Multiscale Inference for NonParametric Time Trends

Marina Khismatullina Michael Vogt

September 2, 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 demonstrate the use of our functions, we analyse two datasets. In order to illustrate the method from Khismatullina and Vogt (2020a), 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 Parker et al. (1992). In order to illustrate the method from Khismatullina and Vogt (2020b), 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 inKhismatullina and Vogt (2020a) and the results of applying it to the temperature data. Section 3 describes the multiscale procedure for comparing different time trends as in Khismatullina and Vogt (2020b) 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).

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

As an illustration for the multiscale testing procedure proposed in Khismatullina and Vogt (2020a), we analyse the CET dataset, which is the longest instrumental record of temperature in the world. It contains the mean monthly surface air temperatures (in degrees Celsius) from the year 1659 to the present. CET datasets are freely available for use under Open Government License and can be downloaded from https://www.metoffice.gov.uk/hadobs/hadcet/. You can load the data using the function `data(temperature, package = "multiscale")`.

```
> require(multiscale)
> data(temperature, package = "multiscale")
> str(temperature)
```

```
 num [1:359] 8.87 9.1 9.78 9.52 8.63 9.34 8.29 9.86 8.52 9.51 ...
```

As you can see, this is an array of length $T = 359$ where each element denotes the mean yearly temperature starting from year 1659 and ending with year 2017.

```
> t_len    <- length(temperature)
> t_len
```

```
[1] 359
```

```
> ts_start <- 1659
```

We assume that the temperature data Y_t follow the nonparametric trend model

$$Y_t = m(t/T) + \varepsilon_t \quad \text{for } t = 1, \dots, T,$$

where m is the unknown time trend of interest. We are interested in identifying local increases/decreases of the trend function m. We assume throughout that m is continuously differentiable on $[0, 1]$. The test problem then can be formulated as follows: Let $H_0(u, h)$ be the hypothesis that m is constant on the interval $[u - h, u + h] \in [0, 1]$, or, equivalently,

$$H_0(u, h) : m'(w) = 0 \text{ for all } w \in [u - h, u + h].$$

We want to test the hypothesis $H_0(u, h)$ simultaneously for many different intervals $[u - h, u + h]$. The overall null hypothesis is thus given by

$$H_0 : \text{ The hypothesis } H_0(u, h) \text{ holds true for all } (u, h) \in \mathcal{G}_T,$$

where $\mathcal{G}_T$ is some set of points (u, h). Specifically, for this application we take into account the default set of points, i.e. all locations u on an equidistant grid $u = 5/T, 10/T, \dots, 355/T$ and all bandwidths $h = 5/T, 10/T, 15/T, \dots, 85/T$. More on the construction of $\mathcal{G}_T$ you can find in Khismatullina and Vogt (2020a) and in the package documentation.

```
> grid <- construct_grid(t_len)
> str(grid$gset * 1)
```

```
'data.frame':        1136 obs. of  2 variables:
 $ u: num  0.0139 0.0279 0.0418 0.0557 0.0696 ...
 $ h: num  0.0279 0.0279 0.0279 0.0279 0.0279 ...
```

Furthermore, the test statistic requires an estimator of the long-run variance $\sigma^2 = \sum_{\ell=-\infty}^{\infty} Cov(\varepsilon_0, \varepsilon_\ell)$ of the error process $\{\varepsilon_t\}$. Here we assume that the error process $\{\varepsilon_t\}$ has the AR(2) structure

$$\varepsilon_t = \sum_{j=1}^{2} a_j \varepsilon_{t-j} + \eta_t,$$

where η_t are i.i.d. innovations with mean 0 and variance ν^2. We estimate the the long-run error variance σ^2 by the procedure from Khismatullina and Vogt (2020a) (with tuning parameters $q = 25$ and $\overline{r} = 10$), which produces the following value:

```
> parameters <- estimate_lrv(data = temperature,
+                            q = 25, r_bar = 10, p = 2)
> cat("Long-run variance is equal to ", parameters$lrv, "\n")
```

```
Long-run variance is equal to  0.7576827
```

```
> sigmahat <- sqrt(parameters$lrv)
```

Details of the estimation procedure together with the description of the tuning parameters are deferred to Section 4.

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

```
> alpha    <- 0.05
> sim_runs <- 5000
```

Since we consider increases and decreases of the function, we are interested in the first derivative of the function:

```
> deriv_order = 1
```

The package currently supports only `deriv_order = 0` for testing $m = 0$ and `deriv_order = 1` for testing $m' = 0$.

Now we are ready to perform the test.

Step 1. Compute the quantile $q_{T,\mathrm{Gauss}}(\alpha)$ by Monte Carlo simulations. Specifically, draw a large number `sim_runs` $= 5000$ samples of independent standard normal random variables $\{Z_t^{(\ell)} : 1 \le t \le T\}$ 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:

$$\Phi_T = \max_{(u,h)\in\mathcal{G}_T} \left\{ \Big| \sum_{t=1}^{T} w_{t,T} Z_t \Big| - \lambda(h) \right\},$$

where $w_{t,T}(u,h)$ are local linear kernel weights with the Epanechnikov kernel, and $\lambda(h) = \sqrt{2\log\{1/(2h)\}}$ is an additive correction term.

Then calculate the empirical $(1-\alpha)$-quantile $\widehat{q}_{T,\mathrm{Gauss}}(\alpha)$ from the values $\{\Phi_T^{(\ell)} : 1 \le \ell \le$ `sim_runs`$\}$. Use $\widehat{q}_{T,\mathrm{Gauss}}(\alpha)$ as an approximation of the quantile $q_{T,\mathrm{Gauss}}(\alpha)$. This step is done with these lines of code (running this can take a while):

```
> quantiles <- compute_quantiles(t_len = t_len, grid = grid,
+                                sim_runs = 10)
> probs <- as.vector(quantiles$quant[1, ])
> pos   <- which.min(abs(probs - (1 - alpha)))
> quant <- quantiles$quant[2, pos]
> quant
```

```
[1] 1.191727
```

Step 2. Compute the kernel averages $\widehat{\psi}_T(u,h)$ as

$$\widehat{\psi}_T(u,h) := \sum\nolimits_{t=1}^{T} w_{t,T}(u,h)Y_t,$$

where, as before, $w_{t,T}(u,h)$ are local linear kernel weights based on the Epanechnikov kernel. Based on these kernel averages, calculate the test statistic

$$\widehat{\Psi}_T = \max_{(u,h)\in\mathcal{G}_T} \left\{ \left| \frac{\widehat{\psi}_T(u,h)}{\widehat{\sigma}} \right| - \lambda(h) \right\}.$$

This step is done with these lines of code:

```
> result <- compute_statistics(data = temperature,
+                              sigma = sigmahat,
+                              grid = grid, deriv_order = deriv_order)
> str(result)
```

```
List of 2
 $ stat          : num 3.14
 $ gset_with_vals:'data.frame':        1136 obs. of  4 variables:
  ..$ u       : num [1:1136] 0.0139 0.0279 0.0418 0.0557 0.0696 ...
  ..$ h       : num [1:1136] 0.0279 0.0279 0.0279 0.0279 0.0279 ...
  ..$ vals    : num [1:1136] -0.263 -1.38 -0.914 0.587 0.128 ...
  ..$ vals_cor: num [1:1136] -2.14 -1.02 -1.49 -1.82 -2.27 ...
  ..- attr(*, "out.attrs")=List of 2
  .. ..$ dim     : Named int [1:2] 71 16
  .. .. ..- attr(*, "names")= chr [1:2] "u" "h"
  .. ..$ dimnames:List of 2
  .. .. ..$ u: chr [1:71] "u=0.01392758" "u=0.02785515" "u=0.04178273" "u=0.0
  .. .. ..$ h: chr [1:16] "h=0.02785515" "h=0.04178273" "h=0.05571031" "h=0.0
```

We get the list with the following elements as the result:

- `stat` denotes $\widehat{\Psi}_T$;
- `gset_with_vals` is a dataframe that contains the normalised kernel average. The dataframe is coded in the following way. Columns `u` and `h` determine the element $(u,h) \in \mathcal{G}_T$ for which we calculate the kernel average. Column `vals` consists of the values of $\frac{\widehat{\psi}_T(u,h)}{\widehat{\sigma}}$, and column `vals_cor` contains the values of $\left|\frac{\widehat{\psi}_T(u,h)}{\widehat{\sigma}}\right| - \lambda(h)$ for the given pair (u,h).

Step 3. Now we carry out the test itself, comparing the normalised values of kernel averagess from Step 2 with the critical value from Step 1. It is done by the following lines of code:

```
> gset         <- result$gset_with_vals
> test_results <- (gset$vals_cor > quant) * sign(gset$vals)
```

```
> gset$test    <- test_results
> str(gset)

'data.frame':        1136 obs. of  5 variables:
 $ u       : num  0.0139 0.0279 0.0418 0.0557 0.0696 ...
 $ h       : num  0.0279 0.0279 0.0279 0.0279 0.0279 ...
 $ vals    : num  -0.263 -1.38 -0.914 0.587 0.128 ...
 $ vals_cor: num  -2.14 -1.02 -1.49 -1.82 -2.27 ...
 $ test    : num  0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "out.attrs")=List of 2
  ..$ dim     : Named int  71 16
  .. ..- attr(*, "names")= chr  "u" "h"
  ..$ dimnames:List of 2
  .. ..$ u: chr  "u=0.01392758" "u=0.02785515" "u=0.04178273" "u=0.05571031"
  .. ..$ h: chr  "h=0.02785515" "h=0.04178273" "h=0.05571031" "h=0.06963788"
```

Now the dataframe `gset` contains everything that was in `result$gset_with_vals` before and an additional column `test`. The values in this column are calculated as follows. It is either `1` if we reject the respective null hypothesis $H_0(u,h)$ and detect an increase in the trend, `0` if we do not reject $H_0(u,h)$. or `-1` if we reject the respective null hypothesis $H_0(u,h)$ and detect an decrease in the trend. In our application we do not detect any decreases in the trend function m:

```
> sum(gset$test == -1)

[1] 0
```

We can now use this dataframe to produce the plots for illustrating the results.

All these steps are not necessary for performing the test, they are already incorporated in the function `multiscale_test()`:

```
> results <- multiscale_test(data = temperature,
+                            sigma = sigmahat,
+                            grid = grid,
+                            alpha = alpha,
+                            deriv_order = deriv_order,
+                            sim_runs = 10)

For the given time series we reject H_0 with probability 0.05 . Psihat_statistic =

> str(results)

List of 4
 $ quant         : num 2.06
 $ stat          : num 3.14
 $ test_matrix   : num [1:16, 1:71] 0 0 0 0 0 0 0 0 0 0 ...
```

```
$ gset_with_vals:'data.frame':        1136 obs. of  5 variables:
 ..$ u       : num [1:1136] 0.0139 0.0279 0.0418 0.0557 0.0696 ...
 ..$ h       : num [1:1136] 0.0279 0.0279 0.0279 0.0279 0.0279 ...
 ..$ vals    : num [1:1136] -0.263 -1.38 -0.914 0.587 0.128 ...
 ..$ vals_cor: num [1:1136] -2.14 -1.02 -1.49 -1.82 -2.27 ...
 ..$ test    : num [1:1136] 0 0 0 0 0 0 0 0 0 0 ...
 ..- attr(*, "out.attrs")=List of 2
 .. ..$ dim     : Named int [1:2] 71 16
 .. .. ..- attr(*, "names")= chr [1:2] "u" "h"
 .. ..$ dimnames:List of 2
 .. .. ..$ u: chr [1:71] "u=0.01392758" "u=0.02785515" "u=0.04178273" "u=0.05571031
 .. .. ..$ h: chr [1:16] "h=0.02785515" "h=0.04178273" "h=0.05571031" "h=0.06963788
```

Now we are ready to present the results.First, we plot the the observed time series.

```
> plot(ts_start:(ts_start + t_len - 1), temperature, type = 'l',
+      lty = 1, xlab = 'year', ylab = 'temperature',
+      ylim = c(min(temperature) - 0.1, max(temperature) + 0.1))
> title(main = "(a) observed yearly temperature", font.main = 1,
+       line = 0.5)
>
```

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.

3 Multiscale Inference for Multiple Nonparametric Regressions

As an illustration for the multiscale method proposed in Khismatullina and Vogt (2020b), 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)
> 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 Khismatullina and Vogt (2020b), $\hat{\sigma}^2$ 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)
+ }
> 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:

```
> 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 $j = 1, 2, \ldots$.

```
> 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
```

```
> 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:

```
> 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 $q_{T,\mathrm{Gauss}}(\alpha)$ by Monte Carlo simulations. Specifically, 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 $\hat{q}_{T,\mathrm{Gauss}}(\alpha)$ from the values $\{\Phi_T^{(\ell)} :$

$1 \leq \ell \leq$ `sim_runs`$\}$. Use $\hat{q}_{T,\mathrm{Gauss}}(\alpha)$ as an approximation of the quantile $q_{T,\mathrm{Gauss}}(\alpha)$.

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] 2.174247
```

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 =  2.18578

> str(results)

List of 5
 $ quant           : num 2.19
 $ 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 Khismatullina and Vogt (2020b).

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)
+ }
> 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 Khismatullina and Vogt (2020b). 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 this section we describe the usage of the estimation procedure of the parameteres of a time series proces. The estimation procedure relies on the following simple observation: If $\{\varepsilon_t\}$ is an AR(p) process, then the time series $\{\Delta_q \varepsilon_t\}$ of the differences $\Delta_q \varepsilon_t = \varepsilon_t - \varepsilon_{t-q}$ is an ARMA(p, q) process of the form

$$\Delta_q \varepsilon_t - \sum_{j=1}^{p} a_j \Delta_q \varepsilon_{t-j} = \eta_t - \eta_{t-q}.$$

As m is Lipschitz, the differences $\Delta_q \varepsilon_t$ of the unobserved error process are close to the differences $\Delta_q Y_{t,T} = Y_{t,T} - Y_{t-q,T}$ of the observed time series. This implies that the differenced time series $\{\Delta_q Y_{t,T}\}$ is approximately an ARMA(p, q) process.

Specifically, we proceed as follows. First, we estimate $\boldsymbol{a}$ by

$$\widetilde{\boldsymbol{a}}_q = \widehat{\boldsymbol{\Gamma}}_q^{-1} \widehat{\boldsymbol{\gamma}}_q,$$

where $\widehat{\boldsymbol{\Gamma}}_q$ denotes the denotes the $p \times p$ sample covariance matrix $\widehat{\boldsymbol{\Gamma}}_q = (\widehat{\gamma}_q(i-j) : 1 \le i, j \le p)$, and $\widehat{\boldsymbol{\gamma}}_q$ denotes the vector of sample autocovariances $\widehat{\boldsymbol{\gamma}}_q = (\widehat{\gamma}_q(1), \ldots, \widehat{\gamma}_q(p))^\top$ with $\widehat{\gamma}_q(\ell) = (T-q)^{-1} \sum_{t=q+\ell+1}^{T} \Delta_q Y_{t,T} \Delta_q Y_{t-\ell,T}$.

This estimator $\widetilde{\boldsymbol{a}}_q$ depends on the tuning parameter q, that is, on the order of the differences $\Delta_q Y_{t,T}$. An appropriate choice of q needs to take care of the following two points: (i) q should be chosen large enough to ensure that the vector $\boldsymbol{c}_q = (c_{q-1}, \ldots, c_{q-p})^\top$ is close to zero. As we have already seen, the constants c_k decay to zero exponentially fast and can be computed from the recursive equations (**??**) for given parameters $a_1, a_2, a_3, \ldots$ In the special case of an AR(1) process, for example, one can readily calculate that $c_k \le 0.0035$ for any $k \ge 20$ and any $|a_1| \le 0.75$. Hence, if we have an AR(1) model for the errors ε_t and the error process is not too persistent, choosing $q \ge 20$ should make sure that $\boldsymbol{c}_q$ is close to zero. Generally speaking, the recursive equations (**??**) can be used to get some idea for which values of q the vector $\boldsymbol{c}_q$ can be expected to be approximately zero. (ii) q

should not be chosen too large in order to ensure that the trend m is appropriately eliminated by taking q-th differences. As long as the trend m is not very strong, the two requirements (i) and (ii) can be fulfilled without much difficulty. For example, by choosing $q = 20$ in the AR(1) case just discussed, we do not only take care of (i) but also make sure that moderate trends m are differenced out appropriately.

When the trend m is very pronounced, in contrast, even moderate values of q may be too large to eliminate the trend appropriately. As a result, the estimator $\widetilde{\boldsymbol{a}}_q$ will have a strong bias. In order to reduce this bias, we refine our estimation procedure as follows: By solving the recursive equations (**??**) with $\boldsymbol{a}$ replaced by $\widetilde{\boldsymbol{a}}_q$, we can compute estimators $\widetilde{c}_k$ of the coefficients c_k and thus estimators $\widetilde{\boldsymbol{c}}_r$ of the vectors $\boldsymbol{c}_r$ for any $r \geq 1$. Moreover, the innovation variance ν^2 can be estimated by $\widetilde{\nu}^2 = (2T)^{-1} \sum_{t=p+2}^{T} \widetilde{r}_{t,T}^2$, where $\widetilde{r}_{t,T} = \Delta_1 Y_{t,T} - \sum_{j=1}^{p} \widetilde{a}_j \Delta_1 Y_{t-j,T}$ and $\widetilde{a}_j$ is the j-th entry of the vector $\widetilde{\boldsymbol{a}}_q$. Plugging the expressions $\widehat{\boldsymbol{\Gamma}}_r$, $\widehat{\boldsymbol{\gamma}}_r$, $\widetilde{\boldsymbol{c}}_r$ and $\widetilde{\nu}^2$ into (**??**), we can estimate $\boldsymbol{a}$ by

$$\widehat{\boldsymbol{a}}_r = \widehat{\boldsymbol{\Gamma}}_r^{-1} (\widehat{\boldsymbol{\gamma}}_r + \widetilde{\nu}^2 \widetilde{\boldsymbol{c}}_r), \tag{2}$$

where r is a much smaller differencing order than q. Specifically, in case (A), we can choose r to be any fixed number $r \geq 1$. Unlike q, the parameter r thus remains bounded as T increases. In case (B), our theory allows to choose any number r with $r \geq (1 + \delta)p$ for some small $\delta > 0$. Since $q/p \to \infty$, it holds that $q/r \to \infty$ as well, which means that r is of smaller order than q. Hence, in both cases (A) and (B), the estimator $\widehat{\boldsymbol{a}}_r$ is based on a differencing order r that is much smaller than q; only the pilot estimator $\widetilde{\boldsymbol{a}}_q$ relies on differences of the larger order q. As a consequence, $\widehat{\boldsymbol{a}}_r$ should eliminate the trend m more appropriately and should thus be less biased than the pilot estimator $\widetilde{\boldsymbol{a}}_q$. In order to make the method more robust against estimation errors in $\widetilde{\boldsymbol{c}}_r$, we finally average the estimators $\widehat{\boldsymbol{a}}_r$ for a few values of r. In particular, we define

$$\widehat{\boldsymbol{a}} = \frac{1}{\overline{r} - \underline{r} + 1} \sum_{r=\underline{r}}^{\overline{r}} \widehat{\boldsymbol{a}}_r, \tag{3}$$

where $\underline{r}$ and $\overline{r}$ are chosen as follows: In case (A), we let $\underline{r}$ and $\overline{r}$ be small natural numbers. In case (B), we set $\underline{r} = (1 - \delta)p$ for some small $\delta > 0$ and choose $\overline{r}$ such that $\overline{r} - \underline{r}$ remains bounded. For ease of notation, we suppress the dependence of $\widehat{\boldsymbol{a}}$ on the parameters $\underline{r}$ and $\overline{r}$. Once $\widehat{\boldsymbol{a}} = (\widehat{a}_1, \ldots, \widehat{a}_p)^\top$ is computed, the long-run variance σ^2 can be estimated by

$$\widehat{\sigma}^2 = \frac{\widehat{\nu}^2}{(1 - \sum_{j=1}^{p} \widehat{a}_j)^2}, \tag{4}$$

where $\widehat{\nu}^2 = (2T)^{-1} \sum_{t=p+2}^{T} \widehat{r}_{t,T}^2$ with $\widehat{r}_{t,T} = \Delta_1 Y_{t,T} - \sum_{j=1}^{p} \widehat{a}_j \Delta_1 Y_{t-j,T}$ is an estimator of the innovation variance ν^2 and we make use of the fact that $\sigma^2 = \nu^2 / (1 - \sum_{j=1}^{p^*} a_j)^2$ for the AR(p^*) process $\{\varepsilon_t\}$.

5 Conclusion

In development.

References

Khismatullina, M. and Vogt, M. (2020a). Multiscale inference and long-run variance estimation in non-parametric regression with time series errors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Khismatullina, M. and Vogt, M. (2020b). Simultaneous statistical inference for epidemic trends: how do the time trends of covid-19 compare across countries. Preprint.

Parker, D. E., Legg, T. P. and Folland, C. K. (1992). A new daily central england temperature series, 1772-1991. *International Journal of Climatology*, **12** 317–342.