GPU Parallel Implementation of Numerical Distribution Functions for Seasonal Unit Root Tests

Javier López-de-Lacalle*

June, 2016
DRAFT VERSION: PLEASE DO NOT CITE WITHOUT PERMISSION

Abstract

This paper describes a parallel implementation of multiple linear regression models that are run on a general-purpose Graphics Processing Unit. Seasonal unit root test statistics are obtained from each fitted regression model. The code has two main applications: simulation of response surfaces for different combinations of parameters and bootstrapping the test statistics. Each of this applications provides a different method to compute p-values of seasonal unit root tests, for which available tabulated critical values are limited. Computing intensive operations are implemented in the CUDA platform and programming model. An interface in the R language and environment is provided for the computation of p-values by means of both approaches.

Key words: seasonal unit root tests, response surface, bootstrap, parallel computing, GPU, CUDA, R.

1 Introduction

The distribution of unit root tests is often non-standard and tabulated values obtained by means of simulations are often employed. In the case of seasonal unit root tests, Hylleberg *et al.* (1990) (hereafter HEGY) obtain the most common critical values for seasonal unit root tests in quarterly data, Beaulieu and Miron (1993) provide tabulated values for monthly data and Cáceres (1996) computes some of the quantiles of the distribution of these statistics in weekly data. Further critical values for the HEGY test in biannual, quarterly, bimonthly and monthly data are given in Franses and Hobijn (1997). The critical values of these test statistics depend on several elements: the model specification (e.g., including a constant and/or trend and/or seasonal dummies), the sample size, the periodicity of the data and lags included in the regression. Tabulated values provided in the papers cited above do not cover all possible combinations of these elements that may arise in practice. Critical values for sample sizes not considered in the original tables can be obtained by interpolation in those tables. For periodicities not considered in the papers mentioned above (e.g., daily data), critical values are not available.

*Universidad del País Vasco/Euskal Herriko Unibertsitatea. E-mail: javier.lopezdelacalle@ehu.eus. Website: http://jalobe.com.

In the last years, the approach described in MacKinnon (1996) has been applied to obtain approximations to the distribution of the unit root test statistics. The idea involves extensive simulations where several quantiles of the distribution of the test statistic are obtained. The results from the simulations are then employed to calculate p-values for any sample size of the data by means of response-surface regressions. MacKinnon (1994) applies this idea in the framework of the Dickey and Fuller test (Dickey and Fuller, 1979), Cheung and Lai (1995) do the same exercise for the augmented Dickey and Fuller test, Harvey and van Dijk (2006) employ the same idea in the HEGY test and Otero and Smith (2012) apply this technique to a variation of the Dickey and Fuller unit root test proposed in Leybourne (1995). Lupi (2009) obtains response surfaces for the covariate-augmented Dickey-Fuller (CADF) test proposed in Hansen (1995) and provides an interface in the R language (R Core Team, 2016) for computing the p-values of the CADF test. Díaz-Emparanza (2014) (DE14) generalizes the calculation of p-values of the HEGY test statistics for series of any periodicity by means of the response surface regression approach. A user interface in the `Gretl` (Cottrell and Lucchetti, 2016) software package for econometric analysis is also provided by DE14.

The bootstrap method has also been investigated as an alternative approach for the computation of p-values of unit root tests. For example, Rayner (1990), Nankervis and Savin (1996) and Park (2003) study the bootstrapped Dickey and Fuller test statistics. Psaradakis (2000) shows the asymptotic validity of the bootstrap in a seasonal autoregressive model with uncorrelated and homoscedastic disturbances and studies the performance in small samples. Burridge and Taylor (2004) (BT04) introduce the bootstrapped HEGY test statistic as a method to estimate the sampling distribution of these statistics. They study the performance in quarterly data with different structures of the innovations.

The purpose of this paper and the code described herein is double. First, a parallel implementation that replicates some of the simulation exercises carried out in DE14 is designed and optimized for a general-purpose Graphics Processing Unit (GPU). Secondly, the code for these simulations is reused and adapted for computing bootstrapped test statistics in the vein of BT04. Both applications of the code provide a different method for the computation of p-values of seasonal unit root tests, The computations to be executed on the GPU are implemented in the CUDA programming model (NVIDIA Corporation, 2015). An interface in the R language is provided for the computation of p-values by means of both approaches.

The remaining of the paper is organized as follows. Section 2 introduces the regression model and the HEGY test statistics. Section 3 describes the optimizations of the code to be run on the GPU and introduces the sweep operator. Section 4 outlines the simulation exercises that were run to obtain the response surfaces of the statistics. Section 5 introduces another application of the code for bootstrapping the HEGY statistics. The usage of the `uroot` R package for the application of the methods and code discussed in the paper is illustrated in Section 7. Section 8 concludes.

2 The HEGY test statistics

The HEGY regression for data of any seasonal periodicity S can be defined as follows:

$$\begin{aligned} \Delta^S y_t &= \alpha + \pi_0 y_{0,t-1} + \textstyle\sum_{j=1}^{S^*} \left(\pi_{j,a} y^a_{j,t-1} + \pi_{j,b} y^b_{j,t-1} \right) + \pi_{S/2} y_{S/2,t-1} + \\ &+ \textstyle\sum_{i=1}^{p} \phi_i \Delta^S y_{t-i} + \varepsilon_t \,, \quad t = 1, \ldots, n \,, \quad S^* = \begin{cases} S/2 - 1 & \text{if } S \text{ is even} \\ \lfloor S/2 \rfloor & \text{if } S \text{ is odd} \end{cases} \,, \end{aligned} \tag{1}$$

where Δ^S is the seasonal differencing operator, n is the sample size, p are the number of lags of the dependent variable (if any are chosen) and ε_t is a Gaussian white noise process, NID$(0, \sigma^2)$. The term related to the coefficient $\pi_{S/2}$ is not included if S is odd. In addition to the constant α, a linear trend and seasonal dummies are often included in applications.

Following DE14, the formulation of the HEGY regressors proposed in Smith *et al.* (2009) is used in the code:

$$y_{0,t} = \sum_{i=0}^{S-1} y_{t-i} \,, \qquad y_{S/2,t} = \sum_{i=0}^{S-1} \cos\left[(i+1)\pi\right] y_{t-i} \,,$$
$$y^a_{j,t} = \sum_{i=0}^{S-1} \cos\left[(i+1)\omega_j\right] y_{t-i} \,, \qquad y^b_{j,t} = -\sum_{i=0}^{S-1} \sin\left[(i+1)\omega_j\right] y_{t-i} \,.$$

For low periodicities, in particular quarterly series, this formulation may be slightly less efficient than the definition based on lags of the dependent variable as defined in the original paper. However, these expressions are convenient because they can be used for any periodicity and therefore the code is simplified.

We are interested in the following test statistics:

- t-test statistics related to the null hypotheses of a unit root at the zero and π frequencies, $\pi_0 = 0$ and $\pi_{S/2} = 0$, denoted t_0 and t_π, respectively.
- F-test statistics for the null hypothesis of a unit root at each pair of complex-conjugate seasonal roots, $\pi_{j,a} = \pi_{j,b} = 0$. Beaulieu and Miron (1993) explain that the asymptotic distribution of these joint test statistics are the same for all $j = 1, \ldots, S^*$.
- F-test statistics for the null hypothesis of a unit root at all seasonal frequencies, $\pi_{S/2} = \pi_{j,a} = \pi_{j,b} = 0$ for all j, denoted $F_{2:S}$.
- F-test statistic for the null hypothesis that all the coefficients related to the HEGY regressors are zero, i.e., unit roots are present at the zero and all the seasonal frequencies as implied by the seasonal differencing operator. This statistic is denoted F_{Δ^S}.

Burridge and Taylor (2001) show by means of Monte Carlo experiments that, under serial correlation in the disturbance term ε_t, the asymptotic distributions of the t-statistics related to each harmonic seasonal frequency are not corrected by augmenting the regression with p lags of the dependent variable, while the joint F-tests at the harmonic frequencies remain pivotal. The latter test statistics are therefore recommended to be used in practice. Following those results, the t-statistics for the null hypotheses of the type $\pi_{j,a} = 0$ or $\pi_{j,b} = 0$ will be omitted in this work, since in practice they can be replaced by the corresponding joint F-test, $F^{a,b}_j$.

3 GPU program optimization

In the last years, the parallel structure of the many-core GPUs has been applied to applications that naturally involve parallel operations but are not necessarily related to graphics operations. That's why sometimes these devices are referred to as general-purpose GPUs. Unlike parallel programming on the CPU, the benefits of parallelization are not ensured by just reusing the original code run on the CPU and compiling it for the GPU. Despite most of the calculations that can be executed on a CPU can also be executed on the GPU, differences in the architecture of both hardware have and impact on the performance of a given program or algorithm. In particular, there are several levels of memory on a GPU that differ in size and behaviour from those on a CPU. For details about the architecture of modern GPUs see, for example, Wilt (2013). Here, it suffices to think of the resources available on the GPU in the following simplified terms: the lower memory space is required by each thread, the more room will be available for more threads to work concurrently and exploit the advantages of parallel processing on these devices.

Having the above in mind, the strategy of the code to be executed on the GPU is to do *on-the-fly* as many computations as possible, i.e., to avoid allocating memory space for data that can take part in the computations without being explicitly stored on memory. To this end, the following major adjustments are carried out on the process of fitting a linear regression model compared to the common algorithms employed by econometric software:

- Stick to compute those elements that necessary to obtain the test statistics (residual sums of squares in the unrestricted and the restricted models).
- A limited memory algorithm is employed: the sweep operator.
- Symmetric matrices are packed into a vector. The indexing of the sweep operator is adjusted in terms of this vector.

The code must stick to do only those computations that are necessary to get the test statistics that were introduced in Section 2. In particular, the target of the computations are the residual sums of squares of the unrestricted regression model and of the restricted versions related to each one of the null hypotheses:

$$\begin{aligned} F_{\Delta^S} &= \left[(RSS - URSS)/S)\right]/\hat{\sigma}^2\,, & t_\pi &= \sqrt{(RSS - URSS)/\hat{\sigma}^2}\,, \\ F_{2:S} &= \left[(RSS - URSS)/(S-1)\right]/\hat{\sigma}^2\,, & t_0 &= \sqrt{(RSS - URSS)/\hat{\sigma}^2}\,, \\ F_{pair} &= \left[(RSS - URSS)/2\right]/\hat{\sigma}^2\,, \end{aligned} \tag{2}$$

where RSS is the residual sum of squares of the corresponding restricted model, URSS is the RSS in the unrestricted model and $\hat{\sigma}^2$ is the variance of the residuals from the unrestricted model, $\hat{\sigma}^2 = URSS/(n - S - \#\text{others})$ where #others stands for the number of additional regressors, e.g., intercept, lags.

A common procedure to fit a linear regression model is to obtain the QR decomposition of the matrix of regressors by means of Householder transformations. See, for example, Pollock (1999, Chapter 8) and Monahan (2001, § 5.6). Although the Householder transformations are in principle an appealing procedure to obtain the RSS of the restricted models, memory requirements

were an important constraint for the GPU, especially for large sample sizes and periodicities. In addition, not all the required RSS are obtained and a matrix inversion is needed in order to obtain the t-test statistics. The process of orthogonalization requires storing the vectors that have been orthogonalized in previous steps and, hence, the orthogonalization of the whole set of vectors cannot be computed independently. This approach was therefore not satisfactory enough for the purpose of reducing memory requirements. Same drawbacks applied to other orthogonalization algorithms: Gram-Schmidt procedure or Givens rotations.

The Frisch-Waugh-Lovell Theorem (FWL Theorem), introduced for example in Davidson and MacKinnon (2004, § 2.4-2.5), was considered as a mean to annihilate some of the regressors from the model (the deterministic components and the lags of the dependent variable). In essence, one of the results of the FWL Theorem is that the Ordinary Least Squares residuals from the following regression models are identical:

$$\begin{aligned} y &= X_1\beta_1 + X_2\beta_2 + u_t\,, \\ M_1 y &= M_1 X_2\beta_2 + \text{residuals}\,, \quad \text{with } M_1 = I - X_1(X_1^\top X_1)^{-1}X_1^\top\,, \end{aligned}$$

where X_1 is a $n \times k_1$ matrix and X_2 is $n \times k_2$ matrix. This result would allow us to obtain the residual sum of squares from the restricted models by means of regressions involving fewer regressors. However, a way to arrive to the transformed variables without incurring in strong memory requirements is still required. At this point, we get stuck in the same difficulties as before. In fact, the task is to orthogonalize the HEGY regressors with respect to the remaining regressor variables. Even for the deterministic regressors –which have a predetermined structure– annihilating these variables did not compensate for the additional memory required for the projection matrix.

The solution that was found to suit the needs for the present task is the sweep operator. The sweep operator performs a series of elementary operations on the rows of a matrix. When applied on a given matrix that we shall define, it provides several information that is helpful for summarizing the results of a fitted linear regression model by ordinary least squares. Although it is not common in modern software, we will see that it is ideally suited for our purposes.

As the matrix on which the sweep operator is applied is symmetric, storing one of the sides of the matrix can be avoided. To do so, the standard algorithm was adapted in terms of a vector that contains one side and the diagonal of the target matrix. Further simplifications of the sweep operator that are possible in this context are discussed in the subsection below and in Appendix A.

Another optimization of the code that turned out to be relevant compared to a standard implementation is that the seasonal random walk that is generated at each iteration is not stored in an array of size n. Instead, just the last S values of the process are stored; the remaining values are generated as needed when the HEGY regressors are obtained. These regressors are in turn accumulated on-the-fly as crossproducts in the target matrix A that will be introduced below. In this way, memory requirements no longer depend on the sample size.

As shown in Appendix B, following this approach memory requirements were considerably reduced.

3.1 The sweep operator

The sweep operator is introduced in Beaton (1964) as a procedure to solve linear systems of equations of the form $A_{11}X = A_{12}$, where A_{11} and A_{22} are respectively a $m_1 \times m_1$ and a $m_1 \times m_2$ matrix and X is a $m_1 \times m_2$ matrix of unknowns. The operator is also described as a tool to find the determinant and rank of a matrix or to obtain the inverse of positive-definite matrices.

Beaton (1964) defines the sweep operator as follows. Let A be an $m \times m$ square positive definite of semi-definite matrix, then sweeping the k-th column of this matrix yields a matrix B with elements:

$$\begin{aligned}
b_{jj'} &= a_{jj} - a_{jk}a_{kj'}/a_{kk} && j, j' = 1, \ldots, k-1, k+1 \ldots, m\,, \\
b_{kj'} &= a_{kj'}/a_{kk} && j' \neq k\,, \\
b_{jk} &= -a_{jk}/a_{kk} && j \neq k\,, \\
b_{kk} &= 1.0/a_{kk}\,.
\end{aligned}$$

If a matrix A is positive definite, then sweeping all the columns results in the inverse of the matrix, A^{-1}. The following result mentioned also in Beaton (1964) is the basis that will make the operator useful in our context. If the matrix A is partitioned as shown below, the matrix C is obtained after sweeping the first m_1 columns:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \xRightarrow{m_1 \text{ sweeps}} \quad C = \begin{bmatrix} A_{11}^{-1} & A_{11}^{-1}A_{12} \\ -A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}.$$

By defining the matrix A conveniently, the above result can be used to solve the system of equations of the ordinary least squares estimator. This approach, described for example in Pollock (1999, Chapter 8) and Monahan (2001, § 5.12), can be summarized as follows. Let

$$y = X\beta + u$$

be a regression model in matrix form where y is of dimension $n \times 1$ and X consists of m regressors, $n \times m$. Then, the matrix A is defined below as a matrix of crossproducts augmented by a diagonal matrix of size m. Upon the above result, it can be checked that sweeping the first m columns leads to a matrix containing relevant information:

$$A = \begin{bmatrix} X^\top X & X^\top y & I_m \\ y^\top X & y^\top y & 0 \end{bmatrix} \quad \xRightarrow{m \text{ sweeps}} \quad \begin{bmatrix} I_m & \hat{\beta} & (X^\top X)^{-1} \\ 0 & y^\top M_x y & -\hat{\beta} \end{bmatrix},$$

with $\hat{\beta} = (X^\top X)^{-1}X^\top y$, i.e., the ordinary least squares estimator and $M_X = I_n - X(X^\top X)^{-1}X^\top$ is the projection matrix that when it multiplies the dependent variable y gives the vector of residuals. The element $y^\top M_x y$ is the sum of squared residuals and will be the target of our computations. If $X^\top X$ is not needed for further computations, the inverse matrix can be overwritten on top of it. Thus, the sweep operator can be summarized in this context as follows:

$$A = \begin{bmatrix} X^\top X & X^\top y \\ y^\top X & y^\top y \end{bmatrix} \quad \xRightarrow{m \text{ sweeps}} \quad \begin{bmatrix} (X^\top X)^{-1} & \hat{\beta} \\ 0 & y^\top M_x y \end{bmatrix}.$$

The sweep operator can therefore be interpreted as a process of column elimination. In fact, if it is applied on the columns of $X^\top X$, then the inverse $(X^\top X)^{-1}$ is obtained. The sweep operator

amounts to carrying out m steps of the Gaussian inversion procedure to the augmented matrix A defined above. As a result, relevant information about the fitted model is generated, in particular the sum of squared residuals will be a valuable output for our purposes.

In our context, the sweep operator has some advantages over other alternatives that make it especially appealing for our application:

- It allows us to minimize memory requirements, which is of paramount importance in our application. The crossproducts can be accumulated in place in the storage matrix A; therefore, there is no need to explicitly allocate memory for the dependent and the regressor variables.
- The indexing of the target symmetric matrix A can be mapped in terms of the indexing in a vector containing only the upper triangular side and the diagonal of the matrix. In our implementation, instead of storing the $m \times m$ matrix A, a vector of length $m \times (m-1)/2+m$ is employed. The reduction in memory requirements was prioritized over the cost of doing some additional operations in order to get the indexing in terms of the vector. Unlike other matrix operations, e.g. the inversion of symmetric packed matrices, the mapping of the indexing from a matrix to a vector is relatively simple in the case of the sweep operator and do not involve a significant burden.
- Sweeping the i-th column is equivalent to removing that regressor from the model. This allows us to design a procedure for computing the residual sum of squares in the restricted models. The procedure is illustrated in Appendix A.
- The whole sweep operator is not needed to get the last element of the output matrix, i.e., the RSS. At some points of the process, this allows us to save some operations.
- The algorithm can be stopped and restarted when required without the need to recompute or reset any of the values in the input matrix A. This is not the case, for example, with the Householder regression, for which we would need to reset the input matrix or allocate temporary arrays in order to obtain the t-statistics after the transformations are applied in order to get the F-statistics.

As shown in Monahan (2001, § 5.12), sweeping the k-th column of the matrix `A` can be implemented as follows (k is the column to be swept and m is the number of columns in matrix A):

```
pivot = A[k,k]
for j = 1,...,m {
  A[k,j] = A[k,j] / pivot
}
for i = 1,...,m
{
  if i != k
  {
    b = A[i,k]
```

```
    for j = 1,...,m {
      A[i,j] = A[i,j] - b * A[k,j]
    }
    A[i,k] = -b / pivot
  }
}
A[k,k] = 1.0 / pivot
```

Marchetti *et al.* (2015) implements a version of the algorithm that sweeps more than one column at a time. Although this would be helpful to eliminate for example the columns related to deterministic terms (e.g., constant) or lags of the dependent variable, we don't use this approach in our implementation since it requires inverting a submatrix. Here, I found it more convenient to sweep one column at a time.

An illustration of the whole procedure for a particular case is given in Appendix A.

4 Response surface regressions

The first application of the code is a simulation exercise that replicates some of the response surfaces obtained in Díaz-Emparanza (2014). The simulations described below were run on the Arina cluster of the UPV/EHU; in particular, the program was run in a node with an Intel® Xeon® processor E5-2680 v2 @2.80GHz and a Tesla® K20 GPU (NVIDIA Corporation, 2014).

4.1 Outline of the simulation exercises

In the simulation exercises, the HEGY test statistics are computed for different combinations of the following elements that characterize the regression model: periodicity, lag orders and sample size. In this replication exercises only the case of an intercept is considered, i.e., linear trend or seasonal dummies are not considered in the auxiliary regression. The simulations were run for the following combinations of parameters:

- Periodicity (S): 4, 5, 6, 7, 12, 24.
- Sample size (n): 48, 52, 64, 76, 100, 124, 152, 200, 300, 400, 500, 780.
- Lags of the dependent variable (p): $0, 1, \ldots, \min(S, 12)$.

For each combination of parameters, 221 quantiles that will represent the response surface are obtained based on 100,000 iterations.

As mentioned in Section 2, the asymptotic distribution of the $F_j^{a,b}$ test statistics for the null $\pi_{1,a} = \pi_{1,b} = 0$ is the same for all the pairs of complex-conjugate seasonal roots. To save computations and storage, only the first one of these F-test statistics will be computed. It will be denoted F_{pair}. As discussed in the same Section above, the t-statistics related to each harmonic seasonal frequency will not be considered. Thus, five test statistics will be computed when the periodicity is even and four statistics when the periodicity is odd: t_0, t_π (if S is even), F_{pair}, $F_{2:S}$ and F_{Δ^S}.

The following setup was used in order to distribute the 100,000 iterations across the CUDA threads:

- If $S < 7$ and $n < 780$, 10 blocks 500 threads per block and 20 iterations per thread.
- If $S < 7$ and $n = 780$, 20 blocks, 200 threads per block and 25 iterations per thread.
- If $S >= 7$, 5 blocks, 200 threads per block and 100 iterations per thread.

4.2 Random number generator

The XORWOW random number generator (RNG) is used to produce a sequence of pseudorandom integers. This generator is a member of the class of Xorshift family of random number generators (Marsaglia, 2003). The implementation of this RNG in the `cuRAND` library returns a sequence with a period greater than 2^{190}. The Box-Muller transform (Box and Muller, 1958) is employed to get draws from the Gaussian distribution, which are required to simulate seasonal random walks upon which the test statistics are obtained. For each draw, this method consumes four elements from the sequence of integers generated by the RNG. Actually, this method returns two draws from the Gaussian distribution, however, in the current version of our simulations the second draw is discarded. Although using both draws would be more efficient, discarding the second draw was more convenient for testing and simplification of the code. As we shall see, this does not pose any problems in terms of exhausting the sequence generated by the RNG.

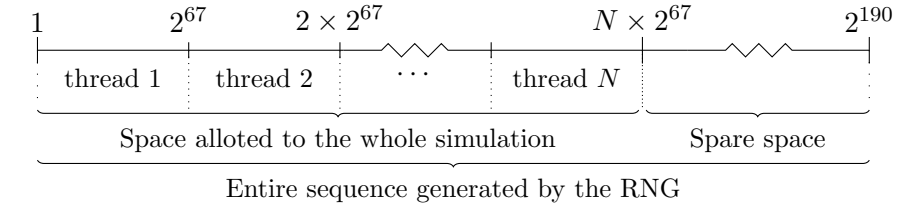
The sequence of integers provided by the XORWOW generator is long enough to cover the needs of the entire simulation that includes all the sample sizes, periodicities and lag orders (in this replication exercise we consider the HEGY regression with a constant term and without any other deterministic components).

$$\text{Required uniform draws} = \left(\sum_s \sum_n \sum_p n_i + 3 \times s \right) \times 100{,}000 \times 4 = 8.80144 \times 10^{10} \,,$$

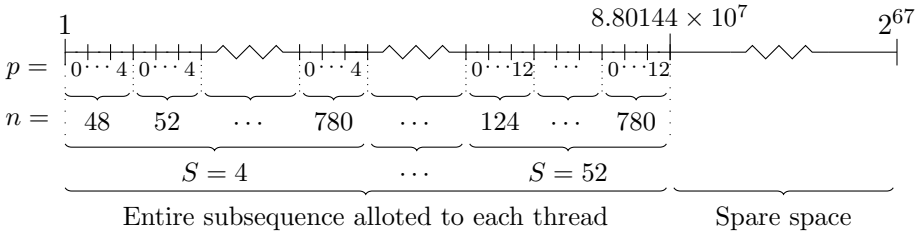
where $\sum_s$ is the sum over the periodicities considered in the exercise, $\{4, 5, 6, 7, 12, 24, 52\}$; $\sum_n$ is the sum over the sample sizes, $\{48, 52, 64, 76, 100, 124, 152, 200, 300, 400, 500, 780\}$, $3 \times s$ observations are added as warming observations; $\sum_p$ is the sum over each lag order (from 0 to the minimum of s and 12). For each combination of parameters (periodicity, sample size, lag order) 100,000 Gaussian seasonal random walks are simulated. Each draw from the Gaussian distribution employs four draws from the uniform distribution. For the periodicity $s = 52$, sample sizes lower than 76 are discarded because for most of the lags the number of regressors is larger than the sample size (non-positive degrees of freedom). Thus, it can be checked that the total number of required uniform draws is 8.80144×10^{10}, lower than 2^{37} and, therefore, much lower than the period of the generator, 2^{190}.

In the parallel execution, each tread could be assigned a different seed of the RNG. However, it is better to use the same seed for the entire simulation (for all combinations of parameters). As stated in the documentation of the library `cuRAND`, sequences generated with the same seed and different sequence numbers will not have statistically correlated values, while some choices of seeds may give statistically correlated sequences. In order to take advantage of the period of the RNG, a single seed is set and each thread is located at points separated by 2^{67} elements in the sequence. This is a convenient distribution of threads because `curand_init` allows us to initialize the state

of the RNG at positions multiple of 2^{67} without having to run the generator up to that point. Figure 1a shows the location of threads along the entire sequence generated by the XORWOW generator.



(a) Distribution of threads in the sequence generated by the XORWOW generator.



(b) Usage of the subsequence alloted to a thread.

Figure 1: Distribution of threads in the sequence generated by the XORWOW generator and usage of the subsequence alloted to a thread.

Figure 1b shows how each thread uses the subsequence of uniform draws that is alloted to it. The following design of the CUDA threads is considered: 1,000 threads distributed across 5 blocks with 200 threads per block; 100 iterations (out of the 100,000) are run by each thread. In that case, the total number of uniform draws consumed by a thread is 8.80144×10^7, lower than 2^{27} and, therefore, lower than the length of the subsequence alloted to each thread, 2^{67}. Other configurations of threads were also used, here we reported the one that employs the largest amount of draws by each thread.

The whole simulation is managed by a bash shell script. Instead of running the simulations in one call to the CUDA program, a call to the program is done for each combination of parameters. Splitting the simulations in this way was helpful for simplifying some parts of the code and assessing the performance of the program in the development version. For each combination of parameters (e.g., $S = 4$, $n = 48$, $p = 0$), 100,000 iterations are run. After that, the state of the RNG (summarized by six integers) is saved to a file. In this way, the state of the RNG for the next call to the program can be initialized at the right location in each subsequence without having to run the generator step by step up to that point in the sequence.

4.3 Results

Figure 2 and Figure 3 show some of the response surfaces obtained in the simulations. The former shows response surfaces for a given periodicity (quarterly data), while in the latter plot, the number of lags is fixed in order to display the response surface with respect to the periodicity and the sample

size.

The results are similar to the critical values tabulated in the reference papers cited in the introduction section above. The Figures reveal that the quantiles of a test statistic vary for different periodicities, lag order and sample sizes. For moderate sample sizes these differences may have a relevant effect on the results of the test for a given significance level. Computing a p-value based on response surfaces seems, therefore, a better option than sticking to the limited critical values reported in the original papers.

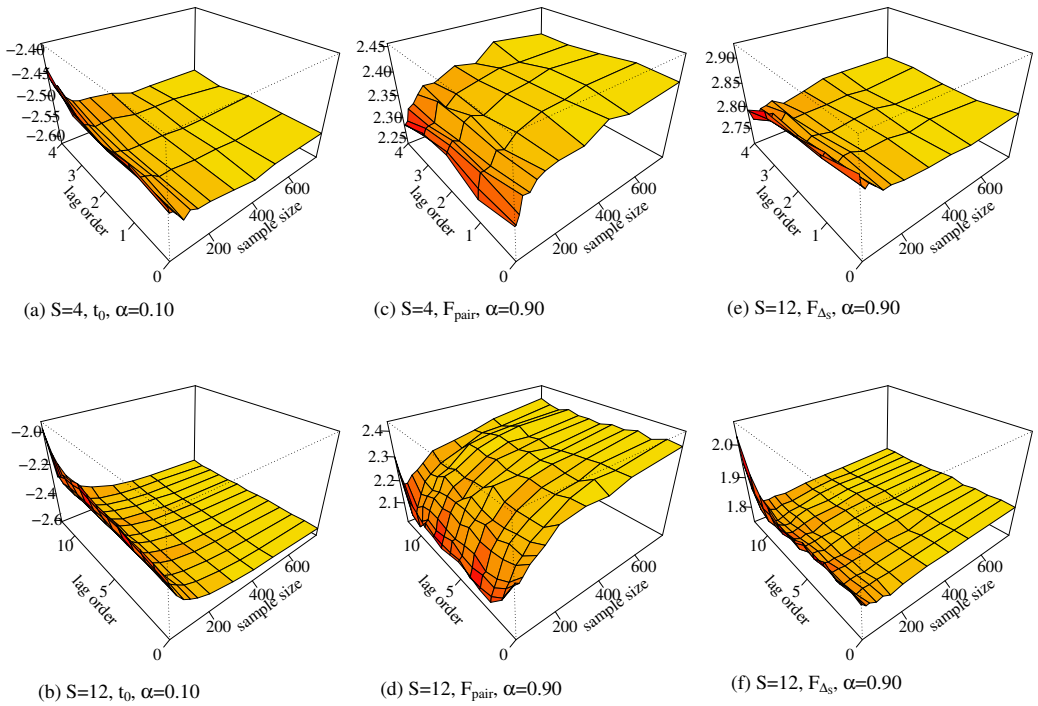


Figure 2: Response surfaces for a given periodicity.

5 Bootstrap

The second application of the code allows us to approximate the sampling distribution of the statistics –and therefore calculate the p-values– from a different approach: bootstrapping. Rather than using precompiled response surfaces, p-values can be computed at run time by bootstrapping the residuals of the auxiliary regression given in Equation 1. The implementation shown here follows Burridge and Taylor (2004). This can be done with minimal changes in the code employed for the simulation of response surfaces. In fact, the same strategy for computing the statistics can be followed. The following options that are common in applications were added to the code:

- Deterministic components: in addition to a constant, a linear trend and/or seasonal dummies can be selected.
- Lag order selection: the lag order, p, in the auxiliary regression for the original and the

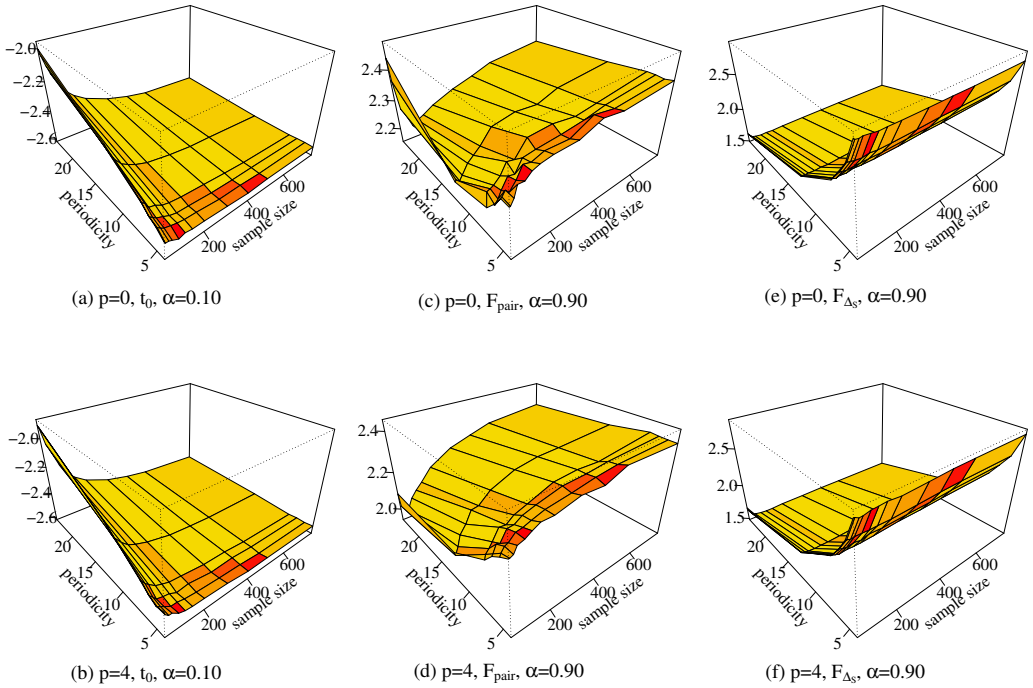


Figure 3: Response surfaces for a given lag order.

bootstrapped data can be chosen based on the AIC, BIC or AICc criteria.

- All the F_j^{ab} test statistics for the null hypothesis of a unit root at each pair of complex-conjugate seasonal roots are computed.
- Instead of generating disturbances from the Gaussian distribution (as it was done in the simulation exercises), the residuals from the original regression are resampled in order to generate the ensemble of bootstrap replicates of the data.

The ensemble of replicates is generated upon the following model:

$$\Delta^S y_t^* = \sum_{i=1}^{p} \hat{\phi}_i \Delta^S y_{t-i}^* + e_t^* \,,$$

where e are the residuals from the regression Equation 1 fitted to the original data. The residuals can be resampled taking into account the season they belong to or regardless of the season. If a lag order greater than zero is chosen for the original data, the estimated autoregressive coefficients, $\hat{\phi}_i$, are included in the model that generates the resampled series. As noted in BT04, the test statistics are not affected by the deterministic terms included in the model for the original data (constant, trend, seasonal dummies) and, hence, they don't need to be included in the model that generates the replicates.

Although the extensions cited above do not involve major changes in the code, new arrays are defined and further memory allocation is required by each thread. As indicated in Table 4, some

of these elements are passed once to the GPU, while others need to be dynamically allocated by each thread on the GPU. Next, the size and purpose of these additional arrays are introduced.

Each bootstrapped statistic is compared with the statistic obtained for the original data, which are stored and passed to the GPU in the array `d_stats0`. If the bootstrapped statistic is beyond the original one in the corresponding tail, then the event is recorded as a 1, and otherwise as a 0. The sum of records over all iterations divided by the number of bootstrap iterations is the estimated p-value. Unlike in the simulation exercises, in a real application we are interested in all the F_j^{ab} statistics. The event of F_j^{ab} statistics larger than the original ones are stored in the array `d_res3`, which requires storage for $N(\lfloor S/2 \rfloor - 1 + S \bmod 2)$ elements, i.e., the number of bootstrap iterations times the number of F_j^{ab} statistics.

The residuals from the auxiliary regression that is fitted for the original data are resampled and employed as innovations in the model under null hypothesis, which is used to generate replicates of the data. The vector of residuals is allocated once and passed to the GPU in the array `d_eps`. The residuals are stacked together by season, which will be helpful if they are resampled by seasons. The length of this vector is the number of observations in differenced series, $\Delta^S y_t$, minus the observations missed due to lags, p. If the regression model for the original data contains lags of the dependent variable, the estimated coefficients are required in order to generate the resampled series. These coefficients are stored in the array `d_arcoefs`. As mentioned before, two resampling schemes are considered: resampling of the whole set of residuals; alternatively, the residuals are split by seasons so that only those residuals belonging to the same season are resampled. The latter is an interesting option in the presence of seasonal heteroscedasticity in the residuals. To do so, the vector `d_csns` contains the cumulative sum of the number of residuals per season. This is used to generate a proper index of the residuals and to identify the location of the residuals belonging to a season in the array `d_eps`.

As in the simulations, it is not needed to store the whole series (resampled series in this case). It is enough to store the last observations in the array `y0` along with the last p disturbances (resampled residuals) that are tracked in the array `dylags`. The length of the array `y0` is $S + p$. Finally, when $S > 4$, the computation of the F_j^{ab} statistics requires a backup copy of the matrix of crossproducts, this is the array `v_backup` of the same length as `vA` in Table 3.

One of the peculiarities of the design proposed in BT04 and the implementation introduced in this paper is the possibility to use a lag selection procedure for each bootstrap replicate. For the original series, the standard definition of the AIC, BIC and AICc criteria is used (i.e., on the value of the log-likelihood function of the fitted model). For the bootstrap replicates, these criteria are defined upon the residual sum squares of the fitted model (RSS):

$$\begin{array}{rcl} \text{AIC} & = & n \log\left(\text{RSS}/n\right) + 2k \\ \text{BIC} & = & n \log\left(\text{RSS}/n\right) + k \log(n) \\ \text{AICc} & = & \text{AIC} + \frac{2k(k+1)}{n-k-1} \end{array} ,$$

where k is the number of parameters in the model and n is the number of available observations. As the sweep operator is applied to obtain the required RSS, these definitions allow us to easily include the lag selection procedure for the bootstrap samples.

As regards the random number generator, similarly to the simulations exercises, each thread is

initialized at a point 2^{67} elements apart in the sequence generated by the XORWOW for a given seed. For sample sizes and iterations common in applications, this gives enough room for threads not to overlap in the sequence of random draws.

6 Simulation exercises

The performance of response surface regressions and the bootstrap for the calculation of p-values of seasonal unit roots tests is assessed by means of simulation exercises.

			t_0			t_π			$F_{3:4}$			$F_{2:4}$			$F_{1:4}$		
T	ϕ	α	I	R	B	I	R	B	I	R	B	I	R	B	I	R	B
6	none	0.05	0.072	0.435	0.075	0.053	0.002	0.048	0.093	0.000	0.036	-	0.000	0.028	-	0.000	0.021
		0.10	0.120	0.559	0.114	0.098	0.034	0.102	0.147	0.000	0.081	-	0.000	0.070	-	0.000	0.051
	−0.50	0.05	0.148	0.550	0.145	0.064	0.004	0.058	0.093	0.000	0.034	-	0.000	0.039	-	0.000	0.089
		0.10	0.223	0.624	0.212	0.108	0.029	0.113	0.150	0.001	0.079	-	0.000	0.086	-	0.001	0.160
	0.50	0.05	0.086	0.448	0.080	0.072	0.006	0.065	0.103	0.000	0.040	-	0.000	0.035	-	0.000	0.027
		0.10	0.137	0.514	0.121	0.124	0.051	0.123	0.163	0.000	0.090	-	0.000	0.083	-	0.000	0.058
	0.85	0.05	0.121	0.451	0.105	0.079	0.013	0.070	0.110	0.001	0.045	-	0.001	0.051	-	0.001	0.071
		0.10	0.176	0.490	0.158	0.126	0.053	0.121	0.166	0.001	0.090	-	0.002	0.092	-	0.003	0.109
15	none	0.05	0.040	0.059	0.046	0.048	0.064	0.055	0.050	0.051	0.045	-	0.074	0.046	-	0.091	0.045
		0.10	0.084	0.115	0.094	0.098	0.125	0.105	0.104	0.106	0.095	-	0.131	0.092	-	0.160	0.093
	−0.50	0.05	0.072	0.096	0.067	0.048	0.064	0.058	0.062	0.063	0.048	-	0.094	0.052	-	0.144	0.078
		0.10	0.121	0.152	0.105	0.097	0.124	0.111	0.114	0.117	0.094	-	0.152	0.010	-	0.213	0.113
	0.50	0.05	0.042	0.067	0.047	0.056	0.072	0.052	0.059	0.061	0.047	-	0.089	0.049	-	0.106	0.048
		0.10	0.092	0.122	0.100	0.105	0.132	0.097	0.114	0.116	0.097	-	0.142	0.093	-	0.177	0.094
	0.85	0.05	0.051	0.082	0.051	0.045	0.061	0.050	0.053	0.054	0.045	-	0.072	0.040	-	0.103	0.044
		0.10	0.108	0.142	0.103	0.093	0.119	0.099	0.104	0.107	0.095	-	0.125	0.091	-	0.178	0.092
30	none	0.05	0.044	0.053	0.048	0.051	0.059	0.055	0.054	0.052	0.051	-	0.073	0.051	-	0.090	0.053
		0.10	0.095	0.110	0.100	0.103	0.117	0.103	0.105	0.103	0.102	-	0.130	0.105	-	0.157	0.097
	−0.50	0.05	0.047	0.055	0.043	0.048	0.056	0.054	0.053	0.051	0.051	-	0.074	0.051	-	0.097	0.043
		0.10	0.095	0.110	0.094	0.101	0.117	0.104	0.105	0.103	0.010	-	0.129	0.101	-	0.166	0.090
	0.50	0.05	0.049	0.059	0.053	0.049	0.056	0.051	0.055	0.053	0.051	-	0.074	0.049	-	0.099	0.049
		0.10	0.099	0.115	0.103	0.099	0.114	0.101	0.107	0.105	0.101	-	0.131	0.101	-	0.170	0.099
	0.85	0.05	0.057	0.065	0.055	0.049	0.057	0.054	0.053	0.052	0.053	-	0.074	0.051	-	0.105	0.050
		0.10	0.104	0.119	0.102	0.099	0.114	0.104	0.104	0.103	0.103	-	0.129	0.106	-	0.179	0.101

Table 1: Empirical levels of the HEGY test in quarterly data. Based on 10,000 series simulated according to the following data generating process: $(1-\phi L)(1-L^4)y_t = \epsilon_t \sim \text{NID}(0,1)$, $t = 1, \ldots, T \times 4$. The lag order in the auxiliary regression is chosen based on the BIC. p-values are obtained by means of the following methods: I, interpolation in the original tables of critical values; R, response surface; B, bootstrap. Cells report the empirical levels to be compared with the nominal levels $\alpha = 0.05$ and 0.10.

		t_0			t_π			$F_{3:4}$			$F_{2:4}$			$F_{1:4}$		
T	ϕ	I	R	B	I	R	B	I	R	B	I	R	B	I	R	B
6	none	0	0	0	0	0	3	0	0	1	-	0	0	-	0	4
	−0.50	0	0	0	0	0	2	0	0	4	-	0	4	-	0	0
	0.50	0	0	0	0	0	3	0	0	2	-	0	1	-	0	4
	0.85	0	0	0	0	0	0	0	0	1	-	0	0	-	0	0
15	none	0	4	4	0	4	4	0	4	4	-	0	4	-	0	4
	−0.50	0	0	0	0	4	4	0	0	4	-	0	0	-	0	0
	0.50	0	3	4	0	0	0	0	1	2	-	0	0	-	0	4
	0.85	0	0	4	0	2	4	0	4	4	-	0	3	-	0	1
30	none	0	4	4	0	4	3	0	4	4	-	0	4	-	0	4
	−0.50	0	4	4	0	3	4	0	3	4	-	0	4	-	0	4
	0.50	0	4	4	0	4	3	0	4	4	-	0	4	-	0	4
	0.85	0	4	1	0	4	4	0	4	4	-	0	4	-	0	4

Table 2: Chi-squared test for uniformity of p-values. Cells report the significance of the Chi-squared test statistic for the null that the p-values lower or equal than 0.10 are uniformly distributed. Significance codes: 0 '4' 0.001 '3' 0.01 '2' 0.05 '1' 0.1 '0' 1 (the higher the code, the better the performance of the corresponding method). Based on simulation exercises described in Table 1.

7 Examples

The `uroot` R package provides an interface to the code and methods introduced above. On the one hand, p-values based on response surfaces can be obtained using `hegy.rs.pvalue`. This function is a port from Gretl code provided by the author of the original paper (Díaz-Emparanza, 2014) and employs the matrices of quantiles obtained on it. Once these matrices have been obtained by simulations, the calculation of p-values for a given time series is not a computing intensive operation and, hence, this can be conveniently implemented in R. On the other hand, bootstrapped p-values can be computed by means of the code described in this paper. This utility requires a CUDA enabled GPU. Since the bootstrap will not in general be as demanding as the simulations described in Section 4, a high performance GPU is not needed. A GPU with compute capability >= 3.0 is nonetheless required. If such a GPU is not available, the package `uroot` can still be installed. A version of the bootstrap method fully implemented in R is provided in the function `hegy.boot.pval`. This latter version does not enjoy the benefits of parallelization; it is intended for debugging and as a workaround for systems that do not have the required hardware.

The HEGY statistics can be obtained through the function `hegy.test`. The example below shows the usage of this function for the logarithms of the quarterly time series of household fuel and power total consumption in the UK. Setting to ones the argument `deterministic`, a constant, a linear trend and seasonal dummies are included in the model. Setting to zero any of the elements in this argument, the corresponding component is omitted in the regression model; e.g., `deterministic = c(1,1,0)` would include a constant and trend but no seasonal dummies. It is often required to include some lags of the dependent variable in order to account for the presence of serial correlation in the disturbances. Below, `lag.method = "BIC"` is set in order to choose a lag order based on the Bayesian information criterion, considering lags up to order `maxlag = 4`.

```
> library(uroot)
> x <- bgt.data[["LHHCONSFUELPOWCU"]]
> (res1 <- hegy.test(x, deterministic = c(1,1,1),
+   lag.method = "BIC", maxlag = 4))

        HEGY test for unit roots

data:  x

      statistic p-value
t_1     -1.3376       1
t_2     -2.1261       1
F_3:4   14.0326  0.9092
F_2:4   11.3717  0.3835
F_1:4    8.8258  0.6104
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Deterministic terms: constant + trend + seasonal dummies
Lag selection criterion and order: BIC, 1
P-values: based on response surface regressions
```

t_1 is the test statistic for the null of a unit root at the frequency zero (long run); t_2 is the test statistic for the null of a unit root at the semiannual frequency, π. F_3:4 is the F-test statistic for the null of complex conjugate unit roots $\pm i$ at the annual frequency $\frac{\pi}{2}$ and at the frequency $\frac{3\pi}{2}$; F_2:4 is a joint test statistic for the null of unit roots at all seasonal frequencies; F_1:4 tests the null of a unit root at the long run and seasonal frequencies. None of the null hypotheses for the presence of unit roots are rejected at the 5% significance level.

By default, p-values are based on the response surface approach. Setting the option pvalue = "bootstrap", the method introduced in Section 5 is run for 1,000 bootstrap replicates.

```
> hegy.test(x, deterministic = c(1,1,1), lag.method = "BIC", maxlag = 4,
+   pvalue = "bootstrap")

        HEGY test for unit roots

data:  x

      statistic p-value
t_1     -1.3376   0.805
t_2     -2.1261   0.254
F_3:4   14.0326   0.002 **
F_2:4   11.3717   0.009 **
F_1:4    8.8258   0.017 *
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Deterministic terms: constant + trend + seasonal dummies
Lag selection criterion and order: BIC, 1
P-values: based on bootstrap (1000 replicates)
```

The values reported under the column named statistics are the test statistics for the original data; therefore, these values are the same as those obtained in the previous example. The p-values based on the bootstrap method differ from those obtained before by means of the response surface approach. The difference is especially relevant for the F_3:4, F_2:4 and F_1:4 test statistics, for which the p-values lead to different conclusions at the 5% significance level; in particular, the null of a unit root at the frequencies tested by these statistics are now rejected.

In order to inspect the possible source of the disagreement in these results, the p-values are obtained below assuming a fixed lag order; in particular, the lag order chosen by the BIC. For the response surface approach, this can be done by setting lag.method = "fixed" and maxlag = 1 (to avoid redundant output, only p-values are printed below):

```
> res2 <- hegy.test(x, deterministic = c(1,1,1),
+   lag.method = "fixed", maxlag = 1)
> print(rbind("p-value" = round(res2$pvalues, 7), "label" = res2$pvlabels),
+   quote = FALSE)

        t_1       t_2       F_3:4 F_2:4 F_1:4
p-value 0.8503363 0.1889914 0     0     0.0048639
label                       ***   ***   **
```

Alternatively, the results obtained in `res1` can be reused and the p-values updated by means of `hegy.rs.pvalue`. Although the previous option is more straightforward and less error-prone, the usage of this function is shown below for completeness:

```
> S <- frequency(x)
> p <- res1$lag.order
> dfr <- df.residual(res1$fit)
> hegy.rs.pvalue(res1$stat["t_1"], type = "zero", deterministic = c(1,1,1),
+   lag.method = "fixed", lag.order = p, S = S, n = dfr, nobsreg = 15)

[1] 0.8503363

> hegy.rs.pvalue(res1$stat["t_2"], type = "pi", deterministic = c(1,1,1),
+   lag.method = "fixed", lag.order = p, S = S, n = dfr, nobsreg = 15)

[1] 0.1889914

> hegy.rs.pvalue(res1$stat["F_3:4"], type = "pair", deterministic = c(1,1,1),
+   lag.method = "fixed", lag.order = p, S = S, n = dfr, nobsreg = 15)

[1] 0

> hegy.rs.pvalue(res1$stat["F_2:4"], type = "seasall",
+   deterministic = c(1,1,1), lag.method = "fixed", lag.order = p, S = S,
+   n = dfr, nobsreg = 15)

[1] 0

> hegy.rs.pvalue(res1$stat["F_1:4"], type = "all", deterministic = c(1,1,1),
+   lag.method = "fixed", lag.order = p, S = S, n = dfr, nobsreg = 15)

[1] 0.004863868
```

Each test statistic requires a different table of quantiles obtained in DE14, therefore, a label is passed through argument `type`; the periodicity, the lag order (the order chosen in `res1` is used here) and the residual degrees of freedom are also required; `nobsreg` is the number of points employed in the response surface regression, the default value employed in `hegy.test` is 15. The resulting p-values are now closer to those obtained based on the bootstrap. When the lag order is fixed to 1, the bootstrapped p-values remain similar to those obtained when the BIC was used:

```
> res3 <- hegy.test(x, deterministic = c(1,1,1), lag.method = "fixed",
+   maxlag = 1, pvalue = "bootstrap")
> print(rbind("p-value" = res3$pvalues, "label" = res3$pvlabels),
+   quote = FALSE)

        t_1   t_2   F_3:4 F_2:4 F_1:4
p-value 0.842 0.173 0     0     0.004
label               ***   ***   **
```

Therefore, in this example, the bootstrap seems to be more robust to the lag order selection method.

In the presence of seasonal heteroscedasticity, it is advisable to resample the residuals taking into account the season they belong to. Setting the option `byseason = TRUE` through argument `boot.args`, the residuals belonging to the same season are resampled and assigned to the observations of the same season in the bootstrap replicate. In this case, no major differences are observed compared to the results shown above for the bootstrap (it can be checked that the variance and level of the residuals are relatively stable and similar across seasons):

```
> res4 <- hegy.test(x, deterministic = c(1,1,1), lag.method = "BIC",
+   maxlag = 4, pvalue = "bootstrap", boot.args = list(byseason = TRUE))
> print(rbind("p-value" = res4$pvalues, "label" = res4$pvlabels),
+   quote = FALSE)

        t_1   t_2  F_3:4 F_2:4 F_1:4
p-value 0.814 0.26 0.001 0.003 0.011
label              ***   **    *
```

The lag order chosen for the original data, in this case $p = 1$, is included in the model that generates the bootstrap replicates. The lag selection procedure is applied for the replicates as well. A different value of `lag.method` or `maxlag` can be defined in argument `boot.args`. If they are not specified, the same lag selection method and maximum lag order than those chosen for the original data are employed when computing the test statistics for the bootstrap replicates. This can be checked by printing the arguments that were used by the bootstrap method, which are stored in the element `bootstrap` of the output object `res4`. There, it can also be checked the seed value used by the random number generator that is instantiated on the GPU. This value can be chosen through the argument `boot.args`, which by default is `seed = 123`. In addition, the lag order chosen for each replicate is available in the element `boot.chosen.lags`. A summary obtained with `table` shows that a lag order equal to 0 or 1 was chosen for most of the replicates.

```
> c(res4$bootstrap$lag.method, res4$bootstrap$maxlag)

[1] "BIC" "4"

> res4$bootstrap$seed

[1] 123
```

```
> table(res4$boot.chosen.lags)

  0   1   2   3   4
404 524  55   8   9
```

The function `hegy.boot.pval` is provided as a workaround for systems where a CUDA capable GPU is not available. This function requires as input the model fitted to the original data, `model0`, and the test statistics obtained from this model, `stats0`. The number of bootstrap replicates is set in argument `nb` and the argument `byseason` indicates whether the residuals should be resampled by seasons or not. For example, bootstrapped p-values for the model with a constant, trend and seasonal dummies and one lag of the dependent variable, can be obtained as follows:

```
> set.seed(1235)
> hegy.boot.pval(x, model0 = res1$fit, stats0 = res1$statistics,
+   deterministic = c(1,1,1), lag.method = "fixed", maxlag = 1,
+   byseason = FALSE, nb = 500)

  t_1   t_2 F_3:4 F_2:4 F_1:4
0.832 0.198 0.000 0.000 0.008
```

A look at the code of this function is helpful as a complement to the description of the procedure given in Section 5. Below, a simplified version that reproduces the results obtained above by means of `hegy.boot.pval` is described. The number of replicates is set to `nb = 500` and the lag order is fixed to `p = 1`. Required information is retrieved from the procedure applied to the original data: the selected lag order, `p`; the fitted model and its residuals, stored respectively in `model0` and `e`; the test statistics, `stats0` and the value of the estimated autoregressive coefficient, `arcoefs`. The bootstrapped p-values will be stored in `bpvals`, which is initialized to zeros since this vector will be a counter keeping track of the statistics that turn out to be more extreme (on the proper tail) than `stats0`.

```
> nb <- 500
> S <- frequency(x)
> p <- 1
> model0 <- res1$fit
> stats0 <- res1$statistics
> e <- residuals(model0)
> arcoefs <- coef(model0)["xregLag1"]
> ns <- length(x) + S + p
> stats.names <- names(stats0)
> bpvals <- rep(0, length(stats0))
> names(bpvals) <- stats.names
```

Next, the seed is set and the main loop begins. Notice that this seed is set for the RNG run on the CPU, which by default in R is the Mersenne-Twister algorithm.

```
> set.seed(1235)
> for (i in seq_len(nb))
+ {
+   be <- sample(e, size = ns, replace = TRUE)
+   m <- outer(c(1, -arcoefs), c(1, rep(0, S-1), -1))
+   pcoefs <- as.vector(tapply(m, row(m) + col(m), sum))
+   bx <- ts(filter(be[-seq_len(S+1)], filter = -pcoefs[-1],
+     method = "recursive", init = rev(be[seq_len(S+1)])),
+     frequency = S, start = start(x))
+
+   bres <- hegy.test(bx, deterministic = c(1,1,1),
+     lag.method = "fixed", maxlag = p, pvalue = "raw")
+
+   for (j in seq_along(bpvals))
+   {
+     if (grepl("t\\_", stats.names[j]))
+     {
+       if (bres$stat[j] <= stats0[j])
+         bpvals[j] <- bpvals[j] + 1
+     } else
+       if (bres$stat[j] >= stats0[j])
+         bpvals[j] <- bpvals[j] + 1
+   }
+ }
> bpvals / nb

  t_1   t_2 F_3:4 F_2:4 F_1:4
0.832 0.198 0.000 0.000 0.008
```

The residuals are resampled with replacement obtaining a vector of size `ns`, i.e., of length such as the number of available observations are the same as in the original series after lossing `S` observations due to the seasonal differencing filter and `p` observations due to lags. Multiplying the coefficients of the autoregressive polynomial, `c(1, -arcoefs)` by the coefficients of the seasonal differencing filter, `c(1, rep(0, S-1), -1)`, yields the filter to be passed to `filter`. This filter generates the bootstrap replicates, `bx`, based on the model under the null hypothesis. Then, the HEGY statistics are computed for `bx`. As the p-values returned by `hegy.test` are not needed here, the least cumbersome method is chosen; `pvalue = "raw"` performs interpolation on the critical values tabulated in the original paper. An inner loop is run over each test statistic in order to track which of the bootstrapped statistics turned out to be beyond the original statistic in the appropriate tail (left tail for t_0 and t_π and right tail for the F-test statistics). Once the loop has finished, the bootstrapped p-values are the number of cases that yielded more extreme statistics divided by the number of bootstrap replicates.

8 Concluding remarks

Graphical Processing Units are nowadays employed to execute parallel applications that has no relation to graphics operations. Hence the name general-purpose GPU. These applications involve in most cases very large data sets to which relatively simple operations are applied. The data set is split into smaller subsets that are processed independently by parallel threads. In this paper, we have seen that GPUs can also be helpful for parallelizing tasks that do not involve large data sets. In particular, a procedure based on the sweep operator was designed to obtain residual sums of squares of several linear regression models. Compared to a conventional implementation, the procedure reduces substantially the amount of required memory allocation, which allowed us to make the most of the GPU architecture. The code was employed in two applications: 1) simulation of response surfaces for seasonal unit root tests, 2) bootstrapping seasonal unit root test statistics. For the latter application, an interface that calls the CUDA program from the R environment is provided in the `uroot` package.

9 Acknowledgements

I thank for technical and human support provided by IZO-SGI SGIker of UPV/EHU and European funding (ERDF and ESF). I am also grateful to Ignacio Díaz-Emparanza for helpful comments and suggestions.

A Procedure to compute the HEGY test statistics

Let us consider the HEGY regression model for a quarterly series, $S = 4$, without deterministic components or lags of the dependent variable. The target matrix is built as follows:

$$D = \begin{bmatrix} \sum y_{0,t}^2 & \sum y_{0,t}y_{S/2,t} & \sum y_{0,t}y_{j,t}^a & \sum y_{0,t}y_{j,t}^b & \sum \Delta^S y_{0,t}y_t \\ \cdots & \sum y_{S/2,t}^2 & \sum y_{S/2,t}y_{j,t}^a & \sum y_{S/2,t}y_{j,t}^b & \sum y_{S/2,t}\Delta^S y_t \\ \cdots & \cdots & \sum (y_{j,t}^a)^2 & \sum y_{j,t}^a y_{j,t}^b & \sum y_{j,t}^a \Delta^S y_t \\ \cdots & \cdots & \cdots & \sum (y_{j,t}^b)^2 & \sum y_{j,t}^b \Delta^S y_t \\ \cdots & \cdots & \cdots & \cdots & \sum \left(\Delta^S y_t\right)^2 \end{bmatrix}.$$

The process to obtain the HEGY statistics consists on sweeping certain columns in order to obtain the RSS of the restricted and the unrestricted models involved in the test statistics. By sweeping up to the i-th column of matrix D, the RSS of a regression model that includes the regressors related to the swept columns is obtained. Put it in other way, the RSS of a model where the coefficients of the remaining regressors are restricted to zero is obtained.

I implemented the sweep operator in terms of a vector containing the upper side of the matrix D stacked by rows. Thus, instead of working with a matrix D of dimension 5×5 we can work with a vector of length 15 that will be denoted v. This is at the cost of some further operations to map the indexing from the matrix to the vector but savings in memory are relevant especially for large S; for example, with $S = 12$, the matrix D contains $13 \times 13 = 169$, while the vector v is of length $(13 \times 14)/2 = 91$.

A.1 Compute the residual sums of squares

Once the matrix D is built and the upper-side is packed into the vector v, the last element of this vector gives the RSS to be used in the statistic F_{Δ^S}. The RSS for each model are stored in a vector named `rss`.

$$rss_1 = v_{15}\,.$$

The RSS of the restricted model related to $F_{2:S}$, i.e., a model containing only the first HEGY regressor, $y_{0,t}$, is obtained by sweeping the first column. As we are interested only in the RSS (not in the inverse of $X^\top X$ or in $\hat{\beta}$, we only need to apply the operations that affect the last element. Thus, the operations reduce to:

$$rss_2 = v_{15} - v_5 \times v_5/v_1\,.$$

The RSS of the restricted model related to F_{pair} is obtained by sweeping the columns 1 and 2. Again, we don't need to apply all the operations, we can stick only to those that will affect the last element:

$$\begin{aligned} a_1 &= v_6 - v_2 \times v_2/v_1 \\ a_2 &= v_9 - v_2 \times v_5/v_1 \\ rss_3 &= rss_2 - a_2 \times a_2/a_1 \end{aligned} \quad .$$

At this point, just by doing 4 subtractions, 4 products and 4 divisions (aside from the operations required to build the matrix D), we have obtained three of the six residual sums of squares we are seeking for. The vector v has not been overwritten. In order to get the RSS of the restricted model related to t_π, the columns 4 and 3 are first swept. In this case, the complete operations of the sweep operator must be done. The following shows the right indexing in terms of the vector v, which gets now modified. Sweep column 4:

$$\begin{aligned} v_1 &= v_1 - v_4 \times v_4/v_{13}\,; & v_2 &= v_2 - v_4 \times v_8/v_{13} \\ v_3 &= v_3 - v_4 \times v_{11}/v_{13}\,; & v_5 &= v_5 - v_4 \times v_{14}/v_{13} \\ v_6 &= v_6 - v_8 \times v_8/v_{13}\,; & v_7 &= v_7 - v_8 \times v_{11}/v_{13} \\ v_9 &= v_9 - v_8 \times v_{14}/v_{13}\,; & v_{10} &= v_{10} - v_{11} \times v_{11}/v_{13} \\ v_{12} &= v_{12} - v_{11} \times v_{14}/v_{13}\,; & v_{15} &= v_{15} - v_{14} \times v_{14}/v_{13} \\ v_4 &= v_4/v_{13}\,; & v_8 &= v_8/v_{13} \\ v_{11} &= v_{11}/v_{13}\,; & v_{14} &= v_{14}/v_{13} \\ v_{13} &= -1/v_{13}\,; \end{aligned} \quad .$$

Sweep column 3:

$$\begin{aligned} v_1 &= v_1 - v_3 \times v_3/v_{10}\,; & v_2 &= v_2 - v_3 \times v_7/v_{10} \\ v_4 &= v_4 - v_3 \times v_{11}/v_{10}\,; & v_5 &= v_5 - v_3 \times v_{12}/v_{10} \\ v_6 &= v_6 - v_7 \times v_7/v_{10}\,; & v_8 &= v_8 - v_7 \times v_{11}/v_{10} \\ v_9 &= v_9 - v_7 \times v_{12}/v_{10}\,; & v_{13} &= v_{13} - v_{11} \times v_{11}/v_{10} \\ v_{14} &= v_{14} - v_{11} \times v_{12}/v_{10}\,; & v_{15} &= v_{15} - v_{12} \times v_{12}/v_{10} \\ v_3 &= v_3/v_{10}\,; & v_7 &= v_7/v_{10} \\ v_{11} &= v_{11}/v_{10}\,; & v_{12} &= v_{12}/v_{10} \\ v_{10} &= -1/v_{10}\,; \end{aligned} \quad .$$

Sweeping the first column will return the RSS related t_π. The whole sweep operator is not required, we can stick to the following operations, which do not modify v:

$$rss_4 = v_{15} - v_5 \times v_5 / v_1 \,.$$

Now, we seek the RSS of the restricted model related t_0 is given by: First, the second column is swept:

$$\begin{array}{rclrcl}
v_1 &=& v_1 - v_2 \times v_2/v_6\,; & v_3 &=& v_3 - v_2 \times v_7/v_6 \\
v_4 &=& v_4 - v_2 \times v_8/v_6\,; & v_5 &=& v_5 - v_2 \times v_9/v_6 \\
v_{10} &=& v_{10} - v_7 \times v_7/v_6\,; & v_{11} &=& v_{11} - v_7 \times v_8/v_6 \\
v_{12} &=& v_{12} - v_7 \times v_9/v_6\,; & v_{13} &=& v_{13} - v_8 \times v_8/v_6 \\
v_{14} &=& v_{14} - v_8 \times v_9/v_6\,; & v_{15} &=& v_{c(15)} - v_9 \times v_9/v_6 \\
v_2 &=& v_2/v_6\,; & v_7 &=& v_7/v_6 \\
v_8 &=& v_8/v_6\,; & v_9 &=& v_9/v_6 \\
v_6 &=& -1/v_6\,; & & &
\end{array} \,.$$

The RSS of the restricted model related t_0 is given by:

$$rss_5 = v_{15} - v_9 \times v_9 / v_6 \,.$$

Finally, the RSS of the unrestricted model is obtained as follows:

$$urss = v_{15} - v_5 \times v_5 / v_1 \,.$$

The indexing shown here changes for models with different periodicity, deterministic terms or lags. For illustration, I showed the actual indices for a particular case. The function `sweep` in the source code shows the general indexing that maps the conventional sweeping operator based on the index of a matrix into the indexing of a vector containing the upper side of the matrix and the diagonal stacked by rows.

A.2 Compute the test statistics

Upon the residual sums of squares obtained in A.1, it is straightforward to get the final test statistics defined in Equation 2:

$$\begin{array}{rclrcl}
F_{\Delta^S} &=& ((rss_1 - urss)/S)/\sigma^2\,, & t_0 &=& \sqrt{(rss_5 - urss)/\sigma^2}\,, \\
F_{2:S} &=& ((rss_2 - urss)/(S-1))/\sigma^2\,, & t_\pi &=& \sqrt{(rss_4 - urss)/\sigma^2}\,, \\
F_{pair} &=& ((rss_3 - urss)/2)/\sigma^2\,, & & &
\end{array}$$

where $\sigma^2 = urss/(n - 2 \times S)$. The sign of the t-statistics is set as follows: if $v_9 - v_2 \times v_5/v_1 < 0$ then $t_\pi = -t_\pi$; if $v_5/v_1 < 0$ then $t_0 = -t_0$.

B Dynamic memory allocation on the GPU

The main purpose of the implementation is to reduce the amount of memory that is employed by each thread. The optimization of the programme that is executed on the GPU achieves a large

Array	Size	Usage
`y0`	S	Last S observations of the seasonal random walk.
`vA`	$k \times (k+1)/2$	Vector containing the upper triangular part and the diagonal of the matrix D of crossproducts.
`dy`	$p+1$	Seasonally differenced series, the disturbances ε_t.
`ypi_row`	S	HEGY regressors at time t.

Table 3: Dynamic memory allocation on the GPU. Size is the number of double precision elements in the array. S is the periodicity of the data; p is the number of lags of dependent variable (if any) and $k = S + c + p + 1$, being c the number of deterministic regressors: e.g., in the simulations $c = 1$ since a constant is included; in the bootstrap, $c = S + 1$ when a constant, a linear trend and seasonal dummies are included.

reduction in the amount of memory that is dynamically allocated. The arrays that are dynamically allocated on the GPU are summarized in Table 3.

A conventional implementation that allocates for each element as much memory as the size of the element would consume much more memory, constraining the amount of threads that can be run simultaneously on the GPU. For example, let's take a model with the following parameters: periodicity $S = 4$, sample size $n = 100$, without constant and with two lags of the dependent variable. A conventional implementation would employ (reading from left to right in Equation 1): $98 + 4 \times 98 + 98 \times 2 = 686$ double precision numbers (98 rows instead of 100 since two observations are lost due to lags); while the code optimized for the GPU employs (adding up the size of the arrays reported in the second column of Table 3): $4 + 7 \times 8/2 + 3 + 4 = 39$ double precision numbers, i.e., 17 times less memory.

As discussed in Section 5, the bootstrap requires additional memory allocation for the arrays reported in Table 4.

Array	Size	Usage
Shared by all threads:		
`d_stats0`	$\lfloor S/2 \rfloor + 3$	Test statistics for the original series.
`d_eps`	$n - p$	Vector of residuals to be resampled.
`d_arcoefs`	p	Autoregressive coefficients estimated in the regression for the original data.
`d_csns`	2 or S	Number of residuals per season (cumulative).
`d_res3`	$N(\lfloor S/2 \rfloor - 1 + S \bmod 2)$	Storage for the F_j^{ab} statistics.
Allocated by each thread:		
`y0`	$S + p$	Last observations of the seasonal random walk (bootstrap replicates).
`dylags`	p	Last p innovations (resampled residuals).
`v_backup`	$k \times (k+1)/2$	Backup of the matrix of crossproducts.

Table 4: New elements required by the bootstrap and further dynamic memory allocation on the GPU. See Table 3.

References

Beaton AE (1964). *The Use of Special Matrix Operators in Statistical Calculus*. Ph.D. thesis, Faculty of the Graduate School of Education of Harvard University. doi10.1002/j.2333-8504.1964.tb00689.x. Reprinted as Educational Testing Service Research Bulletin 64-51. Princeton, New Jersey.

Beaulieu JJ, Miron JA (1993). "Seasonal Unit Roots in Aggregate U.S. Data." *Journal of Econometrics*, **55**(1–2), 305–328. doi10.1016/0304-4076(93)90018-Z.

Box GEP, Muller ME (1958). "A Note on the Generation of Random Normal Deviates." *The Annals of Mathematical Statistics*, **29**(2), 610–611. doi10.1214/aoms/1177706645.

Burridge P, Taylor RAM (2001). "On the Properties of Regression-Based Tests for Seasonal Unit Rroots in the Presence of Higher Order Serial Correlation." *Journal of Business & Economic Statistics*, **19**(3), 374–379. doi10.1198/073500101681019918.

Burridge P, Taylor RAM (2004). "Bootstrapping the HEGY Seasonal Unit Root Tests." *Journal of Econometrics*, **123**(1), 67–87. doi10.1016/j.jeconom.2003.10.029.

Cáceres JJ (1996). "Contraste de Raíces Unitarias en Datos Semanales." *Estadística Española*, **38**(141), 139–159. Instituto Nacional de Estadística.

Cheung YW, Lai KS (1995). "Lag Order and Critical Values of the Augmented Dickey-Fuller test." *Journal of Business and Economic Statistics*, **13**(3), 277–280. URL http://web.calstatela.edu/faculty/klai/KLPaper/JBES95Jy.pdf.

Cottrell A, Lucchetti R (2016). *Gretl User's Guide*. Department of Economics. Wake Forest University. URL http://gretl.sourceforge.net/.

Davidson R, MacKinnon JG (2004). *Econometric Theory and Methods*. Oxford University Press.

Díaz-Emparanza I (2014). "Numerical Distribution Functions for Seasonal Unit Root Tests." *Computational Statistics and Data Analysis*, **76**, 237–247. doi10.1016/j.csda.2013.03.006.

Dickey DA, Fuller WA (1979). "Distribution of the Estimators for Autoregressive Time Series with a Unit Root." *Journal of the American Statistical Association*, **74**(366), 427–431. URL http://www.jstor.org/stable/2286348.

Franses PH, Hobijn B (1997). "Critical Values for Unit Root Tests in Seasonal Time Series." *Journal of Applied Statistics*, **24**(1), 25–47. URL http://repub.eur.nl/pub/2097/.

Hansen BE (1995). "Rethinking the Univariate Approach to Unit Root Testing: Using Covariates to Increase Power." *Econometric Theory*, **11**, 1148–1171. ISSN 1469-4360. doi10.1017/S0266466600009993.

Harvey DI, van Dijk D (2006). "Sample Size, Lag Order and Critical Values of Seasonal Unit Root Tests." *Computational Statistics & Data Analysis*, **50**(10), 2734–2751. doi10.1016/j.csda.2005.04.011.

Hylleberg S, Engle R, Granger C, Yoo B (1990). "Seasonal Integration and Cointegration." *Journal of Econometrics*, **44**(1–2), 215–238. doi10.1016/0304-4076(90)90080-D.

Leybourne SJ (1995). "Testing for Unit Roots Using Forward and Reverse Dickey-Fuller Regressions." *Oxford Bulletin of Economics and Statistics*, **57**(4), 421–571. doi10.1111/j.1468-0084.1995.tb00040.x.

Lupi C (2009). "Unit Root CADF Testing with R." *Journal of Statistical Software*, **32**(1), 1–19. ISSN 1548-7660. doi10.18637/jss.v032.i02.

MacKinnon JG (1994). "Approximate Asymptotic Distribution Functions for Unit-Root and Cointegration Tests." *Journal of Business and Economic Statistics*, **12**(2), 167–176. doi10.1080/07350015.1994.10510005.

MacKinnon JG (1996). "Numerical Distribution Functions for Unit Root and Cointegration Tests." *Journal of Applied Econometrics*, **11**(6), 601–618. URL http://onlinelibrary.wiley.com/doi/10.1002/%28SICI%291099-1255%28199611%2911:6%3C601::AID-JAE417%3E3.0.CO;2-T/abstract.

Marchetti GM, Drton M, Sadeghi K (2015). *ggm: Functions for graphical Markov models*. R package version 2.3, URL http://CRAN.R-project.org/package=ggm.

Marsaglia G (2003). "Xorshift RNGs." *Journal of Statistical Software*, **8**(14), 1–6. doi10.18637/jss.v008.i14.

Monahan JF (2001). *Numerical Methods of Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

Nankervis JC, Savin NE (1996). "The Level and Power of the Bootstrap t Test in the AR(1) Model with Trend." *Journal of Business & Economic Statistics*, **14**(2), 161–168. doi10.2307/1392427.

NVIDIA Corporation (2014). "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210." Whitepaper. URL http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf.

NVIDIA Corporation (2015). *CUDA C Programming Guide*. URL http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

Otero J, Smith J (2012). "Response Surface Models for the Leybourne Unit Root Tests and Lag Order Dependence." *Computational Statistics*, **27**(3), 473–486. doi10.1007/s00180-011-0268-y.

Park JY (2003). "Bootstrap Unit Root Tests." *Econometrica*, **71**(6), 1845–1895. doi10.1111/1468-0262.00471.

Pollock DSG (1999). *A Handbook of Time-Series Analysis Signal Processing and Dynamics*. Academic Press.

Psaradakis Z (2000). "Bootstrap Tests for Unit Roots in Seasonal Autoregressive Models." *Statistics & Probability Letters*, **50**(4), 389–395. doi10.1016/S0167-7152(00)00128-0.

Rayner RK (1990). "Bootstrapping p Values and Power in the First-Order Autoregression: A Monte Carlo Investigation." *Journal of Business & Economic Statistics*, **8**(2), 251–263. doi10.2307/1391988.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Smith R, Taylor A, del Barrio Castro T (2009). "Regression-Based Seasonal Unit Roots." *Econometric Theory*, **25**(2), 527–560. URL http://www.jstor.org/stable/20532451.

Wilt N (2013). *CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley. URL http://www.cudahandbook.com/.