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

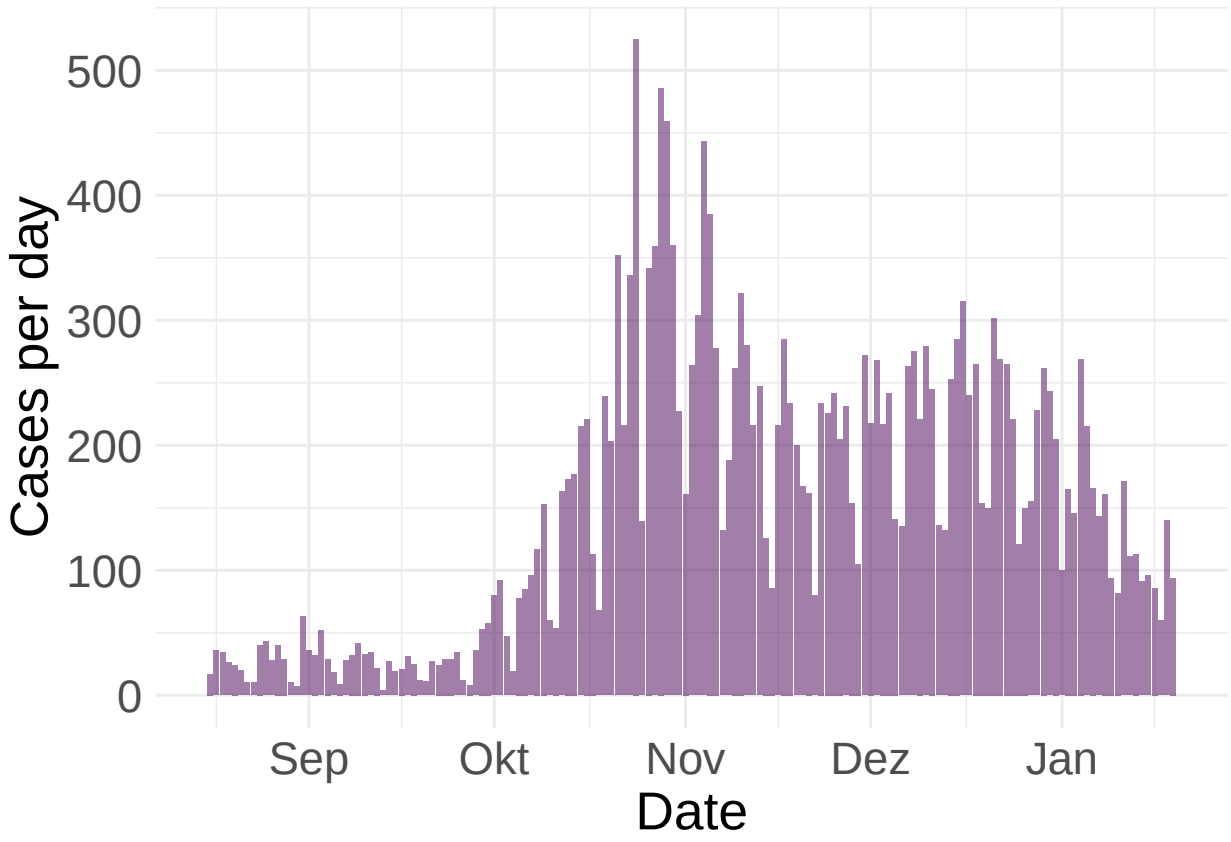
- flow data is loaded with col_names = c('date', 'cases', 'cases_smooth','flow', 'n1_smooth', 'n2_smooth')
- gene data is loaded with col_names = c('date', 'n1', 'n2'),
- missing values in flow data are imputed by linear interpolation:

```
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] 0.000271 0.009958 0.036702 0.066930 0.090491 0.103378 0.105766 0.101069
##   [9] 0.091498 0.079412 0.066541 0.055174 0.044308 0.035169 0.027421 0.021097
##  [17] 0.016365 0.012553 0.009241 0.006998 0.005134 0.003786 0.003024 0.002114
##  [25] 0.001479 0.001118 0.000840 0.000567 0.000415 0.000331 0.000223 0.000171
##  [33] 0.000125 0.000090 0.000067 0.000050 0.000030 0.000022 0.000027 0.000014
##  [41] 0.000004 0.000006 0.000004 0.000006 0.000003 0.000003 0.000002 0.000001
##  [49] 0.000000 0.000001 0.000000 0.000001 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
##
## $Symptoms
##   [1] 0.004607 0.060842 0.119950 0.146859 0.144965 0.127522 0.104273 0.081281
##   [9] 0.061620 0.044772 0.032270 0.022803 0.015629 0.010728 0.007410 0.004959
##  [17] 0.003283 0.002125 0.001411 0.000936 0.000634 0.000401 0.000265 0.000161
##  [25] 0.000107 0.000067 0.000041 0.000033 0.000022 0.000010 0.000003 0.000004
##  [33] 0.000002 0.000002 0.000001 0.000001 0.000000 0.000000 0.000001 0.000000
##  [41] 0.000000 0.000000 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]])
```

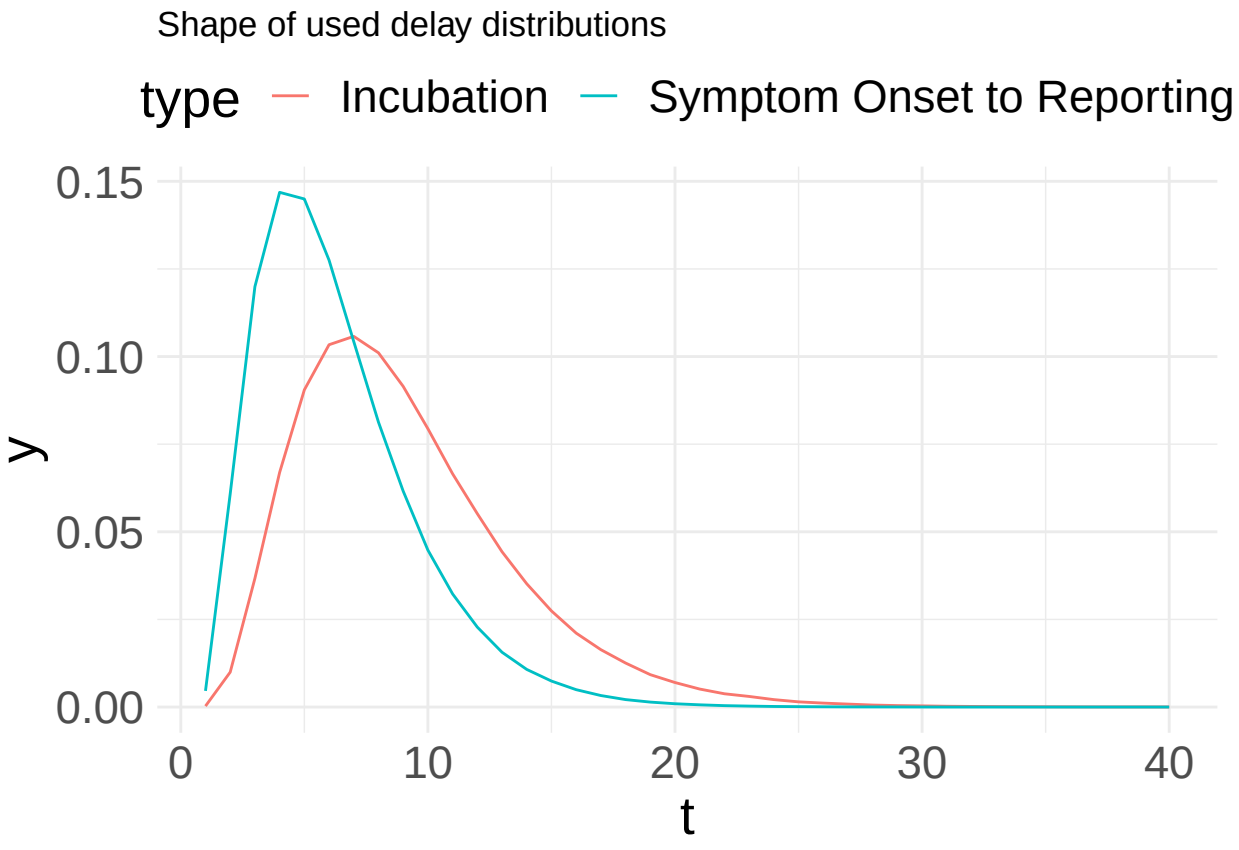
```
## [1] 1
```

```
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,] 0.000276 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [2,] 0.010062 0.000276 0.000000 0.000000 0.000000 0.000000 0.000000
## [3,] 0.036610 0.010062 0.000276 0.000000 0.000000 0.000000 0.000000
## [4,] 0.067133 0.036610 0.010062 0.000276 0.000000 0.000000 0.000000
## [5,] 0.090271 0.067133 0.036610 0.010062 0.000276 0.000000 0.000000
## [6,] 0.103354 0.090271 0.067133 0.036610 0.010062 0.000276 0.000000
## [7,] 0.106364 0.103354 0.090271 0.067133 0.036610 0.010062 0.000276
```

```
# 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
```

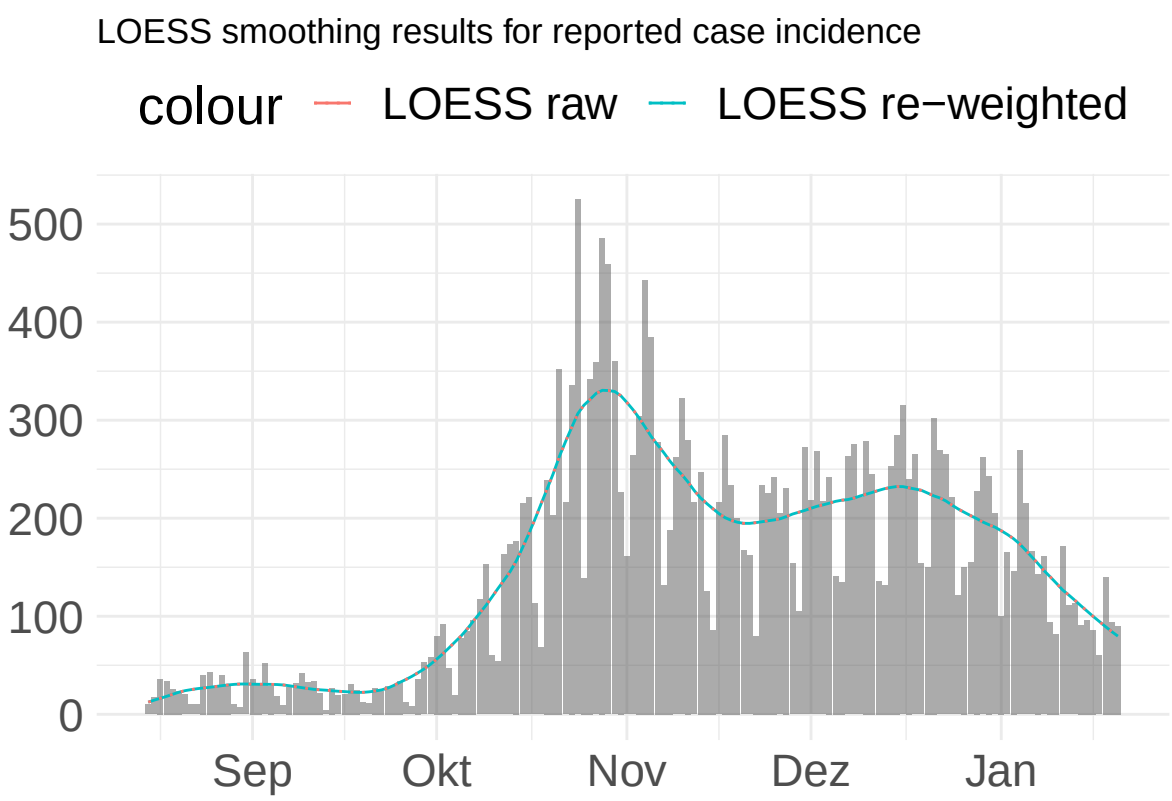
- Smoothing by LOESS

```
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() +
```

```r
  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 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 the

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

  # 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)
}
```

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

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
    )
```

```
new_deconv_data = deconvolveIncidence(orig_cases,
                                        incidence_var = inc_var,
                                        getCountParams('incubation'),
                                        getCountParams(paste0(inc_var, '_zh')),
                                        smooth_param = TRUE, n_boot = 50)
```