WastewaterRe Code Summary

Karim El-Ouaghlidi

4/25/2022

Summary of code used in WastewaterRe Paper by Huisman et. al - wastewater_Zurich.R

- Initial data manipulation (outside of complicated, nested functions)

- Wastewater ("Flow") data

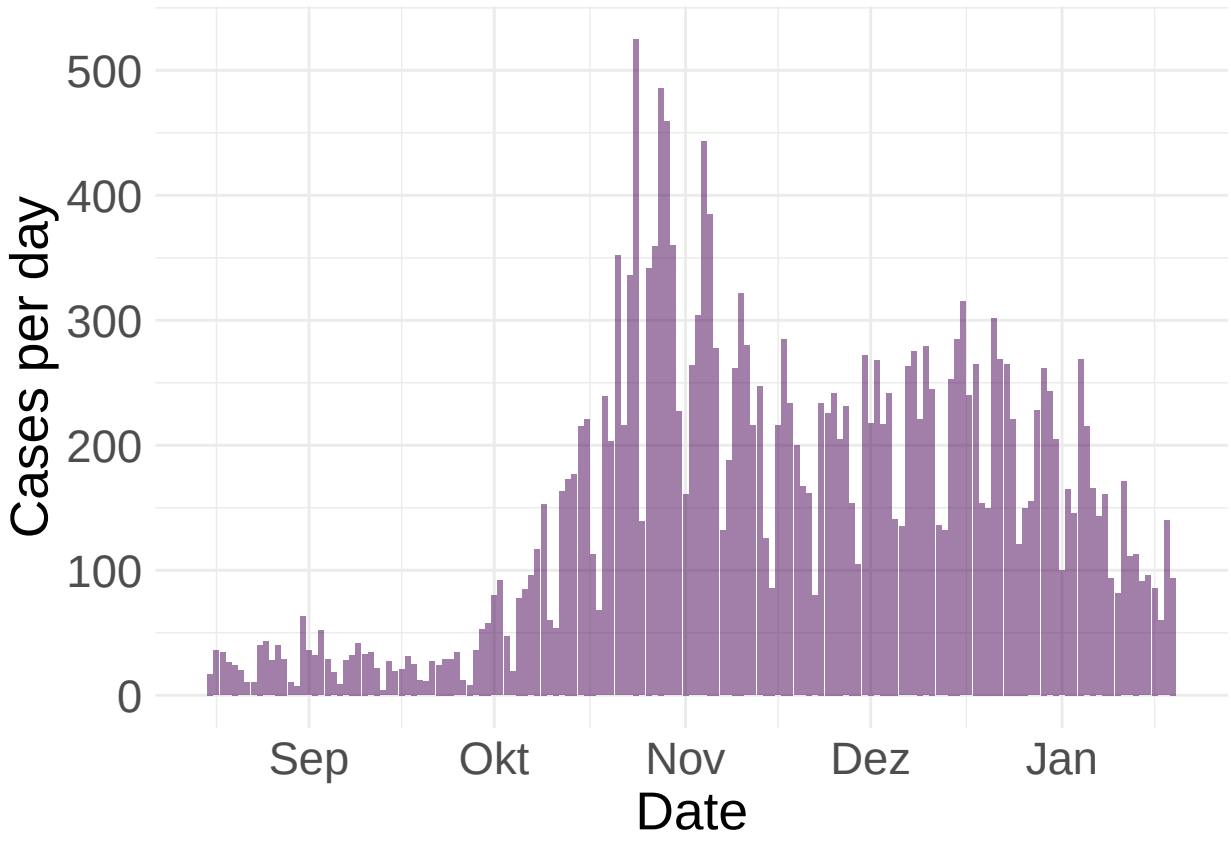
- flow data is loaded with col_names = c('date', 'cases', 'cases_smooth','flow', 'n1_smooth', 'n2_smooth')
- gene data is loaded with col_names = c('date', 'n1', 'n2'),
- missing values in flow data are imputed by linear interpolation:

```
raw_data_ZH <- raw_flow_data_ZH %>%
  left_join(raw_gene_data_ZH, c('date')) %>%
  filter(!is.na(n1),
         date >= as_date("2020-09-01"),
         date <= as_date("2021-01-20"),
         date != as_date("2020-10-29")) %>%
  mutate(orig_data = TRUE) %>%
  complete(date = seq.Date(min(date), max(date), by = 'days')) %>%
  mutate(across(where(is.numeric), ~ zoo::na.approx(.x, na.rm = F) )) %>%
  mutate(region = 'ZH')
```

- Case data

- Catchment level clinical case data: missing values are set to 0
- Intuition: a positive case is most likely reported for the next day if the actual day was missing

```
orig_cases <- read_csv(paste0(dir,'/wastewaterRe/data/ZH_case_incidence_data.csv')) %>%
  filter(date >= as_date("2020-08-15"),
         date <= as_date("2021-01-20")) %>%
  mutate(across(c(-date), ~ ifelse(is.na(.), 0, .)))  %>%
  select(date, confirmed)
```



- Deconvolution of Case data

- Let C denote the observed cases and D denote the delay distribution ("transfer function") from true underlying incidence I_{cc} to C.
- Then, $C = I_{cc} * D$ is the convolution of true incidence I_{cc} and delay function D.
- Thus, in order to obtain an estimate for the underlying true incidence, which we want to use to obtain our Shedding Load Distribution (SLD), we need to do a deconvolution of observed cases and delay distribution.

```
deconv_cases <- data.frame()
Re_cases <- data.frame()
for(inc_var in c('confirmed')){
  new_deconv_data = deconvolveIncidence(orig_cases,
                                        incidence_var = inc_var,
                                        getCountParams('incubation'),
                                        getCountParams(paste0(inc_var, '_zh')),
                                        smooth_param = TRUE, n_boot = 50)
  new_Re = getReBootstrap(new_deconv_data)

  deconv_cases <- bind_rows(deconv_cases, new_deconv_data)
  Re_cases = bind_rows(Re_cases, new_Re)
}
```

```
## estimating Re for data source:  ETH ...
##   Region:  CHE
##     Data type:  infection_confirmed
```

- deconvolveIncidence()

- Let's have a closer look at the called function wrapper:

```
deconvolveIncidence <- function(df, incidence_var = 'n1',
                                IncubationParams, OnsetToCountParams,
                                smooth_param = FALSE, n_boot = 50){
  infection_df <- addUselessColumns(df, inc_var = incidence_var)

  constant_delay_distributions <- list("Simulated" = get_vector_constant_waiting_time_distr(
    IncubationParams$shape, IncubationParams$scale,
    OnsetToCountParams$shape, OnsetToCountParams$scale),
    "Symptoms" = get_vector_constant_waiting_time_distr(
      IncubationParams$shape, IncubationParams$scale,
      0, 0))

  estimatedInfections <- get_infection_incidence_by_deconvolution(
    infection_df,
    is_local_cases = T,
    constant_delay_distribution = constant_delay_distributions[['Simulated']],
    constant_delay_distribution_incubation = constant_delay_distributions[["Symptoms"]],
    max_iterations = 100,
    smooth_incidence = smooth_param,
    empirical_delays = tibble(),
    n_bootstrap = n_boot,
    verbose = FALSE)

  return(estimatedInfections)
}
```

- Apparently, the actual deconvolution function takes as input the infection time series, and the delay distributions 1) incubation and 2) symptom onset to reporting as vectors that sum up to 1.

```
inc_var <- "confirmed"

IncubationParams <- getCountParams('incubation')
OnsetToCountParams <- getCountParams(paste0(inc_var, '_zh'))

constant_delay_distributions <- list("Simulated" = get_vector_constant_waiting_time_distr(
    IncubationParams$shape, IncubationParams$scale,
    OnsetToCountParams$shape, OnsetToCountParams$scale),
    "Symptoms" = get_vector_constant_waiting_time_distr(
      IncubationParams$shape, IncubationParams$scale,
      0, 0))

constant_delay_distributions
```

```
## $Simulated
##   [1] 0.000249 0.009954 0.036585 0.066823 0.090698 0.103189 0.106403 0.100766
##   [9] 0.091679 0.079801 0.066626 0.054616 0.044358 0.035327 0.027431 0.021192
##  [17] 0.016095 0.012188 0.009252 0.006966 0.005238 0.003821 0.002739 0.002111
##  [25] 0.001569 0.001222 0.000834 0.000623 0.000456 0.000338 0.000244 0.000163
##  [33] 0.000107 0.000102 0.000062 0.000050 0.000038 0.000020 0.000020 0.000012
##  [41] 0.000008 0.000008 0.000008 0.000005 0.000001 0.000000 0.000000 0.000001
##  [49] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000001
```

```
##  [57] 0.000000 0.000000 0.000000 0.000000 0.000001 0.000000 0.000000 0.000000
##  [65] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [73] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [81] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [89] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [97] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [105] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [113] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [121] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [129] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [137] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [145] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [153] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [161] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [169] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [177] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [185] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [193] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##
## $Symptoms
##   [1] 0.004537 0.060809 0.119930 0.146395 0.144358 0.127380 0.104666 0.082200
##   [9] 0.061414 0.045133 0.032019 0.022757 0.015633 0.010843 0.007424 0.004925
##  [17] 0.003259 0.002169 0.001461 0.000961 0.000626 0.000390 0.000266 0.000147
##  [25] 0.000120 0.000065 0.000043 0.000019 0.000024 0.000014 0.000005 0.000005
##  [33] 0.000001 0.000001 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [41] 0.000000 0.000001 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [49] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [57] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [65] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [73] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [81] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [89] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
##  [97] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [105] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [113] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [121] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [129] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [137] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [145] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [153] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [161] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [169] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [177] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [185] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [193] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

```
sum(constant_delay_distributions[[1]])
```

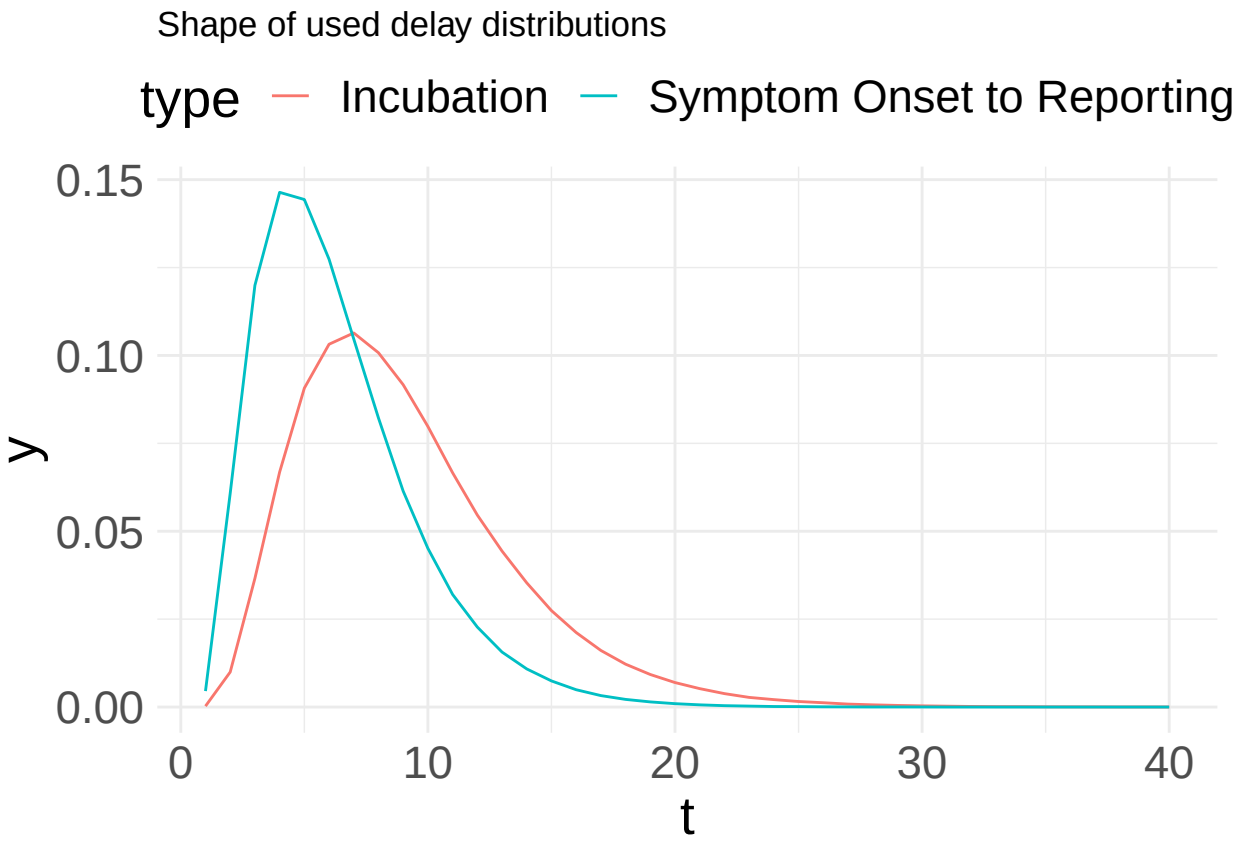
```
## [1] 1
```

```
sum(constant_delay_distributions[[2]])
```

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

```
T_max <- 40
```

```
delay_distribution_inc <- tibble(t = 1:T_max,
                                 y = constant_delay_distributions[[1]][1:T_max],
                                 type = 'Incubation')


delay_distribution_sym <- tibble(t = 1:T_max,
                                   y = constant_delay_distributions[[2]][1:T_max],
                                   type = 'Symptom Onset to Reporting')


delay_distributions <- c(list(delay_distribution_inc),list(delay_distribution_sym)) %>%
  dplyr::bind_rows()

ggplot(delay_distributions) +
  geom_line(aes(x=t, y= y, color = type)) +
  ggtitle("Shape of used delay distributions")
```



- detail question: What goes on inside the following function?! I know what the result is so this is not too important to sort out :)

```
get_vector_constant_waiting_time_distr <- function(shape_incubation,
                                                   scale_incubation,
                                                   shape_onset_to_report,
                                                   scale_onset_to_report,
                                                   length_out = 200,
                                                   n_random_samples = 1E6) {

  F_h <- make_ecdf_from_gammas(shape = c(shape_incubation, shape_onset_to_report),
```

```
                                scale = c(scale_incubation, scale_onset_to_report))

  f <- Vectorize(function(x){
    if(x < 0) {
      return(0)
    } else if(x < 0.5) {
      return(F_h(0.5))
    } else {
      return(F_h(round(x + 1E-8) + 0.5) - F_h(round(x + 1E-8) - 0.5))
    }
  })

# Where the following function is called
make_ecdf_from_gammas <- function(shape, scale, numberOfSamples = 1E6) {
  draws <-
    rgamma(numberOfSamples, shape = shape[1], scale = scale[1]) +
    rgamma(numberOfSamples, shape = shape[2], scale = scale[2])
  return(Vectorize(ecdf(draws)))
}
```

- get_infection_incidence_by_deconvolution()

```
## Actual deconvolution function..

get_infection_incidence_by_deconvolution <- function(
  data_subset,
  constant_delay_distribution,
  constant_delay_distribution_incubation = c(),
  is_onset_data = F,
  is_local_cases = T,
  smooth_incidence = T,
  days_incl = 21,
  empirical_delays  = tibble(),
  n_bootstrap = 5,
  days_further_in_the_past = 30,
  days_further_in_the_past_incubation = 5,
  max_iterations = 100,
  verbose = FALSE) {

  #TODO make the days_further_in_the_past type specific

  if(nrow(data_subset) == 0) {
    return(tibble())
  }

  data_type_subset <- unique(data_subset$data_type)[1]

  # exclude leading zeroes
  data_subset <- data_subset %>%
    arrange(date) %>%
    filter(cumsum(value) > 0)
```

```r
  if(nrow(data_subset) == 0) {
    return(tibble())
  }

  minimal_date <- min(data_subset$date) - days_further_in_the_past
  maximal_date <- max(data_subset$date)
  all_dates <- seq(minimal_date, maximal_date, by = "days")

  is_empirical = (nrow(empirical_delays) > 0)

  if(verbose && is_empirical) {
    cat("\tEmpirical delay distribution available\n")
  }

# We can ignore this as default setting is_onset_data = FALSE

#   if( is_onset_data ) {
#     delay_distribution_matrix_incubation <- get_matrix_constant_waiting_time_distr(
#       constant_delay_distribution_incubation,
#       all_dates)
#
#     initial_delta_incubation <- min(which(cumsum(constant_delay_distribution_incubation) > 0.5)) - 1 s
#
#
#     if(unique(data_subset$region)[1] != "ESP") { # hack to workaround weirdness of Spanish data
#
#       # account for additional right-truncation of onset data (needs to be reported first)
#       if(is_empirical) {
#         delay_distribution_matrix_onset_to_report <- get_matrix_empirical_waiting_time_distr(
#           empirical_delays,
#           seq.Date(min(data_subset$date), max(data_subset$date), by = "days"))
#       } else {
#         delay_distribution_matrix_onset_to_report <- get_matrix_constant_waiting_time_distr(
#           constant_delay_distribution,
#           seq.Date(min(data_subset$date), max(data_subset$date), by = "days"))
#       }
#
#       data_subset <- data_subset %>%
#         complete(date = seq.Date(min(date), max(date), by = "days"), fill = list(value = 0))
#
#       Q_vector_onset_to_report <- apply(delay_distribution_matrix_onset_to_report, MARGIN = 2, sum)
#
#       #TODO remove
#       # if(unique(data_subset$region)[1] == "ESP") { # hack to work around spanish data between sympt
#       #   right_truncation <- 3
#       #   # need to offset the Q vector by how many days were truncated off originally
#       #   Q_vector_onset_to_report <- c(rep(1, right_truncation), Q_vector_onset_to_report[1:(length(
#       # }
#
#       data_subset <- data_subset %>%
#         mutate(value = value / Q_vector_onset_to_report) %>%
#         mutate(value = if_else(value == Inf, 0, value))
#
```

```
#     }
#
#   } else {


# Also, is_empirical = FALSE per default and we don't supply empirical delays.

    # if(is_empirical) {
    #   delay_distribution_matrix_onset_to_report <- get_matrix_empirical_waiting_time_distr(
    #     empirical_delays,
    #     all_dates[(days_further_in_the_past_incubation + 1):length(all_dates)])
    #
    #   delay_distribution_matrix_incubation <- get_matrix_constant_waiting_time_distr(
    #     constant_delay_distribution_incubation,
    #     all_dates)
    #
    #   initial_delta_incubation <- min(which(cumsum(constant_delay_distribution_incubation) > 0.5)) -
    #   initial_delta_report <-  median(empirical_delays$delay, na.rm = T)
    # } else {


      delay_distribution_matrix <- get_matrix_constant_waiting_time_distr(
        constant_delay_distribution,
        all_dates)

      initial_delta <- min(which(cumsum(constant_delay_distribution) > 0.5)) - 1
      # take median value (-1 because index 1 corresponds to zero days)

  #   }
  # }



  results <- list(tibble())
  ## bootstrapping is performed on the log_diff = log_value - log_loess
  ## and is converted back to original scale afterwards
  for (bootstrap_replicate_i in 0:n_bootstrap) {

    if (verbose == T) {
      cat("    Bootstrap replicate: ", bootstrap_replicate_i, "\n")
    }

    if (bootstrap_replicate_i == 0) {
      time_series <- data_subset
    } else {
      time_series <- get_bootstrap_replicate(data_subset)
    }
## Now, the original or bootstrapped time series is smoothed by LOESS
    if (smooth_incidence == T) {
      smoothed_incidence_data <- time_series %>%
        complete(date = seq.Date(min(date), max(date), by = "days"),
                 fill = list(value = 0)) %>%
        mutate(value = getLOESSCases(dates = date, count_data = value, days_incl))
```

```
      raw_total_incidence <- sum(time_series$value, na.rm = TRUE)
      smoothed_total_incidence <- sum(smoothed_incidence_data$value, na.rm = T)
## Reweighting to make sure that sums match??
      if (smoothed_total_incidence > 0) {
        smoothed_incidence_data <- smoothed_incidence_data %>%
          mutate(value = value * raw_total_incidence / smoothed_total_incidence)
      }

    } else {
      smoothed_incidence_data <- time_series  %>%
        complete(date = seq.Date(min(date), max(date), by = "days"),
                 fill = list(value = 0))
    }
    # is_onset_data = FALSE per default

    # if (is_onset_data) {
    #   deconvolved_infections <-  do_deconvolution(smoothed_incidence_data,
    #                                               delay_distribution_matrix = delay_distribution_matr
    #                                               days_further_in_the_past = days_further_in_the_past
    #                                               initial_delta = initial_delta_incubation,
    #                                               max_iterations = max_iterations,
    #                                               verbose = verbose)
    # } else {


    # is_empirical = FALSE as well in our setting

      # if(is_empirical) {
      #   # perform the deconvolution in two steps
      #   deconvolved_symptom_onsets <- do_deconvolution(smoothed_incidence_data,
      #                                                  delay_distribution_matrix = delay_distribution
      #                                                  days_further_in_the_past = days_further_in_the
      #                                                  initial_delta = initial_delta_report,
      #                                                  max_iterations = max_iterations,
      #                                                  verbose = verbose)
      #
      #   deconvolved_infections <- do_deconvolution(deconvolved_symptom_onsets,
      #                                              delay_distribution_matrix = delay_distribution_mat
      #                                              days_further_in_the_past = days_further_in_the_pas
      #                                              initial_delta = initial_delta_incubation,
      #                                              max_iterations = max_iterations,
      #                                              verbose = verbose)
      # } else {

        deconvolved_infections <-  do_deconvolution(smoothed_incidence_data,
                                    delay_distribution_matrix = delay_distribution_matrix,
                                    days_further_in_the_past = days_further_in_the_past,
                                    initial_delta = initial_delta,
                                    max_iterations = max_iterations,
                                    verbose = verbose)

    #   }
    # }
```

```
    deconvolved_infections <- deconvolved_infections %>%
      slice((days_further_in_the_past -5 + 1):n())

    data_type_name <- paste0("infection_", data_type_subset)

    ## dataframe containing results
    deconvolved_infections <- tibble(
      date = deconvolved_infections$date,
      region = unique(time_series$region)[1],
      country = unique(time_series$country)[1],
      source = unique(time_series$source)[1],
      local_infection = is_local_cases,
      data_type = data_type_name,
      replicate = bootstrap_replicate_i,
      value = deconvolved_infections$value
    )

    results <- c(results, list(deconvolved_infections))
  }

  return(bind_rows(results))
}
```

- Essentially, what happens is: the wrapper function creates the vectors of delay distributions for 1) incubation and 2) symptom onset to reporting and supplies it to the actual deconvolution function. Then, the function `get_matrix_constant_waiting_time_distr` creates a T by T lower-diagonal matrix with the vectorized delay distributions as columns starting at the diagonal element (see below for illustration).
- bootstrapping is performed on `log_diff = log_value - log_loess` where `log_value = log(value + 1)`. Afterwards, the bootstrapped time series is transformed back to original scale: `ts_boot = exp(log_diff + log_loess) - 1`. In each of the `n_bootstrap = 50` iterations, the deconvolution is performed.

```
get_matrix_constant_waiting_time_distr <- function(waiting_time_distr,
                                                   all_dates) {
  N <- length(all_dates)

  if(length(all_dates) >= length(waiting_time_distr)) {
    waiting_time_distr <- c(waiting_time_distr, rep(0, times = N - length(waiting_time_distr)))
  }

  delay_distribution_matrix <- matrix(0, nrow = N, ncol = N)
  for(i in 1:N) {
    delay_distribution_matrix[, i ] <-  c(rep(0, times = i - 1 ), waiting_time_distr[1:(N - i + 1)])
  }

  return(delay_distribution_matrix)
}
```

- Let's check what shape the returned matrix `delay_distribution_matrix` has:

```
constant_delay_distributions <- list("Simulated" = get_vector_constant_waiting_time_distr(
    IncubationParams$shape, IncubationParams$scale,
    OnsetToCountParams$shape, OnsetToCountParams$scale),
    "Symptoms" = get_vector_constant_waiting_time_distr(
```

```
      IncubationParams$shape, IncubationParams$scale,
      0, 0))

constant_delay_distribution <- constant_delay_distributions[['Simulated']]

constant_delay_distribution_incubation <- constant_delay_distributions[["Symptoms"]]

dates_seq <- seq(date("2021-01-01"), date("2021-01-01")+6, by = "days")

# Check Matrix for small N
get_matrix_constant_waiting_time_distr(
        constant_delay_distribution,
        dates_seq)
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 0.000297 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [2,] 0.010002 0.000297 0.000000 0.000000 0.000000 0.000000 0.000000
## [3,] 0.036571 0.010002 0.000297 0.000000 0.000000 0.000000 0.000000
## [4,] 0.066968 0.036571 0.010002 0.000297 0.000000 0.000000 0.000000
## [5,] 0.090729 0.066968 0.036571 0.010002 0.000297 0.000000 0.000000
## [6,] 0.103512 0.090729 0.066968 0.036571 0.010002 0.000297 0.000000
## [7,] 0.106210 0.103512 0.090729 0.066968 0.036571 0.010002 0.000297
```

```
# Verify that Matrix indeed only includes vectorized delay distribution
all.equal(constant_delay_distribution[1:7],
        get_matrix_constant_waiting_time_distr(
        constant_delay_distribution, dates_seq)[,1])
```

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

-

"stepwise" function call

```
days_further_in_the_past <- 30

incidence_var <- inc_var <- 'confirmed'

head(orig_cases)
```

```
## # A tibble: 6 x 2
##   date       confirmed
##   <date>         <dbl>
## 1 2020-08-15        10
## 2 2020-08-16        17
## 3 2020-08-17        36
## 4 2020-08-18        34
## 5 2020-08-19        26
## 6 2020-08-20        24
```

```
data_subset <- infection_df <- addUselessColumns(orig_cases, inc_var = incidence_var)

  # exclude leading zeroes
  data_subset <- data_subset %>%
```

```
    arrange(date) %>%
    filter(cumsum(value) > 0)

head(data_subset)
```

```
## # A tibble: 6 x 9
##   date       region value data_type source variable  country     date_type
##   <date>     <chr>  <dbl> <chr>     <chr>  <chr>     <chr>       <chr>
## 1 2020-08-15 CHE       10 confirmed ETH    incidence Switzerland report
## 2 2020-08-16 CHE       17 confirmed ETH    incidence Switzerland report
## 3 2020-08-17 CHE       36 confirmed ETH    incidence Switzerland report
## 4 2020-08-18 CHE       34 confirmed ETH    incidence Switzerland report
## 5 2020-08-19 CHE       26 confirmed ETH    incidence Switzerland report
## 6 2020-08-20 CHE       24 confirmed ETH    incidence Switzerland report
## # ... with 1 more variable: local_infection <lgl>
```

```
  minimal_date <- min(data_subset$date) - days_further_in_the_past
  maximal_date <- max(data_subset$date)
  all_dates <- seq(minimal_date, maximal_date, by = "days")

# Create actual delay_distribution_matrix
delay_distribution_matrix <- get_matrix_constant_waiting_time_distr(constant_delay_distribution,all_dat

#
initial_delta <- min(which(cumsum(constant_delay_distribution) > 0.5)) - 1
```

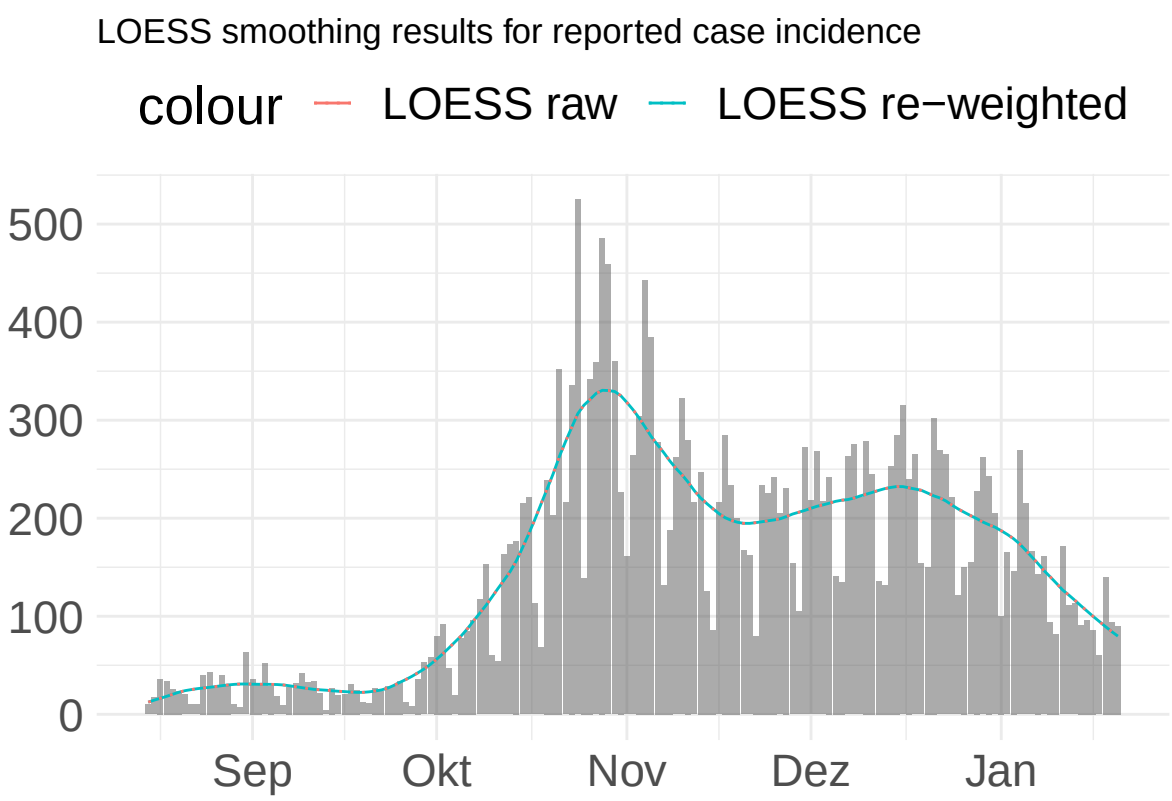
- Smoothing by LOESS

```
days_incl = 21
time_series <- data_subset
smoothed_incidence_data <- time_series %>%
        complete(date = seq.Date(min(date), max(date), by = "days"),
                 fill = list(value = 0)) %>%
        mutate(value = getLOESSCases(dates = date, count_data = value, days_incl))

      raw_total_incidence <- sum(time_series$value, na.rm = TRUE)
      smoothed_total_incidence <- sum(smoothed_incidence_data$value, na.rm = T)
## Reweighting to make sure that sums match
      if (smoothed_total_incidence > 0) {
        smoothed_incidence_data <- smoothed_incidence_data %>%
          mutate(value = value * raw_total_incidence / smoothed_total_incidence)
      }

sum(smoothed_incidence_data$value) == raw_total_incidence
```

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

```
smoothed_incidence_data_raw <- time_series %>%
        complete(date = seq.Date(min(date), max(date), by = "days"),
                 fill = list(value = 0)) %>%
        mutate(value = getLOESSCases(dates = date, count_data = value, days_incl))

ggplot() +
```

```
  geom_bar(aes(x=date, y= value), data = time_series, alpha = 0.5, show.legend = F,
           position = 'identity', stat = 'identity') +
  # geom_line(aes(x = date, y = value, color = 'raw'), data = time_series) +
  geom_line(aes(x= date, y = value, color = 'LOESS re-weighted'), data = smoothed_incidence_data) +
  geom_line(aes(x= date, y = value, color = 'LOESS raw'), linetype = 'dotted', data = smoothed_incidenc
  ggtitle('LOESS smoothing results for reported case incidence')+
  ylab('')+
  xlab('')
```



do_deconvolution()

- After putting together the necessary inputs the actual deconvolution function `do_deconvolution()` is called.
  - `smoothed_incidence_data`
  - `delay_distribution_matrix`
  - `initial_delta`: index at which cumsum of delay distribution vector exceeds median
- `do_deconvolution()` function:

```
do_deconvolution <- function(
  incidence_data,
  days_further_in_the_past = 30,
  verbose = FALSE,
  delay_distribution_matrix,
  initial_delta,
  max_iterations = 100
) {
```

```r
  # use mode of 'constant_delay_distribution'. -1 because indices are offset by one as the delay can be

  first_guess_delay <- ceiling(initial_delta)

  if (verbose) {
    cat("\tDelay on first guess: ", first_guess_delay, "\n")
  }

  first_recorded_incidence <-  with(filter(incidence_data, cumsum(value) > 0), value[which.min(date)])
  last_recorded_incidence <- with(incidence_data, value[which.max(date)])

  minimal_date <- min(incidence_data$date) - days_further_in_the_past
  maximal_date <- max(incidence_data$date)

  first_guess <- incidence_data %>%
    mutate(date = date - first_guess_delay) %>%
    complete(date = seq.Date(minimal_date, min(date), by = "days"),
             fill = list(value = first_recorded_incidence)) %>% # left-pad with first recorded value
    complete(date = seq.Date(max(date), maximal_date, by = "days"),
             fill = list(value = last_recorded_incidence)) %>% # right-pad with last recorded value
    arrange(date) %>%
    filter(date >=  minimal_date)

  original_incidence <- incidence_data %>%
    complete(date = seq.Date(minimal_date, maximal_date, by = "days"),
             fill = list(value = 0)) %>%
    pull(value)

  final_estimate <- iterate_RL(
    first_guess$value,
    original_incidence,
    delay_distribution_matrix = delay_distribution_matrix,
    max_delay = days_further_in_the_past,
    max_iterations = max_iterations,
    verbose = verbose)

  deconvolved_dates <- first_guess %>% pull(date)

  result <- tibble(date = deconvolved_dates, value = final_estimate)

  result <- result %>%
    filter(date <= maximal_date - first_guess_delay)

  return(result)
}
```

```r
# setting function arguments
bootstrap_replicate_i <- 0 # looping index
max_iterations <- 100
verbose <- FALSE
data_type_subset <- unique(data_subset$data_type)[1]
is_local_cases <- TRUE
incidence_data <- smoothed_incidence_data
```

```
  # use mode of 'constant_delay_distribution'. -1 because indices are offset by one as the delay can be

  (first_guess_delay <- ceiling(initial_delta))
```

```
## [1] 7
```

```
  if (verbose) {
    cat("\tDelay on first guess: ", first_guess_delay, "\n")
  }

  (first_recorded_incidence <-  with(filter(incidence_data, cumsum(value) > 0), value[which.min(date)]))
```

```
## [1] 12.38484
```

```
  (last_recorded_incidence <- with(incidence_data, value[which.max(date)]))
```

```
## [1] 79.74476
```

```
  minimal_date <- min(incidence_data$date) - days_further_in_the_past
  maximal_date <- max(incidence_data$date)

  incidence_data %>%
    head(10)
```

```
## # A tibble: 10 x 9
##    date       region value data_type source variable  country     date_type
##    <date>     <chr>  <dbl> <chr>     <chr>  <chr>     <chr>       <chr>
##  1 2020-08-15 CHE     12.4 confirmed ETH    incidence Switzerland report
##  2 2020-08-16 CHE     14.5 confirmed ETH    incidence Switzerland report
##  3 2020-08-17 CHE     16.4 confirmed ETH    incidence Switzerland report
##  4 2020-08-18 CHE     18.3 confirmed ETH    incidence Switzerland report
##  5 2020-08-19 CHE     20.5 confirmed ETH    incidence Switzerland report
##  6 2020-08-20 CHE     22.5 confirmed ETH    incidence Switzerland report
##  7 2020-08-21 CHE     24.1 confirmed ETH    incidence Switzerland report
##  8 2020-08-22 CHE     25.2 confirmed ETH    incidence Switzerland report
##  9 2020-08-23 CHE     26.2 confirmed ETH    incidence Switzerland report
## 10 2020-08-24 CHE     26.9 confirmed ETH    incidence Switzerland report
## # ... with 1 more variable: local_infection <lgl>
```

```
  incidence_data %>%
    tail(10)
```

```
## # A tibble: 10 x 9
##    date       region value data_type source variable  country     date_type
##    <date>     <chr>  <dbl> <chr>     <chr>  <chr>     <chr>       <chr>
##  1 2021-01-11 CHE    127.  confirmed ETH    incidence Switzerland report
##  2 2021-01-12 CHE    122.  confirmed ETH    incidence Switzerland report
##  3 2021-01-13 CHE    116.  confirmed ETH    incidence Switzerland report
##  4 2021-01-14 CHE    111.  confirmed ETH    incidence Switzerland report
##  5 2021-01-15 CHE    105.  confirmed ETH    incidence Switzerland report
##  6 2021-01-16 CHE     99.7 confirmed ETH    incidence Switzerland report
##  7 2021-01-17 CHE     94.6 confirmed ETH    incidence Switzerland report
##  8 2021-01-18 CHE     89.3 confirmed ETH    incidence Switzerland report
##  9 2021-01-19 CHE     84.3 confirmed ETH    incidence Switzerland report
## 10 2021-01-20 CHE     79.7 confirmed ETH    incidence Switzerland report
```

```
## # ... with 1 more variable: local_infection <lgl>
```

```
first_guess <- incidence_data %>%
  mutate(date = date - first_guess_delay) %>%
  complete(date = seq.Date(minimal_date, min(date), by = "days"),
           fill = list(value = first_recorded_incidence)) %>% # left-pad with first recorded value
  complete(date = seq.Date(max(date), maximal_date, by = "days"),
           fill = list(value = last_recorded_incidence)) %>% # right-pad with last recorded value
  arrange(date) %>%
  filter(date >=  minimal_date)

first_guess %>%
  head(10)
```

```
## # A tibble: 10 x 9
##    date       region value data_type source variable country date_type
##    <date>     <chr>  <dbl> <chr>     <chr>  <chr>    <chr>   <chr>
##  1 2020-07-16 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  2 2020-07-17 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  3 2020-07-18 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  4 2020-07-19 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  5 2020-07-20 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  6 2020-07-21 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  7 2020-07-22 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  8 2020-07-23 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
##  9 2020-07-24 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
## 10 2020-07-25 <NA>    12.4 <NA>      <NA>   <NA>     <NA>    <NA>
## # ... with 1 more variable: local_infection <lgl>
```

```
first_guess %>%
  tail(10)
```

```
## # A tibble: 10 x 9
##    date       region value data_type source variable  country     date_type
##    <date>     <chr>  <dbl> <chr>     <chr>  <chr>     <chr>       <chr>
##  1 2021-01-11 CHE     89.3 confirmed ETH    incidence Switzerland report
##  2 2021-01-12 CHE     84.3 confirmed ETH    incidence Switzerland report
##  3 2021-01-13 CHE     79.7 confirmed ETH    incidence Switzerland report
##  4 2021-01-14 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
##  5 2021-01-15 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
##  6 2021-01-16 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
##  7 2021-01-17 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
##  8 2021-01-18 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
##  9 2021-01-19 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
## 10 2021-01-20 <NA>    79.7 <NA>      <NA>   <NA>      <NA>        <NA>
## # ... with 1 more variable: local_infection <lgl>
```

```
original_incidence <- incidence_data %>%
  complete(date = seq.Date(minimal_date, maximal_date, by = "days"),
           fill = list(value = 0)) %>%
  pull(value)



final_estimate <- iterate_RL(
  first_guess$value,
```

```
    original_incidence,
    delay_distribution_matrix = delay_distribution_matrix,
    max_delay = days_further_in_the_past,
    max_iterations = max_iterations,
    verbose = verbose)

  deconvolved_dates <- first_guess %>% pull(date)

  result <- tibble(date = deconvolved_dates, value = final_estimate)

  result <- result %>%
    filter(date <= maximal_date - first_guess_delay)

  restult_RL <- result
```

iterate_RL()

- `do_deconvolution()` is another wrapper function for `iterate_RL()`, which implements the Richardson-Lucy Algorithm.
- This algorithm starts with a shifted and extended version of the original dataset as `initial_estimate = first_guess`. Specifically, the time series of reported cases is shifted by 7 days in the past, 30 days are added to the left and the resulting 30 missing values left and 7 right are imputed using the first and last observation of reported cases, respectively.
- Additional inputs to the function are:
  - `original_incidence`: smoothed original time series of cases
  - `delay_distribution_matrix`: The matrix presented above
  - `threshold_chi_squared = 1`
  - `max_iterations = 100`
  - `max_delay = 30`

Function

```
iterate_RL <- function(
  initial_estimate,
  original_incidence,
  delay_distribution_matrix,
  threshold_chi_squared = 1,
  max_iterations = 100,
  max_delay,
  verbose = FALSE) {

  current_estimate <- initial_estimate
  N <- length(current_estimate)
  N0 <- N - max_delay
  chi_squared <- Inf
  count <- 1

  delay_distribution_matrix <- delay_distribution_matrix[1:length(current_estimate), 1:length(current_es
  truncated_delay_distribution_matrix <- delay_distribution_matrix[(1 + max_delay):NROW(delay_distributi
```

```
  Q_vector <- apply(truncated_delay_distribution_matrix, MARGIN = 2, sum)

  while(chi_squared > threshold_chi_squared & count <= max_iterations) {

    if (verbose) {
      cat("\t\tStep: ", count, " - Chi squared: ", chi_squared, "\n")
    }

    E <- as.vector(delay_distribution_matrix %*% current_estimate)
    B <- replace_na(original_incidence/E, 0)

    current_estimate <- current_estimate / Q_vector *  as.vector(crossprod(B, delay_distribution_matrix)
    current_estimate <- replace_na(current_estimate, 0)

    chi_squared <- 1/N0 * sum((E[(max_delay + 1): length(E)] - original_incidence[(max_delay + 1) : leng
    count <- count + 1
  }

  return(current_estimate)
}
```

Stepwise examination

```
# setting missing arguments
initial_estimate <- first_guess$value
max_delay <- days_further_in_the_past
threshold_chi_squared <- 1

# initialize RL algorithm: starting value is
  current_estimate <- initial_estimate
  # length of output series
  N <- length(current_estimate)
  # length of original series
  N0 <- N - max_delay
  chi_squared <- Inf
  count <- 1

  # subset delay_distribution_matrix
  delay_distribution_matrix <- delay_distribution_matrix[1:length(current_estimate), 1:length(current_es

  # subset delay_distribution_matrix to obtain a 159 x 189 matrix
  # in order to compute weighting vector
  truncated_delay_distribution_matrix <- delay_distribution_matrix[(1 + max_delay):NROW(delay_distributi

  # column sums (sum of weights from original series to incidence estimate)
  Q_vector <- apply(truncated_delay_distribution_matrix, MARGIN = 2, sum)

  while(chi_squared > threshold_chi_squared & count <= max_iterations) {

    if (verbose) {
      cat("\t\tStep: ", count, " - Chi squared: ", chi_squared, "\n")
    }
```

```
    # P x u (wiki)
    E <- as.vector(delay_distribution_matrix %*% current_estimate)
    # d / (P x u)
    B <- replace_na(original_incidence/E, 0)

    current_estimate <- current_estimate / Q_vector *  as.vector(crossprod(B, delay_distribution_matrix

    current_estimate <- replace_na(current_estimate, 0)


    chi_squared <- 1/N0 * sum((E[(max_delay + 1): length(E)] - original_incidence[(max_delay + 1) : leng
    count <- count + 1
  }

(final_estimate <-  current_estimate)
```

```
##   [1]  12.03129  12.01429  11.97833  11.95466  11.92945  11.94037  11.98400
##   [8]  11.98460  11.98466  11.99222  11.99631  12.00785  12.01729  12.02201
##  [15]  12.04036  12.05631  12.07801  12.09261  12.11959  12.15131  12.18766
##  [22]  12.22819  12.27919  12.34037  14.54178  16.53884  18.66581  21.11161
##  [29]  23.55298  25.51549  26.93477  28.05062  28.80114  29.38114  30.11299
##  [36]  30.86677  31.46687  31.96849  32.21024  32.09833  31.77875  31.50753
##  [43]  31.32591  31.06703  30.72711  30.14189  29.15640  28.00582  26.87387
##  [50]  25.78023  24.81179  24.16521  23.65563  22.99370  22.45045  22.08068
##  [57]  21.59997  21.26583  21.28476  21.79234  22.69131  23.84333  25.74037
##  [64]  28.64571  32.06017  35.34603  39.00140  43.16522  47.47466  52.40903
##  [71]  57.98623  63.93842  70.34174  77.00854  83.61032  91.24982  99.78046
##  [78] 107.88923 116.16043 125.22731 134.52248 143.44985 152.62299 164.00945
##  [85] 177.99264 192.84128 208.72858 225.99939 243.19068 260.76497 280.11675
##  [92] 296.85747 312.76462 327.81731 336.66821 342.07027 347.32129 348.77687
##  [99] 346.25926 342.79042 335.69048 325.84509 315.55403 304.33770 292.18620
## [106] 280.35170 270.24136 260.90075 251.20727 242.85431 235.60297 227.65024
## [113] 218.96912 211.71971 206.02085 200.85696 196.04262 192.29178 190.04285
## [120] 188.71099 188.27828 188.99994 190.54861 192.32490 194.17451 195.90618
## [127] 197.88651 200.82107 203.96784 206.34525 208.71727 211.30716 213.66360
## [134] 215.38870 217.25633 219.18098 220.36835 221.07242 222.60904 224.79407
## [141] 226.65334 228.60672 230.73977 232.65494 234.00518 234.94542 234.70448
## [148] 233.19836 231.63610 229.87113 226.54198 223.13174 220.40247 216.67768
## [155] 211.88142 207.51302 203.70518 200.11499 196.72238 193.11841 190.19042
## [162] 187.54769 183.93546 179.97338 175.48214 169.82303 162.95004 155.59946
## [169] 148.25667 140.53153 133.69992 126.75294 119.81223 114.39011 108.82158
## [176] 103.14554  97.71854  92.29851  87.17600  81.96468  76.97901  72.34955
## [183]  71.88838  71.40393  70.91009  70.43287  70.00383  69.68736  69.64088
```

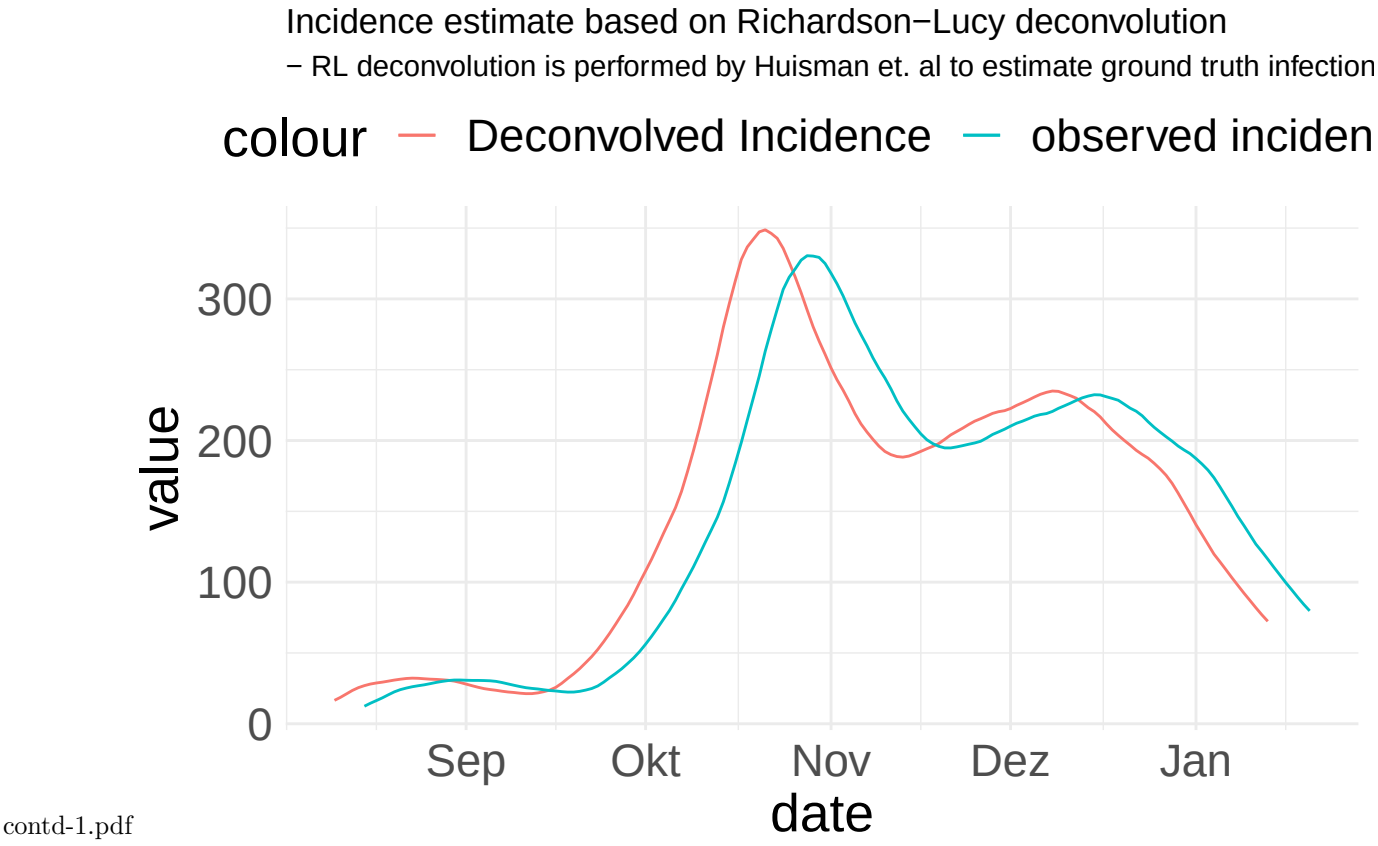
Further stepwise processing

```
## do_deconvolution()
deconvolved_dates <- first_guess %>%
  pull(date)

  result <- tibble(date = deconvolved_dates, value = final_estimate)
```

```
  result <- result %>%
    filter(date <= maximal_date - first_guess_delay)

result %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    date       value
##    <date>     <dbl>
##  1 2020-07-16  12.0
##  2 2020-07-17  12.0
##  3 2020-07-18  12.0
##  4 2020-07-19  12.0
##  5 2020-07-20  11.9
##  6 2020-07-21  11.9
##  7 2020-07-22  12.0
##  8 2020-07-23  12.0
##  9 2020-07-24  12.0
## 10 2020-07-25  12.0
```

```
deconvolved_infections <- result
## get_infection_incidence_by_deconvolution()

deconvolved_infections <- deconvolved_infections %>% slice((days_further_in_the_past -5 + 1):n())

    data_type_name <- paste0("infection_", data_type_subset)

    ## dataframe containing results
    deconvolved_infections <- tibble(
      date = deconvolved_infections$date,
      region = unique(time_series$region)[1],
      country = unique(time_series$country)[1],
      source = unique(time_series$source)[1],
      local_infection = is_local_cases,
      data_type = data_type_name,
      replicate = bootstrap_replicate_i,
      value = deconvolved_infections$value
    )


ggplot() +
  geom_line(aes(x = date, y = value, color = "Deconvolved Incidence"), data = deconvolved_infections) +
  geom_line(aes(x = date, y = value, color = "observed incidence"), data = incidence_data) +
  ggtitle("Incidence estimate based on Richardson-Lucy deconvolution",
          subtitle = "- RL deconvolution is performed by Huisman et. al to estimate ground truth infecti
```



contd-1.pdf

Richardson-Lucy deconvolution applied to cases

- Iterative deconvolution procedure to recover underlying image (or time series in our case) that has been blurred by a *known* point spread function (PSF)
- Converges to the Maximum-Likelihood (MLE) solution under the assumption that the data follows a poisson distribution, which is great for us as we have count data.
- Recall our original deconvolution problem: $C = I_{cc} * D$ where we are interested in estimating the ground truth infection incidence time series I_{cc}.
- Then, $C_t = \sum_j \mathbf{P}_{t,j} \cdot I_{cc}^j$ are the observed cases at day t, where $\mathbf{P}$ is `delay_distribution_matrix`. As mentioned above, this matrix is a T by T lower-diagonal matrix with the vectorized delay distributions as columns starting at the diagonal element created by the function `get_matrix_constant_waiting_time_distr()`. (see above for illustration).
- Following the RL algorithm:

$$I_{cc}^{i+1} = I_{cc}^i \left(\frac{C}{\mathbf{P} \cdot I_{cc}^i} \times \mathbf{P} \right)$$

  , where the division is elementwise.
- For our purpose, however, we need a small adjustment to correct for the fact that we might observe infections that are not fully transmitted into the wastewater:

$$I_{cc}^{i+1} = \frac{I_{cc}^i}{\mathbf{P}^{\mathbf{T}}_{[31:189,1:189]} \cdot \mathbf{1}} \left(\frac{C}{\mathbf{P} \cdot I_{cc}^i} \times \mathbf{P} \right)$$

- RL deconvolution in R

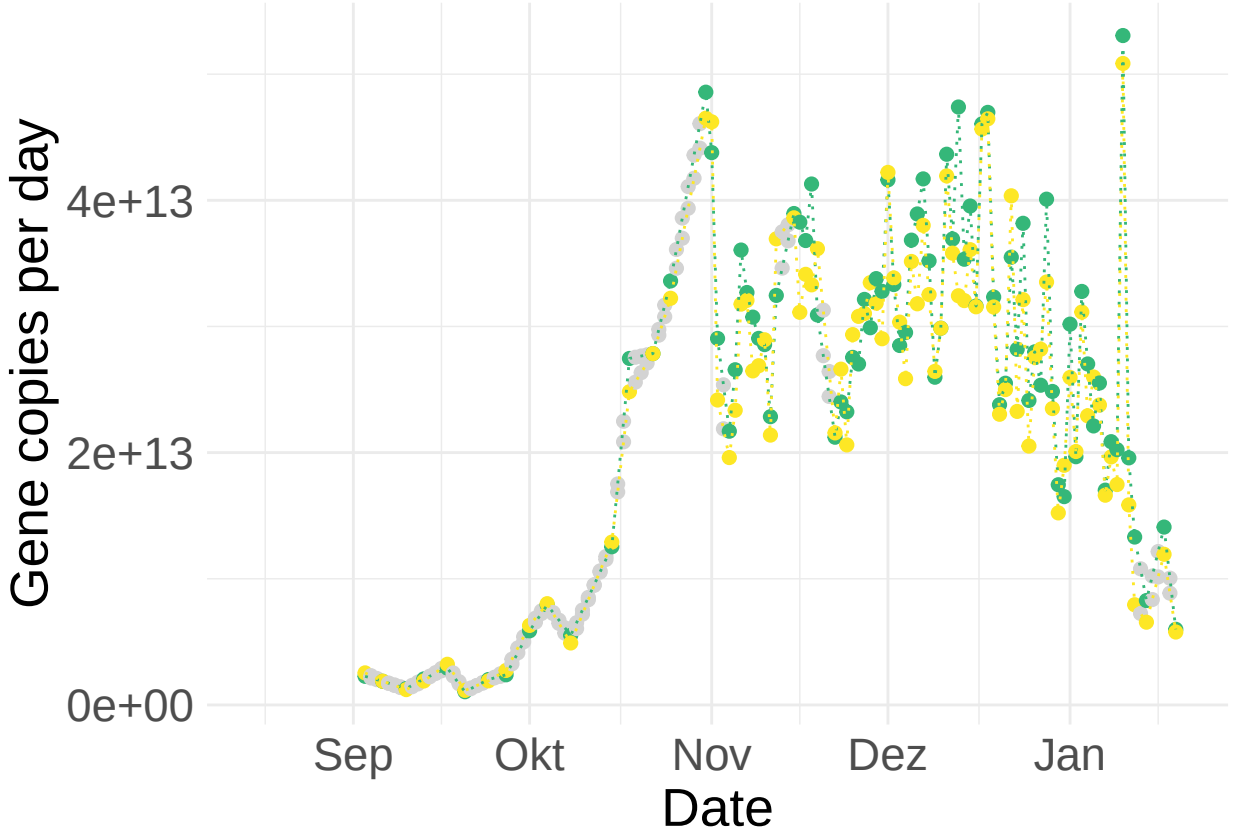
```
## RL deconvolution in R
  Q_vector <- apply(truncated_delay_distribution_matrix, MARGIN = 2, sum)

  while(chi_squared > threshold_chi_squared & count <= max_iterations) {

    if (verbose) {
      cat("\t\tStep: ", count, " - Chi squared: ", chi_squared, "\n")
    }
    # P x u (wiki)
    E <- as.vector(delay_distribution_matrix %*% current_estimate)
    # d / (P x u)
    B <- replace_na(original_incidence/E, 0)

  current_estimate <- current_estimate / Q_vector *  as.vector(crossprod(B, delay_distribution_matrix))

    current_estimate <- replace_na(current_estimate, 0)


    chi_squared <- 1/N0 * sum((E[(max_delay + 1): length(E)] - original_incidence[(max_delay + 1) : leng
    count <- count + 1
```

- Normalisation of Wastewater Data

```
## Normalisation of WW data ####

# Min observed
norm_min <- min(raw_data_ZH$n1, raw_data_ZH$n2)

### ALL Normalised WW DATA ####
ww_data = bind_rows(raw_data_ZH)  %>%
  mutate(norm_n1 = n1/norm_min,
         norm_n2 = n2/norm_min)

ww_data
```

```
## # A tibble: 139 x 12
##    date       cases cases_smooth    flow n1_smooth n2_smooth      n1      n2
##    <date>     <dbl>        <dbl>   <dbl>     <dbl>     <dbl>   <dbl>   <dbl>
##  1 2020-09-03  53           33   158459    1.81e12   2.04e12 2.27e12 2.54e12
##  2 2020-09-04  38.7         33.4 156052.   1.94e12   2.18e12 2.14e12 2.32e12
##  3 2020-09-05  24.3         33.8 153644.   2.06e12   2.31e12 2.00e12 2.10e12
##  4 2020-09-06  10           34.1 151237    2.19e12   2.45e12 1.87e12 1.88e12
##  5 2020-09-07  15.5         32.5 147888.   2.02e12   2.18e12 1.73e12 1.71e12
##  6 2020-09-08  21           30.9 144540.   1.85e12   1.92e12 1.58e12 1.55e12
##  7 2020-09-09  26.5         29.3 141191.   1.68e12   1.66e12 1.44e12 1.38e12
##  8 2020-09-10  32           27.7 137842    1.51e12   1.39e12 1.30e12 1.22e12
##  9 2020-09-11  22.7         27.9 132288.   1.57e12   1.45e12 1.55e12 1.44e12
## 10 2020-09-12  13.3         28.1 126733.   1.62e12   1.50e12 1.81e12 1.67e12
## # ... with 129 more rows, and 4 more variables: orig_data <lgl>, region <chr>,
## #   norm_n1 <dbl>, norm_n2 <dbl>
```

```r
# Plot raw data #####
plot_raw_ww_data <- ww_data %>%
  dplyr::select(date, orig_data, region, N1 = n1, N2 = n2) %>%
  pivot_longer(cols = c(N1, N2)) %>%
  mutate(name_orig = ifelse(!is.na(orig_data), name, 'Imputed'))



ww_data_plot <- ggplot() +
  geom_point(data = plot_raw_ww_data, aes(x=date, y= value, colour = name_orig),
             size = 2, show.legend = F) +
  geom_line(data = plot_raw_ww_data %>% filter(orig_data),
            aes(x=date, y= value,colour = name), linetype = 'dotted', show.legend = F) +
  labs(x = 'Date' , y='Gene copies per day') +
  scale_x_date(limits = c(as_date('2020-08-15'), as_date('2021-01-20')) ) +
  scale_colour_manual(values = c(viridis(4)[3:4], 'lightgrey'),
                      labels = c('N1', 'N2', 'Imputed'),
                      breaks = c('N1', 'N2', 'Imputed'),
                      name = 'Variable') +
  labs(colour = 'Variable')

ww_data_plot
```



- Deconvolution of case data with given SLD

```
##### Deconvolve and Estimate WW Re #####

config_df = expand.grid("region" = c('ZH'),
                        'incidence_var' = c('norm_n1', 'norm_n2'),
                        'FirstGamma' = 'incubation',
                        'SecondGamma' = 'benefield' )

config_df
```

```
##   region incidence_var FirstGamma SecondGamma
## 1     ZH       norm_n1 incubation   benefield
## 2     ZH       norm_n2 incubation   benefield
```

```
getCountParams(as.character(config_df[1, 'FirstGamma']))
```

```
## $shape
## [1] 2.743164
##
## $scale
## [1] 1.932075
```

```
getCountParams(as.character(config_df[1, 'SecondGamma']))
```

```
## $shape
## [1] 0.929639
##
## $scale
## [1] 7.241397
```

```
getCountParams(as.character(config_df[2, 'FirstGamma']))
```

```
## $shape
## [1] 2.743164
##
## $scale
## [1] 1.932075
```

```
getCountParams(as.character(config_df[2, 'SecondGamma']))
```

```
## $shape
## [1] 0.929639
##
## $scale
## [1] 7.241397
```

```
deconv_ww_data <- data.frame()
Re_ww <- data.frame()

for(row_i in 1:nrow(config_df)){
  new_deconv_data = deconvolveIncidence(ww_data %>% filter(region == config_df[row_i, 'region']),
                                        incidence_var = config_df[row_i, 'incidence_var'],
                                        getCountParams(as.character(config_df[row_i, 'FirstGamma'])),
                                        getCountParams(as.character(config_df[row_i, 'SecondGamma'])),
                                        smooth_param = TRUE, n_boot = 50)
```

```
  new_deconv_data <- new_deconv_data %>%
    mutate(incidence_var = config_df[row_i, 'incidence_var'])

  ##### Get Re #####
  new_Re_ww = getReBootstrap(new_deconv_data)
  new_Re_ww <- new_Re_ww %>%
    mutate(variable = config_df[row_i, 'incidence_var'],
           region = config_df[row_i, 'region'])

  deconv_ww_data <- bind_rows(deconv_ww_data, new_deconv_data)
  Re_ww = bind_rows(Re_ww, new_Re_ww)
}
```

```
## estimating Re for data source:  ETH ...
##   Region:  ZH
##     Data type:  infection_norm_n1
## estimating Re for data source:  ETH ...
##   Region:  ZH
##     Data type:  infection_norm_n2
```

- Scanning of different SLDs

- They are scanning across 600 different combinations of mean x sd.

```
#Scan across deconvolution parameters ####

# Uncommented because this takes about 2 hrs per scan
 for (incubationParam in c('zero', 'incubation')){
   for (incidence_var in c('norm_n1')){
     for (canton in c('ZH')){

       proc_data <- ww_data %>% filter(region == canton)
       plotData_subset <- plotData %>% filter(region == canton)

       # the scan values
       meanOpts = seq(0.5, 15, 0.5)
       sdOpts = seq(0.5, 10, 0.5)

       deconv_results = cbind(expand_grid(meanOpts, sdOpts),
                              'rmse_cc' = NA, 'coverage_cc' = NA, 'mape_cc' = NA,
                              'rmse_h' = NA, 'coverage_h' = NA, 'mape_h' = NA)

       for (row_id in 1:nrow(deconv_results)){
         deconv_config = try(deconvolveIncidence(proc_data, incidence_var,
                                                 getCountParams(incubationParam),
                                                 getGammaParams(deconv_results[row_id, 'meanOpts'],
                                                                deconv_results[row_id, 'sdOpts']),
                                                 smooth_param = TRUE, n_boot = 50))

         if('try-error' %in% class(deconv_config)){
           deconv_results[row_id, c('rmse_cc', 'coverage_cc', 'mape_cc')] = c(Inf, 0, Inf)
           deconv_results[row_id, c('rmse_h', 'coverage_h', 'mape_h')] = c(Inf, 0, Inf)
           next
```

```
        }

        Re_config = getReBootstrap(deconv_config)

        deconv_results[row_id, c('rmse_cc', 'coverage_cc', 'mape_cc')] = compareTraces(Re_config, plotl
                                                                                       filter(data_ty
      }

      write_csv(deconv_results, paste0('../scan/deconv_', incubationParam, '_',
                                       canton, '_', incidence_var, '.csv'))

    }
  }
}

###############
# Results of the scan

# Where:

Restimates <- Re_cases %>%
  mutate(region = 'ZH') %>%
  mutate(data_type = recode_factor(data_type,
                           infection_confirmed = "Confirmed cases",
                           infection_hospitalised = "Hospitalized patients",
                           infection_death = "Deaths") ) %>%
  mutate(countryIso3 = 'CHE')

date_ranges <- Re_ww %>%
  group_by(region) %>%
  summarise(min_date = min(date),
            max_date = max(date))

plotData <- Restimates %>%
  filter(region %in% c('ZH'),
         estimate_type == 'Cori_slidingWindow',
         date >= date_ranges[date_ranges$region == 'ZH', ]$min_date,
         date <= date_ranges[date_ranges$region == 'ZH',]$max_date )
```

```
compareTraces <- function(Re_i, Re_j){
  compare_df = Re_i %>%
    left_join(Re_j, by = 'date', suffix = c(".i", ".j")) %>%
    mutate(se = (median_R_mean.i - median_R_mean.j)^2,
           rele = abs((median_R_mean.j - median_R_mean.i)/median_R_mean.j),
           coverage = (median_R_mean.i > median_R_lowHPD.j) & (median_R_mean.i < median_R_highHPD.j) )

  se = compare_df %>% pull(se)
  rele = compare_df %>% pull(rele)
  coverage = compare_df %>% pull(coverage) %>% sum(na.rm = T) /length(Re_i$date)

  rmse = sqrt(sum(se, na.rm = T)/length(Re_i$date))
  mape = sum(rele, na.rm = T)/length(Re_i$date)

  return(c(rmse, coverage, mape))
```

```
}
```