Journal of Statistical Software

February 2020, Volume 92, Code Snippet 2. *doi: 10.18637/jss.v092.c02*

gdpc: An R Package for Generalized Dynamic Principal Components

Daniel Peña
Universidad Carlos III de Madrid

Ezequiel Smucler
Universidad Torcuato Di Tella

Victor J. Yohai
Universidad de Buenos Aires

Abstract

gdpc is an R package for the computation of the generalized dynamic principal components proposed in Peña and Yohai (2016). In this paper, we briefly introduce the problem of dynamical principal components, propose a solution based on a reconstruction criteria and present an automatic procedure to compute the optimal reconstruction. This solution can be applied to the non-stationary case, where the components need not be a linear combination of the observations, as is the case in the proposal of Brillinger (1981). This article discusses some new features that are included in the package and that were not considered in Peña and Yohai (2016). The most important one is an automatic procedure for the identification of both the number of lags to be used in the generalized dynamic principal components as well as the number of components required for a given reconstruction accuracy. These tools make it easy to use the proposed procedure in large data sets. The procedure can also be used when the number of series is larger than the number of observations. We describe an iterative algorithm and present an example of the use of the package with real data.

Keywords: dimensionality reduction, high-dimensional time series, R.

1. Introduction

Dimension reduction is important for the analysis of multivariate time series, particularly for the high-dimensional data sets that are becoming increasingly common in applications, because the number of parameters in the usual multivariate time series models grows quadratically with the number of variables. The first proposal of a dynamic factor model for time series is due to Brillinger (1964, 1981) who proposed to apply standard techniques of factor analysis to the spectral matrix. He also proposed dynamic principal components, which are two sided linear combinations of the data which provide an optimal reconstruction. They are obtained by the inverse Fourier transform of the principal components of the spectral density matrices for each frequency. Geweke (1977) proposed a one-sided generalization of the static factor model where the factors and the innovations are mutually independent and follow covariance stationary linear processes, and applied standard estimation methods for factor analysis to the spectral density matrix instead of the covariance matrix. A similar model was used by Sargent and Sims (1977) who named their model the index model. In this model the factors account for all the cross-correlations among the series, whereas the factors and the innovations account for the autocorrelation of the observed series. Peña and Box (1987) propose a dynamic factor model for a vector of stationary time series with white noise innovations and proved that we can estimate the loading matrix by the eigenvectors corresponding to non null eigenvalues of the lag covariance matrices of the data. Stock and Watson (2002) use dynamic factors for forecasting, by assuming that the variable to forecast and the explanatory variables useful for its forecasting follow a dynamic factor model. Bai and Ng (2002) develop a criterion to consistently estimate the number of factors, which will be used in our package. Hu and Chou (2004) proposed a test for the number of factors and explore the generalization of the model for integrated factors that was carried out by Peña and Poncela (2006). Pan and Yao (2008) include general nonstationary processes for the factors. Lam and Yao (2012) proposed a test for the number of factors based on the eigenvalues of the lag covariance matrices.

Forni, Hallin, Lippi, and Reichlin (2000) generalized Geweke's dynamic factor model by allowing the idiosyncratic components to be autocorrelated and contain weak cross-correlations. The authors proposed to estimate the common components by the projection of the data on the dynamic principal components proposed by Brillinger. Forni, Hallin, Lippi, and Reichlin (2005) proposed a one-sided method of estimation of a dynamic factor model and used the method for forecasting. The forecasts generated with this procedure have been compared to the ones derived by Stock and Watson (2002) and the results are mixed (see Forni, Hallin, Lippi, and Zaffaroni (2017)). A modified forecasting approach was proposed by Forni, Hallin, Lippi, and Zaffaroni (2015), although again the finite sample results are mixed. Hallin and Lippi (2013) give a general presentation of the methodological foundations of dynamic factor models.

Peña and Yohai (2016) proposed a new approach for defining dynamic principal components that is different from the one taken by Brillinger in three ways. First, their generalized dynamic principal components (GDPC) are built without the assumption that the series are stationary. If the data is not stationary, the GDPC minimize a meaningful reconstruction criterion, unlike Brillinger's approach, which minimizes the expected reconstruction mean square error. Second, the GDPC need not be a linear combination of the original series. Third, they are computed directly in the time domain, instead of working in the frequency domain. These GDPC are optimal reconstructors of the observed data but they can also be used to estimate the common part in high-dimensional dynamic factor models. In fact, it has been shown that the GDPC provide consistent estimates (Smucler 2017) of the common part of dynamic factor models. Since the GDPC are based on both leads and lags of the data, like the dynamic principal components defined by Brillinger, they are not useful for forecasting. However, they are useful in general as a way to summarize the information in sets of dependent time series in which the factor structure may not apply. Finally, the optimal reconstructor property of the GDPC is useful for data compression to reduce resources required to store and transmit data. This makes them potentially useful for compression of dependent data and they could be applied for image analysis and other types of spatial data in which dependence is important.

A large number of R packages are available for working with time series. Besides the 'mts' class provided by the **stats** package (R Core Team 2019) several R packages provide classes and methods for handling time-indexed data. For example, the **zoo** package (Zeileis and Grothendieck 2005) provides an S3 class and methods for handling totally ordered indexed observations, in particular irregular time series. The **xts** package (Ryan and Ulrich 2018) is able to uniformly handle R's different time-based data classes. Our **gdpc** package supports 'mts', 'xts' and 'zoo' objects: if the original data is stored in an object of class 'mts', 'xts', or 'zoo', then the principal components and reconstructed time series will also be stored in an object of class 'mts', 'xts', or 'zoo' respectively, with the same date attributes. Among the multivariate time series R packages, the **MTS** package (Tsay and Wood 2018) is a general tool-kit for multivariate time series that includes vector autoregressive moving average (VARMA) models, factor models, and multivariate volatility models. **freqdom** (Hormann and Kidzinski 2017) implements Brillinger's dynamic principal components. **pcdpca** (Kidzinski, Jouzdani, and Kokoszka 2017) extends Brillinger's dynamic principal components to periodically correlated multivariate time series. An extensive comparison of Brillinger's approach to dynamic principal components and GDPC can be found in Peña and Yohai (2016). The **BigVAR** package (Nicholson, Matteson, and Bien 2019) estimates vector autoregressive (VAR) models with structured lasso penalties. Many more packages can be found at `https://CRAN.R-project.org/view=TimeSeries`. To the best of our knowledge, **gdpc** (Peña, Smucler, and Yohai 2020) is the only publicly available implementation of GDPC.

This article is organized as follows. In Section 2 we briefly review the definition of the GDPC and presents a new proposal to apply the procedure in an automatic way. Several possible methods for choosing the number of components and the number of lags used for each component are discussed. In Section 3 we review the iterative algorithm used to compute the GDPC. In Section 4 we illustrate the use of the **gdpc** package using artificial and real data sets. We compare the performance regarding computation time and the reconstruction of non-stationary time series of the implementation of Brillinger's method found in the **freqdom** package to that of GDPC in Section 5. In Section 6 we compare the performance of the different criteria available in the **gdpc** package to choose the number of lags that define the GDPC. Section 7 includes a small simulation study to estimate the typical computation times of the main algorithm for both stationary and non-stationary time series. Finally, conclusions are provided in Section 8.

2. Definition of generalized dynamic principal components

Consider a time series vector $\mathbf{z}_t = (z_{1,t}, \dots, z_{m,t})$, where $1 \leq t \leq T$, and let $\mathbf{Z}$ be the $T \times m$ matrix whose rows are $\mathbf{z}_1, \dots, \mathbf{z}_T$. Here T stands for the number of periods and m for the number of series. We define the first dynamic principal component with k lags as a vector $\mathbf{f} = (f_t)_{-k+1 \leq t \leq T}$, so that the reconstruction of series $z_{j,t}, 1 \leq j \leq m$, as a linear combination of $(f_{t-k}, \dots, f_{t-1}, f_t)$ is optimal with respect to the mean squared error (MSE) criterion. More precisely, given a possible factor $\mathbf{f}$ of length $(T+k)$, an $m \times (k+1)$ matrix of coefficients $\boldsymbol{\beta} = (\beta_{j,h})_{1 \leq j \leq m, 1 \leq h \leq k+1}$ and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m)$, the reconstruction of the original series $z_{j,t}$ is defined as

$$z_{j,t}^{R,b}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \alpha_j + \sum_{h=0}^{k} \beta_{j,h+1} f_{t-h}. \tag{1}$$

The R superscript in $z_{j,t}^{R,b}$ stands for reconstruction and the b superscript stands for backward, due to the reconstruction being defined using the lags of f_t, in constrast with the reconstruction using leads, to be defined later.

The MSE loss function when we reconstruct the m series using $\mathbf{f}$, $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ is given by

$$\text{MSE}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \frac{1}{Tm} \sum_{j=1}^{m} \sum_{t=1}^{T} (z_{j,t} - z_{j,t}^{R,b}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}))^2 \qquad (2)$$

and the values that minimize this MSE are called $(\widehat{\mathbf{f}}, \widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{\alpha}})$. The value of $\widehat{\mathbf{f}}$ is not identified as we can multiply the coefficients $\beta_{j,h+1}$ by a constant and divide f_{t-h} by the same contant and the model is the same. To solve this issue we take $\widehat{\mathbf{f}}$ with zero mean and unit variance. $\widehat{\mathbf{f}}$ is then the first GDPC. Note that for $k = 0$, $\widehat{\mathbf{f}}$ is the first ordinary (non-dynamic) principal component of the data. The second GDPC is defined as the first GDPC of the residuals

$$r_{j,t} = z_{j,t} - z_{j,t}^{R,b}(\widehat{\mathbf{f}}, \widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{\alpha}}), \quad 1 \leq j \leq m, 1 \leq t \leq T.$$

Higher order GDPC are defined in a similar fashion.

Note that the reconstruction given in Equation 1 can be written using leads instead of lags. Suppose to simplify that $\alpha_j = 0$ and $k = 1$ so that we have

$$z_{j,t}^{R,b}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}) = \beta_{j,1} f_t + \beta_{j,2} f_{t-1}.$$

Given $\mathbf{f}, \boldsymbol{\beta}$ and $\boldsymbol{\alpha}$, let $f_{t+1}^* = f_t$, $\beta_{j,2}^* = \beta_{j,1}$, $\beta_{j,1}^* = \beta_{j,2}$ and $\alpha_j^* = 0$. Then

$$z_{j,t}^{R,b}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}) = z_{j,t}^{R,f}(\mathbf{f}^*, \boldsymbol{\beta}^*, \boldsymbol{\alpha}^*) = \beta_{j,2}^* f_{t+1}^* + \beta_{j,1}^* f_t^*,$$

that is the same equation but now using leads. We just shift one position the series of the principal components and interchange the values of the β coefficients to obtain an equivalent representation but now using leads instead of lags. In general given $\mathbf{f}, \boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ we can define

$$f_{t+k}^* = f_t, \quad t = 1 - k, \dots, T, \qquad (3)$$

$$\beta_{j,k+2-g}^* = \beta_{j,g}, \quad 1 \leq g \leq k+1, j = 1, \dots, m, \qquad (4)$$

$$\alpha_j^* = \alpha_j, \quad j = 1, \dots, m \qquad (5)$$

and write

$$z_{j,t}^{R,f}(\mathbf{f}^*, \boldsymbol{\beta}^*, \boldsymbol{\alpha}^*) = \alpha_j^* + \sum_{h=0}^{k} \beta_{j,h+1}^* f_{t+h}^*.$$

Clearly

$$z_{j,t}^{R,f}(\mathbf{f}^*, \boldsymbol{\beta}^*, \boldsymbol{\alpha}^*) = z_{j,t}^{R,b}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha})$$

and hence we see that, without loss of generality, we can use either lags or leads of the principal component to reconstruct the series. Moreover, minimizing the function in Equation 2 is equivalent to minimizing

$$\frac{1}{Tm} \sum_{j=1}^{m} \sum_{t=1}^{T} (z_{j,t} - z_{j,t}^{R,f}(\mathbf{f}, \boldsymbol{\beta}, \boldsymbol{\alpha}))^2.$$

Since the notation is less cumbersome using leads, the derivation for the optimal solution in Section 3 is presented using leads. Moreover, all internal computations in the **gdpc** package are performed using leads. However, since the reconstruction using lags is more intuitive, the final output is passed to the form using lags via Equations 3, 4, 5. It can be shown that the GDPC can also be equivalently defined using a $k/2$ leads and $k/2$ lags of f_t to define the reconstruction, see Peña and Yohai (2016) for details. This explains the noisy character of the GDPC at the ends of the sample, see Figure 4 for example, and why, like Brillinger's DPC, the GDPC are not directly useful for forecasting.

Note that if we are considering p dynamic principal components of order k_i each, $i = 1, \ldots, p$, the number of values required to reconstruct the original series is $\sum_{i=1}^{p}(T+k_i+m(k_i+1)+m)$. In practice, the number of components and the number of lags used for each component need to be chosen. The MSE of the reconstruction decreases when either of these two numbers is increased, but the amount of information needed to be stored for the reconstruction will also increase. The number of components can be chosen so that a pre-specified fraction of the total variance of the data is explained, as is usual in ordinary principal component analysis. Regarding the choice of the number of lags for each component, let k be the number of lags used to fit the component under consideration. Let $\widehat{y}_{j,t} = \widehat{\alpha}_j + \sum_{h=0}^{k} \widehat{\beta}_{j,h+1}\widehat{f}_{t+h}$ be the interpolator used where $y_{j,t} = z_{j,t}$ for the first component and will be equal to the residuals from the fit with the previous components otherwise. Let $r_{j,t} = y_{j,t} - \widehat{y}_{j,t}$, be the residuals from the fit with the component, and $\mathbf{R}$ be the corresponding matrix of residuals from this fit and let $\mathbf{\Sigma} = (\mathbf{R}^\top\mathbf{R})/T$. Given k_{max}, k can be chosen among $0, \ldots, k_{max}$ as the value that minimizes some criterion. This criterion should take into account the MSE of the reconstruction and the amount of information to be stored. The following four criteria are available in **gdpc**:

- Leave-one-out (LOO) cross-validation:

$$LOO_k = \frac{1}{Tm}\sum_{i=1}^{m}\sum_{t=1}^{T}\frac{r_{i,t}^2}{(1-h_{k,tt})^2},$$

  where $h_{k,tt}$ are the diagonal elements of the hat matrix $\mathbf{H}_k = \mathbf{F}_k(\mathbf{F}_k^\top\mathbf{F}_k)^{-1}\mathbf{F}_k^\top$, with $\mathbf{F}_k$ being the $T \times (k+2)$ matrix with rows $(f_{t-k}, f_{t-k+1}, \ldots, f_t, 1)$.

- An AIC type criterion (Akaike 1974):

$$AIC_k = T\log\left(\text{trace}(\mathbf{\Sigma})\right) + m(k+2)2.$$

- A BIC type criterion (Schwarz 1978):

$$BIC_k = T\log\left(\text{trace}(\mathbf{\Sigma})\right) + m(k+2)\log T.$$

- A criterion based on the IC_{p3} proposal of Bai and Ng (2002):

$$BNG_k = \min(T,m)\log\left(\text{trace}(\mathbf{\Sigma})\right) + \log(\min(T,m))(k+1).$$

The AIC and BIC type criteria are known not to work well when the ratio T/m is small; this is to be expected since they are based on the assumption that m is fixed and $T \to \infty$. For the cases in which the ratio T/m is small it is interesting to use a criterion based on both m and T going to infinity. For this reason, we included the criterion BNG_k, based on a proposal of Bai and Ng (2002) for choosing the number of factors in a factor model. We will compare the performance of the criteria in Section 6.

3. Computing the GDPC

To compute the GDPC we note that, given the component, the β coefficients are regression coefficients that can be computed by least squares, and given the β coefficients we have again linear equations that can be easily computed. To write the equations we have to solve we introduce some notation. Define $a \vee b = \max(a,b)$ and $a \wedge b = \min(a,b)$. Let

$$\mathbf{C}_j(\alpha_j) = (c_{j,t,q}(\alpha_j))_{1\leq t\leq T+k, 1\leq q\leq k+1}$$

be the $(T+k)\times(k+1)$ matrix defined by

$$c_{j,t,q}(\alpha_j) = (z_{j,t-q+1} - \alpha_j),$$

when $1 \vee (t-T+1) \leq q \leq (k+1)\wedge t$ and zero otherwise. Let

$$\mathbf{D}_j(\beta_j) = (d_{j,t,q}(\beta_j))$$

be the $(T+k)\times(T+k)$ matrix given by

$$d_{j,t,q}(\beta_j) = \sum_{v=(t-k)\vee 1}^{t\wedge T} \beta_{j,q-v+1}\beta_{j,t-v+1},$$

when $(t-k)\vee 1 \leq q \leq (t+k)\wedge(T+k)$ and zero otherwise. Define

$$\mathbf{D}(\beta) = \sum_{j=1}^{m} \mathbf{D}_j(\beta_j).$$

Differentiating Equation 2 with respect to $\mathbf{f}$, it is easy to show that

$$\mathbf{f} = \mathbf{D}(\boldsymbol{\beta})^{-1}\sum_{j=1}^{m}\mathbf{C}_j(\boldsymbol{\alpha})\boldsymbol{\beta}_j \tag{6}$$

where $\boldsymbol{\beta}_j$, $j=1,\dots,m$, are the rows of $\boldsymbol{\beta}$. Differentiating Equation 2 with respect to $\boldsymbol{\beta}_j$ and α_j we obtain

$$\begin{pmatrix} \boldsymbol{\beta}_j \\ \alpha_j \end{pmatrix} = \left(\mathbf{F}(\mathbf{f})^\top \mathbf{F}(\mathbf{f})\right)^{-1} \mathbf{F}(\mathbf{f})^\top\ \mathbf{z}^{(j)}, \tag{7}$$

where $\mathbf{z}^{(j)} = (z_{j,1},\dots,z_{j,T})^\top$ and $\mathbf{F}(\mathbf{f})$ is the $T\times(k+2)$ matrix with t-th row $(f_t, f_{t+1},\dots,f_{t+k},1)$. To define an iterative algorithm to compute $(\widehat{\mathbf{f}}, \widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{\alpha}})$ it is enough to provide $\mathbf{f}^{(0)}$ and a rule describing how to compute $\boldsymbol{\beta}^{(h)}, \boldsymbol{\alpha}^{(h)}$ and $\mathbf{f}^{(h+1)}$ once $\mathbf{f}^{(h)}$ is known. The following two steps based on Equations 6 and 7 describe a natural rule to perform this recursion.

Step 1: Based on Equation 7, define $\boldsymbol{\beta}_j^{(h)}$ and $\alpha_j^{(h)}$, for $1 \leq j \leq m$, by

$$\begin{pmatrix} \boldsymbol{\beta}_j^{(h)} \\ \alpha_j^{(h)} \end{pmatrix} = \left(\mathbf{F}(\mathbf{f}^{(h)})^\top \mathbf{F}(\mathbf{f}^{(h)})\right)^{-1} \mathbf{F}(\mathbf{f}^{(h)})^\top \mathbf{z}^{(j)}.$$

Step 2: Based on Equation 6, define $\mathbf{f}^{(h+1)}$ by

$$\mathbf{f}^* = \mathbf{D}(\beta^{(h)})^{-1}\mathbf{C}(\boldsymbol{\alpha}^{(h)})\beta^{(h)}$$

and

$$\mathbf{f}^{(h+1)} = (T+k-1)^{1/2}(\mathbf{f}^* - \overline{\mathbf{f}}^*)/|||\mathbf{f}^* - \overline{\mathbf{f}}^*||.$$

The initial value $\mathbf{f}^{(0)}$ can be chosen equal to the ordinary first principal component, completed with k leads. We stop after `niter_max` iterations or when

$$\frac{\mathrm{MSE}(\mathbf{f}^{(h)}, \boldsymbol{\beta}^{(h)}, \boldsymbol{\alpha}^{(h)}) - \mathrm{MSE}(\mathbf{f}^{(h+1)}, \boldsymbol{\beta}^{(h+1)}, \boldsymbol{\alpha}^{(h+1)})}{\mathrm{MSE}(\mathbf{f}^{(h)}, \boldsymbol{\beta}^{(h)}, \boldsymbol{\alpha}^{(h)})} < \texttt{tol}$$

for some user-supplied values `tol` and `niter_max`.

All numerically intensive computations are performed in C++, using the **Armadillo** linear algebra library (Sanderson and Curtin 2016). The C++ code is integrated with R using packages **Rcpp**(Eddelbuettel and François 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014).

In step 1 of the algorithm we need to solve m least squares problems, each with T observations and $k+2$ predictor variables. The worst case complexity for solving each of these least squares problems is $O((k+2)^2T) = O(k^2T)$. Hence the worst case complexity for step 1 of the algorithm is $O(mk^2T)$. The worst case complexity for solving the linear system in step 2 of the algorithm is $O((T+k)^3)$. Hence, at it each iteration the worst case complexity is $O((T+k)^3 + mk^2T)$, which is linear in m. Thus, the algorithm is able to deal with very high-dimensional problems. Note that there are no restrictions on the values of $\mathbf{f}$ and, in particular, we do not assume, as in Brillinger (1981), that the GDPC must be linear combinations of the series. Note that in Equation 6 the computation of the component is linear in the observed data given the $\boldsymbol{\beta}$ parameters. Also, the $\boldsymbol{\beta}$ parameters are estimated as linear functions of the observations given the GDPC by Equation 7. However, these two step estimation leads to a component which, in general, will not be a linear function of the observed series. It is shown in Peña and Yohai (2016) in particular stationary cases that the components are approximately linear combinations of the data. This is a similar result to the one found in the static case with standard principal components where we do not impose this restriction but the optimal solution verifies it. However, when the series are not stationary this additional flexibility leads to values of $\mathbf{f}$ better adapted to the possible nonstationary character of the time series. We will back up this claim with experimental results in Section 5.

4. Using the gdpc package

There are two main functions available to the user: `gdpc` and `auto.gdpc`. We first describe `gdpc`.

4.1. The `gdpc` function and class

The function `gdpc` computes a single GDPC with a number of lags that has to be provided by the user. It has the following arguments:

- `Z`: Data matrix. Each column is a different time series.
- `k`: Integer. Number of lags to use.
- `f_ini`: (Optional). Numeric vector. Starting point for the iterations. If no argument is passed the ordinary first principal component completed with `k` lags is used.
- `tol`: Relative precision. Default is 10^{-4}.
- `niter_max`: Integer. Maximum number of iterations. Default is 500.
- `crit`: A string specifying the criterion to be used to evaluate the reconstruction. Options are `"LOO"`, `"AIC"`, `"BIC"` and `"BNG"`. Default is `"LOO"`.

The output of this function is an S3 object of class '`gdpc`', that is, a list with entries:

- `expart`: Proportion of the variance explained.
- `mse`: Mean squared error.
- `crit`: The value of the criterion of the reconstruction, according to what the user specified.
- `k`: Number of lags used.
- `alpha`: Vector of intercepts corresponding to `f`.
- `beta`: Matrix of loadings corresponding to `f`. Column number j is the vector of $j-1$ lag loadings.
- `f`: Coordinates of the first dynamic principal component corresponding to the periods $1, \dots, T$.
- `initial_f`: Coordinates of the first dynamic principal component corresponding to the periods $-\mathtt{k}+1, \dots, 0$. Only for the case $\mathtt{k} > 0$, otherwise 0.
- `call`: The matched call.
- `conv`: Logical. Did the iterations converge?
- `niter`: Integer. Number of iterations.

`fitted`, `plot` and `print` methods are available for this class.

We illustrate the use of this function with the an artificial data set. First, we load the package and generate the artificial data.

```
R> library("gdpc")
R> set.seed(1234)
R> T <- 200
R> m <- 5000
R> f <- rnorm(T + 1)
R> x <- matrix(0, T, m)
R> u <- matrix(rnorm(T * m), T, m)
R> for (i in 1:m) {
+    x[, i] <- 10 * sin(2 * pi * (i / m)) * f[1:T] +
+      10 * cos(2 * pi * (i / m)) * f[2:(T + 1)] + u[, i]
+  }
```

We use `gdpc` to compute a single GDPC using one lag. The rest of the arguments are the default ones.

```
R> fit <- gdpc(x, k = 1)
R> fit
```

```
            Number.of.lags   LOO   MSE Explained.Variance
Component 1              1 1.017 0.986              0.991
```

The result is stored in `fit` and the code `fit` produces the object to be printed. This shows the number of lags used, the value of the criterion specified by the user (the default `"LOO"` in this case), the MSE of the reconstruction and the fraction of explained variance. It is seen that the reconstruction is excellent, with more than 99% of the variance of the data explained.

Now we can plot the loadings and the principal component by using the `plot` method for the '`gdpc`' class. The method has the following arguments

- `x`: An object of class '`gdpc`', usually the result of `gdpc` or one of the entries of the result of `auto.gdpc`.
- `which`: String. Indicates what to plot, either `"Component"` or `"Loadings"`. Default is `"Component"`.
- `which_load`: Lag number indicating which loadings should be plotted. Only used if `which = "Loadings"`. Default is 0.
- `...`: Additional arguments to be passed to the plotting functions.

Continuing with our example, we plot the loadings and the principal component.

```
R> plot(fit, which = "Loadings", which_load = 0, xlab = "", ylab = "")
R> plot(fit, which = "Loadings", which_load = 1, xlab = "", ylab = "")
R> plot(fit, which = "Component", xlab = "", ylab = "")
```

The result is shown in Figure 1.

The reconstruction of the original series can be obtained using the `fitted` method for the '`gdpc`' class. We store it in `recons`.

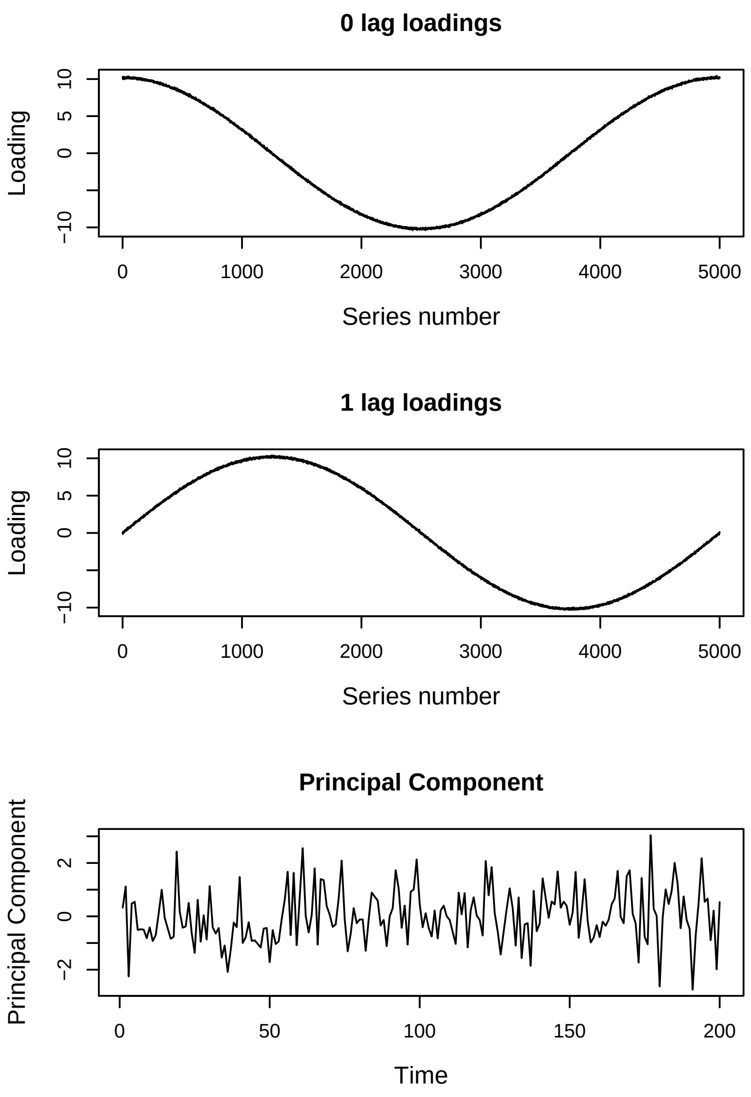

Figure 1: Plot of the loadings and principal component for an artificial data set.

```
R> recons <- fitted(fit)
```

We can compare the approximate amount of storage needed for the original data set and for the `fit` object`x` and the `gdpc` object `fit`.

```
R> object.size(x)

8000216 bytes

R> object.size(fit)

124352 bytes
```

Hence, the amount of memory needed to store `fit` is about 1.6% of that needed to store `x`.

4.2. The `auto.gdpc` function and the ‘`gdpcs`’ class

The function `auto.gdpc` computes *several* GDPC. The number of components can be supplied by the user or chosen automatically so that a given proportion of variance is explained. The number of lags is chosen automatically using one of the criteria listed in Section 2. The function has the following arguments

- `Z`: Data matrix. Each column is a different time series.
- `crit`: A string specifying the criterion to be used to choose the number of lags. Options are `"LOO"`, `"AIC"`, `"BIC"` and `"BNG"`. Default is `"LOO"`.
- `normalize`: Integer. Either 1, 2 or 3. Indicates whether the data should be standardized. Default is 1. See details below.
- `auto_comp`: Logical. If `TRUE` compute components until the proportion of explained variance is equal to `expl_var`, otherwise use `num_comp` components. Default is `TRUE`.
- `expl_var`: A number between 0 and 1. Desired proportion of explained variance (only used if `auto_comp` is `TRUE`). Default is 0.9.
- `num_comp`: Integer. Number of components to be computed (only used if `auto_comp` is `FALSE`). Default is 5.
- `tol`: Relative precision. Default is 10^{-4}.
- `k_max`: Integer. Maximum possible number of lags. Default is 10.
- `niter_max`: Integer. Maximum number of iterations. Default is 500.
- `ncores`: Integer. Number of cores to be used for parallel computations. Default is 1.
- `verbose`: Logical. Should progress be reported? Default is FALSE.

The argument `normalize` indicates whether the data should be normalized. If `normalize` = 1, the data is analysed in the original units, without mean and variance standardization. If `normalize` = 2, the data is standardized to zero mean and unit variance before computing the principal components, but the intercepts and loadings are those needed to reconstruct the original series. If `normalize` = 3 the data are standardized as in `normalize` = 2, but the intercepts and the loadings are those needed to reconstruct the standardized series. Default is `normalize` = 1, and hence the data is analysed in its original units.

The choice of `"LOO"` as the default criterion for choosing the number of lags is justified in Section 6. The optimal number of lags for each component can be computed in parallel, using the R packages **doParallel** (Revolution Analytics and Weston 2018) and **foreach** (Kane, Emerson, and Weston 2013). The argument `ncores` indicates the number of cores to be used for the parallel computations. The default value is 1, and hence by default no parallel computations are performed. If `verbose=TRUE` a message is printed each time a component is succesfully computed.

The output of `auto.gdpc` is an S3 object of class ‘`gdpcs`’, that is, a list of length equal to the number of computed components. The i-th entry of this list is an object of class ‘`gdpc`’ that stores the information of the i-th dynamic principal component, that is, a list with entries

- `expart`: Proportion of the variance explained by the first i components.
- `mse`: Mean squared error of the reconstruction using the first i components.
- `crit`: The value of the criterion of the reconstruction, according to what the user specified.
- `k`: Number of lags chosen.
- `alpha`: Vector of intercepts corresponding to `f`.
- `beta`: Matrix of loadings corresponding to `f`. Column number j is the vector of $j-1$ lag loadings.
- `f`: Coordinates of the i-th dynamic principal component corresponding to the periods $1, \dots, T$.
- `initial_f`: Coordinates of the i-th dynamic principal component corresponding to the periods $-\mathtt{k}+1, \dots, 0$. Only for the case $\mathtt{k} > 0$, otherwise 0.
- `call`: The matched call.
- `conv`: Logical. Did the iterations converge?
- `niter`: Integer. Number of iterations.

`components`, `fitted`, `plot` and `print` methods are available for this class.

We illustrate the use of this function with a real data set, `pricesSP50`, that is part of the package. The data set if formed by fifty series corresponding to the stock prices of the first 50 components of the Standard&Poor's 500 index. Five hundred daily observations starting 1/1/2010 are available. The class of the object storing the data set is '`mts`'.

The data set can be loaded and part of it can be plotted using the following commands

```
R> data("pricesSP50")
R> plot(pricesSP50[, 1:4], main = "Four components of the S&P500 index")
```

The plot is shown in Figure 2.

Next, we apply `auto.gdpc` to the data set, storing the result in `fit_SP`. Since some of the series are significantly more variable than others, we apply the procedure to the normalized data set using the option `normalize`=2. Since convergence is somewhat slow for this data set, we increased the maximum number of iterations from the default 500 to 1000 by setting `niter_max`=1000. We used 8 cores to perform the computation, by setting `ncores`=8. The rest of the arguments are left to their default values. In particular the number of lags for each component is chosen as the value than minimizes the LOO criterion and the number of components is chosen so that the reconstruction explains at least 90% of the variance of the data.

```
R> fit_SP <- auto.gdpc(pricesSP50, normalize = 2, niter_max = 1000,
+    ncores = 8)
```

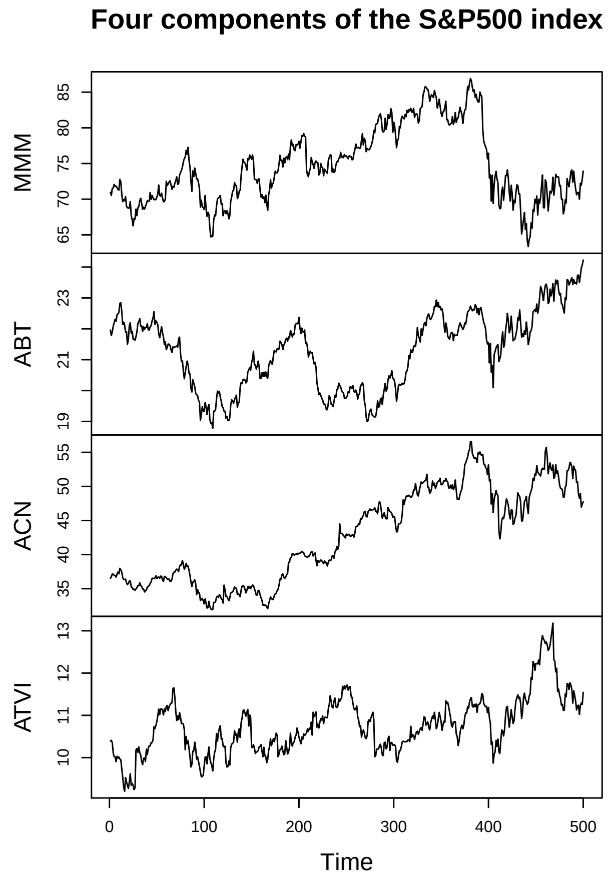

Figure 2: Plot of four components of the S&P500 index.

```
R> fit_SP
```

```
             Number.of.lags   LOO   MSE Explained.Variance
Component 1              10 0.185 0.174              0.826
Component 2               8 0.074 0.071              0.929
```

The whole computation took about 3 minutes on R version 3.4.0 on a computer running OS X El Capitan 10.11.4 64-bit with an Intel Xeon CPU E5-1650 v2 3.50GHz. Entering `fit_SP` produces the object to be printed. A table is printed where row i shows the number of lags used in component i, the value of the criterion specified by the user (the default `"LOO"` in this case) in component i, the MSE of the reconstruction using the components $1, \ldots, i$ and the fraction of explained variance by the reconstruction using components $1, \ldots, i$. Note that the MSEs and the criteria are those of the reconstruction of the normalized series in this case.

We can obtain the reconstruction of the original time series using the `fitted` method for the '`gdpcs`' class. The method has the following arguments

- `object`: An object of class '`gdpcs`', usually the result of `auto.gdpc`.
- `num_comp`: Integer indicating how many components to use for the reconstruction. Default is 1.

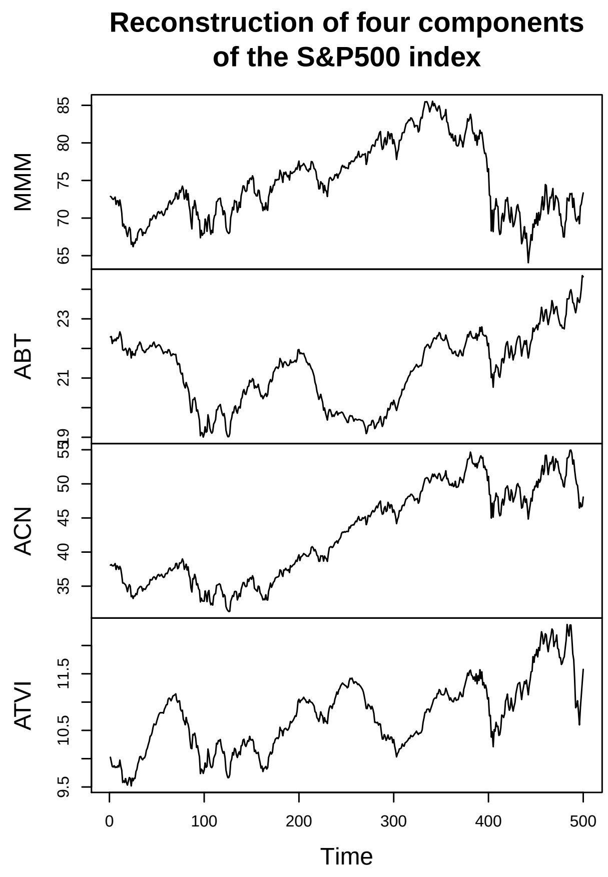

Figure 3: Plot of the reconstruction of four components of the S&P500 index.

- . . . : Additional arguments for compatibility.

Note that the **gdpc** package supports '`mts`', '`xts`' and '`zoo`' objects. The principal components and reconstructed time series will be stored in an object of class '`mts`', '`xts`' or '`zoo`' respectively, the same as the original data, with the same date attributes. Thus, since the `pricesSP50` data was stored in an object of class '`mts`', the reconstructed time series will be of class '`mts`', with the same date attributes.

We store the reconstructed time series using both computed components in an `mts` object called `recons`, assign each of the series its corresponding name, and plot the first four series. The result is shown in Figure 3.

```
R> recons <- fitted(fit_SP, num_comp = 2)
R> colnames(recons) <- colnames(pricesSP50)
R> plot(recons[, 1:4],
+    main = "Reconstruction of four components of the S&P500 index")
```

We can get the dynamic principal components using the `components` method for the class '`gdpcs`'. The method has the following arguments

- `object`: An object of class '`gdpcs`', usually the result of `auto.gdpc`.

- `which_comp`: Numeric vector indicating which components to get. Default is 1.

Since the original data was stored in an object of class ‘`mts`’, the components will also be of class ‘`mts`’. We store the components in an ‘`mts`’ object called `comps`.

```
R> comps <- components(fit_SP, which_comp = c(1, 2))
```

We can either directly plot the `comps` time series matrix or use the `plot` method for the class ‘`gdpcs`’. The method has the following arguments

- `x`: An object of class ‘`gdpcs`’, usually the result of `auto.gdpc`.
- `plot.type`: Argument to be passed to the `plot` for ‘`zoo`’ objects. Used only when the original data set was stored in an object of class ‘`zoo`’. Default is `"multiple"`.
- `which_comp`: Numeric vector indicating which components to plot. Default is 1.
- `...`: Additional arguments to be passed to the plotting functions.

Using the plot method for the ‘`gdpcs`’ we can plot both principal components. The result is shown in Figure 4.

```
R> plot(fit_SP, which_comp = c(1, 2))
```

We can compare the approximate amount of storage needed for the original data set `pricesSP50` and the ‘`gdpcs`’ object `fit_SP`.

```
R> object.size(fit_SP)

31184 bytes

R> object.size(pricesSP50)

204192 bytes
```

Hence, the amount of memory needed to store `fit_SP` is about 14% of that needed to store `pricesSP50`.

5. Reconstructing non-stationary data

We compare the performance of GDPC and the dynamic principal components proposed by Brillinger (1981) by conducting a small simulation study. We consider the following VARI(1,1) model. The data is generated according to

$$\mathbf{z}_t = \mathbf{z}_{t-1} + \mathbf{x}_t,$$

where $\mathbf{x}_t$ satisfies a stationary VAR(1) model

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{u}_t.$$

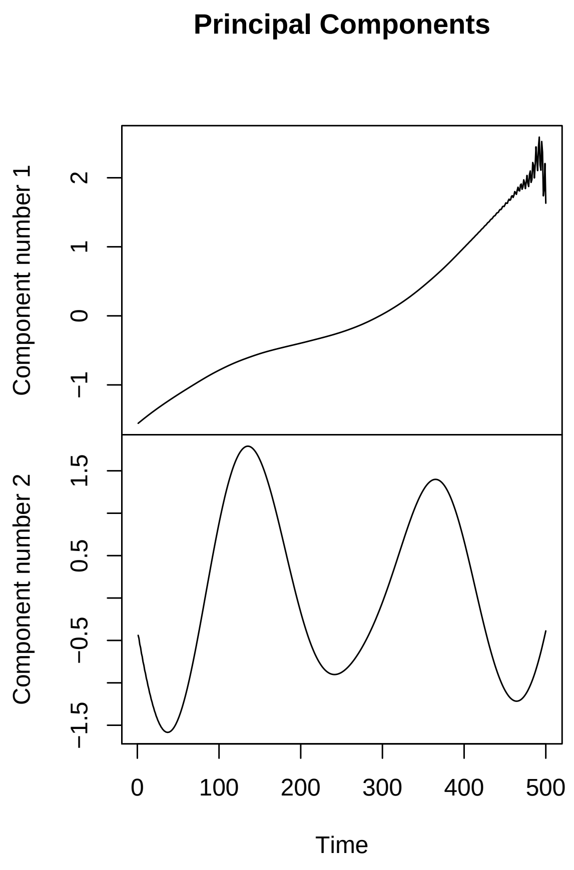

Figure 4: Plot of the first two dynamic principal components of the `pricesSP50` data set.

		GDPC		DPC	
T	m	Time	Explained variance	Time	Explained variance
100	10	1.34	0.95	2.87	0.72
	50	1.51	0.95	21.11	0.69
	100	1.69	0.94	90.09	0.68
200	10	8.76	0.93	2.94	0.69
	50	7.77	0.92	21.74	0.65
	100	7.79	0.91	92.94	0.64

Table 1: Average computation time in seconds and fraction of explained variance for each method.

The $\mathbf{u}_t$ are i.i.d. with a standard multivariate normal distribution. In each replication the matrix $\mathbf{A}$ is generated randomly as $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$, where $\mathbf{V}$ is an orthogonal matrix generated at random with uniform distribution and $\mathbf{\Lambda}$ is a diagonal matrix with diagonal elements are independent random variables with uniform distribution on the $[0, 0.9]$ interval. Note that the generated vector time series is not stationary.

We computed GDPC using one component and $k = 10$ lags and the dynamic principal components of Brillinger (DPC) using one component with 5 leads and 5 lags. We used the `dpca` function from the **freqdom** package (Hormann and Kidzinski 2017) to compute the method proposed by Brillinger. We take $(m, T) \in \{10, 50, 100\} \times \{100, 200\}$ and do 500 replications. Table 1 shows the results. We report the average computation time in seconds and the average fraction of variance explained. We see that the reconstructions obtained with GDPC are much better than those obtained with DPC, with the fraction of variance explained by GDPC being around 0.90 to 0.95, and the fraction of variance explained by DPC being around 0.65 to 0.70. Moreover, the computation times in seconds for GDPC are much lower. For example, for the case of $T = m = 100$, GDPC takes on average 1.69 seconds to compute a solution whereas the same task takes 90 seconds using DPC.

6. The performance of the lag selection criteria

In this section we compare the performance of the four different criteria available in the **gdpc** package to automatically choose the number of lags used to define the GDPC. Since we propose to choose the number of components to explain a given fraction of the variance in the data, we focus on the case in which one component is to be computed. We consider the following two scenarios.

- DFM1: The data is generated as

$$z_{j,t} = b_{0,j} f_t + b_{1,j} f_{t-1} + e_{j,t}, \quad 1 \leq t \leq T, 1 \leq j \leq m.$$

  f_t follows a stationary AR(1) model, $f_t = \theta f_{t-1} + u_t$, with standard normal innovations and θ generated at random on the $(-1, 1)$ interval at each replication. The loadings $b_{0,j}, b_{1,j}$, $j = 1, \dots, m$ are generated at random uniformly on the $(-1, 1)$ interval. The variables $e_{j,t}$ are generated as i.i.d. standard normal. This is a dynamic factor model with one factor and one lag.

- DFM2: The data is generated as

$$z_{j,t} = b_{0,j} f_t + b_{1,j} f_{t-1} + b_{2,j} f_{t-2} + e_{j,t}, \quad 1 \leq t \leq T, 1 \leq j \leq m.$$

  The factor f_t follows a stationary MA(1) model, $f_t = \theta u_{t-1} + u_t$ with standard normal innovations and with θ generated at random at each replication, uniformly on the $(-1, 1)$ interval. The loadings and the errors are generated as in model DFM1. This is a dynamic factor model with one factor and two lags.

We take $(m, T) \in \{5, 10, 20, 200, 800\} \times \{200, 400\}$ and do 500 replications. We report the average number of lags chosen by each criterion and the resulting average reconstruction mean squared error. Tables 2 and 3 show the results.

We see that in the cases in which the dimension of the vector time series is small, say $m = 5$ or $m = 10$, and the sample size is large, the AIC does well, choosing a number of lags close to the true values (1 for DFM1, 2 for DFM2). In this cases BNG tends to underestimate the required number of lags, whereas LOO tends to overestimate it. For moderate sized problems, with $m = 20$, LOO does best all around. For the case of high-dimensional vector time series ($m = 200, 800$), BNG and LOO perform similarly, choosing a number of lags that is on average very close to the truth. In this cases AIC underestimates the number of lags

		DFM1				DFM2			
T	m	BNG	LOO	AIC	BIC	BNG	LOO	AIC	BIC
200	5	0.02	4.14	0.81	0.26	0.00	4.02	1.64	0.54
	10	0.08	2.51	0.43	0.06	0.02	2.73	0.92	0.00
	20	0.28	1.27	0.15	0.00	0.29	2.15	0.01	0.00
	200	1.01	1.01	0.00	0.00	2.00	2.00	0.00	0.00
	800	1.01	1.01	0.00	0.00	2.00	2.00	0.00	0.00
400	5	0.02	3.76	0.95	0.51	0.00	3.83	1.88	1.16
	10	0.08	2.00	0.80	0.20	0.02	2.79	1.75	0.19
	20	0.19	1.15	0.38	0.03	0.18	2.10	0.83	0.00
	200	1.02	1.02	0.00	0.00	2.00	2.01	0.00	0.00
	800	1.01	1.01	0.00	0.00	2.00	2.00	0.00	0.00

Table 2: Average number of lags chosen by each method in scenarios DFM1 and DFM2.

		DFM1				DFM2			
T	m	BNG	LOO	AIC	BIC	BNG	LOO	AIC	BIC
200	5	0.88	0.75	0.80	0.83	0.96	0.75	0.79	0.87
	10	0.97	0.87	0.92	0.98	1.08	0.87	0.95	1.09
	20	1.00	0.93	1.02	1.06	1.09	0.93	1.14	1.15
	200	0.98	0.98	1.11	1.11	0.98	0.98	1.19	1.19
	800	0.98	0.98	1.11	1.11	0.98	0.98	1.20	1.20
400	5	0.88	0.78	0.80	0.82	0.96	0.78	0.79	0.82
	10	0.98	0.89	0.91	0.96	1.08	0.89	0.90	1.04
	20	1.01	0.94	0.98	1.05	1.11	0.94	1.03	1.15
	200	0.99	0.99	1.12	1.12	0.99	0.98	1.20	1.20
	800	0.99	0.99	1.13	1.13	0.99	0.99	1.21	1.21

Table 3: Average reconstruction mean squared error for scenarios DFM1 and DFM2, when the number of lags is chosen by each method.

needed. BIC tends to underestimate the required number of lags in all cases and does not perform well at all in any of the cases considered here.

In Table 3 we see that: in the cases where the methods choose on average a number of lags close to the true values, the reconstruction errors are close to the variance of the idiosyncratic part, which is 1 in this case. In cases where the methods overestimate the number of lags needed, the reconstruction error is smaller than the idiosyncratic variance, see for example the case of $T = 200$, $m = 5$ for the LOO criterion. This is to be expected, since if a larger number of lags than needed is used, the components will also explain part of the variance in the idiosyncratic part. In cases where the number of lags needed is underestimated, the reconstruction error is larger than 1, see for example the case of $T = 200$ and $m = 800$ for the BIC criterion.

Since we believe the main appeal of GDPC is for moderate to large sized panels of time series, the default criterion used in `auto.gdpc` is LOO. However, for small panels better results can be obtained by using the AIC criterion.

7. Computing times

In this section, we estimate the run times of our implementation of the algorithm described in Section 3 for different problem sizes by conducting a small simulation study. Timings were carried out for the `gdpc` function, since it is the one that implements the main numerical algorithm. Timings were carried out on R version 3.4.4 on a computer running Linux Ubuntu 18.04 64-bit with an Intel Core i7-7560U @ 2.40GHz x4.

We take $(m, T) \in \{100, 1000, 2000\} \times \{200, 400\}$ and consider the following two models.

- DFM3: The data is generated according to

$$z_{j,t} = \sin(2\pi(j/m))f_t + \cos(2\pi(j/m))f_{t-1} + e_{j,t}, \quad 1 \leq t \leq T, 1 \leq j \leq m,$$

  where the $e_{j,t}$ are i.i.d. standard normal variables. The vector of factors $\mathbf{f}_t = (f_{1t}, f_{2t})$ is generated according to a vector autoregressive model $\mathbf{f}_t = \mathbf{A}\mathbf{f}_{t-1} + v_t\mathbf{b}$, where the v_t are i.i.d. standard normals, $\mathbf{A}$ is a diagonal matrix with diagonal equal to $(-0.8, 0.7)$ and $\mathbf{b} = (1, 1)^\top$. Note that the resulting time series is stationary.

- DFM4: The data is generated according to

$$z_{j,t} = \sin(2\pi(j/m))f_t + \cos(2\pi(j/m))f_{t-1} + (j/m)f_{t-2} + e_{j,t}, \quad 1 \leq t \leq T, 1 \leq j \leq m,$$

  where the $e_{j,t}$ are i.i.d. standard normal variables. The factor f_t follows an integrated MA(1) model, $f_t = f_{t-1} + \theta f_{t-1} + u_t$, with standard normal innovations and where θ is generated at random uniformly on the $(-0.8, 0.8)$ interval at each replication. Note the the resulting time series is non-stationary.

We used $k = 5$ lags for the first model and $k = 2$ for the second model. We report the average computation times in seconds over 500 replications.

The results are shown in Table 4. We see that in the stationary case (DFM3) the algorithm can compute solutions for data sets with thousands of time series in under 15 seconds. It is seen that the convergence of the algorithm is slower in the non-stationary case (DFM4), since the computing times increase drastically for problems of the same size as before. However, even in the non-stationary case, the algorithm can compute solutions to problems with thousands of time series in under 1 minute. Moreover, note that computations were performed on an ordinary desktop computer, on a high-performance cluster we expect the algorithm to be able to compute the GDPC for tens of thousands of series in a matter of minutes.

T	m	DFM3	DFM4
200	100	0.35	0.92
	1000	2.07	5.89
	2000	3.88	11.50
400	100	1.38	4.48
	1000	6.86	22.01
	2000	12.68	40.46

Table 4: Average computing times in seconds for stationary and non-stationary factor models.

8. Conclusions

The **gdpc** package provides functions to compute the generalized dynamic principal components for a set of time series. These components are useful to reconstruct the series and to describe their underlying dynamic structure. Also, they can be used as estimators of the common part in a dynamic factor model, as shown in Section 6 and by the theoretical results in Smucler (2017). The package is useful for the analysis of large data sets, even when the number of series is much larger than the length of the series. The **gdpc** package is available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=gdpc`, including the `pricesSP50` data set.

Acknowledgments

The authors would like to thank two anonymous reviewers and the editors for their helpful comments. Part of this work was conducted while Ezequiel Smucler was a Ph.D. student at the Deparment of Mathematics at Universidad de Buenos Aires, funded by a CONICET fellowship.

References

Akaike H (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723. doi:10.1109/tac.1974.1100705.

Bai J, Ng S (2002). "Determining the Number of Factors in Approximate Factor Models." *Econometrica*, **70**(1), 191–221. doi:10.1111/1468-0262.00273.

Brillinger DR (1964). "The Generalization of the Techniques of Factor Analysis, Canonical Correlation and Principal Components to Stationary Time Series." Invited Paper at the Royal Statistical Society Conference in Cardiff, Wales.

Brillinger DR (1981). *Time Series: Data Analysis and Theory*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. doi:10.1016/j.csda.2013.02.005.

Forni M, Hallin M, Lippi M, Reichlin L (2000). "The Generalized Dynamic-Factor Model: Identification and Estimation." *The Review of Economics and Statistics*, **82**(4), 540–554. doi:10.1162/003465300559037.

Forni M, Hallin M, Lippi M, Reichlin L (2005). "The Generalized Dynamic Factor Model: One-Sided Estimation and Forecasting." *Journal of the American Statistical Association*, **100**(471), 830–840. doi:10.1198/016214504000002050.

Forni M, Hallin M, Lippi M, Zaffaroni P (2015). "Dynamic Factor Models with Infinite-Dimensional Factor Spaces: One-Sided Representations." *Journal of Econometrics*, **185**(2), 359 – 371. doi:10.1016/j.jeconom.2013.10.017.

Forni M, Hallin M, Lippi M, Zaffaroni P (2017). "Dynamic Factor Models with Infinite-Dimensional Factor Space: Asymptotic Analysis." *Journal of Econometrics*, **199**(1), 74–92. doi:10.1016/j.jeconom.2017.04.002.

Geweke J (1977). "The Dynamic Factor Analysis of Economic Time Series Models." In DJ Aigner, AS Goldberger (eds.), *Latent Variables in Socio-Economic Models*, pp. 365–383. North-Holland Publishing Company, Amsterdam.

Hallin M, Lippi M (2013). "Factor Models in High-Dimensional Time Series – A Time-Domain Approach." *Stochastic Processes and their Applications*, **123**(7), 2678–2695. doi:10.1016/j.spa.2013.04.001.

Hormann S, Kidzinski L (2017). ***freqdom**: Frequency Domain Based Analysis: Dynamic PCA*. R package version 2.0.1, URL https://CRAN.R-project.org/package=freqdom.

Hu YP, Chou RJ (2004). "On The Peña-Box Model." *Journal of Time Series Analysis*, **25**(6), 811–830. doi:10.1111/j.1467-9892.2004.00381.x.

Kane M, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(1), 1–19. doi:10.18637/jss.v055.i14.

Kidzinski L, Jouzdani N, Kokoszka P (2017). ***pcdpca**: Dynamic Principal Components for Periodically Correlated Functional Time Series*. R package version 0.4, URL https://CRAN.R-project.org/package=pcdpca.

Lam C, Yao Q (2012). "Factor Modeling for High-Dimensional Time Series: Inference for the Number of Factors." *The Annals of Statistics*, **40**(2), 694–726. doi:10.1214/12-aos970.

Nicholson W, Matteson D, Bien J (2019). ***BigVAR**: Dimension Reduction Methods for Multivariate Time Series*. R package version 1.0.4, URL https://CRAN.R-project.org/package=BigVAR.

Pan J, Yao Q (2008). "Modelling Multiple Time Series via Common Factors." *Biometrika*, **95**(2), 365–379. doi:10.1093/biomet/asn009.

Peña D, Box GEP (1987). "Identifying a Simplifying Structure in Time Series." *Journal of the American Statistical Association*, **82**(399), 836–843. doi:10.1080/01621459.1987.10478506.

Peña D, Poncela P (2006). "Nonstationary Dynamic Factor Analysis." *Journal of Statistical Planning and Inference*, **136**(4), 1237–1257. doi:10.1016/j.jspi.2004.08.020.

Peña D, Smucler E, Yohai VJ (2020). ***gdpc**: Generalized Dynamic Principal Components*. R package version 1.1.1, URL https://CRAN.R-project.org/package=gdpc.

Peña D, Yohai VJ (2016). "Generalized Dynamic Principal Components." *Journal of the American Statistical Association*, **111**(515), 1121–1131. doi:10.1080/01621459.2015.1072542.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Revolution Analytics, Weston S (2018). ***doParallel**: Foreach Parallel Adaptor for the **parallel** Package*. R package version 1.0.14, URL https://CRAN.R-project.org/package=doParallel.

Ryan JA, Ulrich JM (2018). ***xts**: eXtensible Time Series*. R package version 0.11-2, URL https://CRAN.R-project.org/package=xts.

Sanderson C, Curtin R (2016). "**Armadillo**: A Template-Based C++ Library for Linear Algebra." *The Journal of Open Source Software*, **1**(2). doi:10.21105/joss.00026.

Sargent TJ, Sims CA (1977). "Business Cycle Modeling without Pretending to Have Too Much A Priori Economic Theory." *Working Papers 55*, Federal Reserve Bank of Minneapolis. URL https://ideas.repec.org/p/fip/fedmwp/55.html.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Smucler E (2017). "Consistency of Generalized Dynamic Principal Components in Dynamic Factor Models." *ArXiv e-prints*. doi:10.1016/j.spl.2019.06.012. 1710.11286.

Stock JH, Watson MW (2002). "Forecasting Using Principal Components from a Large Number of Predictors." *Journal of the American Statistical Association*, **97**(460), 1167–1179. doi:10.1198/016214502388618960.

Tsay RS, Wood D (2018). ***MTS**: All-Purpose Toolkit for Analyzing Multivariate Time Series (MTS) and Estimating Multivariate Volatility Models*. R package version 1.0, URL https://CRAN.R-project.org/package=MTS.

Zeileis A, Grothendieck G (2005). "**zoo**: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27. doi:10.18637/jss.v014.i06.

Affiliation:

Daniel Peña
UC3M-BS Institute of Financial Big Data &
Department of Statistics
Universidad Carlos III de Madrid
E-mail: daniel.pena@uc3m.es

Ezequiel Smucler
Department of Mathematics and Statistics
Universidad Torcuato Di Tella
Avenida Figueroa Alcorta 7350
Buenos Aires 1428, Argentina
E-mail: esmucler@utdt.edu

Victor J. Yohai
Instituto de Cálculo
Universidad de Buenos Aires
Ciudad Universitaria, Pabellón 2
Buenos Aires 1428, Argentina
E-mail: vyohai@dm.uba.ar

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
February 2020, Volume 92, Code Snippet 2 *Submitted:* 2017-11-02
doi:10.18637/jss.v092.c02 *Accepted:* 2018-09-09