Multiscale Inference for NonParametric Time Trends

Marina Khismatullina Michael Vogt

July 30, 2020

We present the R package 'multiscale', which performs muliscale tests for nonparametric time trends.

Contents

1 Introduction

The main functions of the **multiscale** package are given in the following list:

- `compute_quantiles()`: Computes the quantiles of the Gaussian version of the statistics that are used to approximate the critical values for the multiscale test; see Sections 2 and 3.
- `compute_statistics()`: Computes the value of the test statistics based on a single time series or multiple time series supplied; see Sections 2 and 3.
- `multiscale_test()`: Performs the test; see Sections 2 and 3.
- `estimate_lrv`: Computes the estimator for the long-run variance of the errors in a nonparametric regression model; see Section 4.

To the best of our knowledge, our **multiscale** package is the first software package that offers the estimation methods of **?** and **?**.

To demonstrate the use of our functions, we analyse two datasets. In order to illustrate the method from **?**, we examine the Central England temperature record, which is the longest instrumental temperature time series in the world. The data are publicly available on the webpage of the UK Met Office. A detailed description of the data can be found in **?**. In order to illustrate the method from **?**, we examine the daily number of infections of COVID-19 across different countries. The data are freely available on the homepage of the European Center for Disease Prevention and Control (`https://www.ecdc.europa.eu`) and were downloaded on 20 July 2020.

The temperature dataset can be obtained from the **multiscale** package using the function data(temperature, package = "multiscale"). The COVID-19 dataset can be obtained from the **multiscale** package using the function data(covid, package = "multiscale").

This vignette is organized as follows. Section 2 presents our mutliscale test for analysing a single time trend as in **?** and the results of applying it to the temperature data. Section 3 describes the multiscale procedure for comparing different time trends as in **?** and displays the results of analysing the COVID-19 data with the help of our test. Section 4 introduces the estimator of the long-run variance which is needed for analyzing a nonparametric regression with errors of class AR(p). Section 5 concludes.

2 Multiscale Inference for a Single Nonparametric Regression with Time Series Errors

In development.

3 Multiscale Inference for Multiple Nonparametric Regressions

As an illustration for the multiscale method proposed in **?**, we analyse the dataset on the daily new cases of infections of COVID-19. The data are freely available on the homepage of the European Center for Disease Prevention and Control (`https://www.ecdc.europa.eu`) and were downloaded on 20 July 2020. You can load the data using the function data(covid, package = "multiscale").

```
require(multiscale)

## Loading required package:  multiscale
```

```
data(covid, package = "multiscale")
str(covid)
```

```
##  num [1:148, 1:42] 15 8 27 25 26 43 0 35 29 38 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:42] "AFG" "ARG" "BEL" "BGD" ...
```

Each entry in the dataset denotes the number of new cases of infection per day and per country. In our dataset, we have data for 42 countries and the longest time series consists of 148 observations.

We assume that the outbreak patterns in different countries follow quasi-Poisson distribution with time-varying intensity parameters. Specifically, we let X_{it} be the number of newly confirmed COVID-19 cases on day t in country i and suppose X_{it} satisfy the following nonparametric regression equation:

$$X_{it} = \lambda_i\Big(\frac{t}{T}\Big) + \sigma\sqrt{\lambda_i\Big(\frac{t}{T}\Big)}\eta_{it}, \tag{1}$$

for $1 \leq t \leq T$ and $1 \leq i \leq n$, where σ is so-called overdispersion parameter that controls the noise variance, and the noise residuals η_{it} have zero mean and unit variance.

In model (1), the outbreak pattern of COVID-19 in country i is determined by the intensity function λ_i. Hence, the question whether the outbreak patterns are comparable across countries amounts to the question whether the intensity functions λ_i have the same shape across countries i.

In order to make the data comparable across countries, we take the day of the 100th confirmed case in each country as the starting date $t = 1$. Obviously, for some countries we have longer time series than for the others because the starting point of the outbreak varies across the countries. For the sake of brevity, we present here the analysis only of the data from five European countries: Germany, Italy, Spain, France and the United Kingdom:

```
covid <- covid[, c("DEU", "GBR", "ESP", "FRA", "ITA")]
covid <- na.omit(covid)
```

As a result, we study $n = 5$ time series of the sample size $T = 137$:

```
n     <- ncol(covid)
t_len <- nrow(covid)
n
```

```
## [1] 5
```

```
t_len
```

```
## [1] 137
```

Some of the time series contain negative values which we replaced by 0. Overall, this resulted in 6 replacements:

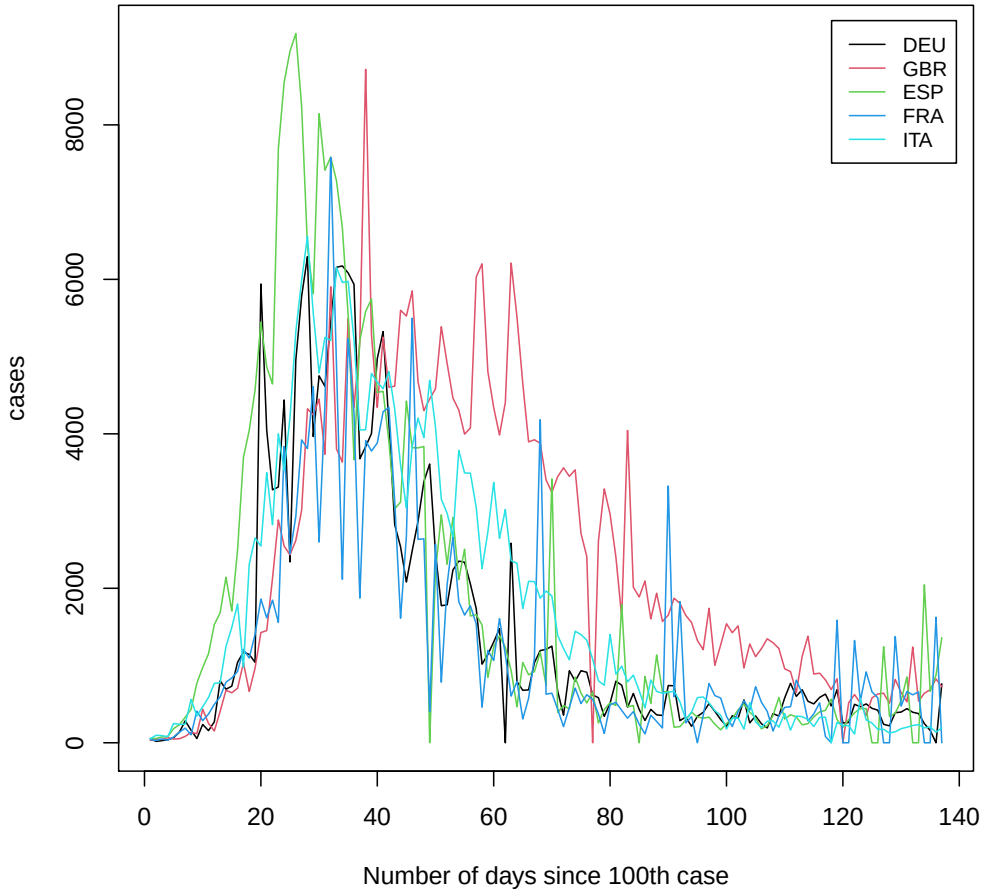
```
sum(covid < 0)
```

```
## [1] 6
```

```
covid[covid < 0] <- 0
```

Here are the plots of the time series:

```
matplot(1:t_len, covid, type = 'l', lty = 1, col = 1:t_len,
        xlab = 'Number of days since 100th case', ylab = 'cases')
legend("topright", legend = c("DEU", "GBR", "ESP", "FRA", "ITA"),
       inset = 0.02, lty = 1, col = 1:t_len, cex = 0.8)
```



In order to be able to implement the test, we first estimate the overdispersion parameter σ. For or each country i, let

$$\hat{\sigma}_i^2 = \frac{\sum_{t=2}^{T}(X_{it} - X_{it-1})^2}{2\sum_{t=1}^{T} X_{it}}$$

and set $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\hat{\sigma}_i^2$. As shown in **?**, $\hat{\sigma}^2$ is a consistent estimator of σ^2 under some regularity conditions.

```
sigma_vec <- rep(0, n)
for (i in 1:n){
  diffs <- (covid[2:t_len, i] - covid[1:(t_len - 1), i])
  sigma_squared <- sum(diffs^2) / (2 * sum(covid[, i]))
  sigma_vec[i] <- sqrt(sigma_squared)
}
```

```r
sigmahat <- sqrt(mean(sigma_vec * sigma_vec))
sigmahat
```

```
## [1] 14.43772
```

Throughout the section, we set the significance level to $\alpha = 0.05$ and the number of the simulations for producing critical values to 5000:

```r
alpha    <- 0.05
sim_runs <- 5000
```

Furthermore, we compare all pairs of countries (i, j) with $i < j$ (hence, $\mathcal{S} = \{1 \leq i < j \leq n\}$), and we choose the family of intervals $\mathcal{F}$ for calculating the test statistics as follows. We consider the intervals of lengths 7 days (1 week), 14 days (2 weeks), 21 days (3 weeks), or 28 days (4 weeks). For each length of the interval, we include all intervals that start at days $t = 1 + 7(j-1)$ and $t = 4 + 7(j-1)$ for $j = 1, 2, \ldots$.

```r
ijset           <- expand.grid(i = 1:n, j = 1:n)
ijset           <- ijset[ijset$i < ijset$j, ]
rownames(ijset) <- NULL
ijset
```

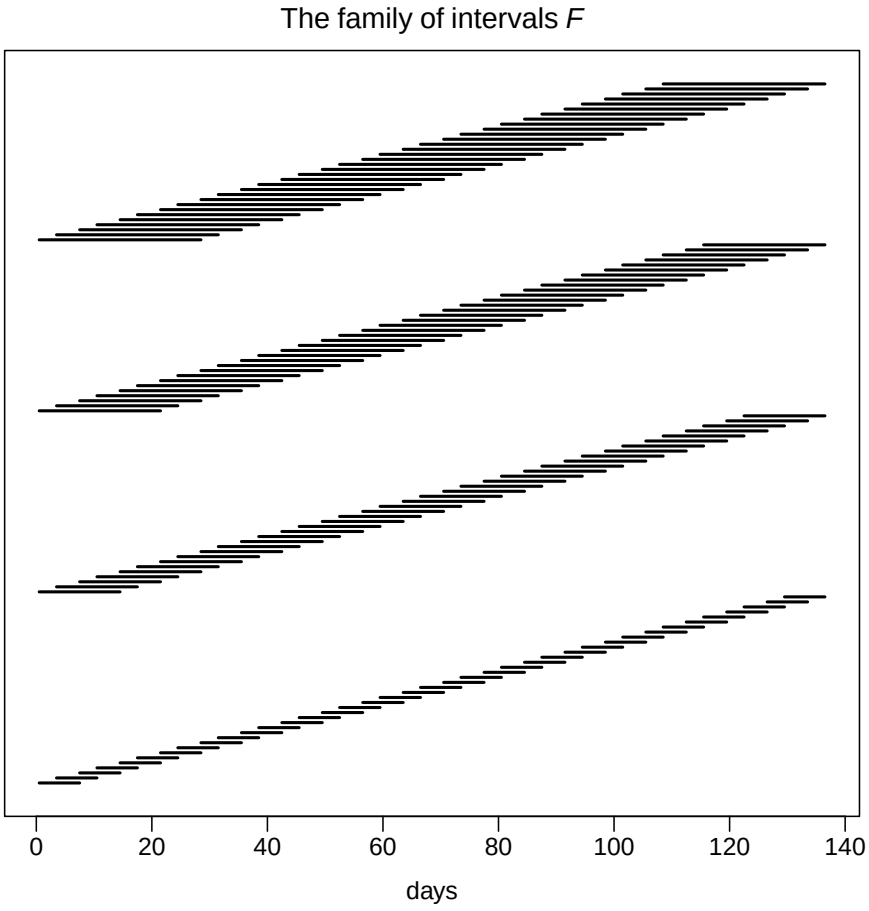
```
##    i j
## 1  1 2
## 2  1 3
## 3  2 3
## 4  1 4
## 5  2 4
## 6  3 4
## 7  1 5
## 8  2 5
## 9  3 5
## 10 4 5
```

```r
grid <- construct_weekly_grid(t_len, min_len = 7, nmbr_of_wks = 4)
```

A graphical presentation of the family $\mathcal{F}$ for our sample size $T = 137$ (as in the application) is given here:

```r
intervals <- data.frame('left' = grid$gset$u - grid$gset$h,
                        'right' = grid$gset$u + grid$gset$h,
                        'v' = 0)
intervals$v <- (1:nrow(intervals)) / nrow(intervals)
```

```
plot(NA, xlim=c(0,t_len),  ylim = c(0, 1 + 1/nrow(intervals)),
     xlab="days", ylab = "", yaxt= "n", mgp=c(2,0.5,0))
title(main = expression(The ~ family ~ of ~ intervals ~ italic(F)),
      line = 1)
segments(intervals$left * t_len, intervals$v,
         intervals$right * t_len, intervals$v,
         lwd = 2)
```



With the help of our multiscale method, we simultaneously test the null hypothesis $H_0^{(i,j,k)}$ that $\lambda_i(\cdot) = \lambda_j(\cdot)$ on the interval $\mathcal{I}_k \in \mathcal{F}$ for each (i, j, k). We denote the length of the intervals from the grid as h_k.

Now we are ready to perfrom the test.

Step 1. Compute the quantile $q_{T,\text{Gauss}}(\alpha)$ by Monte Carlo simulations. Specif-

ically, draw a large number `sim_runs` $= 5000$ samples of independent standard normal random variables $\{Z_{it}^{(\ell)} : 1 \le i \le n, 1 \le t \le T\}$ for $1 \le \ell \le$ `sim_runs`. Compute the value $\Phi_T^{(\ell)}$ of the Gaussian statistic Φ_T for each sample ℓ by the following formula:

$$\Phi_T = \max_{(i,j,k)} a_k\big(|\phi_{ijk,T}| - b_k\big),$$

where

$$\phi_{ijk,T} = \frac{1}{\sqrt{2Th_k}} \sum_{t=1}^{T} \mathbf{1}\Big(\frac{t}{T} \in \mathcal{I}_k\Big)\big\{Z_{it} - Z_{jt}\big\},$$

$a_k = \{\log(e/h_k)\}^{1/2}/\log\log(e^e/h_k)$ and $b_k = \sqrt{2\log(1/h_k)}$. Then calculate the empirical $(1-\alpha)$-quantile $\hat{q}_{T,\text{Gauss}}(\alpha)$ from the values $\{\Phi_T^{(\ell)} : 1 \le \ell \le$ `sim_runs`$\}$. Use $\hat{q}_{T,\text{Gauss}}(\alpha)$ as an approximation of the quantile $q_{T,\text{Gauss}}(\alpha)$.

This step is done with these lines of code:

```
quantiles <- compute_quantiles(t_len = t_len, grid = grid,
                               n_ts = n, ijset = ijset,
                               sigma = sigmahat,
                               sim_runs = sim_runs)

probs <- as.vector(quantiles$quant[1, ])
pos   <- which.min(abs(probs - (1 - alpha)))
quant <- quantiles$quant[2, pos]
quant
```

```
## [1] 2.1995
```

Step 2. Compute the kernel averages $\hat{\psi}_{ijk,T}$ as

$$\hat{\psi}_{ijk,T} := \frac{\sum_{t=1}^{T} \mathbf{1}(\frac{t}{T} \in \mathcal{I}_k)(X_{it} - X_{jt})}{\hat{\sigma}\{\sum_{t=1}^{T} \mathbf{1}(\frac{t}{T} \in \mathcal{I}_k)(X_{it} + X_{jt})\}^{1/2}}$$

together with the scale-adjusted values of individual test statistics for testing the hypothesis $H_0^{(i,j,k)}$ that $\lambda_i = \lambda_j$ on an interval $\mathcal{I}_k$, $a_k\left(|\hat{\psi}_{ijk,T}| - b_k\right)$, where, as before, $a_k = \{\log(e/h_k)\}^{1/2}/\log\log(e^e/h_k)$ and $b_k = \sqrt{2\log(1/h_k)}$. Based on these values, we can calculate the pairwise test statistics

$$\hat{\Psi}_{ij,T} = \max_{\mathcal{I}_k \in \mathcal{F}} a_k\left(|\hat{\psi}_{ijk,T}| - b_k\right)$$

for testing that λ_i and λ_j are different at least on one of the intervals $\mathcal{I}_k \in \mathcal{F}$, as well as the value of the overall test statistics for testing that at least two of the mean functions are different somewhere:

$$\hat{\Psi}_T = \max_{(i,j)\in\mathcal{S}} \hat{\Psi}_{ij,T}.$$

This step is done with these lines of code:

```
result <- compute_statistics(data = covid, sigma = sigmahat,
                             n_ts = n, grid = grid)
str(result)
```

```
## List of 4
##  $ stat            : num 15.5
##  $ stat_pairwise   : num [1:5, 1:5] 0 0 0 0 0 ...
##  $ ijset           :'data.frame': 10 obs. of  2 variables:
##   ..$ i: int [1:10] 1 1 2 1 2 3 1 2 3 4
##   ..$ j: int [1:10] 2 3 3 4 4 4 5 5 5 5
##   ..- attr(*, "out.attrs")=List of 2
##   .. ..$ dim     : Named int [1:2] 5 5
##   .. .. ..- attr(*, "names")= chr [1:2] "i" "j"
##   .. ..$ dimnames:List of 2
##   .. .. ..$ i: chr [1:5] "i=1" "i=2" "i=3" "i=4" ...
##   .. .. ..$ j: chr [1:5] "j=1" "j=2" "j=3" "j=4" ...
##  $ gset_with_values:List of 10
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.15 -2.71 -2.56 -1.95 1.61 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.04 -0.25 1.97 2.66 1.74 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.427 -0.174 2.168 3.389 5.821 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.632 -2.371 -2.073 -2.521 0.363 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.31 -2.28 -1.84 -1.68 -1.5 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.881 -0.634 1.339 2.413 4.709 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
```

```
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.977 -1.104 -0.561 -0.827 -2.728 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.36714 -1.02287 -0.33388 -0.00825 1.66979 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.731 -1.884 -0.101 0.822 1.676 ...
##   ..$ :'data.frame': 140 obs. of  3 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.82 -1.51 -1.26 -1.1 0.43 ...
```

As a result, we get the list with the following elements:

- `stat` denotes $\hat{\Psi}_T$;
- `stat_pairwise` is a matrix that consists of the values of the pairwise statistics $\hat{\Psi}_{ij,T}$;
- `ijset` denotes the set $\int$ and lists all pairwise comparisons that have been performed;
- `gset_with_values` is a list with dataframes that contains the individual test statistics. The order of the dataframes corresponds to the order of the elements in `ijset`, i.e. the results of the first comparison is in the first dataframe, etc. Each dataframe is coded in the following way. Columns `u` and `h` determine the interval $\mathcal{I}_k$ with `u-h` and `u+h` being the left and the right end of the interval respectively. Column `vals` consists of the scale-adjusted values of individual test statistics for testing $H_0^{(i,j,k)}$ for the respective interval $\mathcal{I}_k$.

Step 3. Now we carry out the test itself, comparing the scale-adjusted values of individual test statistics from Step 2with the critical value from Step 1. It is done by the following lines of code:

```
gset_with_values <- result$gset_with_values

for (i in seq_len(nrow(ijset))) {
   test_results <- gset_with_values[[i]]$vals > quant
   gset_with_values[[i]]$test <- test_results
}

str(gset_with_values)

## List of 10
```

```
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -2.15 -2.71 -2.56 -1.95 1.61 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -2.04 -0.25 1.97 2.66 1.74 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE TRUE FALSE TRUE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -1.427 -0.174 2.168 3.389 5.821 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE TRUE TRUE TRUE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -2.632 -2.371 -2.073 -2.521 0.363 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -2.31 -2.28 -1.84 -1.68 -1.5 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -1.881 -0.634 1.339 2.413 4.709 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE TRUE TRUE TRUE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -1.977 -1.104 -0.561 -0.827 -2.728 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -1.36714 -1.02287 -0.33388 -0.00825 1.66979 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
```

```
##   ..$ vals: num [1:140] -2.731 -1.884 -0.101 0.822 1.676 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE TRUE ...
##  $ :'data.frame': 140 obs. of  4 variables:
##   ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   ..$ vals: num [1:140] -1.82 -1.51 -1.26 -1.1 0.43 ...
##   ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

Now each dataframe from `gset_with_values` contains additional column that is either `TRUE` if we reject the respective null hypothesis $H_0^{(i,j,k)}$ or `FALSE` if we do not reject. We can use these dataframes to produce the plots for illustrating the results.

You do not have to perform these steps yourself, the function `multiscale_test()` carries them out automatically for you:

```
results <- multiscale_test(data = covid, sigma = sigmahat,
                           n_ts = n, grid = grid, ijset = ijset,
                           alpha = alpha,
                           sim_runs = sim_runs)
```

```
## We reject H_0 with probability 0.05 . Psihat_statistic =  15.45687 .
##  Number of pairwise rejections =  10 out of  10 . Gaussian quantile value =  2.22
```

```
str(results)
```

```
## List of 5
##  $ quant           : num 2.22
##  $ stat            : num 15.5
##  $ stat_pairwise   : num [1:5, 1:5] 0 0 0 0 0 ...
##  $ ijset           :'data.frame': 10 obs. of  2 variables:
##   ..$ i: int [1:10] 1 1 2 1 2 3 1 2 3 4
##   ..$ j: int [1:10] 2 3 3 4 4 4 5 5 5 5
##   ..- attr(*, "out.attrs")=List of 2
##   .. ..$ dim     : Named int [1:2] 5 5
##   .. .. ..- attr(*, "names")= chr [1:2] "i" "j"
##   .. ..$ dimnames:List of 2
##   .. .. ..$ i: chr [1:5] "i=1" "i=2" "i=3" "i=4" ...
##   .. .. ..$ j: chr [1:5] "j=1" "j=2" "j=3" "j=4" ...
##  $ gset_with_values:List of 10
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.15 -2.71 -2.56 -1.95 1.61 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
```

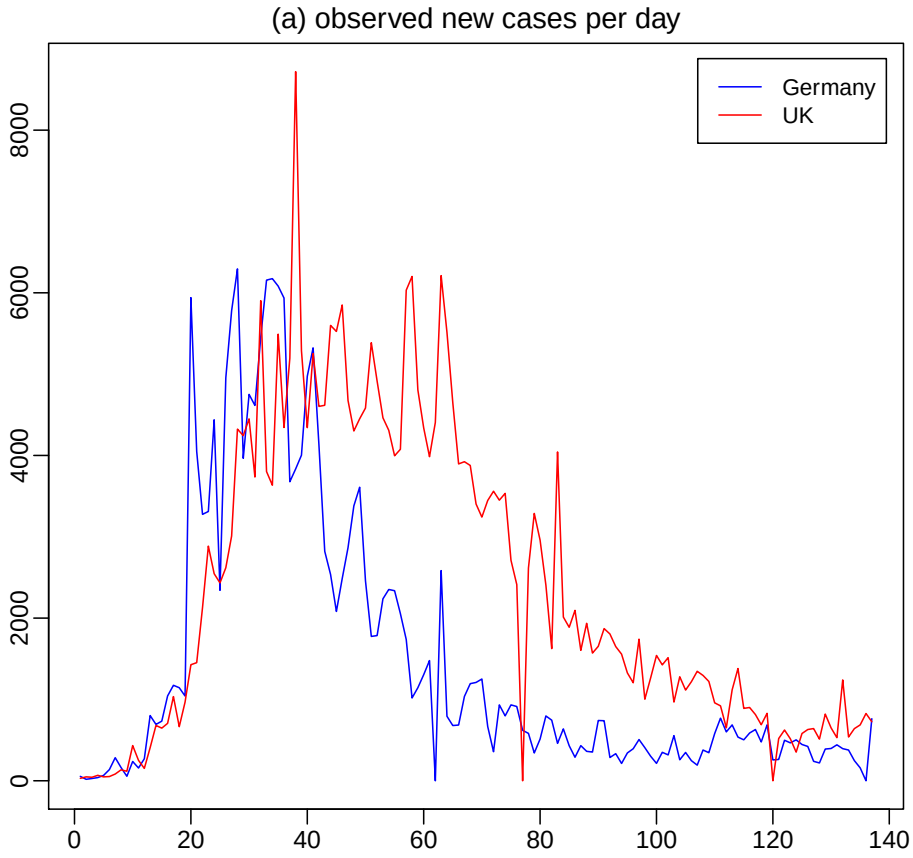
```
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.04 -0.25 1.97 2.66 1.74 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE TRUE FALSE TRUE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.427 -0.174 2.168 3.389 5.821 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE TRUE TRUE TRUE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.632 -2.371 -2.073 -2.521 0.363 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.31 -2.28 -1.84 -1.68 -1.5 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.881 -0.634 1.339 2.413 4.709 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE TRUE TRUE TRUE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.977 -1.104 -0.561 -0.827 -2.728 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.36714 -1.02287 -0.33388 -0.00825 1.66979 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -2.731 -1.884 -0.101 0.822 1.676 ...
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE TRUE ...
##   ..$ :'data.frame': 140 obs. of  4 variables:
##   .. ..$ u   : num [1:140] 0.0292 0.0511 0.0803 0.1022 0.1314 ...
##   .. ..$ h   : num [1:140] 0.0255 0.0255 0.0255 0.0255 0.0255 ...
##   .. ..$ vals: num [1:140] -1.82 -1.51 -1.26 -1.1 0.43 ...
```

```
##   .. ..$ test: logi [1:140] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

Now we are ready to present the results. For the sake of brevity, we only show the results for the pairwise comparisons of Germany ($i = 1$) with the United Kingdom ($j = 2$). This is coded as the first comparison in `ijset`. The remaining figures can be found in **?**.

First, we plot the the observed time series for the two countries.

```
plot(covid[, 1], ylim=c(min(covid[, 1], covid[, 2]),
                        max(covid[, 1], covid[, 2])),
     type="l", col="blue", ylab="", xlab="", mgp=c(1, 0.5, 0))
lines(covid[, 2], col="red")
title(main = "(a) observed new cases per day", font.main = 1, line = 0.5)
legend("topright", inset = 0.02, legend=c("Germany", "UK"),
       col = c("blue", "red"), lty = 1, cex = 0.95, ncol = 1)
```



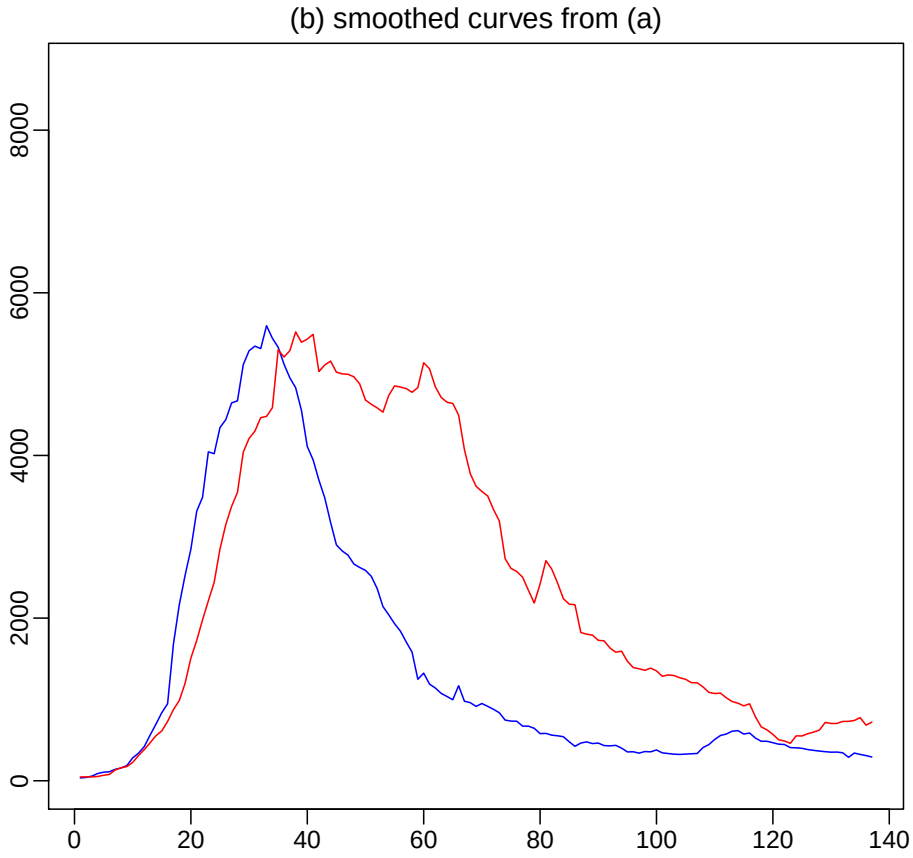
Now we plot the smoothed versions of the time series from (a), that is, the plot shows nonparametric kernel estimates of the two trend functions λ_1 and λ_2, where the bandwidth is set to 7 days and a rectangular kernel is used. This is not necessary but sometimes useful.

```
nadaraya_watson_smoothing <- function(u, data_p, grid_p, bw){
  result      = 0
  norm        = 0
  T_size      = length(data_p)
  result = sum((abs((grid_p - u) / bw) <= 1) * data_p)
  norm = sum((abs((grid_p - u) / bw) <= 1))
  return(result/norm)
}
```

```r
grid_points <- seq(from = 1 / t_len, to = 1, length.out = t_len)
smoothed_1  <- mapply(nadaraya_watson_smoothing, grid_points,
                      MoreArgs = list(covid[, 1], grid_points,
                                      bw = 3.5 / t_len))

smoothed_2  <- mapply(nadaraya_watson_smoothing, grid_points,
                      MoreArgs = list(covid[, 2], grid_points,
                                      bw = 3.5 / t_len))

plot(smoothed_1, ylim=c(min(covid[, 1], covid[, 2]),
                        max(covid[, 1], covid[, 2])),
     type="l", col="blue", ylab="", xlab = "", mgp=c(1,0.5,0))
title(main = "(b) smoothed curves from (a)", font.main = 1, line = 0.5)
lines(smoothed_2, col="red")
```



Finally, we present the results produced by our test. Specifically, we depict in grey the set $\mathcal{F}_{\text{reject}}(1,2)$ of all the intervals $\mathcal{I}_k$ for which the test rejects the null $H_0^{(1,2,k)}$. The minimal intervals in the subset $\mathcal{F}^{\min}_{\text{reject}}(1,2)$ are depicted in black. The definition of the minimal intervals and some dicsussion on the topic are given in **?**. The function that computes minimal intervals can be accessed as `compute_minimal_intervals()`.

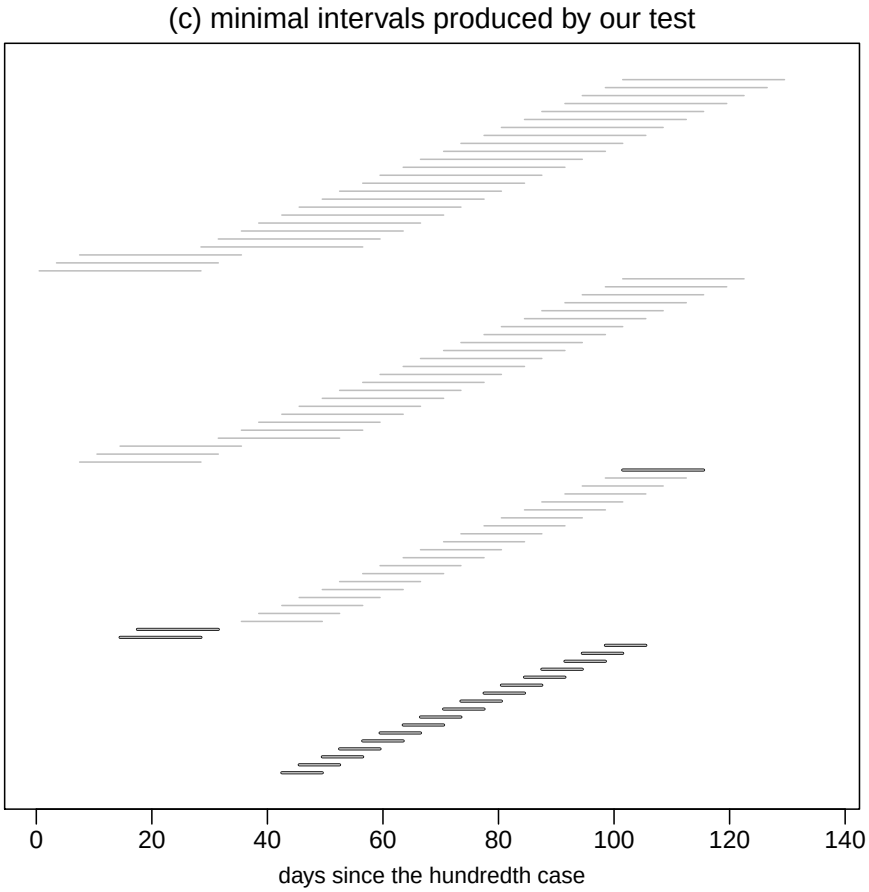
According to theoretical results in this paper, we can make the following simultaneous confidence statement about the intervals plotted below: we can claim, with confidence of about 95%, that there is a difference between the functions λ_1 and λ_2 on each of these intervals.

```
l <- 1 #First comparison in ijset
gset       <- results$gset_with_values[[l]]
reject     <- subset(gset, test == TRUE, select = c(u, h))
```

```
reject_set <- data.frame('startpoint' = (reject$u - reject$h) * t_len,
                         'endpoint' = (reject$u + reject$h) * t_len,
                         'values' = 0)
reject_set$values <- (1:nrow(reject_set)) / nrow(reject_set)

#Produce minimal intervals
reject_min  <- compute_minimal_intervals(reject_set)

plot(NA, xlim=c(0, t_len),  ylim = c(0, 1 + 1 / nrow(reject_set)),
     xlab="", mgp=c(2, 0.5, 0), yaxt = "n", ylab = "")
title(main = "(c) minimal intervals produced by our test",
      font.main = 1, line = 0.5)
title(xlab = "days since the hundredth case", line = 1.7,
      cex.lab = 0.9)
segments(reject_min$startpoint, reject_min$values,
         reject_min$endpoint, reject_min$values, lwd = 2)
segments(reject_set$startpoint, reject_set$values,
         reject_set$endpoint, reject_set$values,
         col = "gray")
```



4 Long-run variance estimator

In development.

5 Conclusion

In development.

References