used for the CVP model.

Fit and visualize the GAM

Now, we are ready to fit the model. This time we use `bam()` which is like `gam()`, but optimized for large datasets. First we fit the model without correcting for autocorrelation of the residuals. The next section will demonstrate how to correct for autocorrelation.

```
gam_cvp <- bam(
      CVP ~
        s(cardiac_index, bs = "cr", k = 40) +
        # The respiratory cycle effect is not really that smooth, as it has a sharp
        # drop at start-expiration. There are different ways to allow this, including
        # fitting separate smooths for inspiration and expiration or using and adaptive
        # smooth (bs = "ad").
        # Here we simply add plenty of knots.
        s(insp_rel_index, bs = "cc", k = 30) +
        # We create the interaction smooth with ti() instead of te() because the main
        # effects, s(cardiac_index) and s(insp_rel_index), are also present in the model.
        ti(
          cardiac_index,
          insp_rel_index,
          bs = c("cr", "cc"),
          k = c(40, 30),
          sp = c(0.2, 2) # 40 x 30 knots makes a highly flexible plane,
          # that will often overfit the data. By fixing the smoothing parameters
          # we can force the interaction effect to be more smooth.
        ) +
        s(time, bs = "cr"), # If the detrending smooth captures some of the
```

```
        # other effects (respiratory or cardiac) it may be necessary to set a high
        # fixed smoothing parameter.
        knots = list(insp_rel_index = c(0, 1)),
        method = "fREML",
        data = cvp_df
    )
```

Visualize the smooth effects of the model.

```
gratia::draw(gam_cvp, residuals = TRUE, rug = FALSE)
```

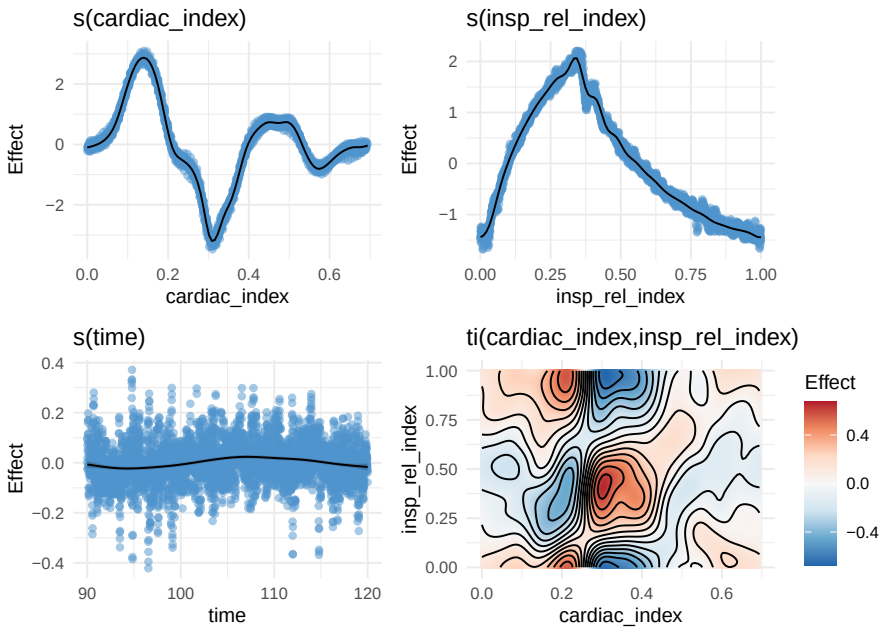


Figure 13: Smooth effects of the CVP GAM.

The model intercept, the mean CVP, is not visualized in the plots above.

```
coef(gam_cvp)[1]
```

```
## (Intercept)
##     8.95121
```

We can visualize how well our model fits our observed CVP.

```
cvp_df_pred <- mutate(cvp_df,
                      pred = predict(gam_cvp),
                      resid = resid(gam_cvp))

cvp_df_pred %>%
  ggplot(aes(time, CVP)) +
  geom_line() +
  geom_line(aes(y = pred), color = "red") +
  geom_line(aes(y = resid), color = "orange")
```

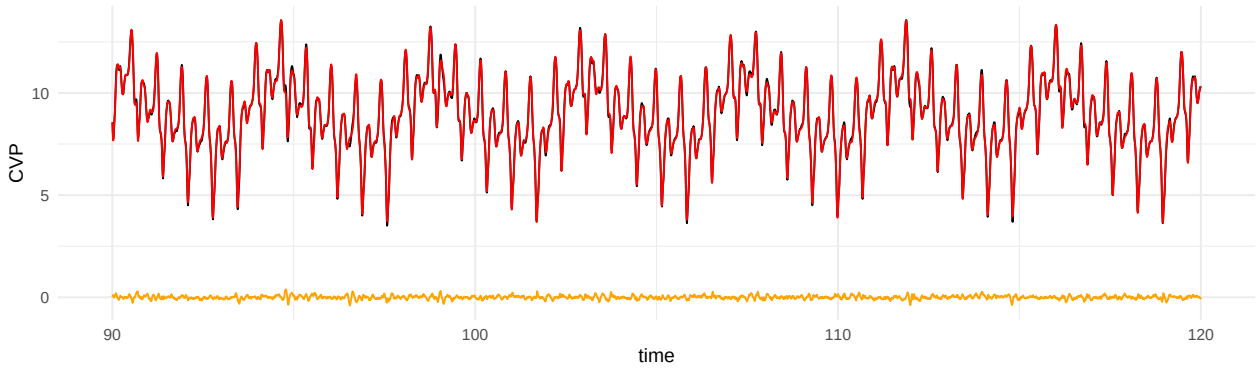


Figure 14: Observed, predicted and residual CVP

In the paper's Figure 6 the fit is visualized as a singe contour plot that sums all the effects except detrending (continuous time). Here is a similar visualization.

```
# Create a new dataset containing a grid of
# `cardiac_index` and `insp_rel_index` to generate predictions for.

# Use a grid with a resolution of 100x100.
cvp_contour_data <- expand.grid(
  # The longest
  cardiac_index = seq(0, 0.7, length.out = 100),
  insp_rel_index = seq(0, 1, length.out = 100),
  time = 999) # We will not use `time` in our prediction, but predict.gam()
              # expects the variable to be present.


cvp_contour_data$CVP_pred <- predict(gam_cvp,
                                     newdata = cvp_contour_data,
                                     exclude = "s(time)")

ggplot(cvp_contour_data, aes(x = cardiac_index, y=insp_rel_index, z = CVP_pred)) +
  geom_contour_filled(binwidth = 1) +
  guides(fill = guide_colorsteps(barheight = 15, title = "CVP"))
```

Autocorrelated residuals

One assumption of a GAM is that the residuals are independent. This will rarely be fulfilled when modelling high resolution waveforms. We can investigate the correlation of the residuals using `acf()`.

```
auto_corr_gam_cvp <- acf(residuals(gam_cvp))
```

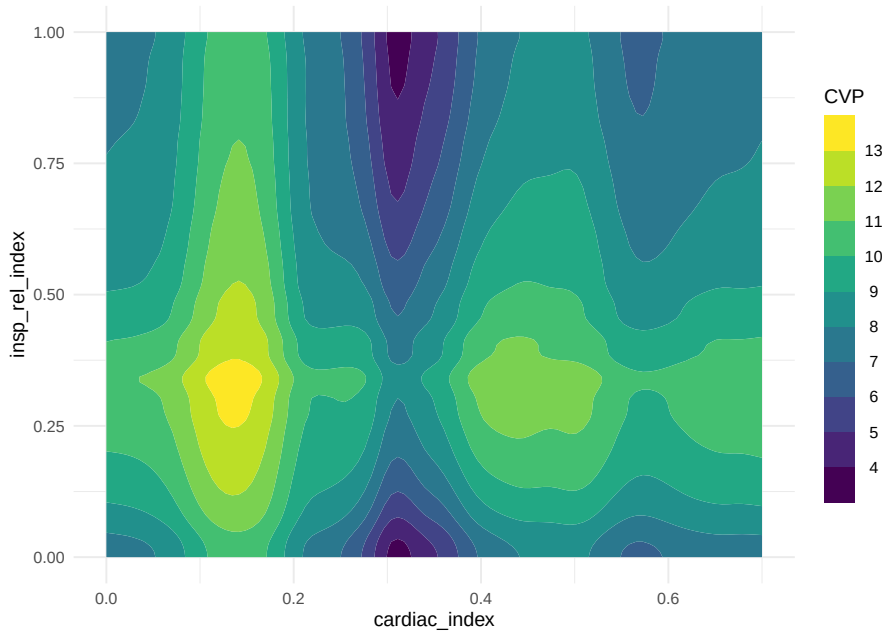
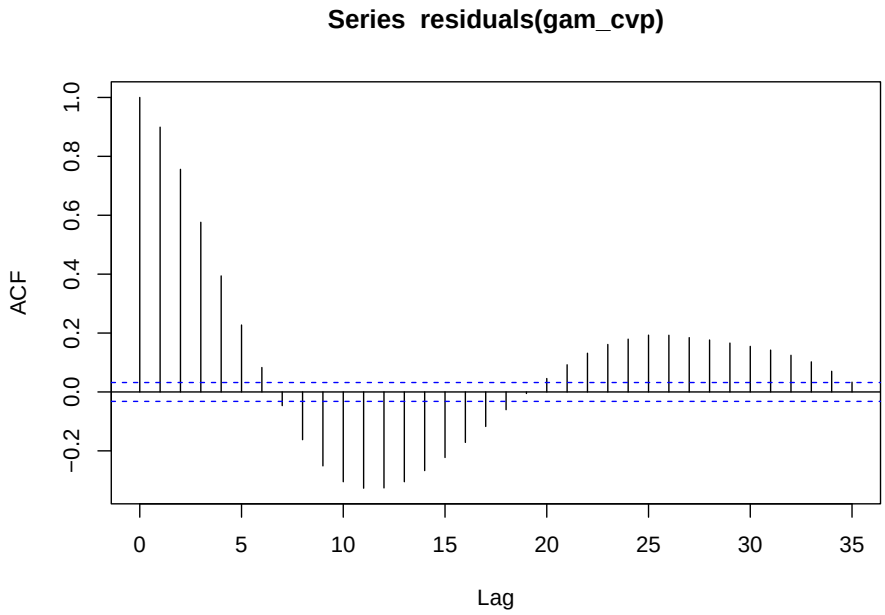


Figure 15: Combined visualization of the CVP GAM, excluding the detrending smooth.



This plot shows the correlation of a residual with the residual 0 to 35 points later. Zero points later correspond to the residual itself, and will always be 1. `bam()` allows modelling the residual errors as an AR(1) model (each residual (ϵ_t) is some proportion (rho) of the previous residual + random error: $\epsilon_t = rho * epsilon_{t-1} + w_t, w_t \sim normal(0, \sigma)$)). `bam()` allows us to specify a fixed AR(1) correlation coefficient with the `rho` parameter, but we have to tune it manually. A good guess is the first-order autocorrelation from a model not accounting for autocorrelation (here `gam_cvp`):

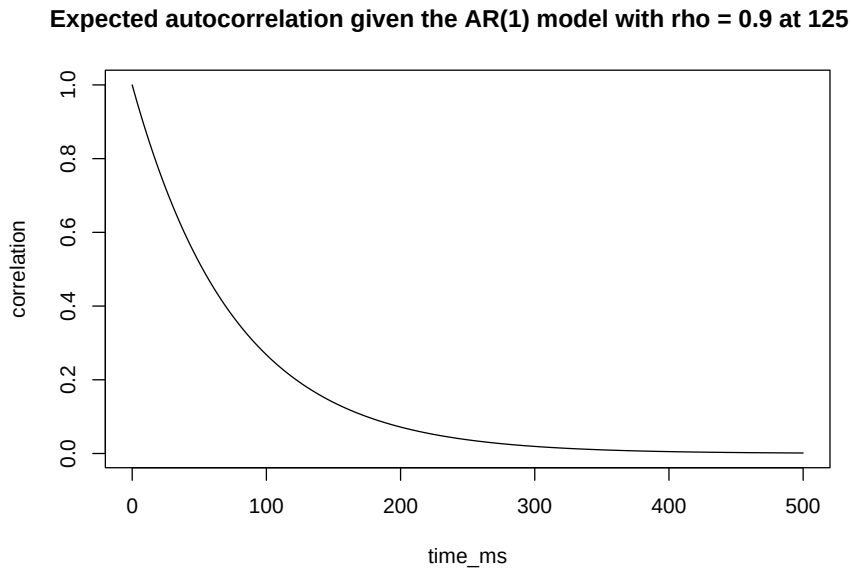
```
auto_corr_gam_cvp$acf[2] # $acf[1] is the unlagged correlation (= 1)
```

```
## [1] 0.8990999
```

The AR(1) model implies that the correlation between residuals drops exponentially with the distance between them. Our CVP waveform has a sample rate of 125 Hz, so a correlation coefficient of 0.9 per sample corresponds to the following relation between correlation and time:

```
time_ms <- 0:500
correlation <- 0.9^((time_ms/1000)*125)

plot(time_ms, correlation, type = "l",
     main = "Expected autocorrelation given the AR(1) model with rho = 0.9 at 125 Hz")
```

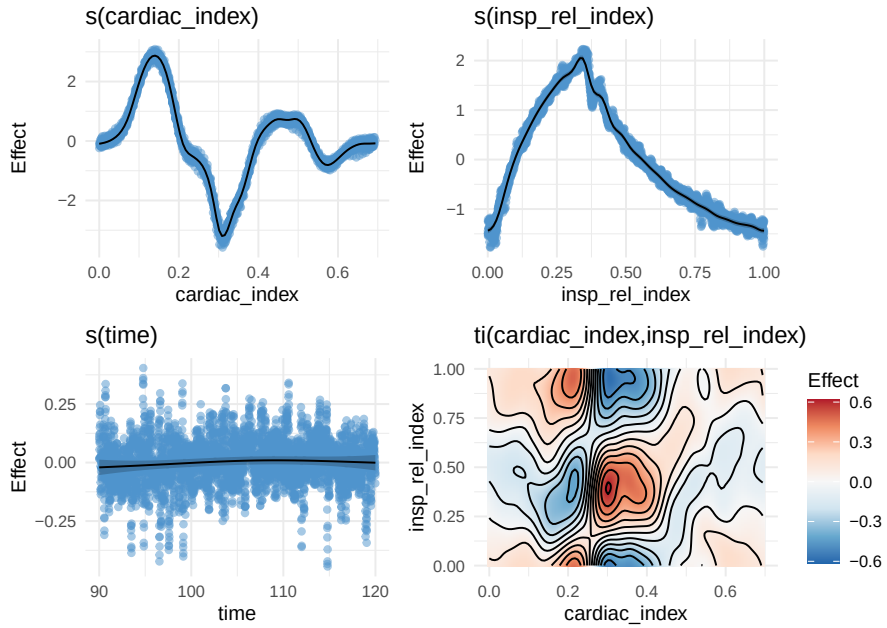


With this model, we expect very little correlation between residuals more than 200 ms apart.

We can use this to fit a GAM with autocorrelated residuals as suggested by van Rij et al, 2019 (https://doi.org/10.1177/2331216519832483).

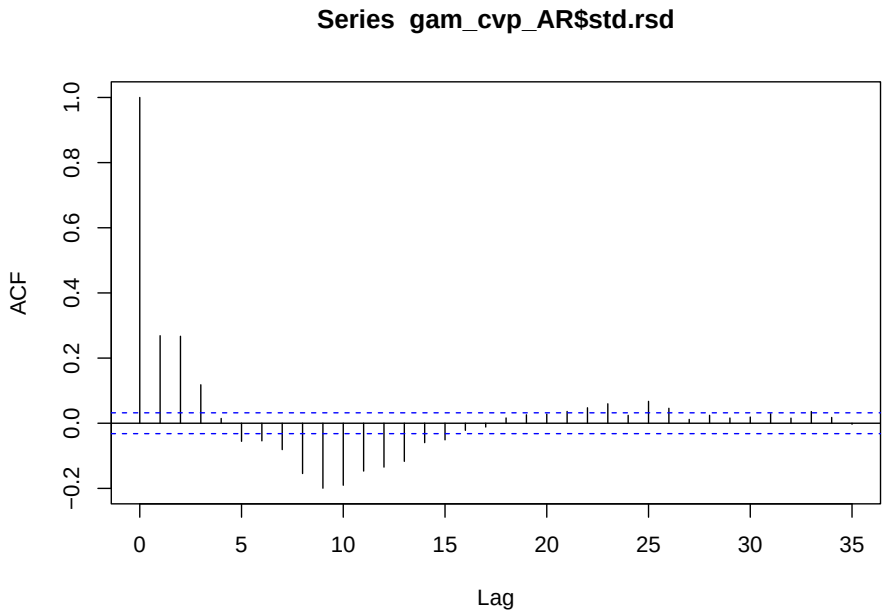
```
# The model i identical to gam_cvp except for the rho parameter.
gam_cvp_AR <- bam(
      CVP ~
        s(cardiac_index, bs = "cr", k = 40) +
        s(insp_rel_index, bs = "cc", k = 30) +
        ti(
          cardiac_index,
          insp_rel_index,
          bs = c("cr", "cc"),
          k = c(40, 30),
          sp = c(0.2, 2)
        ) +
        s(time, bs = "cr"),
        knots = list(insp_rel_index = c(0, 1)),
        method = "fREML",
        rho = 0.9, # correlation coefficient for AR(1) model of the residuals
        data = cvp_df
    )
```

```
gratia::draw(gam_cvp_AR, residual = TRUE, rug = FALSE)
```



The autocorrelation-corrected residuals are stored in `gam_cvp_AR$std.rsd`. Again, we can use `acf()` to visualize the residual autocorrelation.

```
acf(gam_cvp_AR$std.rsd)
```



We can see that this has markedly reduced the autocorrelation in the residuals. The rho chosen with this approach is not guaranteed to be optimal. It is possible to search for the value of rho minimizing the REML-score, but this is computationally expensive as it requires refitting the model for each rho. For detail about estimating autocorrelation of residuals, see Simpson, 2018 (https://doi.org/10.3389/fevo.2018.00149).

Other common challenges

The CVP waveform in the window 30 second earlier is a bit more noisy. We will use that section of data to demonstrate a few common challenges.

```
cvp_df_noise <- sample_cvp$cvp %>%
  filter(between(time, sample_cvp$fluid_start-60, sample_cvp$fluid_start-30)) %>%
  # QRS time - PQ interval = P wave time.
  add_time_since_event(sample_cvp$qrs$time - PQ_interval, prefix = "cardiac") %>%
  add_time_since_event(sample_cvp$insp_start$time, prefix = "insp")


cvp_time <- ggplot(cvp_df_noise, aes(time, CVP)) +
  geom_line()

cvp_insp <- ggplot(cvp_df_noise, aes(insp_rel_index, CVP, group = insp_n)) +
  geom_line()

cvp_p <- ggplot(cvp_df_noise, aes(cardiac_index, CVP, group = cardiac_n)) +
  geom_line()

cvp_time / (cvp_insp + cvp_p)
```

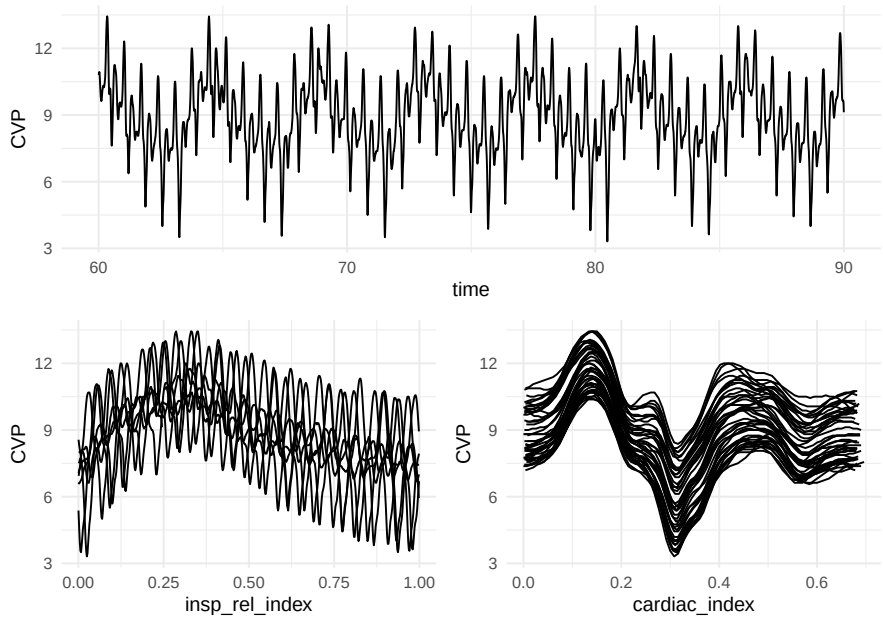


Figure 16: Different visualizations of the noisy data used for the second CVP model.

In the first iteration, we will use fewer knots in the cardiac smooth.

```
gam_cvp_noise1 <- bam(
      CVP ~
        s(cardiac_index, bs = "cr", k = 15) + # only 15 knots
        s(insp_rel_index, bs = "cc", k = 30) +
        #
        ti(
```

```
          cardiac_index,
          insp_rel_index,
          bs = c("cr", "cc"),
          k = c(15, 30) # only 15 knots in the cardiac dimension
        ) +
        s(time, bs = "cr"),
        knots = list(insp_rel_index = c(0, 1)),
        method = "fREML",
        data = cvp_df_noise
    )
```

```
gratia::draw(gam_cvp_noise1, residuals = TRUE, rug = FALSE)
```

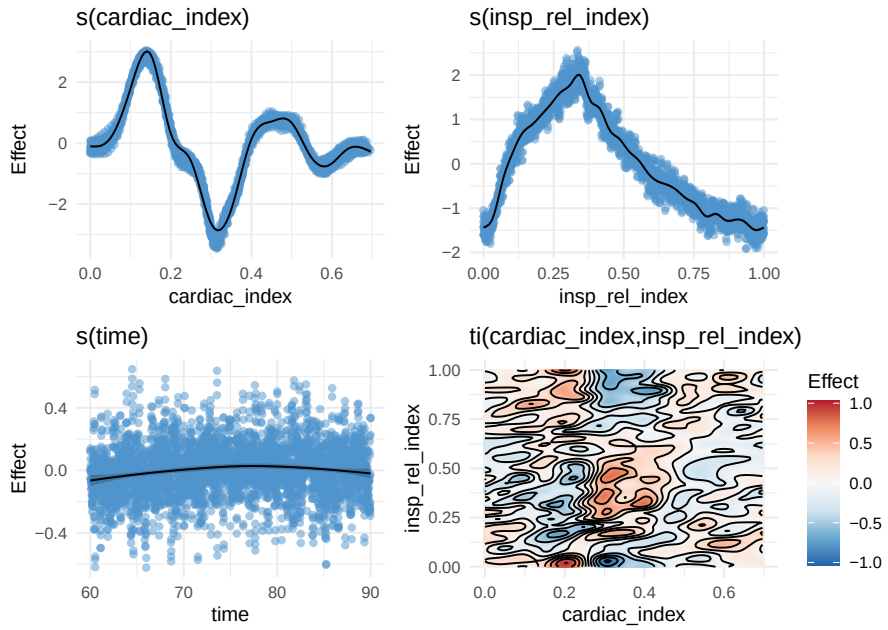
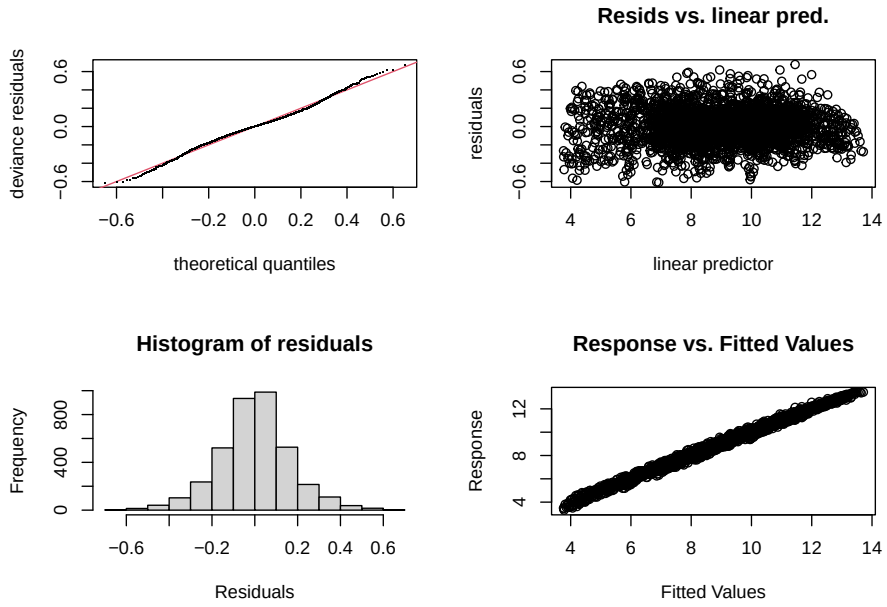


Figure 17: Smooth effects of the noisy CVP GAM.

Here, the interaction smooth clearly overfits the data, and it looks like the cardiac smooth is not flexible enough to match the sharp 'v' peak (the most negative in this sample).

Optimally, the shape of a spline should be limited by the wiggliness penalty (smoothing parameter) and not the number of knots. We can use `gam.check` the see if any splines are limited by their number of knots.

```
old_par = par(mfrow = c(2,2))
gam.check(gam_cvp_noise1)
```




```
##
## Method: fREML   Optimizer: perf newton
## full convergence after 8 iterations.
## Gradient range [-1.973604e-10,1.961613e-10]
## (score -489.8945 & scale 0.03182452).
## Hessian positive definite, eigenvalue range [0.7907991,1880.619].
## Model rank =  444 / 444
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                                    k'    edf k-index p-value
## s(cardiac_index)                14.00  13.99    0.50  <2e-16 ***
## s(insp_rel_index)               28.00  26.74    1.04    0.99
## ti(cardiac_index,insp_rel_index) 392.00 319.46    0.41  <2e-16 ***
## s(time)                           9.00   2.57    0.15  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
par(old_par) # reset par
```

In this table, `k'` is the maximum possible degrees of freedom (d.f.) of the smooth. This is one less than the number knots as one d.f. is used to constrain the spline to have a sum of zero (see `?mgcv::identifiability`). For cyclic splines it is two less than `k`, as the two limiting knots are effectively a single knot. `edf` is the effective degrees of freedom after the smoothing penalty. A `k-index` < 1 indicates there is some residual pattern that is not contained in the smooth. If `k-index` is low and `edf` is close to `k'`, the smooth probably has too few knots. On the other hand, too many knots are not a problem (except for the higher computational cost).

We increase `k` for the cardiac smooth and the cardiac dimension in the interaction. To reduce overfitting from the interaction term, we will add a fixed smoothing parameter.

```
gam_cvp_noise2 <- bam(
      CVP ~
        s(cardiac_index, bs = "cr", k = 40) +
        s(insp_rel_index, bs = "cc", k = 30) +
        #
        ti(
          cardiac_index,
          insp_rel_index,
          bs = c("cr", "cc"),
          k = c(40, 30),
          sp = c(0.2, 2) # fixed smoothing parameters
        ) +
        s(time, bs = "cr"),
        knots = list(insp_rel_index = c(0, 1)),
        method = "fREML",
        data = cvp_df_noise
    )
```

```
gratia::draw(gam_cvp_noise2, residuals = TRUE, rug = FALSE)
```

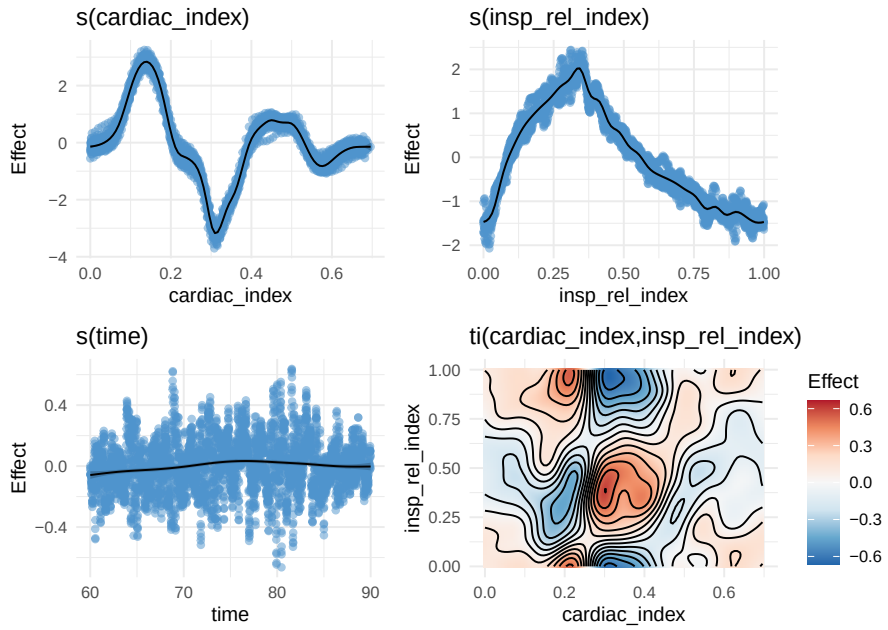
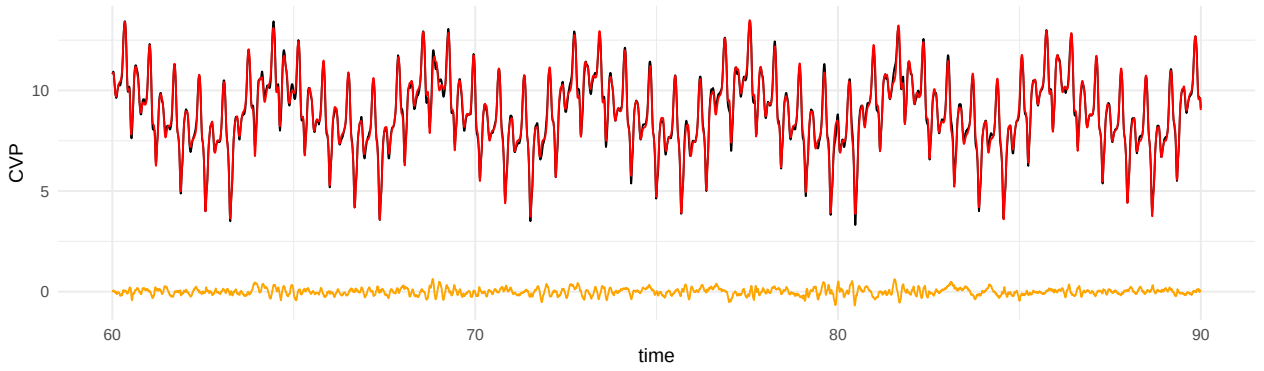


Figure 18: Smooth effects of new version the noisy CVP GAM. This time with more knots in the cardiac smooth and fixed smoothing parameters of the interaction smooth.

```
cvp_df_noise_pred2 <- mutate(cvp_df_noise,
                             pred = predict(gam_cvp_noise2),
                             resid = resid(gam_cvp_noise2))

ggplot(cvp_df_noise_pred2, aes(time, CVP)) +
  geom_line() +
  geom_line(aes(y = pred), color = "red")+
  geom_line(aes(y = resid), color = "orange")
```



The residuals seem to be mostly random. There is slight repeating pattern: an increase in the residuals one third into each respiratory cycle (also visible in the plot of the `s(insp_rel_index)` smooth). This corresponds to the closing of the ventilator solenoid valve at end-inspiration. The sudden drop in pressure makes the ventilator tubing move and disturb the adjacent CVP line.

Session info

Rendered: 2022-02-14 13:26:42

```
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Pop!_OS 21.10
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_DK.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_DK.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_DK.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_DK.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] jenevoldsen_0.1.0        assertthat_0.2.1         broom_0.7.11
##  [4] waveformtools_0.0.0.9000 forcats_0.5.1            stringr_1.4.0
##  [7] readr_2.1.1              tidyr_1.1.4              tibble_3.1.6
## [10] tidyverse_1.3.1          purrr_0.3.4              accelerometry_3.1.2
## [13] patchwork_1.1.1          ggplot2_3.3.5            dplyr_1.0.7
## [16] gratia_0.7.0             mgcv_1.8-36              nlme_3.1-152
##
## loaded via a namespace (and not attached):
```

```
##  [1] httr_1.4.2         viridisLite_0.4.0 jsonlite_1.7.2     splines_4.1.0
##  [5] here_1.0.1         modelr_0.1.8      cellranger_1.1.0   yaml_2.2.1
##  [9] dvmisc_1.1.4       pillar_1.6.4      backports_1.4.1    lattice_0.20-44
## [13] glue_1.6.1         digest_0.6.29     RColorBrewer_1.1-2 rvest_1.0.2
## [17] colorspace_2.0-2   htmltools_0.5.2   Matrix_1.3-4       survey_4.0
## [21] pkgconfig_2.0.3    haven_2.4.3       xtable_1.8-4       mvtnorm_1.1-2
## [25] scales_1.1.1       tzdb_0.2.0        generics_0.1.1     farver_2.1.0
## [29] ellipsis_0.3.2     withr_2.4.3       cli_3.1.1          survival_3.2-11
## [33] magrittr_2.0.1     crayon_1.4.2      mvnfast_0.2.7      readxl_1.3.1
## [37] evaluate_0.14      fs_1.5.2          fansi_1.0.2        MASS_7.3-54
## [41] xml2_1.3.2         tools_4.1.0       data.table_1.14.0  hms_1.1.1
## [45] mitools_2.4        lifecycle_1.0.1   munsell_0.5.0      reprex_2.0.0
## [49] isoband_0.2.5      compiler_4.1.0    rlang_0.4.12       grid_4.1.0
## [53] rstudioapi_0.13    labeling_0.4.2    rmarkdown_2.11.3   codetools_0.2-18
## [57] gtable_0.3.0       DBI_1.1.1         R6_2.5.1           lubridate_1.8.0
## [61] knitr_1.37         fastmap_1.1.0     utf8_1.2.2         rprojroot_2.0.2
## [65] stringi_1.7.6      Rcpp_1.0.7        vctrs_0.3.8        rbenchmark_1.0.0
## [69] tab_4.1.1          dbplyr_2.1.1      tidyselect_1.1.1   xfun_0.29
```