bootUR: An R Package for Bootstrap Unit Root Tests

Stephan Smeekes
Maastricht University

Ines Wilms
Maastricht University

Abstract

Unit root tests form an essential part of any time series analysis. We provide practitioners with a single, unified framework for comprehensive and reliable unit root testing in the R package **bootUR**. The package's backbone is the popular augmented Dickey-Fuller (ADF) test paired with a union of rejections principle, which can be performed directly on single time series or multiple (including panel) time series. Accurate inference is ensured through the use of bootstrap methods. The package addresses the needs of both novice users, by providing user-friendly and easy-to-implement functions with sensible default options, as well as expert users, by giving full user-control to adjust the tests to one's desired settings. Our **OpenMP**-parallelized efficient C++ implementation ensures that all unit root tests are scalable to datasets containing many time series.

Keywords: bootstrap, R, time series, unit roots.

1. Introduction

In this paper, we introduce the **bootUR** package (Smeekes and Wilms 2020) for R (R Core Team 2017), which implements several bootstrap tests for unit roots. Unit root testing is an essential part of any statistical analysis of time series. Given the crucial role of unit root testing in time series analysis, surprisingly few R packages exist that allow for easy and comprehensive unit root testing. The **bootUR** package aims to fill this gap by offering three major contributions to existing R packages. First, it offers a comprehensive, easy-to-use and reliable set of unit root tests not found as generally in other packages. Second, it offers accurate p-values based on bootstrap methods. Third, its functions are not only directly applicable to single time series, but also to datasets consisting of a potentially large set of time series. With these contributions the **bootUR** package provides practitioners with a single source to fill their unit root testing needs.

Proper handling of unit roots is of paramount importance before commencing any form of analysis on the time series of interest. The by far most important use of unit root tests is therefore as a pre-test to determine whether differencing of the series is needed to eliminate the trend and render the time series stationary. Ignoring unit roots, or stochastic trends, essentially invalidates any subsequent statistical analysis: the stochastic trend, and associated non-decaying dependence of the present on the far past of a series, yields standard inference inapplicable. Probably the most famous consequence of ignoring unit roots is the 'spurious regression phenomenon', where one finds seemingly important relations (high R^2s and highly significant t-statistics) between unrelated time series with stochastic trends. These results

have a long history and are well-established and extensively documented in the time series literature. A reader new to unit roots may, for instance, consult Enders (2008) for a classical textbook treatment of this spurious regression phenomenon as well as the more general problems associated with unit roots.

Currently, unit root tests are scattered across several packages in the R environment for statistical computing and graphics, making it difficult for a practitioner to find and apply an appropriate and reliable test. The most popular unit root test is the classical augmented-Dickey Fuller (ADF) test (Dickey and Fuller 1979, 1981). Implementations of the ADF test are incorporated in various packages, in particular **CADFtest** (Lupi 2009), **fUnitRoots** (Wuertz, Setz, and Chalabi 2017), **tseries** (Trapletti, Hornik, and LeBaron 2019), and **urca** (Pfaff 2008).[1]

As we will argue in the next section, most 'standard' unit root tests, such as the ones implemented in these packages, require seemingly innocuous choices from the practitioner regarding model specifications or which test to use, that may have a major impact on the performance of the unit root tests. As its first major contribution, the **bootUR** package instead implements the user-friendly *union of rejections* principle (Harvey, Leybourne, and Taylor 2009, 2012; Smeekes and Taylor 2012) that relieves the user from the burden of having to choose the right specification and performs this task automatically.

Crucially, with the exception of the HEGY seasonal unit root test in the **uroot** package (López-de Lacalle and Boshnakov 2019), current R implementations of unit root tests rely on asymptotic inference when returning critical values or p-values for the unit root test.[2] As is well known in the literature, unit root tests are very sensitive to size distortions in smaller samples due to, for example, neglected serial correlation (Schwert 1989). Size distortions due to features such as time-varying volatility even persist asymptotically (Cavaliere 2005). As a consequence, unit root tests based on asymptotic or numerical p-values (MacKinnon, Haug, and Michelis 1999), which do not take the features of the specific time series into account, are quite unreliable in practice.

The 'boot' in **bootUR** stands for bootstrap since the unit roots tests we provide rely on various bootstrap methods for constructing p-values. The bootstrap approximates the exact distribution of the unit root test statistic by repeatedly drawing new samples from the original sample, thereby capturing the features of the time series of interest that affect the distribution of the test. This ensures that the bootstrap tests in **bootUR** have accurate size properties under very general conditions, which constitutes the second major contribution of our package.

Finally, most datasets contain multiple, sometimes even many, time series to be tested for unit roots, often leading practitioners to apply unit root tests to each time series separately. Such a practice does not only suffer from multiple testing issues, rejecting several tests by chance alone, but also disregards similarities between individual time series which, if exploited, could increase the often limited power of the individual tests. Although some packages provide joint unit root tests for multivariate or panel data (**pdR**, Tsung-wu 2019; **plm**, Croissant and

[1] The **mleur** package (Zhang, Yu, and McLeod 2011) also implements the ADF test, but links to **urca** for this purpose. The package **uroot** (López-de Lacalle and Boshnakov 2019) used to have the ADF test implemented but it is no longer supported in the package's current version, hence disregarded from the overview.

[2] Another exception is the repository **URT** (Mallet 2017), available on GitHub, which includes bootstrap unit root tests. In the remainder, we only focus on packages that are currently maintained on the Comprehensive R Archive Network (CRAN).

Millo 2008)[3], such tests may increase power but do not allow one to determine the properties of individual series. For this goal one would need tests accounting for multiple testing, but proper implementations of multiple testing corrections are currently lacking for unit root tests. Therefore, the third major contribution of **bootUR** is to implement easy tools for applying unit root tests to multivariate time series with automatic multiple testing control.

With these contributions, the **bootUR** package provides a unified framework for easy and comprehensive unit root testing based on the following philosophy. 1) for novice users, the tests should be easy to implement with sensible default options; 2) those default options should lead to reliable and accurate unit root tests, applicable in general situations; 3) expert users, familiar with the unit root literature, should be able to easily tweak and adjust the tests to their desired settings; 4) all tests should be easily scalable to large datasets without additional effort by the user, thereby providing 'automatic' functionality.

To accomplish our philosophy, the package has a simple structure, yet it offers users a wide variety of unit root tests. In particular, unit root tests can *directly* be performed on *single* time series or *multiple* time series. To this end, we deliberately created separate functions that serve these purposes: the functions `boot_df()` and `boot_union()` can be used for single time series, `iADFtest()` for multiple time series without multiple-testing control, `BSQTtest()` and `bFDRtest()` for multiple time series with multiple-testing control, and the `paneltest()` function offers a panel unit root test. For each unit root test, the bootstrap method can be chosen by the end-user. To this end, all functions make use of the universal argument `boot`. Via suitable warning and error messages, user-friendly advise is provided on the (non)-applicability of certain bootstrap methods in certain situations. Finally, *model specifications* (such as deterministic components, lag length selection, detrending methods) are under the user's full control, with the option to have them implemented automatically according the union of unit root tests principle which ensures reliable tests across potentially heterogeneous series. Each function contains many options whose syntax is shared across the package, thereby facilitating usability and control by the end-user.

Finally, we have also added several functions, based around the core functions above, that aid in the practical implementation of the unit root tests. Most importantly, the function `order_integration()` provides an automatic way to determine the order of integration of each series in a dataset, based on a sequence of one of the aforementioned unit root tests. As it also directly outputs the correctly differenced time series that remove all stochastic trends, it provides the user with the option to conduct the entire unit root pre-analysis with a single command. Additionally, we provide several functions that easily allow the user to assess and visualize properties of the data and outcomes of the tests.

The package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=bootUR`. In addition, the latest (development) version is available on GitHub at `https://github.com/smeekes/bootUR`. The core of the package is written in C++ with parallel execution offered by the **OpenMP** (Dagum and Menon 1998) API to ensure scalability to large datasets. We make use of the packages **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013; Eddelbuettel and Balamuta 2017) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014) to facilitate seamless integration with R. Version 0.2.0 of the **bootUR** package and version 4.0.2 of R were used in this paper.

[3]The packages **PANICr** (Bronder 2016) and **punitroots** (Kleiber and Lupi 2012) also provide panel unit root tests, but the former has been removed from CRAN and the latter is only available on R-Forge.

Adhering to the four points of our philosophy not only requires thoughts on how to implement the tests and design the API, but it also requires a careful choice of the appropriate statistical methods. We therefore first consider the problem from a statistical point of view in Section 2, where we discuss the unit root test for single time series and multiple time series, and in Section 3, where we discuss the bootstrap methods. We then continue with the package's implementation in Section 4. Section 5 uses two empirical applications to compare **bootUR**'s unit root functions to implementations in other R packages and illustrate its usefulness for practitioners. Section 6 concludes.

2. Unit Root Tests

We first discuss unit root tests for individual time series (Section 2.2), followed by testing multiple series for unit roots (Section 2.3). In our discussion, paralleling Smeekes and Wijler (2020), we do not focus on theory, but on the issues that arise for practitioners when implementing these tests on their time series. For a more extensive and theoretical overview of unit root testing, we refer the interested reader to Choi (2015).

2.1. Unit Roots

Consider the case where we have T observations from a time series y_t $(t = 1, \ldots, T)$ generated according to the data generating process (DGP)

$$y_t = x_t + \beta^\top d_t, \qquad x_t = \rho x_{t-1} + u_t, \tag{1}$$

where d_t are deterministic functions of time. In particular, three cases are commonly considered: $d_t = 0$ (no deterministic components), $d_t = 1$ (intercept only), and $d_t = (1, t)^\top$ (intercept and linear trend). The error process u_t is allowed to be serially correlated and heteroskedastic. The presence of serial correlation in u_t has to be accounted for in inference. Typically, u_t is modelled as an invertible infinite order linear process, for instance as

$$u_t = \sum_{j=0}^{\infty} \psi_j \epsilon_{t-j} = \sum_{j=1}^{\infty} \phi_j u_{t-j} + \varepsilon_t,$$

where ε_t is typically assumed to be a martingale difference sequence. This linearity motivates the use of adding lagged differences of the time series to account for the serial dependence, as in the classical augmented-Dickey-Fuller (ADF) test (Dickey and Fuller 1979). However, Paparoditis and Politis (2018) show that ADF-type approaches are valid under much more general forms of dependence in u_t.

We focus on testing whether or not y_t contains a unit root, that is on testing

$$H_0 : \rho = 1 \text{ against } \quad H_1 : |\rho| < 1$$

in equation (1). Under the null hypothesis of a unit root, y_t contains a stochastic trend, and equivalently y_t is being said to be integrated of order 1 ($I(1)$), while the alternative postulates that y_t is integrated of order 0 ($I(0)$), which is generally taken as synonymous to y_t being stationary. Here 'integrated of order d' means that y_t should be differenced d times to achieve

Table 1: Overview ADF-test functionalities in existing R packages.

	Package	**bootUR**	**CADFtest**	**fUnitRoots**	**tseries**	**urca**	
	Function	`boot_df()`	`CADFtest()`	`unitrootTest()`	`adf.test()`	`ur.df()`	`ur.ers()`
Deterministic components	Fixed				✓		
	User Control	✓	✓	✓		✓	✓
detrending	1-Step OLS		✓	✓	✓	✓	
	2-Step OLS	✓					
	2-Step QD	✓					✓
Lag selection	User Control	✓	✓	✓	✓	✓	✓
	AIC	✓	✓			✓	
	BIC	✓	✓			✓	
	MAIC	✓	✓				
	MBIC	✓					
	Rescaled	✓					
***p*-value**	Asymptotic		✓	✓	✓		
	Bootstrap	✓					

a process that does not contain a stochastic trend anymore.[4]

2.2. Individual Unit Root Tests

To test the null hypothesis of a unit root, the classical ADF test (Dickey and Fuller 1979, 1981) remains the pre-dominant choice in practice. For this reason it also forms the backbone of the **bootUR** package. However, even in its most basic form, practitioners are required to make several non-trivial choices that have a big impact on its performance. Table 1 summarizes these choices and indicates how the various R packages address each of them. In this section, we first discuss the ADF test and the choices that need to be made, before discussing the union of unit root tests principle proposed by Harvey *et al.* (2009, 2012) which alleviates many of the concerns.

ADF test The ADF t-statistic is the most popular unit root test in practice. Let Δ be the difference operator defined as $\Delta y_t := y_t - y_{t-1}$. If no deterministic components are present, the ADF regression is given by

$$\Delta y_t = \gamma y_{t-1} + \sum_{j=1}^{p} \phi_j \Delta y_{t-j} + \varepsilon_t, \qquad t = p+1, \ldots, T, \tag{2}$$

where the lagged differences of y_t are added to the regression to capture the serial correlation present in u_t. Testing the null of a unit root then boils down to testing the significance of the parameter γ in equation (2).

If the time series y_t is suspected to have deterministic components as well, testing becomes more complicated. The traditional one-step procedure adds the relevant deterministic components directly in (2). However, this may easily lead to confusion on which components to include, as under the null of a unit root, the coefficient of the linear trend cancels out. This has led many to erroneously perform tests including an intercept only, or to perform joint tests on γ and the coefficient of the linear trend (as suggested by Dickey and Fuller

[4]Although stationary is generally used as synonym for $I(0)$, an $I(0)$ process can still be non-stationary, for instance through a shift in the variance. Despite this distinction, we follow tradition and use '$I(0)$' and 'stationary' interchangeably.

1981), which, although correct, is unnecessary and complicates the development of a coherent framework. This one-step procedure is implemented in most **R** packages, see Table 1.

Instead, **bootUR** follows the two-step approach implemented by most modern unit root tests, such as the bootstrap tests considered in Section 3. Here, a first stage regression is run of y_t on the deterministic components d_t, and in the second stage the ADF regression

$$\Delta y_t^d = \gamma y_{t-1}^d + \sum_{j=1}^{p} \phi_j \Delta y_{t-j}^d + \varepsilon_t^d, \qquad t = p+1, \ldots, T, \tag{3}$$

is run on the residuals of the first stage regression, $y_t^d = y_t - \hat{\beta}^\top d_t$, commonly referred to as the detrended time series. The two-step procedure has the advantage that it disconnects the deterministic trend from the stochastic trend, which makes it easier to interpret. This procedure is implemented in the `boot_df()` function of the **bootUR** package, see Table 1.

The most straightforward choice to obtain the parameter $\hat{\beta}$ is by ordinary least squares (OLS), which is asymptotically equivalent to including the trend directly in the ADF regression. Alternatively, inspired by the work of Elliott, Rothenberg, and Stock (1996) and their DF-GLS test, one can obtain $\hat{\beta}$ by a generalized least squares (GLS) type of regression, where the (near-)unit root in y_t is first removed by quasi-differencing (QD); the regression is then performed by OLS for $y_t - \left(1 - \frac{c}{T}\right) y_{t-1}$ on $d_t - \left(1 - \frac{c}{T}\right) d_{t-1}$, where c is a parameter that determines how close to differencing the GLS step is; Elliott *et al.* (1996) recommend that $c = 7$ for the case $d_t = 1$ and $c = 13.5$ for the case $d_t = (1, t)^\top$ to yield tests with good power properties. The DF-GLS, or more accurately DF-QD[5], test is often considered to be more powerful than the ADF test; it is therefore surprising that it does not appear in many R packages; in fact, it seems a version with limited functionality is only available in the package **urca**, see Table 1. The actual relation between the OLS and QD detrended tests is more nuanced though. In particular, as shown by Müller and Elliott (2003) inter alia, the QD test is only more powerful if the initial condition, that is the deviation of the start of the time series from equilibrium, is small. When the initial condition is large, the standard OLS-detrended ADF test is considerably more powerful.

While both options are implemented in the function `boot_df()` for varying choice of d_t, one should realize that the seemingly innocuous issue of including deterministic components presents the practitioner with two difficult choices: which deterministic components to include, and how to perform the detrending. These choices can have a major impact on the performance. If too few deterministic components are included, deterministic trends are detected as stochastic trends, and the test becomes inconsistent. On the other hand, adding too many deterministic components reduces the power of the test considerably, and should also be avoided. As already noted by Campbell and Perron (1991), "A nonrejection of the unit root hypothesis may be due to misspecification of the deterministic components included as regressors" (p. 152). Of course, the trend parameters are not observable which complicates the choice in practice. Typically, the choice whether to include a trend or not is based on visual inspection of the time series. However, trend detection based on a plot is clearly very prone to errors, and even influenced by the resolution and format of the plot. Yet, all unit root tests in current R packages ask the user to make a choice without providing any guidance. Not only does this assume the user knows how to make that choice, but also that the user

[5]To avoid confusion with a 'proper' GLS estimation that also takes into account higher-order serial dependence and heteroskedasticty, we refer to this test as the quasi-differenced (QD) test rather than GLS.

has the opportunity and time to do this manually. The latter may be feasible for a handful of time series, but quickly becomes impossible for a modern high-dimensional dataset with perhaps hundreds of time series.

Similarly, the initial condition is unobservable, such that the user has to make an (un)educated guess as to which detrending method to use (if the package allows for a choice at all). Given the large power differences between the methods, we believe that we may add to Campbell and Perron's (1991) statement about deterministic components the following variation: "A nonrejection of the unit root hypothesis may be due to the chosen detrending method".

The **bootUR** package is, to the best of our knowledge, the first R package which does not force the user to make these choices, but instead offers the function `boot_union()` for a data-driven alternative via the union of rejections principle introduced by Harvey *et al.* (2009, 2012). Before discussing this in detail, we first turn to the third difficult choice a user has to make: selecting the lag length p in equation (3).

The lag length choice concerns a trade off between size distortions incurred from including too few lags to capture all serial correlation, and power loss incurred from including too many lags. Although theory (and some R packages such as **tseries**) generally assume p to be a deterministic function of the sample size, in practice a data-driven selection will clearly be more successful in managing a good trade off between size and power.

Popular choices for automatic data-driven lag length selection are information criteria and sequential tests. Sequential tests consider a sequence of t-tests on the largest lag, starting from the largest model. If the coefficient is found to be insignificant, the lag is removed and the next model considered. In the **bootUR** package, we do not consider sequential testing as, in general, information criteria are more popular and accurate than sequential testing (cf. Cavaliere, Phillips, Smeekes, and Taylor 2015, Remark 3).

Information criteria trade off model fit (through the residual sum of squares) and overfitting (through a penalty on the number of parameters). The lag length is estimated as

$$\hat{p} := \underset{p_{\min}\leq k\leq p_{\max}}{\operatorname{argmin}} \; IC(k), \qquad IC(k) = \ln\hat{\sigma}_k^2 + k\frac{C_T}{T}, \tag{4}$$

where $\hat{\sigma}_k^2 := (T-p_{\max})^{-1}\sum_{t=p_{\max}+1}^{T}(\hat{\varepsilon}_{k,t}^d)^2$ with $\hat{\varepsilon}_{k,t}^d$ the OLS residuals from the ADF regression with lag length k in equation (3), and C_T is a penalty function that differs according to the information criterion used. We consider two penalties: one corresponding to the Akaike information criterion (AIC; $C_T = 2$) and the other to the Bayesian information criterion (BIC; $C_T = \ln T$).

Next to the original criteria, **bootUR** also implements their modified variants proposed by Ng and Perron (2001). These modifications are specifically motivated for lag length selection in the ADF regression. They are given by

$$MIC(k) := \ln\hat{\sigma}_k^2 + k\frac{C_T + \xi_T(k)}{T},$$

where $\xi_T(k) := (\hat{\sigma}_k^2)^{-1}\hat{\gamma}^2\sum_{t=p_{\max}+1}^{T}(y_{t-1}^d)^2$. The lag length is then estimated as in (4), with $IC(k)$ replaced by $MIC(k)$. The modified AIC (MAIC) is obtained by taking $C_T = 2$, the modified BIC (MBIC), by taking $C_T = \ln T$. Ng and Perron (2001) show that the *MICs* yield large size improvements over the *ICs* for the purpose of unit root testing. Perron and Qu (2007) recommend to always use the *MICs* with the OLS rather than QD-detrended data (even

if the unit root test itself makes use of QD detrending) since this improves the test's power properties; **bootUR** follows this recommendation. In addition, there are various seemingly minor aspects of how the lag selection is implemented that influence its performance, such as how many observations are used to calculate the residual sum of squares. Ng and Perron (2005) provide a detailed study and guidelines for these choices; **bootUR** implements the scheme they recommend as optimal.

Cavaliere *et al.* (2015) find that heteroskedasticity affects the performance ofinformation criteria, leading to less accurate choices of p and consequent power loss of the unit root tests. They propose rescaled information criteria, where the time series y_t is rescaled with a nonparametric estimate of its (time-varying) standard deviation, thereby eliminating the heteroskedastictiy. The information criterion is then applied to this rescaled series. These rescaled ICs are generally more powerful in the presence of heteroskedasticity, yet very similar to the original ones without. **bootUR** therefore performs the rescaling by default (with the option not to consider it) since it is a safe choice and relieves the user of the burden to check whether heteroskedasticity is present.

Union of rejections test As mentioned above, choosing the right deterministic components to include and the right detrending method to use, is crucial to obtain tests with good power properties. However, making an informed, data-driven, choice is complicated. While deterministic trends can in principle be consistently detected, in practice a trend test will only detect large trends. Part of the problem is that such a test must be valid under both the unit root null and the stationary alternative - since we can only test for unit roots afterwards. The failure of such tests to detect trends means that based on such a pre-test one will often decide not to include a trend when it should have been included, which is the one scenario that must be avoided due to the test's inconsistency. Detecting a large initial condition is even more complicated, so a reliable data-driven pre-test is not an option.

Harvey *et al.* (2009, 2012) take a different approach based on a very simple principle. Roughly speaking, for both specification issues, we have one powerful test and one not powerful test. A logical step would therefore be to perform both tests and reject whenever one of them rejects the null hypothesis - the logic being that the one rejecting is then the powerful one. With two tests performed simultaneously, one must control for multiple testing and adjust the tests with a Bonferroni-type adjustment to control size at the desired level. Harvey *et al.* (2009) introduced this union of rejections idea for the two specification issues separately, while Harvey *et al.* (2012) combined the two approaches to consider a union of four tests – intercept only or intercept with trend in combination with OLS or QD detrending – that guards against both uncertainty over the trend and the initial condition. While the size correction makes the union test strictly less powerful than the optimal test, the power loss turns out to be small and this disadvantage is far out-weighted by the fact that the union test never breaks down unlike the individual tests.

This makes the union test a safe option for quick or automatic unit root testing where careful manual specification is not viable, and makes it therefore very suitable for **bootUR**'s philosophy that the default option provides a reliable and accurate test, for which no in depth knowledge is needed about either the data or the applicability of various unit root tests. Moreover, it scales easily to large datasets with many series, where careful manual considerations about these specifications are not possible regardless of the expertise of the user.

The **bootUR** package implements the bootstrap version of the union test developed by Smeekes and Taylor (2012), which uses the bootstrap both for determining the appropriate size correction and for obtaining the test's p-values. The test statistic takes the form

$$UR = \min\left(\frac{s}{c^{\mu*}_{QD}(\alpha)}QD^{\mu}, \frac{s}{c^{\tau*}_{QD}(\alpha)}QD^{\tau}, \frac{s}{c^{\mu*}_{ADF}(\alpha)}ADF^{\mu}, \frac{s}{c^{\tau*}_{ADF}(\alpha)}ADF^{\tau}\right), \tag{5}$$

where ADF and QD are the ADF and QD detrended tests, and superscript μ and τ respectively indicating whether the series are demeaned or detrended. The critical values $c^{\cdot}_{\cdot}(\alpha)$ are determined in a preliminary bootstrap step as the individual level α critical values of the four tests; weighting with their inverse is needed to bring the four tests on the same scale. The variable s is a scaling factor to which the statistics are scaled. Any $s < 0$ suffices to preserve the left-tail rejection region; in **bootUR** we scale to -1. This bootstrap union test is made available through the function `boot_union()`.

Finally, note that this union-based approach still requires one to select the lag lengths in each of the four ADF regressions. To this end, any of the four information criteria, AIC, BIC, MAIC and MBIC can be used.

Of course, various other unit root tests exist and are implemented in R, such as the Phillips and Perron (1988) (PP) test (**urca**), the seasonal HEGY test (**uroot**) and the KPSS (Kwiatkowski, Phillips, Schmidt, and Shin 1992) stationarity test (**fUnitRoots**, **tseries**, **urca**). We intentionally do not implement those in **bootUR** to avoid overloading the user with choices that are not easy to justify. Many of such tests, such as HEGY or KPSS have different testing setups that require careful consideration and understanding, while others, such as the PP test, suffer from serious size distortions even the bootstrap cannot fix. Moreover, the ADF test is by far the most popular in practice, and therefore we feel including such tests would only confuse the user, and a better approach is to provide a simple, coherent and reliable testing structure instead. In this line we can also mention the covariate-augmented test of Hansen (1995) implemented in the **CADFtest** package, which exploits correlation with *known stationary* covariates to improve power. While this is interesting if one wants to test a single series and has a set of stationary covariates at hand, this approach is difficult to implement if one has a dataset in which all series need to be tested for unit roots, and hence these series are not available as covariates. In such a setting it makes more sense to pool the tests, as done by panel unit root tests discussed in the next section.

2.3. Multiple Unit Root Tests

Practitioners often make use of several time series in their analysis, and typically need to test all for unit roots. While performing a unit root test for each series separately is normal practice for a small number of time series, this becomes more complicated if the number of series is large. First, performing many unit root tests simultaneously suffers from multiple testing issues as the probability of incorrect classifications increases with the number of performed tests. Second, we would like to exploit the similarity between different time series to improve the power of the unit root tests, in particular if the time dimension is relatively small.

In the **bootUR** package, we consider three different ways to approach the testing problem with multiple time series. First, the simplest option of ignoring the test multiplicity issue by just performing unit root tests separately for each series. To this end, the function `iADFtest()` from **bootUR** can be used. While not very appealing from a theoretical point of view, there

are practical reasons why one may still prefer this conceptually straightforward setup. We will explore this further in Section 4 when we elaborate on the package's functions. Second, we consider the traditional approach of panel unit root tests, where on pools the information in all series to obtain a more powerful test. The function `paneltest()` offers such a test. Third, we can consider individual tests but then with appropriate control of multiple testing error rates. **bootUR** considers two such tests, namely `BSQTtest()` and `bFDRtest()`.

Surprisingly, despite the large literature on this topic, software implementations for multiple unit roots are mostly lacking. While there is some support for panel unit root testing as discussed hereafter, methods to control multiple testing in the context of unit root testing are, to the best of our knowledge, not available. While several general purpose multiple testing packages exist, using these in a proper way with unit root tests requires considerable effort and expertise from the user. For instance, some standard corrections may be overly conservative, such as the Bonferroni correction, or only applicable under specific conditions on the dependence, such as Benjamini and Hochberg's (1995) method to control the false discovery rate. As argued by for instance Romano, Shaikh, and Wolf (2008b), bootstrap methods for controlling multiple testing allow for general forms of dependence and avoid being too conservative. However, such bootstrap methods need to be integrated with the unit root testing, which is the approach taken in **bootUR**.

Throughout this section, we use the following notation. Consider N time series for which one would like to test the presence of a unit root. We denote their respective individual unit root test statistics by UR_i, $1 \leq i \leq N$. Typically these would correspond to one of the tests discussed in Section 2.2. Without loss of generality, we assume that rejections occur for small values of the test statistic.

Panel Unit Root Tests Panel unit root tests view the multiple time series as a coherent panel dataset, and exploit the similarity between such time series to pool the information in them and achieve more powerful tests. They have a long tradition in econometrics, see e.g. Breitung and Pesaran (2008) or Choi (2015) for reviews. A typical panel unit root test has the null hypothesis that all series have a unit root. Rejection of this null hypothesis is then typically interpreted as evidence that a 'significant proportion' of the series is stationary. However, how large that proportion is, or which series are stationary is not revealed by the test. This makes panel unit root tests difficult to interpret, and limits their usefulness as pre-tests when determining the order of integration of each time series in a dataset. Nonetheless, the panel unit root null hypothesis may be interesting in its own right. Moreover, Pesaran (2012) suggests to use panel unit root tests as an initial screening tool for analyzing multiple series; if the panel unit root test rejects the null, this indicates that the individual series need to examined further; if not, treating the full dataset as $I(1)$ may be a reasonable choice. For these reasons **bootUR** also includes some functionality to test the panel unit root hypothesis, although this is not our main focus given the interpretational difficulties.

We implement the bootstrap Group-Mean (GM) test of Palm, Smeekes, and Urbain (2011)

$$GM = \frac{1}{N}\sum_{i=1}^{N} UR_i,$$

in the function `paneltest()` which is based on averaging the unit root test statistics UR_i ($1 \leq i \leq N$) of the N individual time series. This test is valid under very general forms of

dependence within the dataset, yet does not require modelling it. This in contrast to tests based on common factor models, which either require a complicated multi-step approach, or risk eliminating the unit roots by eliminating the common factors, thereby risking false rejections (Bai and Ng 2004, 2010). In contrast, the bootstrap test of Palm *et al.* (2011) is an easy, off-the-shelf method that fits **bootUR**'s philosophy. Panel unit roots tests are scarcely available for R users. Currently, only two packages with panel unit root tests, namely **plm** and **pdR**, are being maintained. The package **plm** was the first to offer panel unit root tests and provides the tests introduced in Maddala and Wu (1999); Choi (2001); Levin, Lin, and Chu (2002); Im, Pesaran, and Shin (2003). However, none of them allow for cross-sectional dependence (see Kleiber and Lupi 2011 for a discussion). The package **pdR** offers the panel unit root test of Chang (2002) and the seasonal test of Hylleberg, Engle, Granger, and Yoo (1990).

Multiple Testing Given the ambiguity of a panel unit root test's outcome, most practitioners will need to go one step further and determine the order of integration for each series in their dataset. In order to properly rank and compare different series, the individual test statistics should have the same marginal distributions. Then, the ranking

$$UR_{(1)} \leq \ldots \leq UR_{(R)} \leq UR_{(R+1)} \leq \ldots \leq UR_{(N)}, \tag{6}$$

corresponds to a ranking from 'most significant' to 'least significant', when the i-th order statistic of $UR_1, \ldots, UR_N$ is denoted by $UR_{(i)}$. To ensure the comparability of these statistics, nuisance parameters need to be eliminated from the distribution of the test statistics. **bootUR** does this automatically for the union test by scaling all test statistics in (5) towards $s = -1$; if the user chooses to set specifications manually, it is up to the user to choose them such that any nuisance parameters are eliminated.

The goal is to find an appropriate cut-off point R such that the null of a unit root is rejected for all statistics less than or equal to $UR_{(R)}$, while it is not rejected for all statistics larger. How this threshold is determined, depends on how one controls for multiple testing. **bootUR** implements two ways to do this: the sequential testing procedure of Smeekes (2015), which also encompasses the Step-M method of Romano and Wolf (2005) to control the family-wise error rate (FWE), and the false discovery rate (FDR) controlling approach of Romano *et al.* (2008b); Moon and Perron (2012).

Sequential Quantile Test Smeekes (2015) proposes a straightforward and fast-to-implement Bootstrap Sequential Quantile Test (BSQT) for multiple unit root testing, that acts as an intermediate between panel unit root testing and full multiple testing control. The method proceeds by sequentially testing groups of time series for unit roots, where the user decides the group sizes. At step 1, we test whether the first p_1 series are stationary. Here 'first' does not refer to the order in the dataset (which is arbitrary), but to the most significant tests as found via (6). If the null hypothesis that all p_1 units have a unit root cannot be rejected, the test stops. If we do observe a rejection, we move on to the second group where we test if the first p_2 are stationary. However, as we already concluded that the first p_1 units are stationary, in this second step the actual test is whether the next $p_2 - p_1$ units are stationary as well. We continue this testing procedure until no rejection is observed anymore or we tested all series in the dataset. The BSQT can be performed by using the function `BQSTtest()`.

More formally, let $p_1, \ldots, p_K$ be the number of series to be tested as stationary in each of the steps $k = 1, \ldots, K$. In the sequential step k we then test

$$H_0 : p_{k-1} \text{ series are } I(0); \qquad \text{against} \qquad H_1 : p_k \text{ series are } I(0).$$

As the first test should have as H_0 that all units are $I(0)$, $p_0 = 0$ by default. Furthermore, $p_K = N$ to complete the testing procedure. The number of steps K and the intermediate numbers $p_1, \ldots, p_{K-1}$ can be chosen by the practitioner. Instead of thinking in terms of p_k series, it may be easier to think in terms of quantiles q_k, and set $p_k = [q_k N]$. A practitioner may for instance think "I want to split my series in 10 equally-sized groups." In that case the practitioner simply sets $q_k = 0.1k$.

We acknowledge that the choice of $\{p_k\}$ does require input and consideration from the user, but unlike 'obscure' statistical arguments related to detrending for instance, the choice for $\{p_k\}$ can be done simply based on the nature of the dataset and the desired level of precision of the practitioner. Smeekes (2015) shows that if p_k units are found to be $I(0)$, the probability that the true number of stationary series lies outside the interval $[p_{k-1}, p_{k+1}]$ is at most the chosen significance level of the test. Finding that p_k series are $I(0)$ should therefore be interpreted as finding that the number of $I(0)$ series is in the interval $[p_{k-1}, p_{k+1}]$. In the end, if $p_2, \ldots, p_{K-1}$ are chosen sensibly and not spaced too far apart, the series that lie in the 'uncertain interval' are likely those series which are 'just about' significant, and correspond to time series with a ρ parameter very close to 1. The practical consequences of incorrect classification of these series are typically small, as their behavior makes them fit reasonably well in both classes of $I(1)$ and $I(0)$ series.

One special case worth mentioning – set as the default in `BQSTtest()` – is when we set $p_k = k$, such that each series gets tested sequentially. Not only does this remove uncertainty about the interpretation of the result, but Smeekes (2015) also shows that in this case the BSQT method coincides with the popular Step-M method of Romano and Wolf (2005) to control the familywise error rate (FWE). The FWE is defined as the probability of making at least one false rejection, and is typically controlled via the Bonferroni or Holm (1979) approach. Romano and Wolf (2005) show that the Step-M method is considerably more powerful than the aforementioned approaches, as the bootstrap method it is based on can capture the true dependence between the series, and therefore does not have to be valid also in worst case scenarios. However, one should still realize that the FWE is very strict and overly conservative if N is large, and this particular implementation of BSQT is mainly suitable for relatively small datasets.

FDR-controlled Test The false discovery rate (FDR), originally proposed by Benjamini and Hochberg (1995), is defined as $FDR = \mathbb{E}\left[\frac{F}{R}\mathbb{1}(R > 0)\right]$, where R denote the total number of rejections, and F the number of false rejections. It is more appropriate for larger N than the FWE, as it aims to control the proportion of false rejections to the total, rather than the probability of a single false rejection. Romano *et al.* (2008b) develop a bootstrap method to control the FDR, and show that unlike the classical way to control FDR, the bootstrap is appropriate under very general forms of dependence between series. Moon and Perron (2012) applied this method to unit root testing, and it is their method that is implemented in the `bFDRtest()` function of the **bootUR** package.

The bootstrap algorithm to control FDR, however, is quite complex, which is likely the reason why, to our knowledge, it is not available in R outside of **bootUR**. The algorithm proceeds

Table 2: Bootstrap methods and their ability to deal with serial correlation, general forms of heteroskedasticity, cross-sectional dependence and unbalancedness of the data.

Bootstrap method	Serial correlation	Heteroskedasticity	Cross-sectional dependence	Unbalancedness
SB	✓			
MBB	✓		✓	
SWB	✓	✓		✓
DWB	✓	✓	✓	✓
BWB	✓	✓	✓	✓
AWB	✓	✓	✓	✓

sequentially, in a step-down way, by starting to test the 'most' significant series (i.e. the one with the smallest unit root test statistic). This statistic is then compared to an appropriate bootstrap-based critical value, where the bootstrap evaluates all possible scenarios in terms of false and true rejections given the current stage of the algorithm. If the null can be rejected for the current series, the algorithm proceeds to the next 'most' significant series and the procedure is repeated. The algorithm stops as soon as the null cannot be rejected. Full details can be found in Romano, Shaikh, and Wolf (2008a). While the algorithm is hard to understand intuitively, the practitioner using the `bFDRtest()` function does not have to worry about this, as our fast C++ implementation does all the heavy lifting, such that this FDR-controlling test becomes a method like any other. As for the other multiple time series methods, FDR control can be combined with any unit root test specification considered in Section 2.2, although we recommend the default union test for the reasons described there.

To decide on whether to use BSQT or FDR control, relative sample sizes can be considered. The Monte Carlo comparison of Smeekes (2015) reveals that the FDR-controlling test is somewhat more accurate when the sample size T is at least of equal magnitude as the number of time series N, whereas the BSQT method is clearly preferable when T is much smaller than N, since the FDR-controlling test then suffers from a lack of power.

3. Bootstrap-based Inference

We rely on bootstrap methods to obtain critical values and/or p-values for all of the unit root tests discussed in Section 2. In the **bootUR** package, six bootstrap methods are implemented: the sieve bootstrap (SB), moving block bootstrap (MBB), sieve wild bootstrap (SWB), dependent wild bootstrap (DWB), block wild bootsrap (BWB) and autoregressive wild bootstrap (AWB). Their properties are summarized in Table 2, and discussed more extensively below. As immediately apparent from Table 2, any 'off-the-shelf' time series bootstrap method may be used to counteract size distortions arising from neglected serial correlation (Schwert 1989); whereas a wild bootstrap method is needed to deal with general forms of heteroskedasticity (Cavaliere and Taylor 2008, 2009a). General forms of cross-sectional dependence can be captured by any bootstrap method apart from the sieve ones.

Next to correcting the size of unit root tests, bootstrap methods have other advantages. First, the bootstrap offers an automatic p-value. This means no additional steps have to be taken to obtain p-values, such as done in packages **CADFtest** or **fUnitRoots** for example. Second, the bootstrap directly allows for implementation of multiple testing techniques such as those

discussed above. Moreover, as already mentioned, as the bootstrap captures the dependence between series, it allows for less conservative, and hence more powerful, tests than methods which use worst case scenarios to ensure validity. Second, it guards against misspecification and uncertainty regarding the lag length selection in the ADF. As **bootUR** re-selects the lag lengths within the bootstrap replications, it automatically takes effects of lag selection into account. This, coupled with the fact that the bootstrap captures any dependence missed by the lagged differences in the ADF regression, adds another layer of protection to the tests.

3.1. Sieve bootstrap

The sieve bootstrap (SB) has been extensively considered in the context of unit root testing; see among others Psaradakis (2001), Chang and Park (2003), Paparoditis and Politis (2005), Palm, Smeekes, and Urbain (2008) and Smeekes (2013). It estimates the dependence as an autoregressive (AR) process, resamples the residuals of the AR fit, and then re-applies the AR model recursively to place the dependence back into the bootstrap sample. This simple and intuitive setup has made it historically popular among practitioners. **bootUR** determines the required order of the AR model by the order of the ADF model, combining these in a single step as they should conceptually coincide.

While it is able to capture general forms of serial dependence (Kreiss, Paparoditis, and Politis 2011), it is mostly suited for tests on single time series. Smeekes and Urbain (2014b) show that it is not suited to capture general forms of cross-sectional dependence, making it invalid for joint or multiple testing. The **bootUR** package therefore advises to only use it for unit root testing of a single series or on multivariate series without multiple testing control, throwing a warning to alert the user otherwise. When still applied multivariately (against better judgment perhaps), users should also realize that each time series is required to be observed over the same periods, which we refer to as balanced datasets. This often forces practitioners to delete observations for series that have been observed for a longer period, a practice that is wasteful. The reason for this limitation is that resampling step of the sieve bootstrap would reshuffle the missing values, creating bootstrap sample with 'holes' in it.

3.2. Moving block bootstrap

The moving block bootstrap (MBB) is another traditional bootstrap method that has not only been used for univariate unit root testing in Paparoditis and Politis (2003), but also for multivariate unit root testing in Moon and Perron (2012) and Smeekes (2015), as well as for panel unit root testing in Palm *et al.* (2011). It works by dividing the data in overlapping blocks of data and resampling those blocks to create bootstrap series by laying them end-to-end. The blocks are taken in the time dimension and encompass all series. This way the MBB can accommodate any form of serial dependence as long as it 'fits' into an adequately sized block, which is a wide class. Unlike the SB, the MBB can also handle general forms of dependence between series, including but not limited to common factor structures. From a practical point of view an attractive features is that it can be applied without requiring one to model the serial and/or cross-sectional dependence. Palm *et al.* (2011) show its validity for mixed $I(1)/I(0)$ panel datasets under such general forms of dependence.

The block length ℓ is set automatically by **bootUR** as a function of the sample size, following a rule proposed by Palm *et al.* (2011) that they showed to perform well in many different circumstances. However, it is easily adjusted by the user to experiment with different lengths

and assess the sensitivity of the results for varying block lengths.

The MBB still has, however, two disadvantages: it cannot handle unbalanced datasets and is sensitive to unconditional heteroskedasticity. The latter makes its use in various application domains, such as macro-economics or finance, problematic. To handle both issues, users should switch to one of the wild bootstrap methods available in **bootUR**.

3.3. Sieve wild bootstrap

The wild, or multiplier, bootstrap (Mammen 1993; Davidson and Flachaire 2008) is known to be robust against general forms of heteroskedasticity, however it cannot handle serial dependence. Nonetheless, if combined with a sieve bootstrap, we get the best of both worlds. That is, by replacing the resampling step applied to the residuals of the AR model with a multiplication by independent and identically distributed (iid) random variables with mean zero and variance one, we obtain the sieve wild bootstrap (SWB). Cavaliere and Taylor (2009a,b) and Smeekes and Taylor (2012) among others apply this sieve wild bootstrap for bootstrap unit root testing. The method is perfectly suited to individual unit root testing, but due the AR estimation, suffers from the same inability to capture complex dependence across series as explained by Smeekes and Urbain (2014b) for the SB. Hence, the **bootUR** package warns against its use in multivariate settings. For the generation of the iid random variables, **bootUR** uses the normal distribution, which is the same choice as the unit root papers cited above.

3.4. Dependent, Block and Autoregressive wild bootstrap

The three remaining bootstrap methods implemented in the package are all wild bootstrap methods adjusted to deal with dependence. However unlike the SWB, here the multiplicative random variables themselves are adjusted to be dependent over time. This setup allows these bootstrap methods to capture complex serial and cross-series dependence structures as well as (unconditional) heteroskedasticity. In addition, no resampling takes place for the DWB, such that missing values 'stay in their place' which makes the method applicable to unbalanced datasets. These bootstrap methods therefore tick all the boxes in Table 2, making them very suitable for unit root testing.

The three wild bootstrap methods only differ in how the multiplier variables are made time-dependent. The dependent wild bootstrap method (DWB), originally introduced by Shao (2010), draws random variables from a T-dimensional $N(0, \Sigma)$ distribution, where the elements in Σ decrease with the distance between them. Shao (2010) proposes to use a kernel function to achieve this, along with a bandwidth ℓ which ensures that variables more than ℓ time points apart are independent. This way ℓ has a similar interpretation as the block length in the MBB. Rho and Shao (2019) and Smeekes and Urbain (2014a) study the DWB for unit root testing, the latter focusing on multivariate settings.

We consider two more variations. The block wild bootstrap (BWB) (Shao 2011; Zhang and Cheng 2014) is a direct alternative to the MBB, where for each block of size ℓ, we use the same multiplier variable, and the variables are independent between blocks. The autoregressive wild bootstrap (AWB) (Smeekes and Urbain 2014a; Friedrich, Smeekes, and Urbain 2020) generates the multiplier variables as a first-order autoregressive process. Unlike the BWB and DWB who have a block length ℓ tuning parameter, the tuning parameter of the AWB is the first-order AR parameter. To be able to use the same tuning parameter ℓ, we use

the conversion formula proposed by Smeekes and Urbain (2014a) and Friedrich *et al.* (2020) that writes the AR parameter as a function of ℓ, though **bootUR** also allows to set the AR parameter directly. The default setting for ℓ in **bootUR** uses the same rule as for the MBB, which was also tested for the three wild bootstrap methods by Smeekes and Urbain (2014a). They also provide theoretical results on the validity of these methods under general forms of dependence and heteroskedasticity.

For completeness, in Algorithm 1 we present the six bootstrap methods and their role in the general bootstrap algorithm. Note that the outcome of the bootstrap algorithm is a collection of bootstrap unit root test statistics UR_i^b for the series $i = 1, \ldots, N$ and bootstrap replications $b = 1 \ldots, B$. How these are then used depends on the multiple testing approach taken. For instance, if we ignore multiple testing, we simply calculate the bootstrap p-values

$$p_i^* = \frac{1}{B} \sum_{b=1}^{B} I(UR_i^b < UR_i), \qquad i = 1, \ldots, N.$$

For the BSQT and FDR tests more involved processing is needed; for details we refer to Smeekes (2015) and Romano *et al.* (2008a) respectively.

4. An introduction to the bootUR package

The **bootUR** package has a simple structure with twelve user-accessible functions. Section 4.1 presents three functions to check if the data are suitable to be bootstrapped. Sections 4.2 and 4.3 introduce the six core functions for unit root testing on respectively individual and multiple time series. Section 4.4 presents three useful functions for determining the order of integration of each series in a particular dataset.

The package's functions will now be presented together with examples of their specific use. To this end, we make use of the dataset `MacroTS` which contains a collection of 20 macroeconomic time series taken from Eurostat and comes with the package. A complete description of the data can be obtained by typing `?MacroTS` in R. The following examples require that the **bootUR** ackage and the data have been loaded. As random number generation is required to draw bootstrap samples, we first set the seed of the random number generator to obtain reproducible results:

```
R> library(bootUR)
R> data("MacroTS")
R> set.seed(155776)
```

4.1. Checking data suitability

To check if a particular dataset is suitable to be bootstrapped, three simple functions can be used, namely `check_missing_insample_values()`, `find_nonmissing_subsample()` and `plot_missing_values()`. While the bootstrap tests do not work with missing data, unbalanced datasets are generally allowed (see Table 2). The function `check_missing_insample_values()` checks if a particular dataset contains missing values. Its usage is extremely simple, as it only requires the data as input,

```
R> check_missing_insample_values(MacroTS)
```

Algorithm 1: Multivariate Bootstrap Unit Root Tests

Let $y^d_{i,t} = y_{i,t} - d_t\hat{\beta}$, where $\hat{\beta}$ is obtained by OLS;

for $i \in \{1, \dots, N\}$ **do**

  Estimate (3) for $\{y_{i,t}\}_{t=1}^T$, determining p by an appropriate criterion, obtaining estimates $(\hat{\gamma}_i, \hat{\phi}_{i,1}, \dots, \hat{\phi}_{i,p})$;

  Set $\hat{u}_{i,t} = \Delta y^d_{i,t} - \hat{\rho}_i y^d_{i,t-1}$ and $\hat{\varepsilon}_{i,t} = \Delta y^d_{i,t} - \hat{\rho}_i y^d_{i,t-1} - \sum_{j=1}^p \hat{\phi}_{i,j}\Delta y^d_{i,t-j}$, with $y_{-p+1}, \dots, y_o = 0$;

end

for $b \in \{1, \dots, B\}$ **do**

  if *SB* **then**

    Generate $s_1, \dots, s_T$ from a Uniform distribution on $\{1, \dots, T\}$;

    Set $u^b_{i,t} = \sum_{j=1}^p u^b_{i,t-j} + \varepsilon^b_{i,t}$ with $\varepsilon^b_{i,t} = \hat{\varepsilon}_{i,s_t}$ and $u_{-p+1}, \dots, u_0 = 0$ for $i = 1, \dots, N$ and $t = 1, \dots, T$;

  else if *MBB* **then**

    Generate $s^b_1, \dots, s^b_{\lceil T/\ell \rceil}$ from a Uniform distribution on $\{1, \dots, T - \ell + 1\}$;

    for $m \in \{1, \dots, \lceil T/\ell \rceil\}$ **do**

      Set $u^b_{i,t} = \hat{\varepsilon}_{i,s^b_t}$ for $i = 1, \dots, N$ and $t = (m-1)\ell + 1, \dots, m\ell$;

    end

  else if *SWB* **then**

    Generate $\xi^b_1, \dots, \xi^b_T$ from a $N(0,1)$ distribution;

    Set $u^b_{i,t} = \sum_{j=1}^p u^b_{i,t-j} + \varepsilon^b_{i,t}$ with $\varepsilon^b_{i,t} = \xi^b_t\hat{\varepsilon}_{i,t}$ and $u_{-p+1}, \dots, u_0 = 0$ for $i = 1, \dots, N$ and $t = 1, \dots, T$;

  else if *DWB* **then**

    Generate $\zeta^b_1, \dots, \zeta^b_T$ from a $N(0,1)$ distribution and let $\xi^b = (\xi^b_1, \dots, \xi^b_T)^\top = \Sigma^{1/2}(\zeta^b_1, \dots, \zeta^b_T)^\top$ with $(\sigma_{s,t})_{s,t=1}^T = K\left(\frac{|s-t|}{\ell}\right)$ for the kernel function $K(\cdot)$ defined in Shao (2010);

    Set $u^b_{i,t} = \xi^b_t\hat{u}_{i,t}$ for $i = 1, \dots, N$ and $t = 1, \dots, T$;

  else if *BWB* **then**

    Generate $\xi^b_1, \dots, \xi^b_{\lceil T/\ell \rceil}$ from a $N(0,1)$ distribution;

    for $m \in \{1, \dots, \lceil T/\ell \rceil\}$ **do**

      Set $u^b_{i,t} = \xi^b_m\hat{u}_{i,t}$ for $i = 1, \dots, N$ and $t = (m-1)\ell + 1, \dots, m\ell$;

    end

  else if *AWB* **then**

    Generate $\zeta^b_2, \dots, \zeta^b_T$ from a $N(0, 1-\gamma^2)$ distribution and let $\xi^b_t = \gamma\xi^b_{t-1} + \zeta^b_t$ with $\xi^b_1 \sim N(0,1)$;

    Set $u^b_{i,t} = \xi^b_t\hat{u}_{i,t}$ for $i = 1, \dots, N$ and $t = 1, \dots, T$;

  Set $y^b_{i,t} = \sum_{s=1}^t u^b_{i,s}$;

  Let $UR^b_i = UR(y^b_{i,1}, \dots, y^b_{i,T})$, where $UR(\cdot)$ denotes the chosen unit root test.

end

which can be a vector, matrix, data frame or in time series format (e.g. `ts`, `zoo` or `xts`). It returns an N-dimensional Boolean vector which indicates for each series whether missing values are present (`TRUE`) or not (`FALSE`).

If a dataset contains series with different starting and end points, the bootstrap methods SWB, DWB, BWB and AWB can still be used. The function `find_nonmissing_subsample()` lets users check the start and end points of each series as follows:

```
R> sample_check <- find_nonmissing_subsample(MacroTS)
R> sample_check
```

```
$range
      GDP_BE GDP_DE GDP_FR GDP_NL GDP_UK CONS_BE CONS_DE CONS_FR CONS_NL CONS_UK
first      1      1      1      5      1       1       1       1       5       1
last     100    100    100    100    100     100     100     100     100     100
      HICP_BE HICP_DE HICP_FR HICP_NL HICP_UK UR_BE UR_DE UR_FR UR_NL UR_UK
first       9       9       9       9       9     1     1     1     1     1
last      100     100     100     100     100   100   100   100   100   100

$all_equal
[1] FALSE
```

The output slot `range` returns a $(2 \times N)$-matrix displaying the first and last non-missing value for each series, the logical slot `all_equal` provides a quick check to see if all time series have the same non-missing indices (`TRUE`) or not (`FALSE`).

Finally, to display missingness in the dataset, we can use

```
R> plot_missing_values(MacroTS, show_names = TRUE)
```

which displays present cell values in green, missing values at the start or end ('Unbalanced NAs') in purple and internal missing values in red, see Figure 1. Only the latter are problematic for the wild bootstrap methods, while the purple values also need to be avoided for the resampling-based bootstraps.

4.2. Individual Unit Root Tests

bootUR has two functions to perform a bootstrap unit root test on a single series: `boot_df()` for a standard ADF test and `boot_union()` for a union test. Below, we start by discussing the many options users can tweak in `boot_df()`. As **bootUR** shares its syntax across the various functions, the majority of function arguments remains identical across **bootUR**'s functions, which facilitates usability and control by the end-user. In the remainder, we therefore only highlight the differences compared to the `boot_df()` function.

ADF test To perform a standard ADF bootstrap unit root test on a single series, the `boot_df()` function can be used. The function is structured as follows:

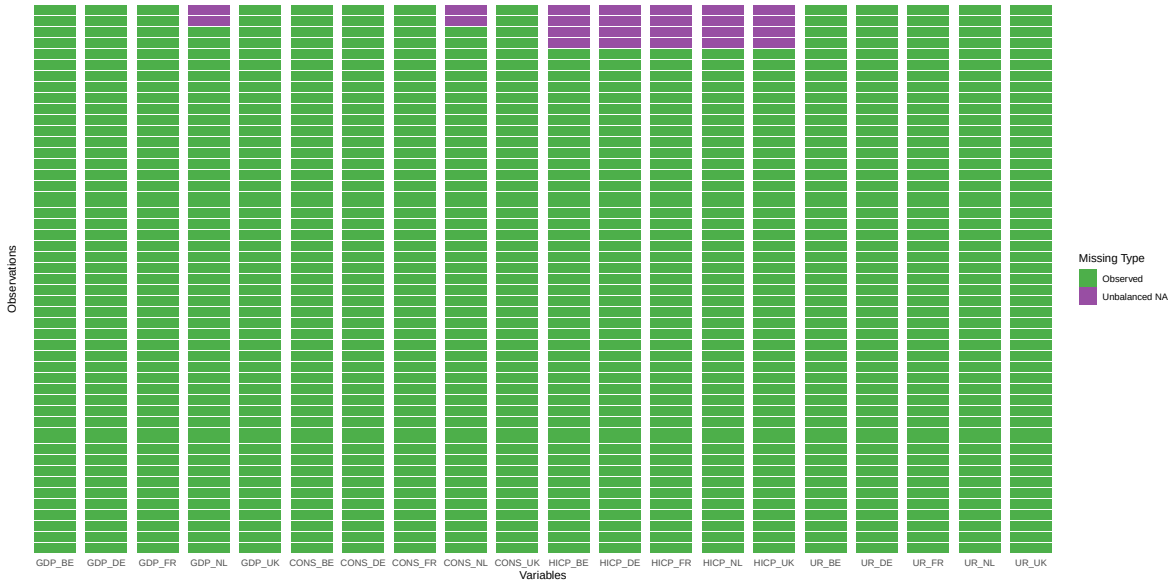

Figure 1: Missingness for the dataset `MacroTS`.

```
boot_df(y, level = 0.05, boot = "AWB", B = 1999, l = NULL, ar_AWB = NULL,
  p_min = 0, p_max = NULL, ic = "MAIC", dc = 1, detr = "OLS", ic_scale = TRUE,
  verbose = FALSE, show_progress = FALSE, do_parallel = FALSE, nc = NULL)
```

The minimum required input is `boot_df(y)`, where the time series `y` can be a vector or a time series object. All other arguments are set to sensible default values for reliable, accurate and generally applicable unit root testing. Yet, users are able to easily tweak all arguments to their desired settings.

The remaining arguments in the first line relate to the bootstrap specifications, including the desired significance level of the test (`level`), bootstrap method (`boot`) and number of bootstrap replications (`B`). If a user chooses the bootstrap method `"MBB"`, `"DBB"` `"BWB"` or `"AWB"`, the desired block length can be controlled via the argument `l`. By default, we use `l =` $\lfloor 1.75 \cdot T^{1/3} \rfloor$, as recommended in Palm *et al.* (2011). While for the first three, this argument concerns the genuine block length, for the latter, the block length is transformed into an autoregressive parameter `ar_AWB` via the formula $0.01^{(1/l)}$ as in Smeekes and Urbain (2014a); this can be overwritten by setting `ar_AWB` directly.

The set of arguments on the second line relates to the ADF regression. The deterministic components can be tweaked via the argument `dc`, the type of detrending via `detr`. The remaining arguments concern the lag length selection: `p_min` and `p_max` respectively control the minimum and maximum lag length, the information criterion can be selected via the argument `ic` and the option `ic_scale` lets practitioners choose to use the rescaled information criteria of Cavaliere *et al.* (2015). To overwrite data-driven lag selection with a pre-specified lag length, users can simply put both `p_min` and `p_max` equal to the desired lag length.

The arguments `verbose` and `show_progress` allow additional information to be printed: the option `verbose = TRUE` prints easy to read output on the unit root test to the console, the option `show_progress = TRUE` provides live progress updates on the bootstrap. The latter is particularly useful for large values of the argument `B`. Finally, the option `do_parallel =`

`TRUE` allows the bootstrap to be executed in parallel on systems where **OpenMP** is supported; the argument `nc` allows users to specify how many cores should be used for the parallel loops. By default, all but one cores are used. If the parallel option is selected on a system where **OpenMP** is not supported, evaluation will simply be serial.

We illustrate the bootstrap ADF test on Dutch GDP, with the sieve bootstrap (`boot = "SB"`) as used by Palm *et al.* (2008) and Smeekes (2013). An intercept and linear time trend are added as deterministic components and detrending is done via both OLS and QD.

```
R> GDP_NL <- MacroTS[, 4]
R> adf_out <- boot_df(GDP_NL, boot = "SB", dc = 2, detr = c("OLS", "QD"),
+   verbose = TRUE, do_parallel = TRUE)
```

Since `verbose = TRUE`, the outcome of the unit root test (test statistic and p-value) can be easily read from the console. Both tests indicate that the unit root null cannot be rejected:

```
Bootstrap DF Test with SB bootstrap method.
----------------------------------------
Type of unit root test performed: detr = OLS, dc = intercept and trend
test statistic        p-value
    -2.5152854      0.1310655
----------------------------------------
Type of unit root test performed: detr = QD, dc = intercept and trend
test statistic        p-value
    -1.5965001      0.4187094
```

Union of rejections test To perform a bootstrap union unit root test on a single series, the `boot_union()` function can be used. It shares all its arguments with `boot_df()` except for `dc` and `detr` which are omitted since `boot_union()` implicitly uses `dc = c(1,2)` and `detr = c("OLS", "QD")`, then combines the outcomes of the four unit root tests, as in equation (5), to produce a single p-value.

The bootstrap union test for Dutch GDP with the sieve wild bootstrap as proposed by Smeekes and Taylor (2012) can be obtained via

```
R> union_out <- boot_union(GDP_NL, boot = "SWB", verbose = TRUE,
+   do_parallel = TRUE)

Bootstrap Test with SWB bootstrap method.
Bootstrap Union Test:
The null hypothesis of a unit root is not rejected at a significance
                    level of 0.05.
test statistic        p-value
    -0.6701345      0.6433217
```

4.3. Multiple Unit Root Tests

Below, we discuss the various approaches **bootUR** offers to approach the testing problem with multiple series.

Separate Unit Root Tests To perform individual ADF tests on multiple time series simultaneously without multiple testing control, the function `iADFtest()` can be used:

```
iADFtest(y, level = 0.05, boot = "AWB", B = 1999, l = NULL, ar_AWB = NULL,
  union = TRUE, p_min = 0, p_max = NULL, ic = "MAIC", dc = NULL, detr = NULL,
  ic_scale = TRUE, verbose = FALSE, show_progress = FALSE, do_parallel = FALSE,
  nc = NULL)
```

Compared to the syntax of `boot_df()`, it has one additional argument, namely `union` which controls whether a bootstrap union test is used (`TRUE`) or not (`FALSE`). If `union = TRUE` (default), the arguments `dc` and `detr` are ignored, and a warning message is returned if the user would have provided specifications for these anyway. If set to `FALSE`, the deterministic components and detrending methods can be specified as for the `boot_df()` function. Furthermore, since the bootstrap is performed for all series simultaneously, the bootstrap methods `"SB"` or `"MBB"`, that cannot handle unbalanced datasets, should not be used. If the user were to specify these anyway, the function will revert to splitting the bootstrap up and performing it separately for each time series. A warning message is then returned to alert the user. If a vector (or univariate time series) instead of a matrix (multivariate time series) is given for `y`, a single unit root test is performed; in this case the function acts as an alternative to `boot_df()` and `boot_union()`.[6]

We illustrate the function's usage by performing individual ADF tests with the `"MBB"` bootstrap on the first five series of the unbalanced dataset `MacroTS`, which correspond to the real Gross Domestic Product in Belgium, Germany, France, the Netherlands and the United Kingdom respectively.

```
R> iADF_out <- iADFtest(MacroTS[, 1:5], boot = "MBB", verbose = TRUE,
+   do_parallel = TRUE)

There are 0 stationary time series.
       test statistic    p-value
GDP_BE     -0.8135022 0.36618309
GDP_DE     -1.1076021 0.08804402
GDP_FR     -0.6301366 0.76188094
GDP_NL     -0.8210610 0.41370685
GDP_UK     -0.7207147 0.53876938

Warning message:
In check_inputs(y = y, BSQT_test = BSQT_test, iADF_test = iADF_test,  :
  Missing values cause resampling bootstrap to be executed for each time
  series individually.
```

None of the time series is stationary, as printed to the console together with detailed information on the value of the test statistic and p-value for each time series (since `verbose = TRUE`). The warning message alerts the user about the resampling `"MBB"` bootstrap method being unable to handle unbalanced datasets and the corrective action that is taken to this end.

[6] In fact, internally these functions call `iADFtest()`.

The user can easily access all information through the list with two components that is returned:

```
R> iADF_out
```

```
$rej_H0
[1] FALSE FALSE FALSE FALSE FALSE

$ADF_tests
       test statistic    p-value
GDP_BE     -0.8135022 0.36618309
GDP_DE     -1.1076021 0.08804402
GDP_FR     -0.6301366 0.76188094
GDP_NL     -0.8210610 0.41370685
GDP_UK     -0.7207147 0.53876938
```

The slot `rej_H0` contains a vector of length N indicating for each series whether the unit root null is rejected (`TRUE`) or not (`FALSE`). The slot `ADF_tests` contains the values of the test statistics and p-values. For the union test, the output is arranged per time series. If no union test is performed, the output is arranged per time series, type of deterministic component and detrending method.

Panel Unit Root Test To perform a panel unit root test, the function `paneltest()` can be used. It shares its syntax with `iADFtest()`. Unlike for the latter, usage of the `"MBB"` or `"SB"` bootstrap methods for a panel unit root test on unbalanced datasets will result in an error– not a warning –since the unbalancedness cannot be reverted. Therefore, users should switch to one of the wild bootstrap methods. Besides, sieve bootstrap methods can be used, but they are not suited to capture general forms of dependence across units (see Table 2). The code therefore warns users against their usage.

We illustrate the usage of the panel unit root test on the five GDP time series with the `"DWB"` bootstrap of Shao (2010) and Rho and Shao (2019):

```
R> panel_out <- paneltest(MacroTS[, 1:5],  boot = "DWB", verbose = TRUE,
+   do_parallel = TRUE)
```

```
Panel Bootstrap Group-Mean Union Test
The null hypothesis that all series have a unit root, is not
                  rejected at a significance level of 0.05.
     test statistic   p-value
[1,]     -0.8371329 0.2956478
```

The outcome of the test is printed on the console (since `verbose = TRUE`). Since the null is not rejected, treating all five GDP series as $I(1)$ is reasonable.

Sequential Quantile Test To perform the BSQT for multiple unit root testing, the function `BSQTtest()` should be used. It has one additional argument compared to the `paneltest()` function, namely `q` which sets the group sizes. These can either be set in units or in quantiles. To split the series in, for instance, K equally sized groups, use `q = 0:K / K`. By the convention of Smeekes (2015), the first entry of the vector should be equal to zero, while the second entry indicates the end of the first group, and so on. If the initial zero value or the final value (N or 1 for quantiles) are accidentally omitted, the function automatically adds them back. The default `q = 0:NCOL(y)` corresponds to the Step-M method of Romano and Wolf (2005). Regarding the bootstrap methods, the same warning and error messaging as for the `paneltest()` apply.

We illustrate the BSQT on the five GDP series with the `"AWB"` (default) bootstrap method of Smeekes and Urbain (2014a) and Friedrich *et al.* (2020):

```
R> BSQT_out <- BSQTtest(MacroTS[, 1:5], verbose = TRUE, do_parallel = TRUE)

There are 0 stationary time series.
Details of the BSQT sequential tests:
       Unit H0 Unit H1 Test statistic   p-value
Step 1       0       1      -1.045657 0.3346673
```

The number of stationary time series is printed to the console (`verbose = TRUE`), together with details on the number of series p_k to be tested as stationary in step k under H_0 and H_1 (first two columns), and the test-statistic and p-value (last two columns) for each of the sequential steps until no rejection occurs. The latter information is also accessible through the output slot `BSQT_sequence`, details on the (non) rejection of the unit root null for each of the series separately can be accessed via the slot `rej_H0`, similar to the function `iADFtest()`.

FDR-controlled Test To perform a multiple unit root test by controlling the FDR, the function `bFDRtest()` should be used. Its arguments are the same as for the other multivariate unit root tests, though the meaning of the argument `level` changes from the regular significance level to the FDR level. We illustrate it here with the `"BWB"` bootstrap method of Shao (2011) and Smeekes and Urbain (2014a):

```
R> bFDR_out <- bFDRtest(MacroTS[, 1:5], boot = "BWB", verbose = TRUE,
+   do_parallel = TRUE)

There are 0 stationary time series
Details of the FDR sequential tests:
       test statistic critical value
GDP_DE     -0.9813749      -1.346138
```

The procedure developed by Romano *et al.* (2008b); Moon and Perron (2012) does not provide p-values, therefore critical values are returned instead here. All information can be accessed via the output slots `rej_H0` and `FDR_sequence`, which reports all tests until no rejection occurs.

4.4. Determining series' order of integration

Finally, **bootUR** offers three useful functions for determining the order of integration of each series in dataset: `order_integration()`, `diff_mult()` and `plot_order_integration()`. The main function is `order_integration()` which applies the 'Pantula principle' (Pantula 1989) to determine the order of integration of each series

```
order_integration(y, max_order = 2, test = NULL, plot_orders = FALSE, ...)
```

The argument `max_order` sets the maximum order of integration that should be considered for each series. Generally the default of two should generally suffice, with series of order three or higher only very rarely occurring in practice. The user can choose the unit root test through the argument `test` depending on whether a single (`"boot_df"` or `"boot_union"`) or a multiple time series (`"iADFtest"`, `"BSQTtest"` or `"bFDRtest"`) is considered. All arguments used in these functions can also be passed on via `order_integration()`.

The Pantula principle works as follows. It starts by setting $d =$ `max_order` $- 1$ and testing for a unit root on the $\Delta^d y_t$ series. The series for which the unit root null cannot be rejected are classified as $I(d+1)$ and subsequently removed from the dataset. In the next step, $d = d - 1$ and the remaining series are tested and classified accordingly. Under the default `max_order = 2`, this second round involves testing the series in levels and classifying them as either $I(1)$ (if the unit root null is not rejected) or $I(0)$ (if the null is rejected).

The function returns a list with two elements. The slot `diff_data` contains a matrix whose columns are $\Delta^{d_i} y_{i,t}$ with d_i indicating the order of integration of the i^{th} series ($i = 1, \ldots, N$). This matrix is generated by the user-accessible function `diff_mult(y, d)`, where `y` is the original dataset and `d` is an N-dimensional vector indicating each series' order of integration. It contains the same number of rows as the original dataset (since the default setting `keep_NAs = TRUE` in `diff_mult()` is used), thereby indicating lost observations as missing. It can be tweaked if a practitioner directly makes use of this function. Note that the object of the input series (e.g. matrix, data frame or time series object) is preserved for the differenced series. The output slot `order_int` makes the vector `d` containing the found orders of integration available to the end-user.

Finally, if the argument `plot_orders` in the function `order_integration()` is set to `TRUE`, a plot is provided which displays each series' order of integration. To this end, it uses the function `plot_order_integration(d)` with minimal required input being the same vector `d`. This function is also made accessible if the end-user wishes to further adjust the display of the variable names, legend and colours through its optional arguments `show_names`, `show_legend`, `names_size`, `legend_size` and `cols`.

5. Applications

We illustrate the methods on two datasets, the `MacroTS` dataset which comes with the package, and the FRED-QD dataset, which is widely used for macro-economic analysis.

MacroTS The `MacroTS` dataset contains $N = 20$ macro-economic time series collected from Eurostat (https://ec.europa.eu/eurostat/data/database) and is included in the package. Quarterly observations from 1995-2019 ($T = 100$) are available on GDP, consumption,

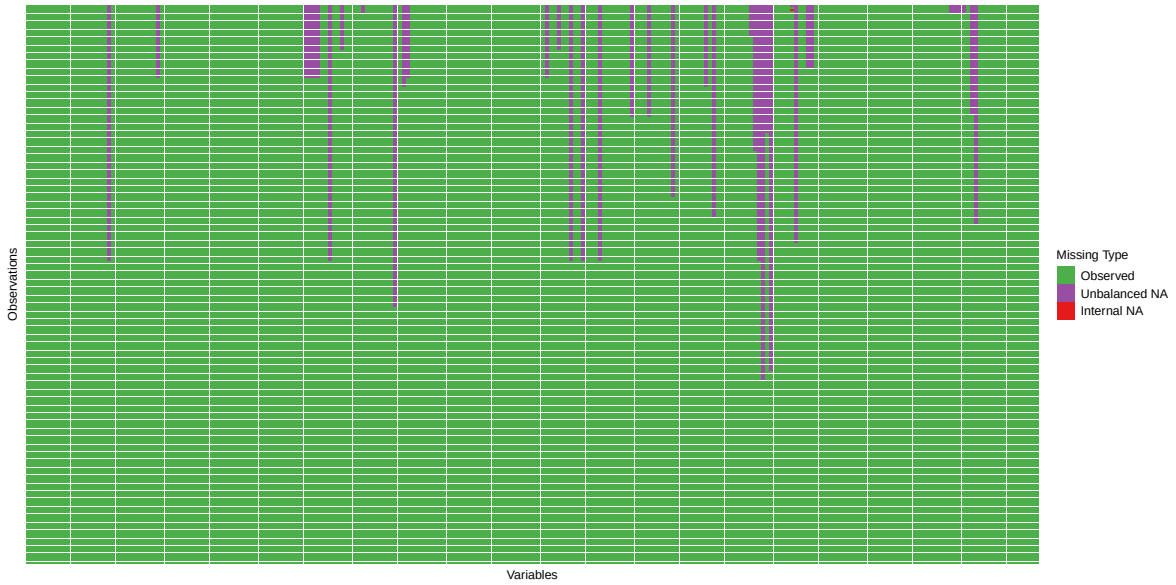

Figure 2: Missingness for the dataset FRED-QD.

inflation and unemployment for Belgium, Germany, France, the Netherlands and the United Kingdom. The dataset is unbalanced, see Figure 1.

FRED-QD This is a quarterly version of the monthly Federal Reserve Economic Data database introduced in McCracken and Ng (2016). It contains $N = 248$ macro-economic time series and was imported into R using the commands

```
R> FRED_url <- url("https://s3.amazonaws.com/files.fred.stlouisfed.org/fred-md/
 + quarterly/2020-06.csv")
R> FRED_QD <- read.csv(FRED_url)
```

This paper uses the data from 1959 Quarter 2 to 2019 Quarter 4 ($T = 244$) to avoid possible structural breaks due to the COVID-19 pandemic in 2020. If a researcher wishes to import the up-to-date version of the dataset, `2020-06.csv` should be changed to `current.csv`. As can be seen from Figure 2, the dataset contains one internal NA, since the third observation of variable 188 (`UMCSENTx`: Consumer Expectations) is missing while the second observation is not. **bootUR** cannot handle internal missing values but this can be easily fixed by setting the second observation to `NA`, which results in the first three observations of this variable being 'unbalanced NAs' that can be handled by **bootUR**. The resulting dataset then contains 38 macro-economic indicators with missing values at the start of the sample. Finally, note that all FRED-QD series have been classified into $I(0), I(1), I(2)$ by the transformation codes provided in McCracken and Ng (2020). However, the authors themselves indicate several discrepancies between these codes and the outcome of unit root tests. We therefore use the transformation codes as a benchmark for the classifications obtained through the unit root tests but do not necessarily consider the classification closest to theirs to be the best.

Since some of the macro-economic series are likely to be $I(2)$, we use the `order_integration()` function (with its defaults) to implement the Pantula principle. All unit root tests in the

Table 3: p-values of the panel unit root test on all series in differences and all series in levels, for both the `MacroTS` and FRED-QD dataset.

Series	`MacroTS`	FRED-QD
in first differences	< 0.001	< 0.001
in levels	0.087	0.168

bootUR are performed with their default settings, which means that union tests are performed with the `AWB` bootstrap method, and lag length selection is done via the re-scaled MAIC. Throughout this section, a significance level of 5% is used. For `BSQTtest()`, the default (i.e. Step-M method) is reported as well as results for evenly spaced 0.1 quantiles (`q = 0:10/10`, for `MacroTS`, FRED-QD), and 0.05 quantiles (`q = 0:20/20`, for FRED-QD).

We compare **bootUR**'s unit root tests to the R packages reported in Table 1. We hereby use the following specifications: For the function `CADFtest()` (package **CADFtest**), we perform ADF-regressions with intercept and trend (`type = "trend"`), and lag length selection with MAIC (`criterion = "MAIC"`) thereby considering a maximum of $\lfloor 12 \cdot (T/100)^{1/4} \rfloor$ lags, set via the argument `max.lag.y`. These lag length specifications correspond to the defaults used in **bootUR**. From the existing unit root packages, we find **CADFtest** to be the one with the most appealing API which also allows full user control of important model specifications and provides easy to read off results. As such we consider this package to be our main reference point, but we also include results from the other packages to evaluate the sensitivity of the test outcomes on the chosen package. For `unitrootTest()` (package **fUnitRoots**), we perform ADF-regressions with intercept and trend (`type = "ct"`). By default, one lagged difference is included. For `adf.test()` (package **tseries**), we use its default settings which implies ADF-regressions with intercept and trend and the number of lags fixed to $\lfloor (T-1)^{1/3} \rfloor$, a deterministic function of the sample size. For `ur.df()` (package **urca**), we use ADF-regressions with intercept and trend (`type="trend"`), lag length selection via AIC (`selectlags = "AIC"`), thereby considering a maximum of $\lfloor 12 \cdot (T/100)^{1/4} \rfloor$ lags, set via the argument `lags`. Finally, for `ur.ers()` (package **urca**), we use an intercept and trend for detrending (`model` = "trend"). By default, four lagged differences are included in the ADF-regression. Unlike the other packages, **urca** only comes with critical values to judge the significance of the unit root test, the p-value is not reported, see Table 1. As discussed in Lupi (2009), the p-value reported under `summary()` is computed using the t-distribution, which is incorrect under the unit root null. Finally, only the packages **CADFtest** and **fUnitRoots** can handle missing values, for the other packages, we removed missing values prior to performing the unit root tests.

Before applying the various unit root tests to the two datasets, we perform the `paneltest()` (with default settings) to all series taken in first differences, and to all series in levels. Table 3 reports the p-values of the panel unit root tests. For both datasets, the panel unit root tests on the series in first differences indicates that the unit root null is rejected, thereby indicating that a 'significant proportion' of the series is stationary in first differences (hence not $I(2)$). The panel unit root test on the series in levels indicates non-rejection of the unit root null.

To shed further light on the order of integration for each of the individual series, the bootstrap unit root tests are applied and compared to the implementations from other R packages. Figure 3 presents the obtained orders of integration on the `MacroTS` dataset, Figures 4 and 5 on the FRED-QD dataset.

Figure 3: Classification of the `MacroTS` dataset into $I(0), I(1), I(2)$.

Globally speaking, most unit root tests agree upon a series' classification into $I(0), I(1), I(2)$, which is comforting. Still, several interesting remarks can be made. First, the results of `iADFtest` are fairly similar to `BSQTtest` and `bFDRtest` but it classifies a considerable amount of series as $I(0)$ instead of $I(1)$ on the FRED-QD dataset. This illustrates that ignoring multiple testing can quickly lead to a considerable number of misclassifications on such large datasets. Second, among the `BSQTtest` procedures, the default Step-M method tends to classify more series as $I(2)$ than the other two procedures. As discussed in Section 2.3, we only recommend its usage for small datasets. On the smaller `MacroTS` dataset, for instance, the two versions of the `BSQTtest` show more agreement than on the larger FRED-QD dataset. Third, `bFDRtest` tends to classify more series as $I(1)$ than the other tests. For a more elaborate discussion of this tendency, we refer the interested reader to Smeekes and Wijler (2020). Fourth, among the unit root tests from the other R packages, `CADFtest()` produces most similar results to **bootUR**. The function `unitrootTest()` detects far less series as $I(2)$. While different implementation of these unit root test do produce different results and it thus matters which test is used in practice, we do find that the unit root tests in the **bootUR** package tend to produce more stable results with respect to the series' order of integration.

6. Summary

This paper presents the R package **bootUR** that provides a unified framework for bootstrap unit root testing on single and multiple time series. To this end, the package builds upon the popular augmented Dickey-Fuller (ADF) test with a union of rejections principle. Unlike existing packages on unit root tests, **bootUR** (i) provides a large collection of easy-to-use, fully-controllable and reliable unit root tests, including the union of rejections test which is set as default to enable quick, automatic unit root testing, (ii) ensures accurate inference through bootstrap methods with easy-to-read output (including p-values), (iii) allows for testing the presence of unit roots in datasets containing many time series by relying on fast C++ implementations.

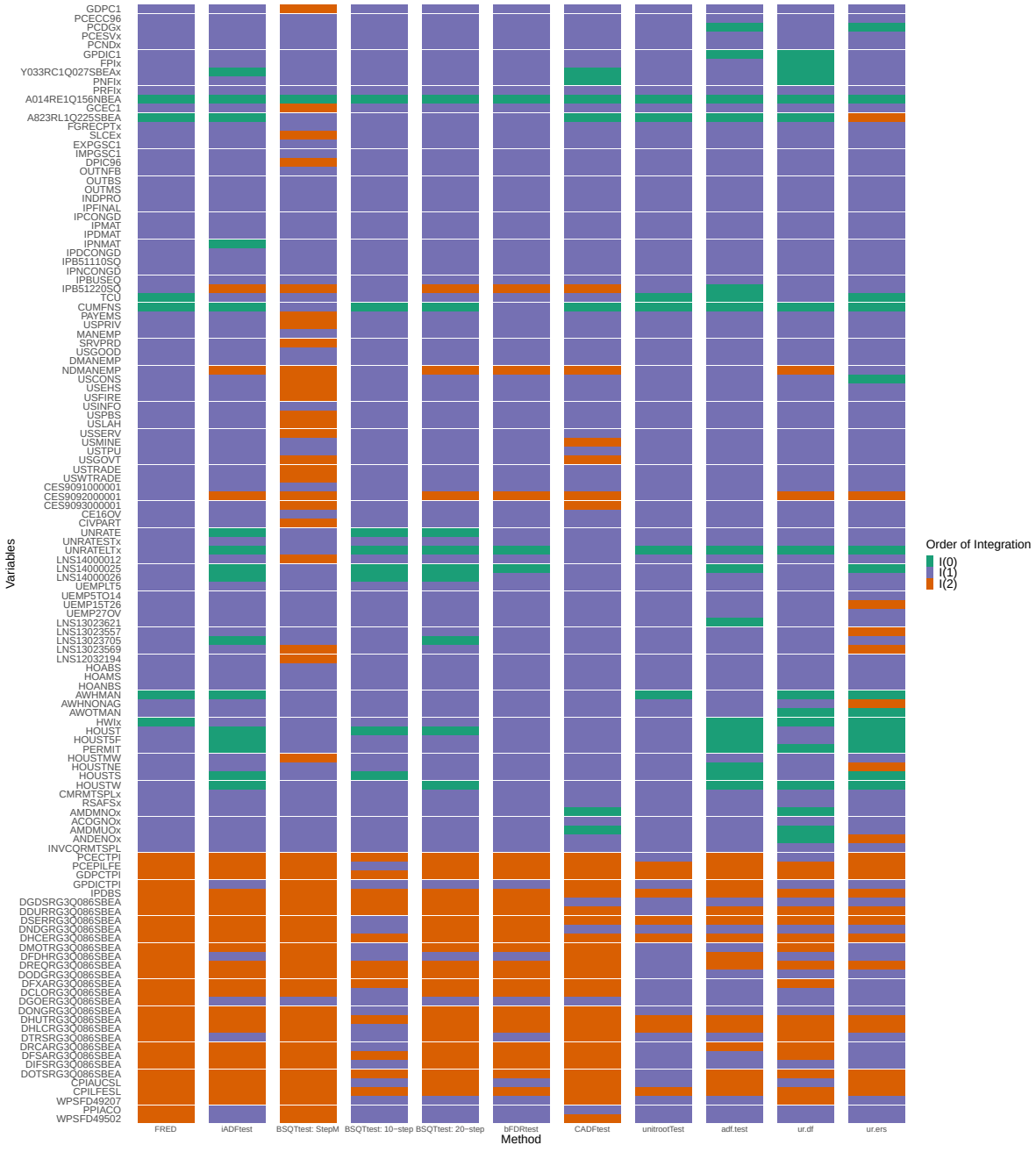

Figure 4: Classification of the first half of time series in the FRED-QD dataset into $I(0), I(1), I(2)$.

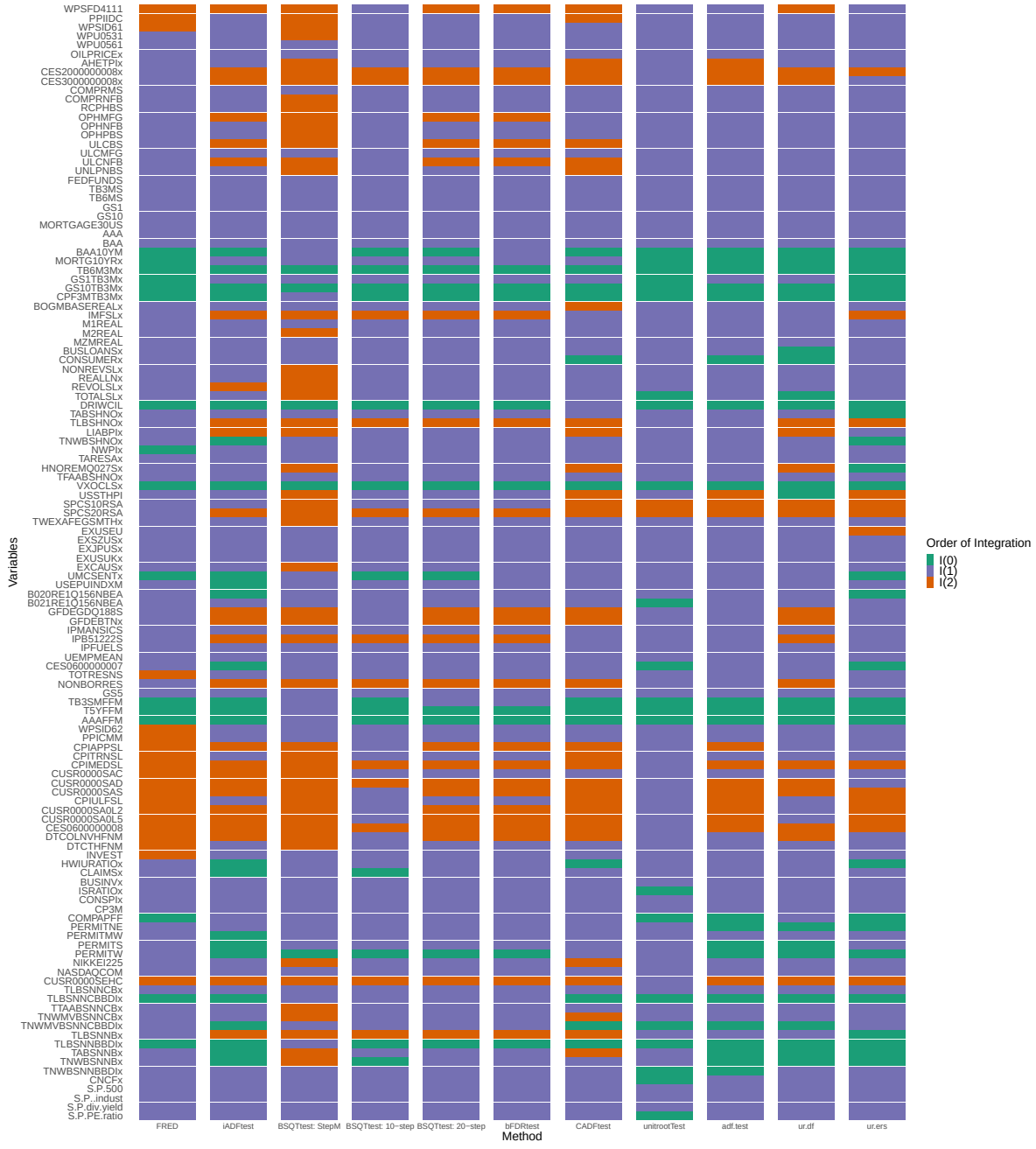

Figure 5: Classification of the second half of time series in the FRED-QD dataset into $I(0), I(1), I(2)$.

Acknowledgments

The first author was financially supported by the Netherlands Organization for Scientific Research (NWO) under grant number 452-17-010, the second author by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 832671. We gratefully acknowledge the comments and checks provided by Robert Adamek, Rui Jorge Almeida, Nalan Baştürk, Caterina Schiavoni and Etiënne Wijler on earlier versions of the package. All remaining errors are our own.

References

Bai J, Ng S (2004). "A PANIC attack on unit roots and cointegration." *Econometrica*, **72**(4), 1127–1177.

Bai J, Ng S (2010). "Panel unit root tests with cross-section dependence: A further investigation." *Econometric Theory*, **26**(4), 1088–1114.

Benjamini Y, Hochberg Y (1995). "Controlling the false discovery rate: a practical and powerful approach to multiple testing." *Journal of the Royal Statistical Society: Series B*, **57**(1), 289–300.

Breitung J, Pesaran MH (2008). "Unit roots and cointegration in panels." In *The Econometrics of Panel Data*, pp. 279–322. Springer.

Bronder S (2016). ***PANICr**: PANIC Tests of Nonstationarity*. R package version 1.0.0, URL https://CRAN.R-project.org/package=PANICr.

Campbell JY, Perron P (1991). "Pitfalls and opportunities: What macroeconomists should know about unit roots." *NBER Macroeconomics Annual*, **6**, 141–201.

Cavaliere G (2005). "Unit root tests under time-varying variances." *Econometric Reviews*, **23**(3), 259–292.

Cavaliere G, Phillips PCB, Smeekes S, Taylor AMR (2015). "Lag length selection for unit root tests in the presence of nonstationary volatility." *Econometric Reviews*, **34**(4), 512–536.

Cavaliere G, Taylor AMR (2008). "Bootstrap unit root tests for time series with nonstationary volatility." *Econometric Theory*, **24**(1), 43–71.

Cavaliere G, Taylor AMR (2009a). "Bootstrap M unit root tests." *Econometric Reviews*, **28**(5), 393–421.

Cavaliere G, Taylor AR (2009b). "Heteroskedastic time series with a unit root." *Econometric Theory*, pp. 1228–1276.

Chang Y (2002). "Nonlinear IV unit root tests in panels with cross-sectional dependency." *Journal of Econometrics*, **110**(2), 261–292.

Chang Y, Park JY (2003). "A sieve bootstrap for the test of a unit root." *Journal of Time Series Analysis*, **24**(4), 379–400.

Choi I (2001). "Unit root tests for panel data." *Journal of International Money and Finance*, **20**(2), 249–272.

Choi I (2015). *Almost All About Unit Roots: Foundations, Developments, and Applications*. Cambridge University Press.

Croissant Y, Millo G (2008). "Panel Data Econometrics in R: The **plm** Package." *Journal of Statistical Software*, **27**(2), 1–43. doi:10.18637/jss.v027.i02.

Dagum L, Menon R (1998). "**OpenMP**: an industry standard API for shared-memory programming." *Computational Science & Engineering, IEEE*, **5**(1), 46–55.

Davidson R, Flachaire E (2008). "The wild bootstrap, tamed at last." *Journal of Econometrics*, **146**, 162–169.

Dickey DA, Fuller WA (1979). "Distribution of estimators for autoregressive time series with a unit root." *Journal of the American Statistical Association*, **74**(366a), 427–431.

Dickey DA, Fuller WA (1981). "Likelihood ratio statistics for autoregressive time series with a unit root." *Econometrica*, pp. 1057–1072.

Eddelbuettel D (2013). *Seamless R and C++ Integration with **Rcpp***. Springer, New York. doi:10.1007/978-1-4614-6868-4. ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta JJ (2017). "Extending R with C++: A Brief Introduction to **Rcpp**." *PeerJ Preprints*, **5**, e3188v1. ISSN 2167-9843. doi:10.7287/peerj.preprints.3188v1. URL https://doi.org/10.7287/peerj.preprints.3188v1.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ integration." *Journal of Statistical Software*, **40**(8), 1–18.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R with high-performance C++ linear algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063.

Elliott G, Rothenberg TJ, Stock JH (1996). "Efficient tests for an autoregressive unit root." *Econometrica*, **64**(4), 813–836.

Enders W (2008). *Applied Econometric Time Series*. 4th edition. John Wiley & Sons.

Friedrich M, Smeekes S, Urbain JP (2020). "Autoregressive wild bootstrap inference for nonparametric trends." *Journal of Econometrics*, **214**(1), 81–109.

Hansen BE (1995). "Rethinking the univariate approach to unit root testing: Using covariates to increase power." *Econometric Theory*, pp. 1148–1171.

Harvey DI, Leybourne SJ, Taylor AMR (2009). "Unit root testing in practice: dealing with uncertainty over the trend and initial condition." *Econometric Theory*, **25**(3), 587–636.

Harvey DI, Leybourne SJ, Taylor AMR (2012). "Testing for unit roots in the presence of uncertainty over both the trend and initial condition." *Journal of Econometrics*, **169**(2), 188–195.

Holm S (1979). "A simple sequentially rejective multiple test procedure." *Scandinavian Journal of Statistics*, **6**, 65–70.

Hylleberg S, Engle RF, Granger CW, Yoo BS (1990). "Seasonal integration and cointegration." *Journal of Econometrics*, **44**(1-2), 215–238.

Im KS, Pesaran MH, Shin Y (2003). "Testing for unit roots in heterogeneous panels." *Journal of Econometrics*, **115**(1), 53–74.

Kleiber C, Lupi C (2011). "Panel unit root testing with R." URL https://r-forge.r-project.org/scm/viewvc.php/*checkout*/pkg/inst/doc/panelUnitRootWithR.pdf.

Kleiber C, Lupi C (2012). ***punitroots***: *Tests for Unit Roots in Panels of (Economic) Time Series, With and Without Cross-sectional Dependence*. R package version 0.0-2, URL https://r-forge.r-project.org/projects/punitroots/.

Kreiss JP, Paparoditis E, Politis DN (2011). "On the range of validity of the autoregressive sieve bootstrap." *Annals of Statistics*, **39**, 2103–2130.

Kwiatkowski D, Phillips PC, Schmidt P, Shin Y (1992). "Testing the null hypothesis of stationarity against the alternative of a unit root." *Journal of Econometrics*, **54**(1-3), 159–178.

Levin A, Lin CF, Chu CSJ (2002). "Unit root tests in panel data: asymptotic and finite-sample properties." *Journal of Econometrics*, **108**(1), 1–24.

López-de Lacalle J, Boshnakov GN (2019). ***uroot***: *Unit Root Tests for Seasonal Time Series*. R package version 2.1-0, URL https://CRAN.R-project.org/package=uroot.

Lupi C (2009). "Unit root CADF testing with R." *Journal of Statistical Software*, **32**(2), 1–19.

MacKinnon JG, Haug AA, Michelis L (1999). "Numerical distribution functions of likelihood ratio tests for cointegration." *Journal of Applied Econometrics*, **14**(5), 563–577.

Maddala GS, Wu S (1999). "A comparative study of unit root tests with panel data and a new simple test." *Oxford Bulletin of Economics and Statistics*, **61**(S1), 631–652.

Mallet O (2017). ***URT***: *Fast Unit Root Tests and OLS regression in C++ with wrappers for R and Python*. URL https://github.com/olmallet81/URT.

Mammen E (1993). "Bootstrap and wild bootstrap for high dimensional linear models." *Annals of Statistics*, **21**, 255–285.

McCracken M, Ng S (2020). "FRED-QD: A quarterly database for macroeconomic research." *Working Paper 26872*, National Bureau of Economic Research.

McCracken MW, Ng S (2016). "FRED-MD: A monthly database for macroeconomic research." *Journal of Business & Economic Statistics*, **34**(4), 574–589.

Moon HR, Perron B (2012). "Beyond panel unit root tests: Using multiple testing to determine the non stationarity properties of individual series in a panel." *Journal of Econometrics*, **169**(1), 29–33.

Müller UK, Elliott G (2003). "Tests for unit roots and the initial condition." *Econometrica*, **71**(4), 1269–1286.

Ng S, Perron P (2001). "Lag length selection and the construction of unit root tests with good size and power." *Econometrica*, **69**(6), 1519–1554.

Ng S, Perron P (2005). "A note on the selection of time series models." *Oxford Bulletin of Economics and Statistics*, **67**, 115–134.

Palm FC, Smeekes S, Urbain JP (2008). "Bootstrap unit root tests: comparison and extensions." *Journal of Time Series Analysis*, **29**(1), 371–401.

Palm FC, Smeekes S, Urbain JP (2011). "Cross-sectional dependence robust block bootstrap panel unit root tests." *Journal of Econometrics*, **163**(1), 85–104.

Pantula SG (1989). "Testing for unit roots in time series data." *Econometric Theory*, **5**(2), 256–271.

Paparoditis E, Politis DN (2003). "Residual-based block bootstrap for unit root testing." *Econometrica*, **71**(3), 813–855.

Paparoditis E, Politis DN (2005). "Bootstrapping unit root tests for autoregressive time series." *Journal of the American Statistical Association*, **100**, 545–553.

Paparoditis E, Politis DN (2018). "The asymptotic size and power of the augmented Dickey-Fuller test for a unit root." *Econometric Reviews*, **37**(9), 955–973.

Perron P, Qu Z (2007). "A simple modification to improve the finite sample properties of Ng and Perron's unit root tests." *Economics Letters*, **94**(1), 12–19.

Pesaran MH (2012). "On the interpretation of panel unit root tests." *Economics Letters*, **116**(3), 545–546.

Pfaff B (2008). *Analysis of Integrated and Cointegrated Time Series with R*. Second edition. Springer, New York. ISBN 0-387-27960-1, URL http://www.pfaffikus.de.

Phillips PC, Perron P (1988). "Testing for a unit root in time series regression." *Biometrika*, **75**(2), 335–346.

Psaradakis Z (2001). "Bootstrap tests for an autoregressive unit root in the presence of weakly dependent errors." *Journal of Time Series Analysis*, **22**, 577–594.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rho Y, Shao X (2019). "Bootstrap-assisted unit root testing With piecewise locally stationary errors." *Econometric Theory*, **35**(1), 142–166.

Romano JP, Shaikh AM, Wolf M (2008a). "Control of the false discovery rate under dependence using the bootstrap and subsampling." *Test*, **17**(3), 417–442.

Romano JP, Shaikh AM, Wolf M (2008b). "Formalized data snooping based on generalized error rates." *Econometric Theory*, **24**(2), 404–447.

Romano JP, Wolf M (2005). "Stepwise multiple testing as formalized data snooping." *Econometrica*, **73**(4), 1237–1282.

Schwert GW (1989). "Tests for unit roots: a Monte Carlo investigation." *Journal of Business and Economic Statistics*, **7**(1), 147–159.

Shao X (2010). "The dependent wild bootstrap." *Journal of the American Statistical Association*, **105**(489), 218–235.

Shao X (2011). "A bootstrap-assisted spectral test of white noise under unknown dependence." *Journal of Econometrics*, **162**(2), 213–224.

Smeekes S (2013). "Detrending bootstrap unit root tests." *Econometric Reviews*, **32**(8), 869–891.

Smeekes S (2015). "Bootstrap sequential tests to determine the order of integration of individual units in a time series panel." *Journal of Time Series Analysis*, **36**(3), 398–415.

Smeekes S, Taylor AMR (2012). "Bootstrap union tests for unit roots in the presence of nonstationary volatility." *Econometric Theory*, **28**(2), 422–456.

Smeekes S, Urbain JP (2014a). "A multivariate invariance principle for modified wild bootstrap methods with an application to unit root testing." *GSBE Research Memorandum RM/14/008*, Maastricht University.

Smeekes S, Urbain JP (2014b). "On the applicability of the sieve bootstrap in time series panels." *Oxford Bulletin of Economics and Statistics*, **76**(1), 139–151.

Smeekes S, Wijler E (2020). "Unit roots and cointegration." In P Fuleky (ed.), *Macroeconomic Forecasting in the Era of Big Data*, volume 52 of *Advanced Studies in Theoretical and Applied Econometrics*, chapter 17, pp. –541–584. Springer.

Smeekes S, Wilms I (2020). ***bootUR**: Bootstrap Unit Root Tests*. R package version 0.2.0, URL https://CRAN.R-project.org/package=bootUR.

Trapletti A, Hornik K, LeBaron B (2019). ***tseries**: Time Series Analysis and Computational Finance*. R package version 0.10-47, URL https://CRAN.R-project.org/package=tseries.

Tsung-wu H (2019). ***pdR**: Threshold Model and Unit Root Tests in Cross-Section and Time Series Data*. R package version 1.7, URL https://CRAN.R-project.org/package=pdR.

Wuertz D, Setz T, Chalabi Y (2017). ***fUnitRoots**: Rmetrics - Modelling Trends and Unit Roots*. R package version 3042.79, URL https://CRAN.R-project.org/package=fUnitRoots.

Zhang X, Cheng G (2014). "Bootstrapping high dimensional time series." ArXiv e-print 1406.1037.

Zhang Y, Yu H, McLeod AI (2011). "Maximum likelihood unit root test." *Working Paper*.

Affiliation:

Stephan Smeekes, Ines Wilms
Quantitative Economics, School of Business and Economics
Maastricht University
Tongersestraat 53
6211 LM Maastricht, the Netherlands
E-mail: s.smeekes@maastrichtuniversity.nl, i.wilms@maastrichtuniversity.nl
URL: https://www.stephansmeekes.nl, https://sites.google.com/view/iwilms