R-Code for the Essay 'A compartative simulation study of the packages `mgcv` and `gamlss` in the case of location-scale modelling with zero-inflated count data'

Alexander Piehler

3/23/2020

Introduction

This is a simulation study comparing the performance of the packages `mgcv` and `gamlss` with respect to their capabilities when dealing with modelling the distribution parameters of a Zero Inflated Poisson distribution in a regression context.

This Markdown-document is meant to supplement the essay 'A compartative simulation study the of the packages `mgcv` and `gamlss` when dealing with location-scale modelling with Zero-Inflated Count Data'.

This documents shows the functions of and computations for the simulation. It also shows the plots that were used in said essay.

Setup

We start off by loading the required packages.

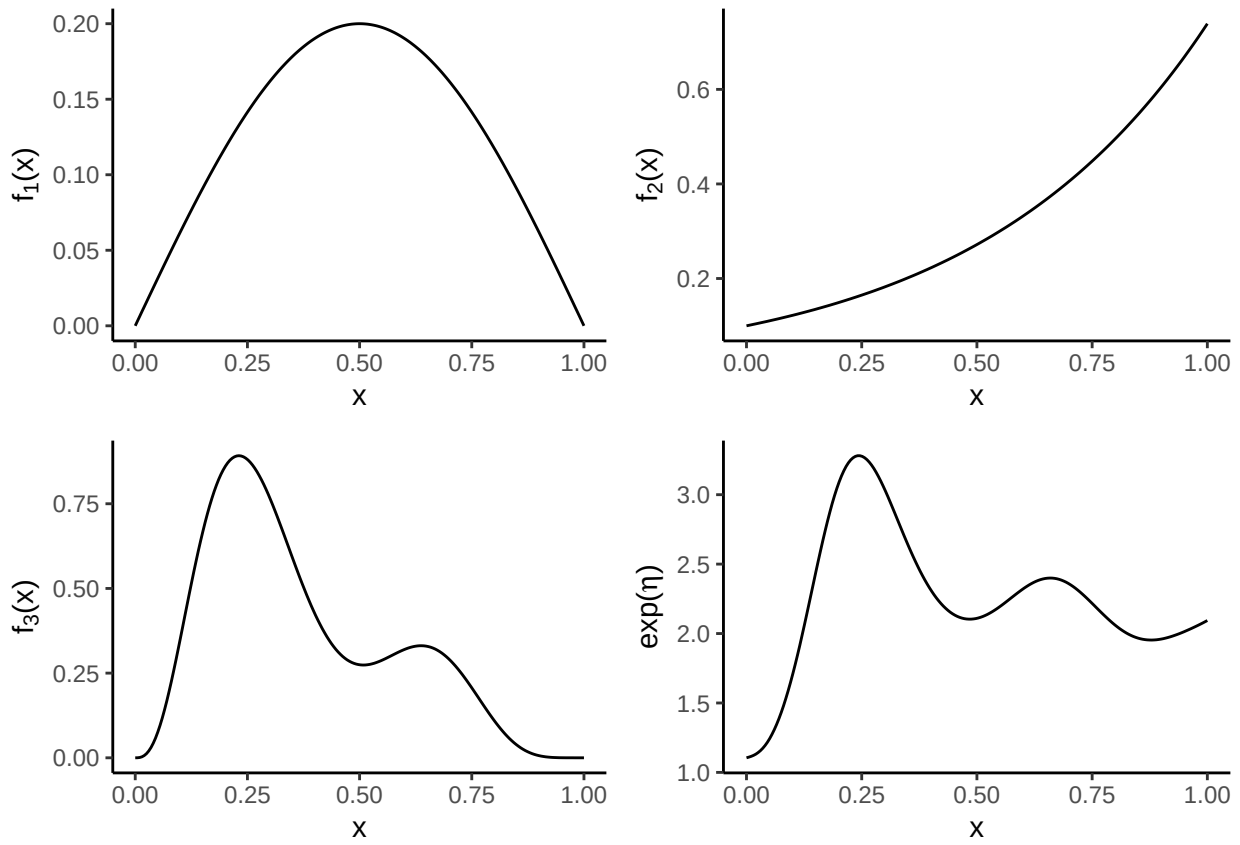
```
# a bunch of required packages
packages_required <- c("tidyverse",
                       "mgcv",
                       "gamlss",
                       "mvtnorm",
                       "gridExtra",
                       "furrr",
                       "latex2exp",
                       "gridExtra",
                       "xtable")

# check which ones are not installed
not_installed <- packages_required[!packages_required %in%
    installed.packages()[, "Package"]]

# install them
if (length(not_installed) > 0) {
  lapply(
    not_installed,
    install.packages,
    repos = "http://cran.us.r-project.org",
    dependencies = TRUE
  )
}

# load them
sapply(packages_required, library, character.only = TRUE, quietly = TRUE)
```

These functions constitute the true underlying nonlinear functions. These functions are the same one used in Wood, Pya, and Säfken (2017). A scaling factor was added to reduce the maximum value of the function. If not, these functions would yield `NaN` values when used in the linear predictor of λ due to integer overflow.

```
func_1 <- function(x, scaling = 0.10) {
  2 * sin(pi * x) * scaling
}

func_2 <- function(x, scaling = 0.10) {
  exp(2 * x) * scaling
}

func_3 <- function(x, scaling = 0.10) {
  (0.2 * x ^ 11 * (10 * (1 - x)) ^ 6 + 10 * (10 * x) ^ 3 * (1 - x) ^ 10 ) *
    scaling
}
```

The following plot shows the true form of $log(\lambda) = 0.10(f_1(x_1) + f_2(2) + f_3(3))$

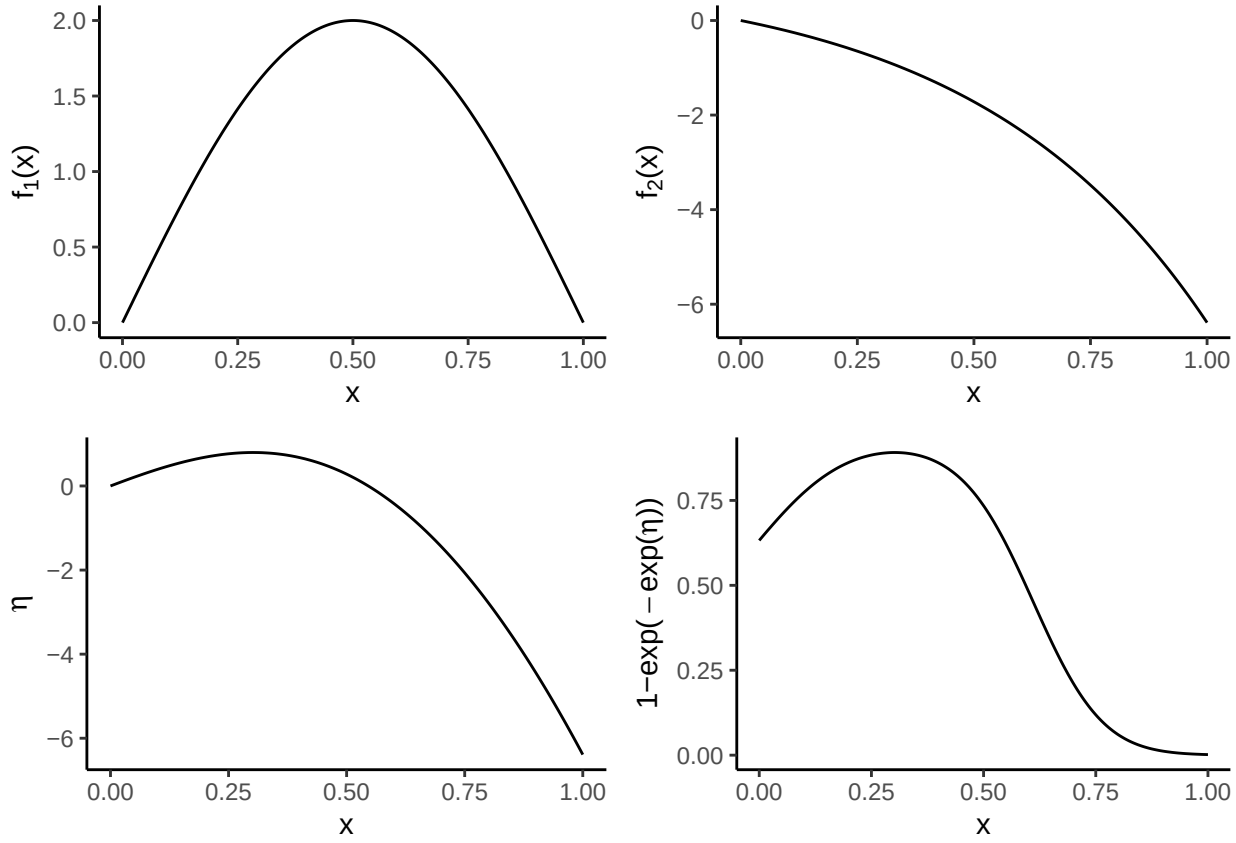
```
x <- seq(from = 0,
         to = 1,
         length.out = 200)

y_1 <- func_1(x)

y_2 <- func_2(x)

y_3 <- func_3(x)

y_all <- func_1(x) + func_2(x) + func_3(x)

p_a <-
  qplot(x, y_1, geom = "line") +
  theme_classic() +
  ylab(TeX("$f_1(x)$"))
p_b <-
  qplot(x, y_2, geom = "line") +
  theme_classic() +
  ylab(TeX("$f_2(x)$"))
p_c <-
  qplot(x, y_3, geom = "line") +
  theme_classic() +
  ylab(TeX("$f_3(x)$"))
p_d <-
  qplot(x, exp(y_all), geom = "line") + theme_classic() +
  ylab(TeX("$\\exp(\\eta)$"))

functions_lambda <- arrangeGrob(p_a, p_b, p_c, p_d,
                                ncol = 2)

functions_lambda <- grid.arrange(functions_lambda)
```



These plots show the true function $log(-log(1-\pi)) = 1 + 0.7(f_1(x_1) - f_2(2))$ with with complementary-log-log-link.

```
x <- seq(from = 0,
         to = 1,
         length.out = 200)

y_1 <- func_1(x, scaling = 1)

y_2 <- -func_2(x, scaling = 1) + 1

y_all <- func_1(x, scaling = 1) - func_2(x, scaling = 1) + 1

p_a <-
  qplot(x, y_1, geom = "line") +
  theme_classic() +
  ylab(TeX("$f_1(x)$"))
p_b <-
  qplot(x, y_2, geom = "line") +
  theme_classic() +
  ylab(TeX("$f_2(x)$"))
p_c <-
  qplot(x, y_all, geom = "line") +
  theme_classic() +
  ylab(TeX("$\\eta$"))
p_d <-
  qplot(x, binomial(link = "cloglog")$linkinv(y_all), geom = "line") +
```

```
  theme_classic() +
  ylab(TeX("$1-\\exp(-\\exp(\\eta))$"))

functions_pi <- arrangeGrob(p_a, p_b, p_c, p_d,
                            ncol = 2)

functions_pi <- grid.arrange(functions_pi)
```



This function generates a list with 4 independent variables. It can be chosen if the variables should be correlated.

```
### GENERATE RANDOM VARIABLES
# input: n
# output: list with 4 independent variables

generate_independent <-
  function(n,
           correlated = FALSE,
           correlation = NULL) {
    if (correlated) {
      correlation_matrix <- matrix(correlation, nrow = 4, ncol = 4)
      diag(correlation_matrix) <- 1

      correlated_vars <- rmvnorm(n,
                                 mean = rep(0, 4),
                                 sigma = correlation_matrix)
```

```
    correlated_unif_vars <- pnorm(correlated_vars)

  } else {
    correlated_unif_vars <- matrix(runif(n * 4), ncol = 4)
  }

  var_list <- asplit(correlated_unif_vars, 2)
  names(var_list) <- paste0("x", 1:4)

  var_list
}
```

This function takes the independent variables generated by `generate_independent()` and outputs a list with the linear predictor η_π and η_λ as well as a dataframe containing the independent variables and the simulated zero-inflated count variable.

```
### GENERATE ZERO INFLATED POISSON
# input: list of independent variables
# output: linear predictor of pi, linear predictor of lambda, dataframe with vars

generate_zip <- function(ind_var,
                         noise_level_lambda = 1,
                         noise_level_pi = 1,
                         zero_inf_control = 1,
                         scaling_lambda = 0.15,
                         scaling_pi = 0.7) {
  ## Simulate probability of potential presence
  eta1 <-
    (func_1(ind_var$x1, scaling = scaling_pi) -
       func_2(ind_var$x2, scaling = scaling_pi) +
       zero_inf_control) * noise_level_pi

  p <- binomial(link = "cloglog")$linkinv(eta1)
  y <- as.numeric(runif(length(eta1)) < p) ## 1 for presence, 0 for absence

  ind <- y > 0
  eta2 <-
    (
      func_2(ind_var$x1, scaling = scaling_lambda) +
        func_2(ind_var$x2, scaling = scaling_lambda) +
        func_3(ind_var$x3, scaling = scaling_lambda)
    ) * noise_level_lambda
  lambda <- poisson()$linkinv(eta2)
  y[ind] <- rpois(lambda[ind], lambda[ind])

  df <- data.frame(
    y,
    x1 = ind_var$x1,
    x2 = ind_var$x2,
    x3 =  ind_var$x3,
    x4 = ind_var$x4,
    prob_of_presence = p
  )
```

```
  list(
    "eta_pi" = eta1,
    "eta_lambda" = eta2,
    "df" = df
  )
}
```

The scaling factors with whom the linear predictor will be multiplied in order to obtain different noise levels, are chosen in such a way that the resulting approximate r^2 are: 0.5 for high noise, 0.63 for medium noise, and 0.8 for low noise.

```
set.seed(1991)

x_for_plot <- generate_independent(200)

noise_lvl_lam <- c(1, 3, 1.5, 1.25)

various_noise_levels_lam <- vector(mode = "list", length = length(noise_lvl_lam))

for (i in seq_along(noise_lvl_lam)) {
  various_noise_levels_lam[[i]] <- generate_zip(x_for_plot,
                                                noise_level_lambda = noise_lvl_lam[i],
                                                noise_level_pi = 1,
                                                scaling_lambda = 0.15,
                                                scaling_pi = 0.7,
                                                zero_inf_control = 1)

}

r2_lam <- NULL

for (i in 2:length(various_noise_levels_lam)) {
  SSE <- sum((various_noise_levels_lam[[1]]$eta_lambda -
                various_noise_levels_lam[[i]]$eta_lambda)^2)

  SSR <- sum((mean(various_noise_levels_lam[[1]]$eta_lambda) -
                various_noise_levels_lam[[i]]$eta_lambda)^2)

  SST <- SSE + SSR

  r2_lam <- c(r2_lam, SSR/SST)

}

noise_level_lambda_curve <- data.frame(scaling_factor = noise_lvl_lam[-1],
                                       r2 = r2_lam)
noise_level_lambda_curve
```

```
##   scaling_factor        r2
## 1           3.00 0.5378143
## 2           1.50 0.6718349
## 3           1.25 0.8055133
```

For the linear predictor of π the scaling factors for the respective noise levels are chosen in such a way that the approximate r^2 is 0.6 for high noise, 0.75 for medium noise and 0.9 for low noise.

```r
noise_lvl_pi <- c(1, 6.0, 2.2, 1.4)

various_noise_levels_pi <- vector(mode = "list", length = length(noise_lvl_pi))

for (i in seq_along(noise_lvl_pi)) {
  various_noise_levels_pi[[i]] <- generate_zip(x_for_plot,
                                               noise_level_lambda = 1,
                                               noise_level_pi = noise_lvl_pi[i],
                                               scaling_lambda = 0.15,
                                               scaling_pi = 0.7,
                                               zero_inf_control = 1)

}

r2_pi <- NULL

for (i in 2:length(various_noise_levels_pi)) {
  SSE <-
    sum((
      various_noise_levels_pi[[1]]$eta_pi -
        various_noise_levels_pi[[i]]$eta_pi
    ) ^ 2)

  SSR <-
    sum((
      mean(various_noise_levels_pi[[1]]$eta_pi) -
        various_noise_levels_pi[[i]]$eta_pi
    ) ^ 2)

  SST <- SSE + SSR

  r2_pi <- c(r2_pi, SSR / SST)

}

noise_level_pi_curve <- data.frame(scaling_factor = noise_lvl_pi[-1],
                                   r2 = r2_pi)
noise_level_pi_curve
```

```
##   scaling_factor       r2
## 1            6.0 0.5800643
## 2            2.2 0.7528551
## 3            1.4 0.9148906
```

This function takes the simulated data and computes one `gam` and one `gamlss`. The basis dimensions for the coviariates in the linear predictor of λ were analogously to Wood et al. (2016) chosen to be x_1: 10, x_2: 10, x_3: 15, and x_4: 8. For the linear predictor of π the basis dimensions were x_1: 10, x_2: 10, and x_4: 8. The function computes the MSE of the prediction from `gam` and `gamlss` on the response scale by comparing it to the reall values of the linear predictor. This is done for the linear predictor of π and λ. Furthermore the information if the algorithm reached convergenced is saved as well as the run time each algorithm. Should the algoritm fail, this is saved to and all other values receive a `NA`-value.

```r
### COMPUTE MODEL FUNCTION
# input: data_list from "generate_zip()"
# output: list with mse_lambda, mse_pi, convergence, cpu time
```

```r
compute_model <- function(data_list, type = "gam") {
  if (type == "gam") {
    tryCatch(
      {
        start_time <- Sys.time()

        mod <-
          mgcv::gam(
            list(
              y ~
              s(x1, k = 10, bs = "ps") +
                s(x2, k = 10, bs = "ps") +
                s(x3, k = 15, bs = "ps") +
                s(x4, k = 8, bs = "ps"),
              ~
              s(x1, k = 10, bs = "ps") +
                s(x2, k = 10, bs = "ps") +
                s(x4, k = 8, bs = "ps")
            ),
            family = ziplss(),
            data = data_list$df
          )

        end_time <- Sys.time()

        time_diff <- log10(as.numeric(
          difftime(end_time,
            start_time,
            units = "secs"
          )
        ))

        convergence <- mod$outer.info$conv == "full convergence"

        mse_lambda <- sum((data_list$eta_lambda -
                             predict(mod, type = "link")[, 1])^2)
        mse_pi <- sum((data_list$eta_pi -
                         predict(mod, type = "link")[, 2])^2)

        remove(mod)

        return(matrix(c(
          mse_lambda, mse_pi, convergence, time_diff, FALSE
        ), nrow = 1))
      },
      error = function(e) {
        end_time <- Sys.time()

        time_diff <- log10(as.numeric(
          difftime(end_time,
            start_time,
            units = "secs"
          )
```

```
      ))
      matrix(c(NA, NA, FALSE, time_diff, TRUE), nrow = 1)
    }
  )
} else {
  tryCatch(
    {
      start_time <- Sys.time()

      mod <-
        gamlss::gamlss(
          formula = y ~
          pb(x1, max.df = 10, method = "ML") +
            pb(x2, max.df = 10, method = "ML") +
            pb(x3, max.df = 15, method = "ML") +
            pb(x4, max.df = 8, method = "ML"),
          sigma.formula = ~
          pb(x1, max.df = 10, method = "ML") +
            pb(x2, max.df = 10, method = "ML") +
            pb(x4, max.df = 8, method = "ML"),
          family = ZIP(sigma.link = "cloglog"),
          data = data_list$df,
          method = RS(30),
          control = gamlss.control(trace = FALSE)
        )

      end_time <- Sys.time()

      time_diff <- log10(as.numeric(
        difftime(end_time,
          start_time,
          units = "secs"
        )
      ))

      convergence <- mod$converged

      mse_lambda <- sum((data_list$eta_lambda -
                           predict(mod, what = "mu", type = "link"))^2)
      mse_pi <- sum((data_list$eta_pi -
                       predict(mod, what = "sigma", type = "link"))^2)

      remove(mod)

      matrix(c(mse_lambda, mse_pi, convergence, time_diff, FALSE), nrow = 1)
    },
    error = function(e) {
      end_time <- Sys.time()

      time_diff <- log10(as.numeric(
        difftime(end_time,
          start_time,
          units = "secs"
```

```
          )
        ))
        matrix(c(NA, NA, FALSE, time_diff, TRUE), nrow = 1)
      }
    )
  }
}
```

This function computes the actual simulation and combines all functions from before.

```
### COMPUTE INNER SIMULATION
# input: none
# output: list of x replicates of a simulation for a given model

compute_inner_simulation <- function(replicates = 300,
                                     n = 400,
                                     correlated = FALSE,
                                     correlation = NULL,
                                     noise_level_lambda = c(1.25, 1.5, 3),
                                     # low, med, high
                                     noise_level_pi = c(1.4, 2.2, 6),
                                     # low, med, high
                                     all_combinations = TRUE,
                                     scaling_lambda = 0.15,
                                     scaling_pi = 0.7,
                                     zero_inf_control = 1) {
  independent_variables <- map(
    rep(list(n), replicates),
    generate_independent,
    correlation = correlation,
    correlated = correlated
  )

  if (all_combinations) {
    names_list <-
      expand.grid(c("llow", "lmed", "lhigh"), c("plow", "pmed", "phigh"))
    names_list <- paste(names_list[, 1], names_list[, 2], sep = "_")

    noise_list <- expand.grid(noise_level_lambda, noise_level_pi)
  } else{
    names_list <-
      data.frame(c("llow", "lmed", "lhigh"), c("plow", "pmed", "phigh"))
    names_list <- paste(names_list[, 1], names_list[, 2], sep = "_")

    noise_list <- data.frame(Var1 = noise_level_lambda, Var2 = noise_level_pi)
  }

  simulation_results <- vector(mode = "list",
                               length = length(names_list))

  names(simulation_results) <- names_list

  for (i in seq_along(simulation_results)) {
    simulation_results[[i]] <- map(
```

```
      independent_variables,
      generate_zip,
      zero_inf_control = zero_inf_control,
      noise_level_lambda = noise_list[i, 1],
      noise_level_pi = noise_list[i, 2],
      scaling_lambda = scaling_lambda,
      scaling_pi = scaling_pi
    )
  }
    for (i in seq_along(simulation_results)) {
      plan(multisession)

      simulated_model_gam <-
        map(transpose(
          future_map(simulation_results[[i]],
                     compute_model,
                     type = "gam"),
          .names = c(
            "mse_lambda",
            "mse_pi",
            "converged",
            "duration",
            "failed"
          )
        ),
        unlist)

      simulated_model_gamlss <-
        map(transpose(
          future_map(simulation_results[[i]],
                     compute_model,
                     type = "gamlss"),
          .names = c(
            "mse_lambda",
            "mse_pi",
            "converged",
            "duration",
            "failed"
          )
        ),
        unlist)

      simulation_results[[i]] <- list("gam" = simulated_model_gam,
                                      "gamlss" = simulated_model_gamlss)
    }

    simulation_results
}
```

This function takes the output of the simulation and computes the comparasions metrics.

```
compute_comparison_metrics <- function(simulation_results) {

  comparison_list <- vector(mode = "list", length = length(simulation_results))
```

```r
for (i in seq_along(simulation_results)) {

  names(comparison_list)[i] <- names(simulation_results)[i]
  gam_model <- simulation_results[[i]][[1]]
  competitor_model <- simulation_results[[i]][[2]]


  # Compute differences

  included_obs <- !is.na(gam_model$mse_pi) == !is.na(competitor_model$mse_pi)

  diff_lambda <-
    competitor_model$mse_lambda[included_obs] -
    gam_model$mse_lambda[included_obs]

  diff_pi <-
    competitor_model$mse_pi[included_obs] - gam_model$mse_pi[included_obs]

  std_diff_lambda <-
    (diff_lambda - mean(diff_lambda, na.rm = TRUE)) / sd(diff_lambda,
                                                         na.rm = TRUE)

  std_diff_pi <-
    (diff_pi - mean(diff_pi, na.rm = TRUE)) / sd(diff_pi, na.rm = TRUE)

  # Compute paired one-sided t-test with significance at alpha = 0.05
  # Hypothesis:
  # H0: diff <= 0 (gam is equal or worse) vs. H1: diff > 0 (gam is better)

  ttest_lambda <-
    t.test(
      competitor_model$mse_lambda[included_obs],
      gam_model$mse_lambda[included_obs],
      alternative = "greater",
      paired = TRUE
    )

  ttest_pi <-
    t.test(
      competitor_model$mse_pi[included_obs],
      gam_model$mse_pi[included_obs],
      alternative = "greater",
      paired = TRUE
    )

  differences <-
    data.frame(
      differences = c(std_diff_lambda, std_diff_pi),
      type = c(rep("mu", length(std_diff_lambda)),
               rep("pi", length(std_diff_pi))),
      significant = c(
        rep(ttest_lambda$p.value < 0.05, length(std_diff_lambda)),
        rep(ttest_pi$p.value < 0.05, length(std_diff_pi))
```

```
        )
      )

    # Computing time

    duration <- data.frame(
      duration = c(gam_model$duration, competitor_model$duration),
      model = c(
        rep("mgcv::ziplss()", length(gam_model$duration)),
        rep("gamlss::ZIP()", length(competitor_model$duration))
      )
    )

    # Convergence rate
    gam_conv <- mean(gam_model$converged, na.rm = TRUE)

    competitor_conv <-
      mean(competitor_model$converged, na.rm = TRUE)

    convergence <- c("gam convergence" = gam_conv,
                     "competitor convergence" = competitor_conv)

    # Failure rate
    gam_failure <- mean(gam_model$failed, na.rm = TRUE)

    competitor_failure <-
      mean(competitor_model$failed, na.rm = TRUE)

    failure <- c("gam failure rate" = gam_failure,
                     "competitor failure rate" = competitor_failure)

    # Store list
    comparison_list[[i]] <- list("differences" = differences,
                                 "duration" = duration,
                                 "convergence" = convergence,
                                 "failure" = failure)
  }
  comparison_list
}
```

This function computes the plots with the comparison metrics.

```
get_plots <- function(comparison_list,
                      noise_level = c("low", "medium", "high"),
                      correlation_type = "uncorrelated",
                      count_type = "low count") {
  # Compute boxplots for differences
  df_diff <- NULL
  lengths <- NULL

  for (i in seq_along(comparison_list)) {
    df_diff <- rbind(df_diff, comparison_list[[i]]$differences)

    lengths <- c(lengths, nrow(comparison_list[[i]]$differences))
```

```
}

df_diff$noise_level <- rep(
  noise_level
  ,
  lengths
)

df_diff$noise_level <- factor(df_diff$noise_level,
                              levels = c("low", "medium", "high"),
                              ordered = TRUE)

plot_diff <- ggplot(data = df_diff, aes(
  x = noise_level,
  y = differences,
  color = factor(significant)
)) +
  geom_boxplot(size = 0.5 ,fill = "grey", outlier.shape = NULL) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  scale_color_manual(values = c("TRUE" = "green4", "FALSE" = "black")) +
  theme_classic() +
  facet_wrap( ~ type,
              labeller = label_parsed) +
  ggtitle(paste(
    "Differences in MSE for the",
    correlation_type,
    count_type,
    "case"
  )) +
  theme(axis.title = element_blank(),
        legend.position = "none")

print(plot_diff)

# Compute boxplot for computation times

df_comp <- NULL

for (i in seq_along(comparison_list)) {
  df_comp <- rbind(df_comp, comparison_list[[i]]$duration)
}

df_comp$type <- paste(
    correlation_type,
    count_type,
    "case"
  )

plot_comp <- ggplot(data = df_comp, aes(x = model,
                                        y = duration)) +
  geom_boxplot(size = 0.5) +
  theme_classic() +
  ggtitle(paste(
```

```
      "log computation time for the",
      correlation_type,
      count_type,
      "case"
    ))

  list(plot_diff, plot_comp, df_diff, df_comp)

}
```

Execution of simulation

```
set.seed(1991)
```

The simualtion for uncorrelated covariates for the three different noise levels:

```
starttime1 <- Sys.time()

simulation_uncorr <-
  compute_inner_simulation(
    replicates = 300,
    n = 400,
    correlated = FALSE,
    correlation = NULL,
    noise_level_lambda = c(1.25, 1.50, 3),
    noise_level_pi = c(1.4, 2.2, 6),
    all_combinations = FALSE,
    scaling_lambda = 0.1
  )

endtime1 <- Sys.time()
```

The simualtion for uncorrelated covariates and a low count response variable for the three different noise levels:

```
starttime2 <- Sys.time()

simulation_uncorr_lc <-
  compute_inner_simulation(
    replicates = 300,
    n = 400,
    correlated = FALSE,
    correlation = NULL,
    noise_level_lambda = c(1.25, 1.50, 3),
    noise_level_pi = c(1.4, 2.2, 6),
    all_combinations = FALSE,
    scaling_lambda = 0.05,
    scaling_pi = 1,
    zero_inf_control = 1
  )

endtime2 <- Sys.time()
```

The simualtion for correlated covariates for the three different noise levels:

```
starttime3 <- Sys.time()

simulation_corr <-
  compute_inner_simulation(
    replicates = 300,
    n = 400,
    correlated = TRUE,
    correlation = 0.9,
    noise_level_lambda = c(1.25, 1.50, 3),
    noise_level_pi = c(1.4, 2.2, 6),
    all_combinations = FALSE,
    scaling_lambda = 0.1,
    scaling_pi = 1,
    zero_inf_control = 1
  )

endtime3 <- Sys.time()
```

The simualtion for correlated covariates and a low count response variable for the three different noise levels:

```
starttime4 <- Sys.time()

simulation_corr_lc <-
  compute_inner_simulation(
    replicates = 300,
    n = 400,
    correlated = TRUE,
    correlation = 0.9,
    noise_level_lambda = c(1.25, 1.50, 3),
    noise_level_pi = c(1.4, 2.2, 6),
    all_combinations = FALSE,
    scaling_lambda = 0.05,
    scaling_pi = 1,
    zero_inf_control = 1
  )

endtime4 <- Sys.time()
```

```
time_diff <- as.numeric(
  difftime(endtime1, starttime1) +
    difftime(endtime2, starttime2) +
    difftime(endtime3, starttime3) +
    difftime(endtime4, starttime4)
)

paste("Overall computing time for complete simuation was",
      round(time_diff, 2),
      "hours")
```

```
## [1] "Overall computing time for complete simuation was 46.45 hours"
```

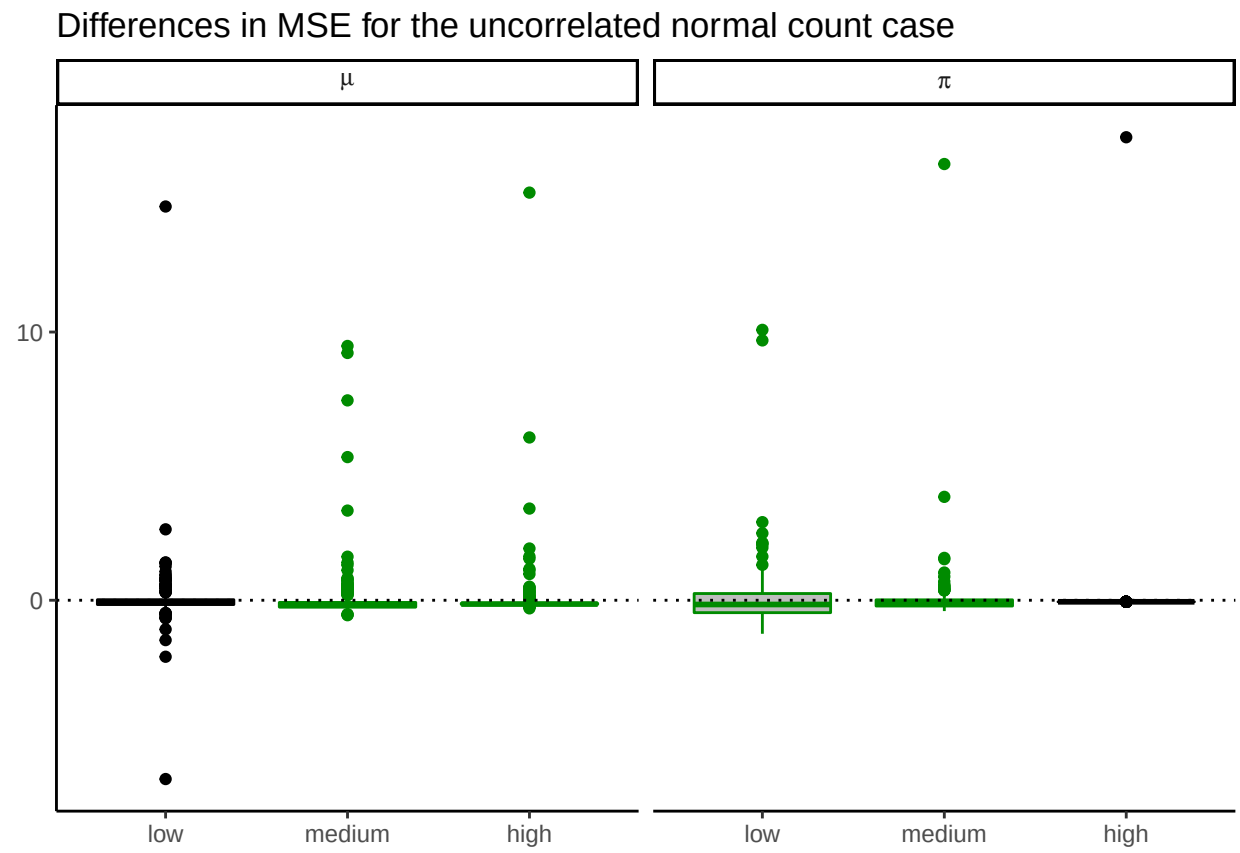
The comparison metrics for all 4 simulations are computed

```
comp_sim_uncorr <- compute_comparison_metrics(simulation_uncorr)
comp_sim_uncorr_lc <- compute_comparison_metrics(simulation_uncorr_lc)
comp_sim_corr <- compute_comparison_metrics(simulation_corr)
```

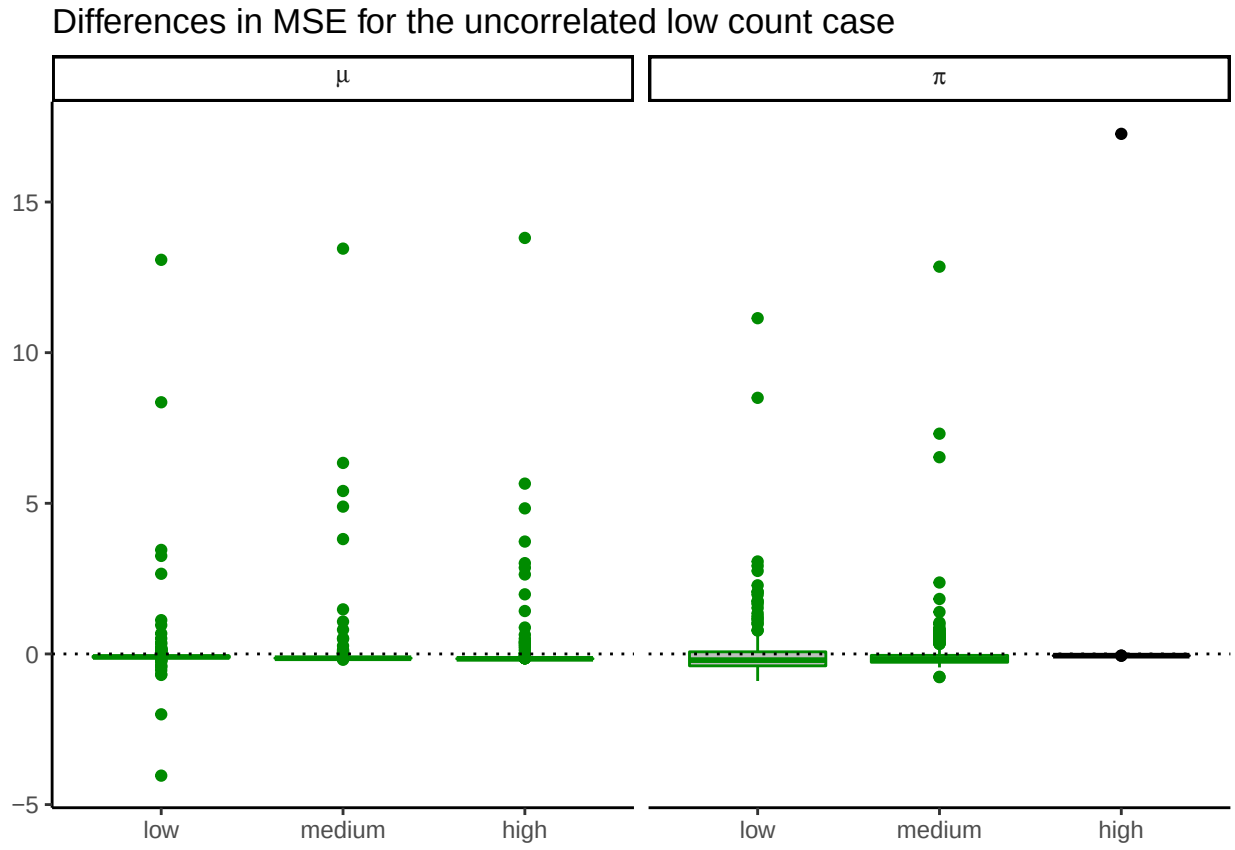
```
comp_sim_corr_lc <- compute_comparison_metrics(simulation_corr_lc)
```

The plots show the differences in MSE between the models for the different data situations:

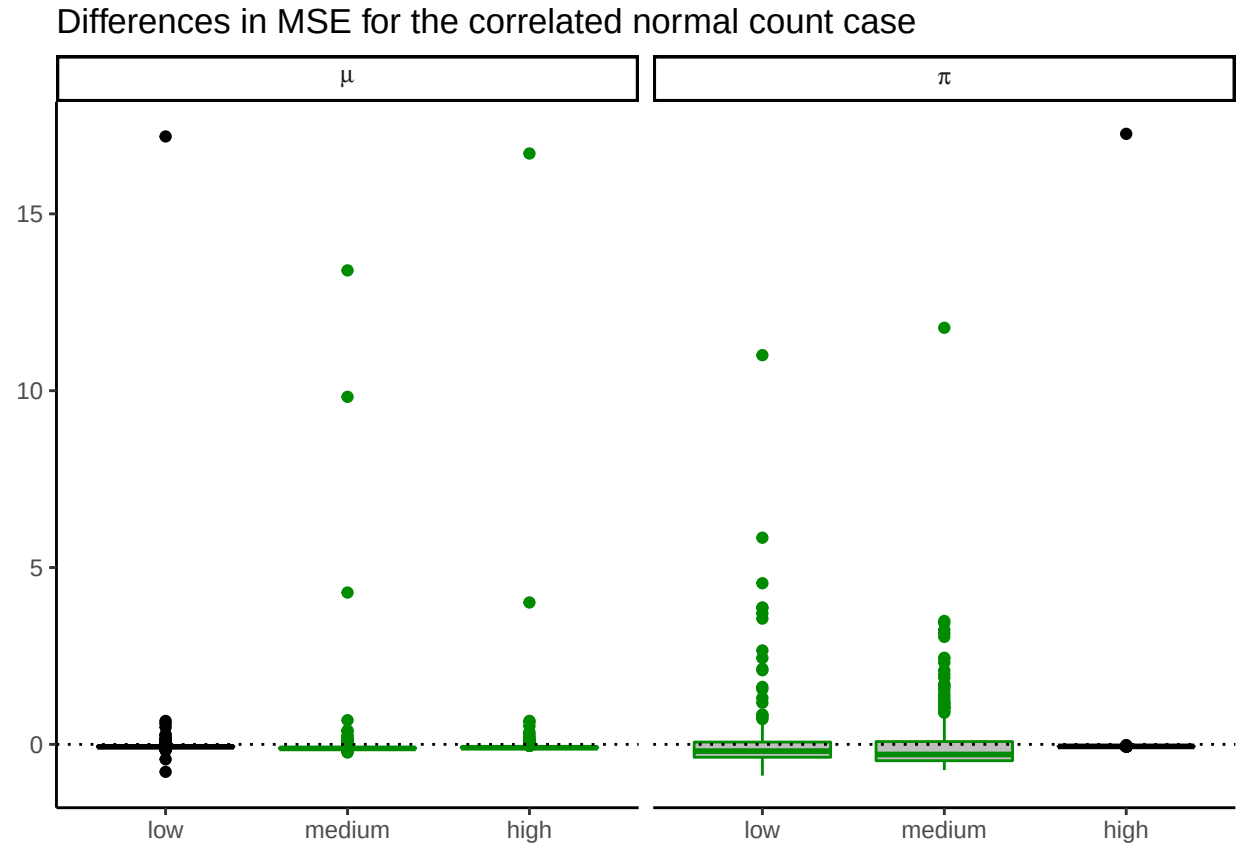
```
plots_uncorr <- get_plots(
  comp_sim_uncorr,
  correlation_type = "uncorrelated",
  count_type = "normal count"
)
```



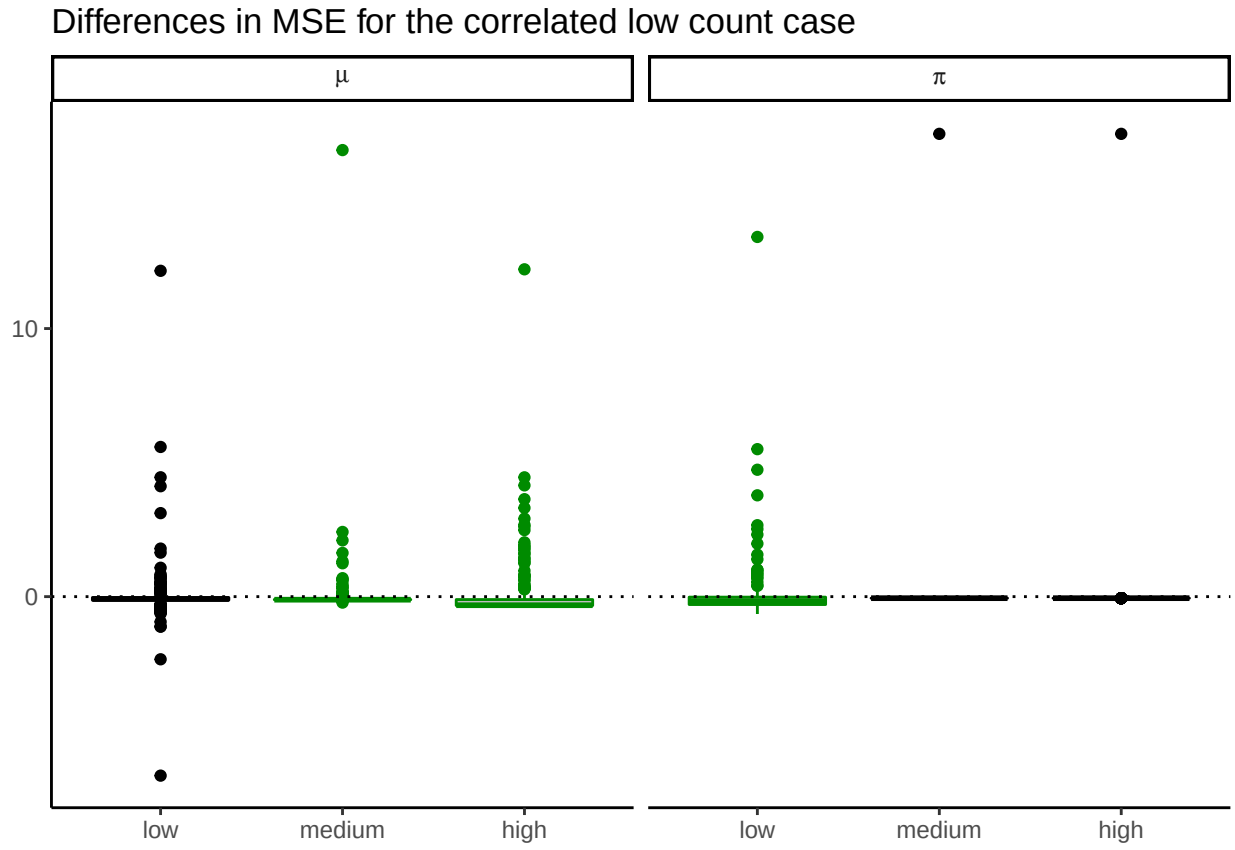

```
plots_uncorr_lc <- get_plots(
  comp_sim_uncorr_lc,
  correlation_type = "uncorrelated",
  count_type = "low count"
)
```




```
plots_corr <- get_plots(
  comp_sim_corr,
  correlation_type = "correlated",
  count_type = "normal count"
)
```




```
plots_corr_lc <- get_plots(
  comp_sim_corr_lc,
  correlation_type = "correlated",
  count_type = "low count"
)
```



This plot shows the computation times for the models:

```
data_frame_computing_times <- reduce(list(plots_uncorr[[4]],
                                          plots_uncorr_lc[[4]],
                                          plots_corr[[4]],
                                          plots_corr_lc[[4]]),
                                     rbind)

comp_times_plot <- ggplot(data = data_frame_computing_times,
                          aes(x = model,
                              y = duration)) +
  geom_boxplot() +
  facet_wrap(~ type) +
  theme_classic() +
  ggtitle(TeX("log_{10} computation times"))
```

This table shows the convergence rates of the models for the different data situations:

```
comparisons <- list(comp_sim_uncorr,
                    comp_sim_uncorr_lc,
                    comp_sim_corr,
                    comp_sim_corr_lc)

convergence_table <- matrix(NA,
                            nrow = 4,
                            ncol = 3)
```

```r
rownames(convergence_table) <- c("uncorrelated",
                                 "uncorrelated / low count",
                                 "correlated",
                                 "correlated / low count")

colnames(convergence_table) <- c("low", "medium", "high")

convergence_table_gamlss <- convergence_table
convergence_table_mgcv <- convergence_table


for(i in seq_along(comparisons)){
  for(j in seq_along(comparisons[[i]])){
    convergence_table_mgcv[i,j] <- comparisons[[i]][[j]][["convergence"]][1]
    convergence_table_gamlss[i,j] <- comparisons[[i]][[j]][["convergence"]][2]
  }
}

convergence_table_gamlss
```

```
##                              low    medium      high
## uncorrelated             0.9666667 0.8500000 0.6233333
## uncorrelated / low count 0.9100000 0.7866667 0.6066667
## correlated               0.9066667 0.7900000 0.2033333
## correlated / low count   0.9400000 0.7566667 0.2633333
```

```r
convergence_table_mgcv
```

```
##                          low medium      high
## uncorrelated               1      1 0.9966667
## uncorrelated / low count   1      1 1.0000000
## correlated                 1      1 1.0000000
## correlated / low count     1      1 1.0000000
```

This is the correlation coefficient for the convergence rates of `gamlss` and the mean difference in MSE

```r
mean_difference <- NULL
convergence_rates <- NULL
for (i in seq_along(comparisons)) {
  for (j in seq_along(comparisons[[i]])) {
    mean_difference  <-
      c(mean_difference,
        mean(comparisons[[i]][[j]]$differences$differences, 0.05))
    convergence_rates <-
      c(convergence_rates, comparisons[[i]][[j]][["convergence"]][2])
  }
}

correlation_matrix <- data.frame(mean_difference, convergence_rates)
cor(correlation_matrix)
```

```
##                  mean_difference convergence_rates
## mean_difference        1.00000000       -0.08389183
## convergence_rates     -0.08389183        1.00000000
```

This table shows the failure rates of the models for the different data situations:

```
comparisons <- list(comp_sim_uncorr,
                    comp_sim_uncorr_lc,
                    comp_sim_corr,
                    comp_sim_corr_lc)

failure_table <- matrix(NA,
                            nrow = 4,
                            ncol = 3)

rownames(failure_table) <- c("uncorrelated",
                                 "uncorrelated / low count",
                                 "correlated",
                                 "correlated / low count")

colnames(failure_table) <- c("low", "medium", "high")

failure_table_gamlss <- failure_table
failure_table_mgcv <- failure_table


for(i in seq_along(comparisons)){
  for(j in seq_along(comparisons[[i]])){
    failure_table_mgcv[i,j] <- comparisons[[i]][[j]][["failure"]][1]
    failure_table_gamlss[i,j] <- comparisons[[i]][[j]][["failure"]][2]
  }
}

failure_table_gamlss
```

```
##                           low medium high
## uncorrelated                0      0    0
## uncorrelated / low count    0      0    0
## correlated                  0      0    0
## correlated / low count      0      0    0
```

```
failure_table_mgcv
```

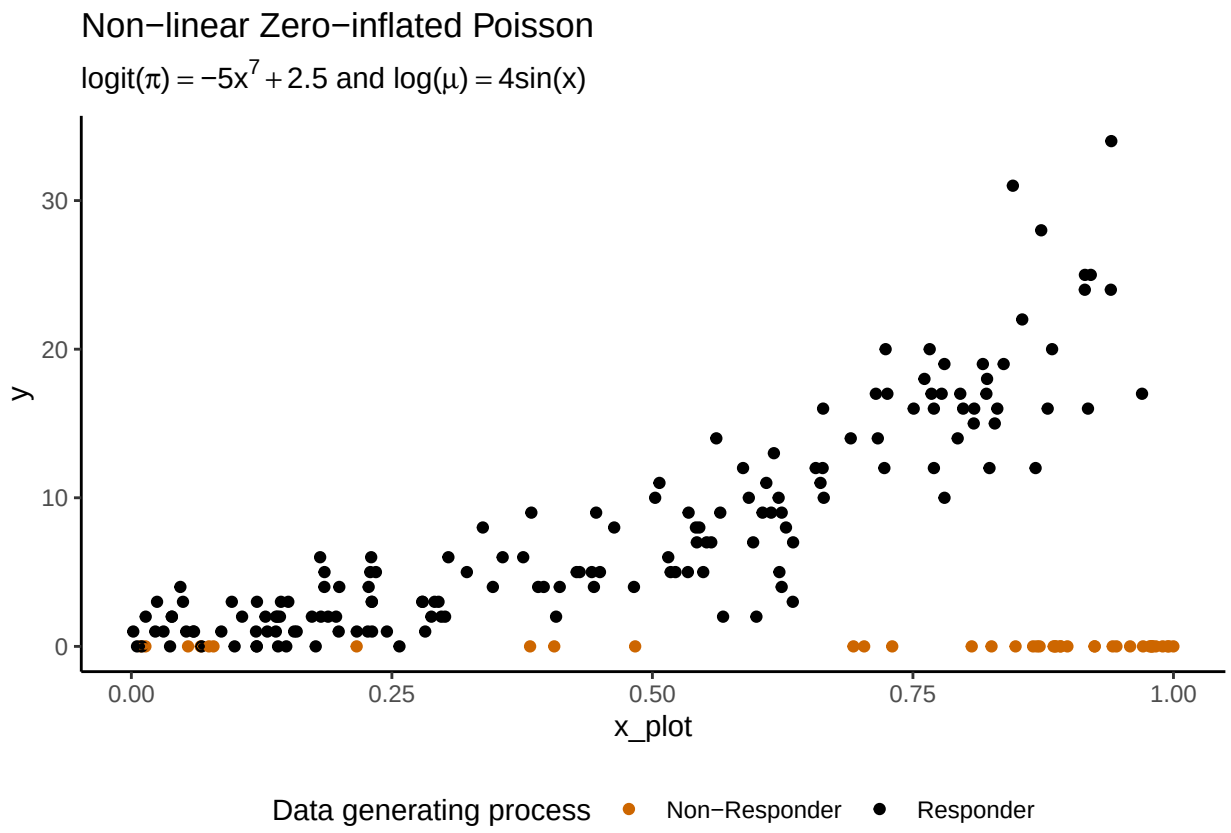
```
##                           low medium high
## uncorrelated                0      0    0
## uncorrelated / low count    0      0    0
## correlated                  0      0    0
## correlated / low count      0      0    0
```

Miscellaneous

Figure 1 in the essay.

```
x_plot <- runif(200)
lambda_plot <- poisson()$linkinv(sin(x_plot) * 4)
pi_plot <- binomial()$linkinv(-5 * x_plot ^ 7 + 2.5)
pois_sim  <- rbinom(x_plot, 1, (pi_plot))
ind <- pois_sim == 1

pois_sim[ind] <- rpois(x_plot[ind], lambda_plot[ind])
```

```
zero_inf_plot <-
  qplot(x_plot, pois_sim, color = factor(ind)) +
  geom_point(size = 0.5) + theme_classic() +
  ylab("y") +
  ggtitle("Non-linear Zero-inflated Poisson", subtitle = (TeX(
    "$logit(\\pi) = -5x^7 + 2.5$ and $log(\\mu) = 4sin(x)$"
  ))) +
  scale_color_manual(
    values = c("FALSE" = "darkorange3", "TRUE" = "black"),
    name = "Data generating process",
    labels = c("Non-Responder", "Responder")
  ) +
  theme(legend.position = "bottom")

zero_inf_plot
```



Session Info

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
```

```
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  splines   stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] xtable_1.8-4         latex2exp_0.4.0      furrr_0.1.0
##  [4] future_1.16.0        gridExtra_2.3        mvtnorm_1.0-11
##  [7] gamlss_5.1-3         gamlss.dist_5.1-3    MASS_7.3-51.4
## [10] gamlss.data_5.1-4    mgcv_1.8-28          nlme_3.1-140
## [13] forcats_0.4.0        stringr_1.4.0        dplyr_0.8.3
## [16] purrr_0.3.3          readr_1.3.1          tidyr_1.0.0
## [19] tibble_2.1.3         ggplot2_3.2.1        tidyverse_1.3.0.9000
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.3        lubridate_1.7.4   lattice_0.20-38   listenv_0.8.0
##  [5] assertthat_0.2.1  zeallot_0.1.0     digest_0.6.23     R6_2.4.1
##  [9] cellranger_1.1.0  backports_1.1.5   reprex_0.3.0      evaluate_0.14
## [13] httr_1.4.1        pillar_1.4.2      rlang_0.4.2       lazyeval_0.2.2
## [17] readxl_1.3.1      rstudioapi_0.10   Matrix_1.2-17     rmarkdown_1.18
## [21] labeling_0.3      munsell_0.5.0     broom_0.5.2       compiler_3.6.1
## [25] modelr_0.1.5      xfun_0.11         pkgconfig_2.0.3   globals_0.12.5
## [29] htmltools_0.4.0   tidyselect_0.2.5  codetools_0.2-16  crayon_1.3.4
## [33] dbplyr_1.4.2      withr_2.1.2       grid_3.6.1        jsonlite_1.6
## [37] gtable_0.3.0      lifecycle_0.1.0   DBI_1.0.0         magrittr_1.5
## [41] scales_1.1.0      cli_1.1.0         stringi_1.4.3     farver_2.0.1
## [45] fs_1.3.1          xml2_1.2.2        generics_0.0.2    vctrs_0.2.0
## [49] tools_3.6.1       glue_1.3.1        hms_0.5.2         survival_2.44-1.1
## [53] yaml_2.2.0        colorspace_1.4-1  rvest_0.3.5       knitr_1.26
## [57] haven_2.2.0
```

Wood, Simon N., Natalya Pya, and Benjamin Säfken. 2017. "Smoothing Parameter and Model Selection for General Smooth Models." *Journal of the American Statistical Association* 111 (516): 1548–63. https://doi.org/10.1080/01621459.2016.1180986.