Mixed Frequency Data Sampling Regression Models: the R Package midasr

Eric Ghysels
University of North Carolina

Virmantas Kvedaras
Vilnius University

Vaidotas Zemlys
Vilnius University

Abstract

The implementation of MIDAS approach in the R package **midasr** is described within the framework of regressions with functional constraints on parameters.

Keywords: MIDAS, specification test.

1. Introduction

This paper[1] introduces a R package **midasr** for regression modelling with mixed frequency data. Ghysels (2013) has developed a package for MATLAB that deals with the estimation and information criteria-based specification of the mixed frequency data sampling (MIDAS) regressions as well as forecasting and nowcasting of low frequency series. In a slightly more genparameters $d := p + (k+1)(h+1)$ can be very large in terms of the number n of available observations of y_t. Since the estimation of the model can easily become infeasible, whenever either larger differences in frequencies or more variables and/or higher lag orders prevail, Ghysels et al. (2002) introduced a sufficiently flexible parametric restriction to be imposed on the original parameters, thus greatly reducing the number of parameters to be estimated: from d, which potentially is infinite, to much fewer number of parameters in a restricted model q which is assumed to be always considerably less than the number of observations available at the lowest frequency. If the parameters of an underlying data generating process did follow a certain functional constraint that is perfectly or well approximated by a constraint function chosen by a researcher, then full or great efficiency gains could be expected to be gained from the imposed constraints. Figure 1 plots the standard error of estimation (left figure) and a mean squared error (MSE) of the 2-norm of parameter estimates in an unconstrained and constrained simple model with correct and approximate restriction (see its characterization in Appendix A). As can be seen, even an incorrect constraint might be useful whenever the number of degrees of freedom in an unconstrained model is low and, consequently, one cannot rely on the large sample properties of unconstrained estimators. Furthermore, this approach seems to be necessary in the Classical framework whenever estimation is simply infeasible because of the lack of degrees of freedom.

1.1. Frequency alignment

Let n_i stand for the number of observations available per each frequency indexed by $i \in$

[1] This is a version 1 of the document, published in 2013-10-11

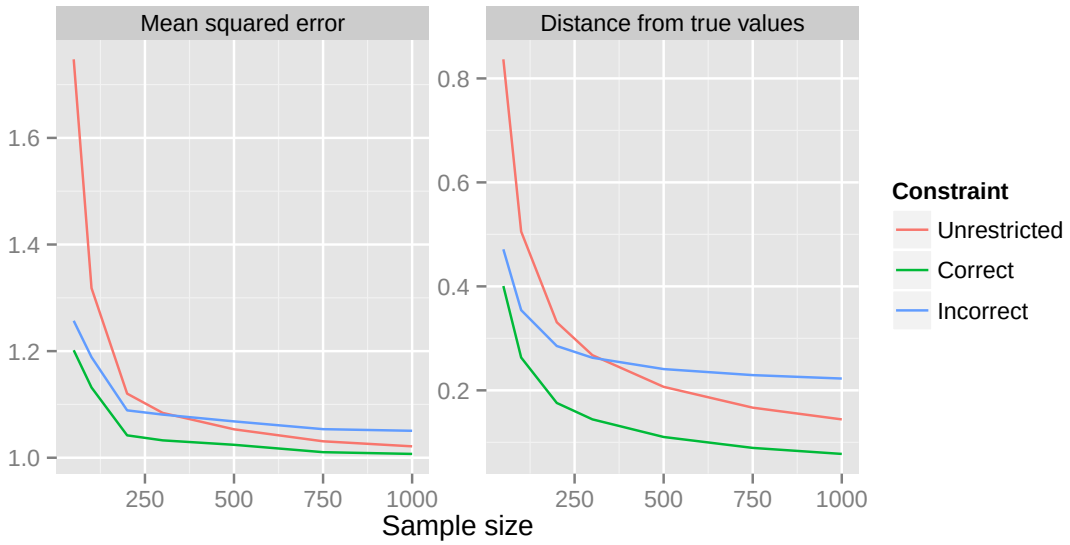

Figure 1: Mean squared error and distance to true coefficient values

$\{0, 1, \ldots, H\}$. Let m_i to represent a constant number of (potentially) higher-frequency observations per the lowest-frequency period. It holds $n_i = nm_i$, $i = 0, 1, \ldots, L$, with $n_0 = n$. Furthermore, denote by $\boldsymbol{x}^{(i)} := \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_{nm_i}^{(i)}, \right)'$ the observation vector of a i^{th} frequency component (explanatory variable) of vector $\boldsymbol{x}_{t,0}$.

In order to model variables of different frequencies in the framework of representation (**??**), the first step is to map the frequencies of a dataset that consists of observation vectors that have different lengths n_i. In linear MIDAS models, the mapping follows a simple time-ordering aggregation scheme (for details on this, see Kvedaras and Račkauskas 2010), where a vector of observations of each higher-frequency series $\boldsymbol{x}^{(i)}$ is transformed into the corresponding $k+1$ variables of the low frequency as follows:

$$\underset{n_i \times 1}{\boldsymbol{x}^{(i)}} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_{tm_i}^{(i)} \\ \vdots \\ x_{(n-1)m_i}^{(i)} \\ x_{nm_i}^{(i)} \end{bmatrix} \rightarrow \begin{bmatrix} x_{l_i m_i}^{(i)} & x_{l_i m_i - 1}^{(i)} & \cdots & x_{l_i m_i - k}^{(i)} \\ x_{(l_i+1)m_i}^{(i)} & x_{(l_i+1)m_i - 1}^{(i)} & \cdots & x_{(l_i+1)m_i - k}^{(i)} \\ \vdots & \vdots & \cdots & \vdots \\ x_{tm_i}^{(i)} & x_{tm_i - 1}^{(i)} & \cdots & x_{tm_i - k}^{(i)} \\ \vdots & \vdots & \cdots & \vdots \\ x_{(n-1)m_i}^{(i)} & x_{(n-1)m_i - 1}^{(i)} & \cdots & x_{(n-1)m_i - k}^{(i)} \\ x_{nm_i}^{(i)} & x_{nm_i - 1}^{(i)} & \cdots & x_{nm_i - k}^{(i)} \end{bmatrix} = \underset{(n-l_i+1)\times k}{\boldsymbol{X}^{(i)}}, \qquad (1)$$

where l_i is the smallest positive integer insuring $l_i m_i - k \geq 0$ i.e. that all the needed lags of i^{th} high-frequency are still available at low-frequency periods. When $k > m_i$, this implies that the effective number of low-frequency observations is reduced to $(n - l_i + 1)$, since the missing data appear in low-frequency periods smaller than l_i. A typical row of the matrix on the right side of eq. (1) framed by vertical dots represents a contemporaneous observation of a variable with its k lags that enter the conditional expectation of eq. (**??**) through the lag

polynomial i.e.

$$E\left(\alpha(B)y_t\middle|\boldsymbol{y}_{t,1},\boldsymbol{x}_{t,0}\right)=\boldsymbol{\beta}(L)'\boldsymbol{x}_{t,0}=\sum_{i=0}^{h}\sum_{j=0}^{k}\beta_j^{(i)}x_{tm_i-j}^{(i)}.$$

The package allows to align the frequencies for each series in this way as described in section 2.

1.2. Estimation

As was already mentioned, if no restrictions were placed on the parameters of eq. (**??**) and it were estimated directly by the ordinary least squares (OLS), then a U-MIDAS model would be under consideration, with an alternative to employ the Bayesian approachFurthermore, a consistent non-parametric approach could be used to estimate the underlying parameters of a function. However, none of these approaches uses a parametric functional constraint and, therefore, after the alignment of data frequencies, they can be directly performed using already available R packages. Whereas the **midasr** package aims at the estimation of mixed frequency models with some parametric functional constraint.

Model (**??**) is a linear model in terms of variables, but, if any of the functional constraints on parameters were non-linear, it would become non-linear with respect to hyper-parameters $\boldsymbol{\gamma}$. Hence, in the general case, we use in function `midas_r` the non-linear least squares (NLS) estimator of parameters $\boldsymbol{\gamma}$ of a restricted model (**??**) as defined by

$$\widehat{\boldsymbol{\gamma}}=\underset{\boldsymbol{\gamma}\in\mathbb{R}^q}{\operatorname{argmin}}\sum_{\lceil(k+1)/m\rceil}^{n}\left(\alpha(B)y_t-\boldsymbol{f}_{\boldsymbol{\gamma}}(L)'\boldsymbol{x}_{t,0}\right)^2, \tag{2}$$

where the lag polynomial of constrained parameters is defined by

$$\boldsymbol{f}_{\boldsymbol{\gamma}}(z)=\sum_{j=0}^{k}\boldsymbol{f}_{\boldsymbol{\gamma},j}z^j$$

with

$$\boldsymbol{f}_{\boldsymbol{\gamma},j}=\Big(f_0(\boldsymbol{\gamma}_0;j),\ldots,f_i(\boldsymbol{\gamma}_i;j),\ldots,f_h(\boldsymbol{\gamma}_h;j)\Big)'$$

for each $(i,j)\in\{0,1,\ldots,h\}\times\{0,1,\ldots,k\}$.

A number of numerical algorithms are ready-available in R for the solution of this problem. By default, the `optim` optimization function is used with optional choices of optimization algorithms in it. However, a user can also choose within function `midas_r` from other available alternatives in R such as `nls`, customizing the desired algorithm which is suitable for the problem at hand.

The efficiency of the estimator and consistency of the standard errors depend on whether the errors of the model are spherical. We leave the aspect of efficiency of estimation to be considered by a user, however the heteroscedasticity and autocorrelation (HAC) robust standard errors are optionally available relying on the implementation available in package **sandwich** (see Zeileis 2004).

If all the functional relations $f_i(\cdot)$ were non-constraining identity mappings, then the NLS estimator would be equivalent to the ordinary least squares (OLS) problem in terms of the

original parameters. For convenience, such a U-MIDAS version can be dealt with directly using a different function `midas_u` of the package (see an illustration in section 2) or a standard `lm` function, given that the alignment of data frequencies has been already performed as discussed in the previous section.

1.3. Taxonomy of aggregates-based MIDAS models

Based on the parsimony of representation argument, the higher-frequency part of conditional expectation of MIDAS regressions is often formulated in terms of aggregates as follows

$$\begin{aligned}\boldsymbol{\beta}(L)'\boldsymbol{x}_{t,0} &= \sum_{i=0}^{h}\sum_{j=0}^{k}\beta_j^{(i)}x_{tm_i-j}^{(i)} \\ &= \sum_{i=0}^{h}\sum_{r=0}^{p}\lambda_r^{(i)}\tilde{x}_{t-r}^{(i)},\end{aligned} \tag{3}$$

with some low-frequency number of lags $p \in \mathbb{N}$ and (directly unobservable) low-frequency aggregates

$$\tilde{x}_{t-r}^{(i)} := x_{t-r}^{(i)}(\boldsymbol{\delta}_{i,r}) = \sum_{s=1}^{m_i} w_r^{(i)}(\boldsymbol{\delta}_{i,r}; s)x_{(t-1-r)m_i+s}^{(i)}$$

that depend on a weighting (aggregating within a low-frequency period) function $w_r(\boldsymbol{\delta}_{i,r}; s)$ with its hyper-parameter vector $\boldsymbol{\delta}_{i,r}$, which, in the general case, can vary with each variable/frequency and/or the low-frequency lag order $r \in \mathbb{N}$. Here the aggregation weights are usually non-negative and, for identification of parameters $\{\lambda_r^{(i)}\}_{i=0,r=0}^{h,p}$, satisfy the normalization constraint such as $\sum_{s=0}^{m_i-1} w_r(\boldsymbol{\delta}_{i,r}; s) = 1$. To get the weights that add to one within a low-frequency, it is convenient to define a weighting function in the following form

$$\forall\ i, r \quad w_r^{(i)}(\boldsymbol{\delta}_{i,r}; s) = \frac{\psi_r^{(i)}(\boldsymbol{\delta}_{i,r}; s)}{\sum_{j=1}^{m_i}\psi_r^{(i)}(\boldsymbol{\delta}_{i,r}; j)},\ s = 1, \ldots, m_i, \tag{4}$$

given some underlying function $\psi_r^{(i)}(\cdot)$. Provided that the later function is non-negatively-valued (and the denominator is positive), the resulting weights in eq. (4) are also non-negative. Table 1 provides a list of some underlying functions producing, when used in eq. (4), the usual weighting schemes with non-negative weights (whenever the parameter space of underlying functions is appropriately bounded, which in some cases is also needed for identification of hyper-parameters). In order to avoid heavy notation, indices i and r—that are connected respectively with frequency/variable and the lag order—are dropped in the table.

Some other weighting functions that do not have a representation as in eq. (4) are also available in the package such as (non-normalized) `almonp` and the polynomial specification with step functions `polystep`.

However, the choice of a particular weighting function in the MIDAS regression with aggregates represents only one restriction imposed on $\boldsymbol{\beta}(L)$ out of many other choices to be made. To see this, let us note that aggregates-based MIDAS regressions can be connected with the

Resulting (normalized) weighting scheme	$\psi(\boldsymbol{\delta}; s) := \psi_r^{(i)}(\boldsymbol{\delta}_{i,r}; s)$	**Related midasr function**
Exponential Almon lag polynomial	$\psi(\boldsymbol{\delta}; s) = \exp\left(\sum_{j=1}^{p} \delta_j s^j\right),\ p \in \mathbb{N}$, where $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_j, \ldots, \delta_p)' \in \mathbb{R}^p$.	`nealmon`
Beta (analogue of probability density function)	$\psi(\boldsymbol{\delta}; s) = x_s^{\delta_1 - 1}(1 - x_s)^{\delta_2 - 1}$, where $x_s := \xi + (1-\xi)h(s),\ h(s) := (s-1)/(m-1)$, with some marginally small quantity $\xi > 0$, and $\boldsymbol{\delta} = (\delta_1, \delta_2)' \in \mathbb{R}_+^2$.	`nbeta`
Gompertz (analogue of probability density function)	$\psi(\boldsymbol{\delta}; s) = z(s)e^{-\delta_1 z(s)}$, where $z(s) = \exp(\delta_2 s)$, and $\boldsymbol{\delta} = (\delta_1, \delta_2)' \in \mathbb{R}_+^2$.	`gompertzp`
Log-Cauchy (analogue of probability density function)	$\psi(\boldsymbol{\delta}; s) = s^{-1}\left(\delta_2^2 + (\ln s - \delta_1)^2\right)^{-1}$, where $\boldsymbol{\delta} = (\delta_1, \delta_2)' \in \mathbb{R} \times \mathbb{R}_+$.	`lcauchyp`
Nakagami (analogue of probability density function)	$\psi(\boldsymbol{\delta}; s) = s^{2\delta_1 - 1}\exp(-\delta_1/\delta_2 s^2)$, where $\boldsymbol{\delta} = (\delta_1, \delta_2)'$, $\delta_1 \geq 0.5, \delta_2 \in \mathbb{R}_+$.	`nakagamip`

Table 1: Some usual weighting schemes in aggregation-based MIDAS specifications.

following restrictions on the conditional expectation of model (**??**):

$$
\begin{aligned}
E\left(\alpha(B)y_t|\boldsymbol{y}_{t,1}, \{\boldsymbol{x}_{t,0}^{(i)}\}_{j=0}^{k}\right) &= \boldsymbol{\beta}(L)'\boldsymbol{x}_{t,0} \\
&= \sum_{i=0}^{h}\sum_{r=0}^{p} \lambda_r^{(i)} \tilde{x}_{t-r}^{(i)}, \\
&= \sum_{i=0}^{h}\sum_{r=0}^{p} \lambda_r^{(i)} \sum_{s=1}^{m_i} w_r^{(i)}(\boldsymbol{\delta}_{i,r}; s) x_{(t-1-r)m_i+s}^{(i)}, \\
\Big|_{w_r^{(i)}(\cdot)=w_r(\cdot)} &= \sum_{i=0}^{h}\sum_{r=0}^{p} \lambda_r^{(i)} \sum_{s=1}^{m_i} w_r(\boldsymbol{\delta}_{i,r}; s) x_{(t-1-r)m_i+s}^{(i)}, \\
\Big|_{w_r(\cdot)=w(\cdot)} &= \sum_{i=0}^{h}\sum_{r=0}^{p} \lambda_r^{(i)} \sum_{s=1}^{m_i} w(\boldsymbol{\delta}_{i,r}; s) x_{(t-1-r)m_i+s}^{(i)}, \\
\Big|_{\boldsymbol{\delta}_{i,r}=\boldsymbol{\delta}_i} &= \sum_{i=0}^{h}\sum_{r=0}^{p} \lambda_r^{(i)} \sum_{s=1}^{m_i} w(\boldsymbol{\delta}_i; s) x_{(t-1-r)m_i+s}^{(i)}, \\
\Big|_{\lambda_r^{(i)}=\lambda^{(i)}} &= \sum_{i=0}^{h} \lambda^{(i)} \sum_{r=0}^{p}\sum_{s=1}^{m_i} w(\boldsymbol{\delta}_i; s) x_{(t-1-r)m_i+s}^{(i)},
\end{aligned}
\tag{5}
$$

As can be seen—and leaving aside other less intuitive restrictions—depending on the choice of a particular MIDAS specification with aggregates, it can impose restrictions on the equality of

- the applied weighting scheme/function across variables and/or frequencies ($\forall\, i,\ \ w_r^{(i)}(\cdot) = w_r(\cdot)$);

- the applied weighting scheme/function across all low-frequency lags $r = 0, 1, \ldots, p$ of aggregates ($\forall\ r,\quad w_r(\cdot) = w(\cdot)$);
- parameters of the weighting functions in each lag ($\forall\ r,\quad \boldsymbol{\delta}_{i,r} = \boldsymbol{\delta}_i$);
- impact of contemporaneous and lagged aggregates for all lags ($\forall\ r,\quad \lambda_r^{(i)} = \lambda^{(i)}$).

Furthermore, let s_i stand for an enumerator of i^{th} higher-frequency periods within a low-frequency period. Then, noticing that, given a frequency ratio m_i, there is a one-to-one mapping between higher-frequency index $j \in \mathbb{N}$ and a pair $(r, s_i) \in \mathbb{N} \times \{1, 2, \ldots, m_i\}$

$$j = rm_i + s_i,$$

it holds

$$f_i(\boldsymbol{\gamma}_i; rm_i + s_i) = \lambda_r^{(i)} w_r^{(i)}(\boldsymbol{\delta}_{i,r}; s). \tag{6}$$

Hence, it is easy to see that the aggregates-based MIDAS induces a certain periodicity of the functional constraint f_i in eq. (**??**) as illustrated bellow using a stylized case where all the restrictions are imposed in eq. (5):

$$\begin{array}{cccc|cccc|c} f_i(\cdot,0), & f_i(\cdot,1), & \ldots & f_i(\cdot,m-1) & f_i(\cdot,m), & f_i(\cdot,m+1), & \ldots & f_i(\cdot,2m-1) & \ldots \\ \lambda^{(i)}w(\cdot,1), & \lambda^{(i)}w(\cdot,2), & \ldots & \lambda^{(i)}w(\cdot,m) & \lambda^{(i)}w(\cdot,1), & \lambda^{(i)}w(\cdot,2), & \ldots & \lambda^{(i)}w(\cdot,m) & \ldots \end{array},$$

for any $i \in \{0, 1, \ldots, h\}$.

From eq. (6) it is clear that any specification of MIDAS models which relies on aggregates is a special case of representation (**??**) with just a specific functional constraint on parameters. On the other hand, not every general constraint $\boldsymbol{\beta}(L)$ can be represented using periodic aggregates. For instance, in the above characterized example the correspondence necessarily breaches whenever there exists at least one frequency i, for which none of $p \in \mathbb{N}$ satisfies $k = pm_i - 1$.

1.4. Specification selection and adequacy testing

Besides the usual considerations about the properties of the error term ,there are two main questions about the specification of the MIDAS models. First, suitable functional constraints need to be selected, since their choice will affect the precision of the model. Second, the appropriate maximum lag orders need to be chosen.

One way to address both issues together is to use some information criterion to select the best model in terms of the parameter restriction and the lag orders using either in- or out-of-sample precision measures. Functions `midas_r_ic_table` and `amidas_table` of the package allow to make an in-sample choice using some usual information criteria, such as AIC and BIC, and a user-specified list of functional constraints[2]. However, this does not insure that the model selected is an adequate one from the statistical point of view: even the best model from a set of wrong ones can still be a poor approximation of the underlying true process.

For instance, whenever the autoregressive terms in model (**??**) are present ($p > 0$), it was pointed out by Ghysels et al. (2007) that, in the general case $\boldsymbol{\phi}(L) = \boldsymbol{\beta}(L)/\alpha(B)$ will have seasonal pattern thus corresponding to some seasonal impact of explanatory variables on the dependent one in a pure distributed lag model (i.e. without autoregressive terms). To avoid

[2]Although aimed at forecasting, function `select_forecasts` can also be used to perform the selection of models relying on their out-of-sample performance.

such an effect whenever it is not (or is believed to be not) relevant, Clements and Galvão (2008) proposed to us a common factor restriction that, in more general case, becomes a common polynomial restriction with a constraint on the polynomial $\boldsymbol{\beta}(L)$ to satisfy a factorization $\boldsymbol{\beta}(L) = \alpha(B)\boldsymbol{\phi}(L)$, so that inverting equation (**??**) in terms of the polynomial $\alpha(B)$ leaves $\boldsymbol{\phi}(L)$ unaffected i.e. without creating/destroying any (possibly absent) seasonal pattern of the impact of explanatory variables. However, there is little if any knowledge a priori whether the impact in the distributed lag model should be seasonal or not. Hence, an explicit testing of adequacy of the model and, in particular, of the imposed functional constraint could be of great help here.

Let $\boldsymbol{\beta}$ denote a vector of all coefficients of polynomial $\boldsymbol{\beta}(z)$ defined in eq. (**??**), while $\boldsymbol{f}_{\boldsymbol{\gamma}}$ stand for the corresponding vector of coefficients restricted by a (possibly incorrect) functional constraint in $\boldsymbol{f}_{\boldsymbol{\gamma}}(z)$. Let $\widehat{\boldsymbol{\beta}}$ denote the respective OLS estimates of unconstrained model i.e. where functional restrictions of parameters are NOT taken into account. Let $\hat{\boldsymbol{f}}_{\boldsymbol{\gamma}} := \boldsymbol{f}_{\boldsymbol{\gamma}}\big|_{\boldsymbol{\gamma}=\widehat{\boldsymbol{\gamma}}}$ denote a vector of the corresponding quantities obtained from the restricted model relying on the NLS estimates $\widehat{\boldsymbol{\gamma}}$ as defined in eq. (2). Denote by $\boldsymbol{\alpha}$, $\widehat{\boldsymbol{\alpha}}$, and $\widehat{\boldsymbol{\alpha}}_{\gamma}$ the corresponding vectors of coefficients of polynomial $\alpha(z)$, its OLS estimates in an unrestricted model, and its NLS estimates in a restricted model[3]. Let $\boldsymbol{\theta} := (\boldsymbol{\alpha}', \boldsymbol{\beta}')'$, $\widehat{\boldsymbol{\theta}} := (\widehat{\boldsymbol{\alpha}}', \widehat{\boldsymbol{\beta}}')'$, and $\widetilde{\boldsymbol{\theta}} := (\widehat{\boldsymbol{\alpha}}_{\boldsymbol{\gamma}}', \hat{\boldsymbol{f}}_{\boldsymbol{\gamma}}')'$ to signify the corresponding vectors of all coefficients in eq. (**??**). Then, under the null hypothesis of $\exists\, \boldsymbol{\gamma} \in \mathbb{R}^q$ such that $\boldsymbol{f}_{\boldsymbol{\gamma}} = \boldsymbol{\beta}$, it holds

$$(\widehat{\boldsymbol{\theta}} - \widetilde{\boldsymbol{\theta}})'\boldsymbol{A}(\widehat{\boldsymbol{\theta}} - \widetilde{\boldsymbol{\theta}}) \sim \chi^2(d-q),$$

where $\boldsymbol{A}$ is a suitable normalisation matrix (see Kvedaras and Zemlys 2012 for a standard and Kvedaras and Zemlys 2013 for a HAC-robust versions of the test), and $q = \dim(\boldsymbol{\gamma})$ and $d = \dim(\boldsymbol{\theta})$ stand for the number of parameters in a restricted and unrestricted models, respectively. Functions `hAh.test` and `hAhr.test` of the package implement the described testing as will be illustrated hereafter.

1.5. Forecasting

Let us write model (**??**) for period $t+1$ as

$$y_{t+1} = \boldsymbol{\alpha}'\boldsymbol{y}_{t,0} + \boldsymbol{\beta}(L)'\boldsymbol{x}_{t+1,0} + \varepsilon_{t+1}, \tag{7}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_p)'$ is a vector of parameters of the autoregressive terms. This representation is well suited for (one step ahead) conditional forecasting of y_{t+1}, provided that the information on the explanatory variables is available. If it were absent, forecasts of $\boldsymbol{x}_{t+1,0}$ would be also necessary from a joint process of $\{y_t, \boldsymbol{x}_{t,0}\}$ which might be difficult to specify and estimate correctly, especially, bearing in mind the presence of data with mixed frequencies. Instead, a direct approach to forecasting is often applied in the MIDAS framework. Namely, given an information set available up to a moment t defined by $\mathcal{I}_{t,0} = \{\boldsymbol{y}_{t,j}, \boldsymbol{x}_{t,j}\}_{j=0}^{\infty}$, an ℓ-step ahead direct forecast

$$\tilde{y}_{t+\ell} = E\left(y_{t+\ell}\middle|\mathcal{I}_{t,0}\right) = \boldsymbol{\alpha}_{\ell}'\boldsymbol{y}_{t,0} + \boldsymbol{\beta}_{\ell}(L)'\boldsymbol{x}_{t,0},\ \ell \in \mathbb{N}, \tag{8}$$

can be formed leaning on a model linked to a corresponding conditional expectation

$$y_{t+\ell} = \boldsymbol{\alpha}_{\ell}'\boldsymbol{y}_{t,0} + \boldsymbol{\beta}_{\ell}(L)'\boldsymbol{x}_{t,0} + \varepsilon_{\ell,t},\ E\left(\varepsilon_{\ell,t}\middle|\mathcal{I}_{t,0}\right),$$

[3]Recall that unconstrained $\boldsymbol{\alpha}$ elements make a subset of parameter vector $\boldsymbol{\gamma}$ of a constrained model.

where $\boldsymbol{\alpha}_\ell$ and $\boldsymbol{\beta}_\ell(L)$ are the respective horizon ℓ-specific parameters. Notice that, in principle, these conditional expectations have a form of representation (**??**) with certain restrictions on the original lag polynomials of coefficients. Hence, in the general case, the suitable restrictions for each ℓ will have a different form.

Given periods $\ell = 1, 2, \ldots,$ and a selected model or a list of specifications to be considered, package **midasr** provides the point forecasts corresponding to the estimated analogue of eq. (8) evaluates the precision of different specifications, and performs weighted forecasting using the framework defined in Ghysels (2013).

2. Implementation in midasr package

2.1. Data handling

From the data handling point of view, the key specificity of the MIDAS model is that the length of observations of variables observed at various frequencies differs and needs to be aligned as described in section **??**. There is no existing R function which performs such a transformation and package **midasr** gives a solution to these challenges. The basic functionality of data handling is summarized in Table 2.

Function	Description	Example	Notes
`mls(x,k,m)`	Stacks a HF data vector x into a corresponding matrix of observations at LF of size $\frac{\dim x}{m} \times \dim k$: from the first to the last HF lag defined by vector k.	`mls(x,2:3,3)`	$\frac{\dim x}{m}$ must be an integer (NA are allowed). For $m = 1$, the function produces lags of x that are defined by vector argument k, e.g., `mls(x,2:3,1)` yields a dataset containing the lags x_{t-2} and x_{t-3} of x_t.
`fmls(x,k,m)`	Same as `mls`, except that k is a scalar and the $k+1$ lags are produced starting from 0 up to k.	`fmls(x,2,3)`	`fmls(x,2,3)` is equivalent to `mls(x,0:2,3)`.
`dmls(x,k,m)`	Same as `fmls`, apart that the resulting matrix contains $k+1$ first-order HF differences of x.	`dmls(x,2,3)`	`mls(x,1,1)` can be used in `dmls` to get stacking of lagged differences, e.g., `dmls(mls(x,1,1),2,3)`.

Table 2: A summary of basic data handling functionality in package *midasr*.

Function `fmls(x,k,m)` performs exactly the transformation defined in eq. (1), converting an observation vector x of a given (potentially) higher-frequency series into the corresponding stacked matrix of observations of $(k+1)$ low-frequency series (contemporaneous with k lags) as defined by the maximum lag order k and the frequency ratio m. For instance, given a series of twelve observations

```
x <- 1:12
```

we get the following result

```
fmls(x, k = 2, m = 3)

##      X.0/m X.1/m X.2/m
## [1,]     3     2     1
## [2,]     6     5     4
## [3,]     9     8     7
## [4,]    12    11    10
```

i.e. three variables (a contemporaneous and two lags) with four low-frequency observations ($n = 12/m$).

Function `mls` is slightly more flexible as the lags included can start from a given order rather than from zero, whereas function `fmls` uses a full lag structure. `dmls` performs in addition a first-order differencing of the data which is convenient when working with integrated series.

A couple of aspects should be taken into account when working with series of different frequencies.

- It is assumed that the numbers of observations of different frequencies match exactly through the frequency ratio ($n_i = nm_i$), and the first and last observations of each series of different frequency are correspondingly aligned (possibly using `NA` to account for some missing observations for series of higher frequency).
- Because of different lengths of series of various frequencies, the data in the model cannot be kept in one `data.frame`. An R object `list` needs to bee used if one intends to keep the data in a single object, but it is not required for the further modelling.

2.2. An example of simulated MIDAS regression

Using the above data handling functions, it is straightforward to simulate a response series from the MIDAS regression as a data generating process (DGP). For instance, suppose one is willing to generate a low-frequency response variable y in the MIDAS with two higher-frequency series x and z where the impact parameters satisfy the exponential Almon lag polynomials of different orders as follows:

$$\begin{gathered} y_t = 2 + 0.1t + \sum_{j=0}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=0}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_t, \\ x_{\tau_1} \sim n.i.d.(0,1), \quad z_{\tau_2} \sim n.i.d.(0,1), \quad \varepsilon_t \sim n.i.d.(0,1), \end{gathered} \tag{9}$$

where $(x_{\tau_1}, z_{\tau_2}, \varepsilon_t)$ are independent for any $(\tau_1, \tau_2, t) \in \mathbb{Z}^3$, and

$$\beta_j^{(i)} = \gamma_0^{(i)} \frac{\exp\left(\sum_{s=1}^{q_i-1} \gamma_s^{(i)} j^s\right)}{\sum_{j=0}^{d_i-1} \exp\left(\sum_{s=1}^{q_i-1} \gamma_s^{(i)} j^s\right)}, \ i = 1, 2,$$

where $d_1 = k_1 + 1 = 8$ is a multiple of the frequency ratio $m_1 = 4$, whereas $d_2 = k_2 + 1 = 17$ is not a multiple of $m_2 = 12$. Here $q_1 = 2$, $q_2 = 3$ with parametrizations

$$\begin{aligned} \boldsymbol{\gamma}_1 &= (1, -0.5)', \\ \boldsymbol{\gamma}_2 &= (2, 0.5, -0.1)', \end{aligned}$$

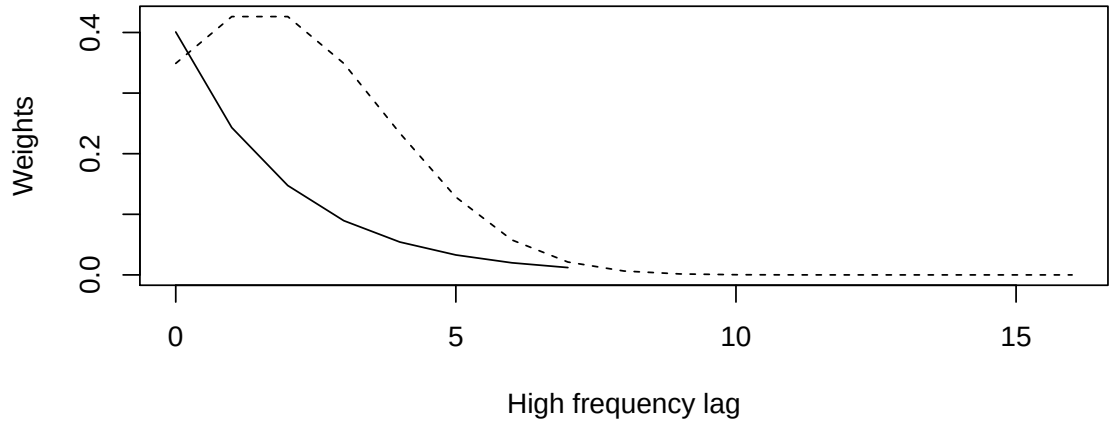

Figure 2: Shapes of impact

that yield the shapes of impact as plotted in Figure 2.

The following R code produces a series according to the DGP characterized above:

```
set.seed(1001)
## Number of low-frequency observations
n <- 250
## Linear trend and higher-frequency explanatory variables
## (e.g. quarterly and monthly)
trend <- c(1:n)
x <- rnorm(4 * n)
z <- rnorm(12 * n)
## Exponential Almon polynomial constraint-consistent
## coefficients
fn.x <- nealmon(p = c(1, -0.5), d = 8)
fn.z <- nealmon(p = c(2, 0.5, -0.1), d = 17)
## Simulated low-frequency series (e.g. yearly)
y <- 2 + 0.1 * trend + mls(x, 0:7, 4) %*% fn.x + mls(z, 0:16,
    12) %*% fn.z + rnorm(n)
```

It is of interest to note that the impact of variable x can be represented using aggregates-based MIDAS, whereas the impact of z cannot.

2.3. Some specification examples of MIDAS regressions

Suppose now that we have (only) observations of y, x, and z that are stored as vectors, matrices, time series, or list objects in R, and our intention is to estimate a MIDAS regression model as in eq. (9):

a) without restricting the parameters (as in U-MIDAS) and using the OLS;

b) with the exponential Almon lag polynomial constraint on parameters (as in function `nealmon`) and using the NLS.

The OLS estimation as in case a) is straightforwardly performed using

```
eq.u <- lm(y ~ trend + mls(x, k = 0:7, m = 4) + mls(z, k = 0:16,
    m = 12))
```

or, equivalently

```
eq.u <- midas_r(y ~ trend + mls(x, 0:7, 4) + mls(z, 0:16, 12),
    start = NULL)
```

The following R code estimates the constrained case b) using function `midas_r` and reports the NLS estimates $\widehat{\gamma}$ of hyper-parameters with the related summary statistics.

```
eq.r <- midas_r(y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z,
    0:16, 12, nealmon), start = list(x = c(1, -0.5), z = c(2,
    0.5, -0.1)))
summary(eq.r)

##
##  Formula y ~ trend + mls(x, 0:7, 4, nealmon) + mls(z, 0:16, 12, nealmon)
##
##  Parameters:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.988196   0.115299   17.24  < 2e-16 ***
## trend        0.099883   0.000777  128.57  < 2e-16 ***
## x1           1.353343   0.151220    8.95  < 2e-16 ***
## x2          -0.507566   0.096670   -5.25  3.3e-07 ***
## z1           2.263473   0.172815   13.10  < 2e-16 ***
## z2           0.409653   0.155685    2.63  0.00905 **
## z3          -0.072979   0.020392   -3.58  0.00042 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Residual standard error: 0.932 on 242 degrees of freedom
```

As you can see the syntax of the function `midas_r` is similar to the standard R function `nls`. The model is specified via familiar `formula` interface. The lags included and functional restriction used can be individual to each variable and are specified within the respective `mls`, `fmls`, or `dmls` function used with `midas_r`. It is necessary to provide a list of starting values for each variable with restricted coefficients, since it implicitly defines the number of hyper-parameters of the constraint functions to be used for each series.

The main difference from the function `nls` is that there is a greater choice of numerical optimisation algorithms. The function `midas_r` is written in a way that in theory it can use any R optimisation function. The choice is controlled via `Ofunction` argument. Currently it is possible to use functions `optim` and `nls` which are present in standard R installation and function `spg` from the package `BB`. The additional arguments to the aforementioned functions can be specified directly in the call to `midas_r`. So for example if we want to use the optimisation algorithm of Nelder and Mead, which is the default option in function `optim` we use the following code

```
eq.r2<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:16,12,nealmon),
               start=list(x=c(1,-0.5),z=c(2,0.5,-0.1)),
               Ofunction="optim", method="Nelder-Mead")
```

If we want to use Golub-Pereyra algorithm for partially linear least-squares models implemented in function `nls` we use the following code

```
eq.r2<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:16,12,nealmon),
               start=list(x=c(1,-0.5), z=c(2,0.5,-0.1)),
               Ofunction="nls", method="plinear")
```

It is possible to reestimate the NLS problem with the different algorithm using as starting values the final solution of the previous algorithm. For example it is known, that the default algorithm in `nls` is sensitive to starting values. So first we can use the standard Nelder-Mead algorithm to find "more feasible" starting values and then use the `nls` to get the final result:

```
eq.r2<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:16,12,nealmon),
               start=list(x=c(1,-0.5),z=c(2,0.5,-0.1)),
               Ofunction="optim",method="Nelder-Mead")
eq.r2<-midas_r(eq.r2,Ofunction="nls")
```

The output of the optimisation function used can be found by inspecting the element `opt` of `midas_r` output.

```
eq.r2<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:16,12,nealmon),
               start=list(x=c(1,-0.5),z=c(2,0.5,-0.1)),
               Ofunction="optim", method="Nelder-Mead")
eq.r2$opt

## $par
## (Intercept)       trend          x1          x2          z1          z2
##     1.98565     0.09990     1.35252    -0.50816     2.26318     0.40886
##          z3
##    -0.07284
##
## $value
```

```
## [1] 210
##
## $counts
## function gradient
##      502       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
```

Here we observe that Nelder-Mead algorithm evaluated the cost function 502 times.

The optimisation functions in R report the status of the convergence of optimisation algorithm by the numeric constant, 0 indicating succesful convergence. This code is reported as the element `convergence` of the `midas_r` output.

```
eq.r2$convergence

## [1] 1
```

In this case the convergence was not succesfull. The help page of function `optim` indicates that convergence code 1 means that iteration limit was reached.

In order to improve the convergence it is possible to use user defined gradient functions. To use them it is necessary to define gradient function of the restriction. For example for `nealmon` restriction the gradient function is defined in the following way:

```
nealmon.gradient <- function(p,d,m) {
    i <- 1:d
    pl <- poly(i,degree=length(p)-1,raw=TRUE)
    eplc <- exp(pl%*%p[-1])[,,drop=TRUE]
    ds <- apply(pl*eplc,2,sum)
    s <- sum(eplc)
    cbind(eplc/s,p[1]*(pl*eplc/s-eplc%*%t(ds)/s^2))
}
```

The naming convention `restriction_name.gradient` is important, currently this is the way `midas_r` "knows" about the gradient functions. To use this function for optimisation it is necessary to set `user.gradient=TRUE` in call to `midas_r`:

```
eq.r2<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:16,12,nealmon),
               start=list(x=c(1,-0.5),z=c(2,0.5,-0.1)),
               user.gradient=TRUE)
```

This way `midas_r` calculates the exact gradient of the NLS problem (2) using the specified gradient function of the restriction. For all the types of the restrictions referenced in table 3 the gradient functions are specified in the package **midasr**.

The gradient and the hessian of the NLS problem are supplied as the output of `midas_r`. The numerical approximation of the hessian is calculated using the package **numDeriv**, the exact gradient is calculated if `user.gradient=TRUE` and the numerical approximation otherwise. Having the gradient and hessian calculated allows to check whether the necessary and sufficient conditions for the convergence are satisfied. This is perfomed by function `deriv_test` which calculates the euclidean norm of the gradient and the eigenvalues of the hessian. It then tests whether the norm of gradient is close to zero and whether the eigen values are positive.

```
deriv_tests(eq.r, tol = 1e-06)

## $first
## [1] FALSE
##
## $second
## [1] TRUE
##
## $gradient
## [1]  0.0042441  0.0503346 -0.0008513  0.0021252  0.0004805 -0.0004505
## [7] -0.3851794
##
## $eigenval
## [1] 1.048e+07 5.888e+04 3.664e+02 1.221e+02 8.117e+01 5.148e+01 4.596e+01
```

To retrieve a vector of constrained estimates $\tilde{\theta}$ (and, hence, also $\hat{\boldsymbol{f}} = \boldsymbol{f}_{\boldsymbol{\gamma}}\big|_{\boldsymbol{\gamma}=\hat{\boldsymbol{\gamma}}}$) that corresponds to the vector $\boldsymbol{\theta}$ ($\boldsymbol{\beta}$, respectively), function `midas_coef` can be used as follows

```
midas_coef(eq.r)

## (Intercept)       trend          x1          x2          x3          x4
##   1.988e+00   9.988e-02   5.481e-01   3.300e-01   1.986e-01   1.196e-01
##          x5          x6          x7          x8          z1          z2
##   7.197e-02   4.332e-02   2.608e-02   1.570e-02   3.347e-01   4.050e-01
##          z3          z4          z5          z6          z7          z8
##   4.235e-01   3.827e-01   2.989e-01   2.018e-01   1.177e-01   5.932e-02
##          z9         z10         z11         z12         z13         z14
##   2.584e-02   9.728e-03   3.165e-03   8.898e-04   2.162e-04   4.539e-05
##         z15         z16         z17
##   8.237e-06   1.292e-06   1.750e-07
```

In the example provided above, a functional constraint was imposed directly on $\boldsymbol{\beta}(L)$ terms corresponding to each series without the usage of aggregates. Relying on the relationship (6), it is always possible to write such an explicit general constraint from an aggregates-based one. For convenience of a user, function `amweights` can be used to form several standard periodic functional constraints with 'typical' restrictions explicated in eq. (4). For instance,

```
amweights(p = c(1, -0.5), d = 8, m = 4, weight = nealmon, type = "C")

## [1] 0.4551 0.2760 0.1674 0.1015 0.4551 0.2760 0.1674 0.1015
```

with `type="C"` corresponds to a fully restricted version of aggregates-based expression (4) apart the cross-restriction on the equality of weighting schemes between different variables/frequencies. Notice that the code above repeats the result of

```
nealmon(p = c(1, -0.5), d = 4)

## [1] 0.4551 0.2760 0.1674 0.1015
```

twice ($d/m = 2$), as implied by the number of periods at higher-frequency (`d=8`) and the frequency ratio (`m=4`). In this way, function `amweights` can be used to define explicitly a new functional constraint relying on the relationship (6). Alternatively, one can indicate directly within function `midas_r` that the aggregates-based restriction must be used as follows

```
eq.r2 <- midas_r(y ~ trend + mls(x, 0:7, 4, amweights, nealmon,
    "C") + mls(z, 0:16, 12, nealmon), start = list(x = c(1, -0.5),
    z = c(2, 0.5, -0.1)))
```

where the first variable follows and aggregates-based MIDAS restriction scheme. Notice that the selection of alternative types "A" and "B" are connected with specifications having a larger number of parameters (see Table 3), hence the list of starting values needs to be adjusted to account for an increase in the number of (potentially unequal) impact parameters.

It should be also noted that, whenever the aggregates-connected restrictions are used, the number of periods must be a multiple of the frequency ratio. For instance, the current lag specification for variable z is not consistent with this requirement and cannot be represented through the (periodic) aggregates, but either `mls(z,0:11,12,amweights,nealmon,"C")` or `mls(z,0:23,12,amweights,nealmon,"C")` would be valid expressions from the code implementation point of view.

Table 3 summarizes and provides various other examples of correspondence between `midas_r` coding and the analytical specifications of MIDAS regressions.

2.4. Adequacy testing of restrictions

Given a MIDAS model estimated with `midas_r`, the empirical adequacy of the functional restrictions can be tested under quite standard assumptions (see Kvedaras and Zemlys 2012 and Kvedaras and Zemlys 2013) using functions `hAh.test` and `hAhr.test` of the package. In the case of a stationary series $\{y_t\}$ they can be applied directly, whereas whenever $\{y_t\}$ is cointegrated with explanatory variables, a special transformation needs to be applied before the testing (see e.g. Bilinskas and Zemlys 2013). The `hAh.test` can be used whenever errors of the process are independently and identically distributed, whereas the `hAhr.test` uses a HAC-robust version of the test. We should just point out that, whenever no significant HAC in the residuals are observed, we would suggest using `hAh.test` which would then have more

Description	**Code example**	**Analytical expression**	**Notes**
Different constraint functions	`midas_r(y~mls(x,0:7,4,nealmon)+ mls(z,0:16,12,gompertzp), start=list(x=c(1,-0.5),z=c(1,0.5,0.1)))`	$y_t = c + \sum_{j=0}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=0}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_t$	Constraints on $\beta_j^{(i)}$, $i = 1, 2$ are given by different functions.
Partial constraint (only on z)	`midas_r(y~mls(x,0:7,4)+mls(z,0:16,12, nealmon),start=list(z=c(1,-0.5)))`	$y_t = c + \sum_{j=0}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=0}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_t$	x enters linearly with unconstrained $\beta_j^{(1)}$.
With unrestricted autoregressive terms	`midas_r(y~mls(y,1:2,1)+mls(x,0:7,4, nealmon),start=list(x=c(1,-0.5)))`	$y_t = c + \sum_{j=1}^{2} \alpha_j y_{t-j} + \sum_{j=0}^{7} \beta_j x_{4t-j} + \varepsilon_t$	Autoregressive terms enter linearly with unconstrained coefficients.
With a common factor restriction	`midas_r(y~mls(y,1:2,1,"*")+mls(x,0:7,4, nealmon),start=list(x=c(1,-0.5)))`	$\alpha(B) y_t = c + \alpha(B)\lambda(L) x_{4t} + \varepsilon_t,$	Here coefficients of $\lambda(z)$ are assumed to satisfy nealmon restriction.
With autoregr. parameters restricted by a function	`midas_r(y~mls(y,1:6,1,nealmon) +mls(x,0:7,4,nealmon), start=list(y=c(1,-0.5),x=c(1,-0.5)))`	$y_t = c + \sum_{j=1}^{6} \alpha_j y_{t-j} + \sum_{j=0}^{7} \beta_j x_{4t-j} + \varepsilon_t$	Autoregressive parameters α_j, $j = 1, \ldots, 6$ are constrained to satisfy nealmon restriction.
Aggregates-based (Case A)	`midas_r(y~mls(x,0:7,4,amweights, nealmon,"A"),start=list(x=c(1,1,1,-0.5)))`	$y_t = c + \sum_{r=0}^{1} \lambda_r \sum_{s=1}^{4} w(\boldsymbol{\delta}_r; s) x_{4(t-1-r)+s} + \varepsilon_t$	The same weighting scheme (not parameters) is used in aggregation.
Aggregates-based (Case B)	`midas_r(y~mls(x,0:7,4,amweights, nealmon,"B"),start=list(x=c(1,1,-0.5)))`	$y_t = c + \sum_{r=0}^{1} \lambda_r \sum_{s=1}^{4} w(\boldsymbol{\delta}; s) x_{4(t-1-r)+s} + \varepsilon_t$	The same weights are used in aggregation.
Aggregates-based (Case C)	`midas_r(y~mls(x,0:7,4,amweights, nealmon,"C"),start=list(x=c(1,-0.5)))`	$y_t = c + \lambda \sum_{r=0}^{1} \sum_{s=1}^{4} w(\boldsymbol{\delta}; s) x_{4(t-1-r)+s} + \varepsilon_t$	A common impact parameter of lags and the same weights are used in aggregation.
With a user-defined constraint	`midas_r(y~mls(x,0:101,4,fn), start= list(x=c(0,0)))`	$y_t = c + \sum_{j=0}^{101} \beta_j x_{4t-j} + \varepsilon_t,$ $\beta_j = \gamma_1 (j+1)^{\gamma_2},\ j = 0, 1, \ldots, 101.$	`fn` in the `midas_r` code is as e.g. provided in the note of the table.

Table 3: Some functionality of specification of MIDAS regressions in *midasr* package.

Note: A function `fn` in the last example of the table is defined e.g. by `fn<-function(p,d){p[1]*c(1:d)^p[2]}`

precise test sizes in small samples. In the case of integrated series $\{y_t\}$ which is co-integrated with explanatory variables, some other alternatives are available (see Kvedaras et al 2013).

For illustration, let us use the name `eq.r` of an estimated model as in the previous subsections. Then the functions produce, respectively,

```
hAh.test(eq.r)

##
##  hAh restriction test
##
## data:
## hAh = 16.55, df = 20, p-value = 0.6818

hAhr.test(eq.r)

##
##  hAh restriction test (robust version)
##
## data:
## hAhr = 14.85, df = 20, p-value = 0.7847
```

Here the value of a test statistic, the degree of freedom (the number of binding constraints on parameters in eq. (**??**)), and the empirical significance of the null hypothesis that a functional constraint is adequate are reported.

As can be seen, such a specification, which in fact corresponds to the underlying DGP, cannot be rejected at the usual significance levels, whereas e.g. reducing the number of hyper-parameters of functional constraint of variable z to only two instead of three is quite strongly rejected using either version of the test:

```
eq.rb<-midas_r(y~trend+mls(x,0:7,4,nealmon)+mls(z,0:12,12,nealmon),
               start=list(x=c(1,-0.5),z=c(2,-0.1)))
hAh.test(eq.rb)

##
##  hAh restriction test
##
## data:
## hAh = 36.89, df = 17, p-value = 0.00348

hAhr.test(eq.rb)

##
##  hAh restriction test (robust version)
##
## data:
## hAhr = 32.88, df = 17, p-value = 0.01168
```

Whenever the empirical adequacy cannot be reject at some appropriate level of significance for a collection of models, we could further rely on information criteria to make the selection of the best candidate(s).

2.5. Model selection

Suppose that we want to investigate which out of several functional constraints—for instance, the normalized ("nealmon") or non-normalized ("almonp") exponential Almon lag polynomials, or with polynomial of order 2 or 3, and so on—are better suited in a MIDAS regression model of y on x and z (possibly different for each variable). Since the best maximum number of lags can differ with a functional constraint and/or variable/frequency, let us first define using **midasr** function `expand_weights_lags` the sets of potential models corresponding to each explanatory variable as follows

```
set.x<-expand_weights_lags(weights=c("nealmon","almonp"),
                           from=0,to=c(5,10),m=1,
                           start=list(nealmon=c(1,-1),almonp=c(1,0,0)))
set.z <- expand_weights_lags(c("nealmon","nealmon"),
                             0,c(10,20),1,
                             start=list(nealmon=c(1,-1),nealmon=c(1,-1,0)))
```

Here, for each variable, vector (or list) `weights` defines the potential restrictions to be considered and a list `start` gives the appropriate starting values defining implicitly the number of hyper-parameters per a function.

The potential lag structures are given by the following ranges of high-frequency lags: from [`from`; `m`$*$ min(`to`)] to [`from`; `m`$*$ max(`to`)]. When aggregates-based modelling is involved using `amweights` in `midas_r`, `m` can be set to the frequency ratio which ensures that the considered models (lag structures) are multiples of it. Otherwise, we would recommend to operate with high-frequency lag structures without changing the default value $m = 1$.

Then, the set of potential models is defined as all possible different combinations of functions and lag structures with a corresponding set of starting values. A simple example bellow illustrates the result in order to reveal the underlying structure, which, besides the understanding of it, is otherwise not needed for a user.

```
expand_weights_lags(weights=c("nealmon","nbeta"),
                    from=1,to=c(2,3),m=1,
                    start=list(nealmon=c(1,-1),nbeta=rep(0.5,3)))

##   weights lags           starts
## 1 nealmon  1:2         c(1, -1)
## 2 nealmon  1:3         c(1, -1)
## 3   nbeta  1:2 c(0.5, 0.5, 0.5)
## 4   nbeta  1:3 c(0.5, 0.5, 0.5)
```

Given the sets of potential specifications for each variable as defined above, the estimation of all the models is performed by

```
eqs.ic <- midas_r_ic_table(y ~ trend + mls(x, 0, m = 4) + fmls(z,
    0, m = 12), table = list(z = set.z, x = set.x))
```

The function `midas_r_ic_table` returns a summary table of all models together with the corresponding values of the usual information criteria and the empirical sizes of adequacy testing of functional restrictions of parameters. The result of derivative tests and the convergence status of optimisation function is also returned.

The summary table is a `data.frame` where each row corresponds to candidate model, so this table can be manipulated in the usual R way. The table can be accessed as `table` element of the list returned by `midas_r_ic_table`. The list of fitted `midas_r` objects of all candidate models can be accessed as `candlist` element. It is possible to inspect each candidate model and fine-tune its convergence if necessary.

```
eqs.ic$candlist[[5]] <- midas_r(eqs.ic$candlist[[5]], Ofunction = "nls")
```

The summary table can be recalculated by simply passing the fine-tuned list in to the function `midas_r_ic_table` again.

```
midas_r_ic_table(eqs.ic)
```

It should be pointed out that there is no need to provide the weighting function nor a specific lag order in the `mls` functions in such a case, since they are defined by the respective potential sets of models under option `table` in function `midas_r_ic_table`. Any provided values with `mls` (or other similar functions) are over-written by those defined in `table`.

Finally, the best model in terms of a selected information criterion in a restricted or unrestricted model then is simply obtained by using

```
modsel(eqs.ic, IC = "AIC", type = "restricted")
```

which also prints the usual summary statistics as well as the testing of adequacy of the applied functional restriction using, by default, the `hAh.test`. A word of caution is needed here to remind that, as it is usual, the empirical size of a model corresponding to a complex model-selection procedure might not correspond directly to a nominal one of a single-step estimation.

2.6. Forecasting

Conditional forecasting (with confidence intervals, etc) using unrestricted U-MIDAS models that are estimated using `lm` can be performed using standard R functions e.g. `predict.lm`. Conditional point prediction given a specific model is also possible relying on a standard `predict` function.

The function `predict` works in a similar manner to `predict.lm`. It takes the new data, transforms it to apropriate matrix and multiplies it by the coefficients. Suppose we want to produce the forecast $\hat{y}_{T+1|T}$ for the model (9). To produce this forecast we need the data $x_{4(T+1)}, ..., x_{4T-3}$ and $z_{12(T+1)}, ..., z_{12T-4}$. It would be tedious to calculate precise amount of

data each time we want to perform forecasting exercise. To alleviate this problem package **midasr** provides the function `forecast`. This function assumes that the model was estimated with the data up to low frequency index T. It then assumes that the new data is the data after the low frequency T and then calculates appropriate forecast. For example suppose that we have new data for one low frequency period for the model (9). Here is how the forecast for one period would look like:

```
newx <- rnorm(4)
newz <- rnorm(12)
forecast(eq.rb, newdata = list(x = newx, z = newz, trend = 251))

## [1] 28.29
```

In MIDAS literature it is more common to estimate models which do not require new data for forecasting

$$y_{t+h} = 2 + 0.1t + \sum_{j=0}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=0}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_{t+h},$$

where h is the desired forecasting horizon. This model can be rewritten as

$$y_t = 2 + 0.1t + \sum_{j=4h}^{7} \beta_j^{(1)} x_{4t-j} + \sum_{j=12h}^{16} \beta_j^{(2)} z_{12t-j} + \varepsilon_t,$$

Then it can be estimated using `midas_r`. For such model we can get forecasts $\hat{y}_{T+h|T}, ..., \hat{y}_{T+1|T}$ using the explanatory variable data up to low frequency index T. To get these forecasts using function `forecast` we need to supply `NA` values for explanatory variables. Here is the example for $h = 1$:

```
eq.f <- midas_r(y ~ trend + mls(x, 4 + 0:7, 4, nealmon) + mls(z,
    12 + 0:16, 12, nealmon), start = list(x = c(1, -0.5), z = c(2,
    0.5, -0.1)))
forecast(eq.f, newdata = list(x = rep(NA, 4), z = rep(NA, 12),
    trend = 251))

## [1] 27.2
```

Note that we still need to specify value for trend.

In addition, package **midasr** provides a general flexible environment for out-of-sample prediction, forecast combination, and precision evaluation of restricted MIDAS models using function `select_and_forecast`. If exact models were known for different forecasting horizons, it can also be used just to report various in- and out-of-sample prediction characteristics of the models. In the general case, it also performs an automatic selection of the best models for each forecasting horizon from a set of potential specifications defined by all combinations of functional restrictions and lag orders to be considered, and produces forecast combinations according to a specified forecast weighting scheme.

In general, the definition of potential models in function `select_and_forecast` is similar to that one used in the model selection analysis of the previous section. However, the key specificity is faced here due to the fact that different best specifications are most likely to be related with each low-frequency forecasting horizon $\ell = 0, 1, 2, \ldots$. Therefore a set of potential different models (parameter restriction functions and lag orders) to be considered for each horizon and variable needs to be defined among others.

Suppose that, as in the previous examples, we have variables x and z with frequency ratios $m_1 = 4$ and $m_2 = 12$, respectively. Suppose that we intend to consider forecasting of y up to three low-frequency periods $\ell \in \{1, 2, 3\}$ ahead. It should be noted that, in terms of high-frequency periods, they correspond to $\ell m_1 \in \{4, 8, 12\}$ for variable x, and $\ell m_2 \in \{12, 24, 36\}$ for variable z. Thus these variable-specific vectors define the lowest lags[4] of high-frequency period to be considered for each variable in the respective forecasting model (option `from` in function `select_and_forecast`). Suppose further that in all the models we want to consider specifications having not less than 10 high-frequency lags and not more than 15 for each variable. This defines a range up to which maximum high-frequency lag the potential models will be considered for each low-frequency horizon period $\ell \in \{1, 2, 3\}$. Hence, for each variable, three corresponding pairs $(\ell m_1 + 10, \ell m_1 + 15)$, $\ell \in \{1, 2, 3\}$ will define the upper bounds of ranges to be considered (option `to` in function `select_and_forecast`). For instance, for variable x, three pairs $(14, 19)$, $(18, 23)$, and $(22, 27)$ correspond to $\ell = 1, 2$, and 3 and together with that defined in option `from` (see `x=(4,8,12)`) imply that the following ranges of potential models will be under consideration for variable x:

$\ell = 1$: from $[4 - 14]$ to $[4 - 19]$,

$\ell = 2$: from $[8 - 18]$ to $[8 - 23]$,

$\ell = 3$: from $[12 - 22]$ to $[12 - 27]$.

The other options of function `select_and_forecast` do rather not require further explanation

```
cbfc<-select_and_forecast(y~trend+mls(x,0,4)+mls(z,0,12),
                          from=list(x=c(4,8,12),z=c(12,24,36)),
                          to=list(x=rbind(c(14,19),c(18,23),c(22,27)),
                                  z=rbind(c(22,27),c(34,39),c(46,51))),
                          insample=1:200,outsample=201:250,
                          weights=list(x=c("nealmon","almonp"),
                              z=c("nealmon","almonp")),
                          wstart=list(nealmon=rep(1,3),almonp=rep(1,3)),
                          IC="AIC",
                          seltype="restricted",
                          ftype="fixed",
                          measures=c("MSE","MAPE","MASE"),
                          fweights=c("EW","BICW","MSFE","DMSFE")
)
```

[4]Including lags smaller than that would imply that more information on explanatory variables is available and, in fact, $\ell - 1$ forecasting horizon is actually under consideration.

The names of weighting schemes are taken from MIDAS Matlab toolbox Ghysels (2013). Similarly forecasting using rolling and recursive model estimation samples defined therein Ghysels (2013) is supported by setting option `seltype="rolling"` or `seltype="recursive"`. Then, among others,

```
cbfc$accuracy$individual
cbfc$accuracy$average
```

report, respectively:

- the best forecasting equations (in terms of a specified criterion out of the above-defined potential specifications), and their in- and out-of-sample forecasting precision measures for each forecasting horizon;
- the out-of-sample precision of forecast combinations for each forecasting horizon.

The above example illustrated a general usage of function `select_and_forecast` including selection of best models. Now suppose that a user is only interested in evaluating a one step ahead forecasting performance of a given model. Suppose further that he/she a priori knows that the best specifications to be used for this forecasting horizon $\ell = 1$ is with

- `mls(x,4:12,4,nealmon)` with hyper-parameters `x=c(2,10,1,-0.1)` (the first one representing an impact parameter and the last three being the hyper-parameters of the normalized weighting function), and
- `mls(z,12:20,12,nealmon)` with hyper-parameters `z=c(-1,2,-0.1)` i.e. with one hyper-parameter less in the weighting function.

Given already preselected and evaluated models, user can use the function `average_forecast` to evaluate the forecasting perfomance. To use this function at first it is necessary to fit the model and then pass it to function `average_forecast` specifying the in-sample and out-of-sample data, accuracy measures and weighting scheme in a similar manner to `select_and_forecast`

```
mod1 <- midas_r(y ~ trend + mls(x, 4:14, 4, nealmon) + mls(z, 12:22, 12, nealmon),
                start=list(x=c(10,1,-0.1),z=c(2,-0.1)))
avgf <- average_forecast(list(mod1),
                         data=list(y=y,x=x,z=z,trend=trend),
                         insample=1:200,outsample=201:250,
                         type="fixed",
                         measures=c("MSE","MAPE","MASE"),
                         fweights=c("EW","BICW","MSFE","DMSFE"))
```

It should be also pointed out that the forecast combinations in function `select_and_forecast` are obtained only from the forecasts linked to different restriction functions on parameters. The forecasts related to different lag specifications are not combined, but the best lag order is chosen in terms of a given information criterion. If there is a need to get forecast

combinations for a group of models which the user selected using other criteria, the function `average_forecast` should be used in a manner outlined in the previous example.

3. Empirical illustrations

3.1. Forecasting GDP growth

We replicate the example provided in Ghysels (2013). In particular we run MIDAS regression to forecast quarterly GDP growth with monthly Employment growth. The forecasting equation is the following

$$y_{t+1} = \alpha + \rho y_t + \sum_{h=0}^{8} \theta_h x_{3t-h} + \varepsilon_t,$$

where y_t is the log difference of quarterly seasonally adjusted real US GDP and x_{3t} is the log difference of monthly total employment non-farms payroll. The data is taken from St. Louis FRED website.

First we load the data and perform necessary transformations.

```
library(quantmod)
gdp <- getSymbols("GDP", src = "FRED", auto.assign = FALSE)

payems <- getSymbols("PAYEMS", src = "FRED", auto.assign = FALSE)
y <- window(ts(gdp, start = c(1947, 1), frequency = 4), end = c(2011,
    2))
x <- window(ts(payems, start = c(1939, 1), frequency = 12), end = c(2011,
    7))
yg <- log(y/lag(y, -1)) * 100
xg <- log(x/lag(x, -1)) * 100
nx <- ts(c(NA, xg, NA, NA), start = start(x), frequency = 12)
ny <- ts(c(rep(NA, 33), yg, NA), start = start(x), frequency = 4)
```

The last two lines are needed to equalise the sample sizes, which are different in the original data. We simply add additional `NA` values at the beginning and the end of the data. The graphical representation of the data is shown in figure 3.1

To specify the model for `midas_r` function we rewrite it in the following equivalent form:

$$y_t = \alpha + \rho y_{t-1} + \sum_{h=3}^{11} \theta_h x_{3t-h} + \varepsilon_t,$$

As in Ghysels (2013) we restrict the estimation sample from the first quarter of 1985 to the first quarter of 2009. We evaluate the models with the Beta polynomial, Beta with non-zero and U-MIDAS weight specifications.

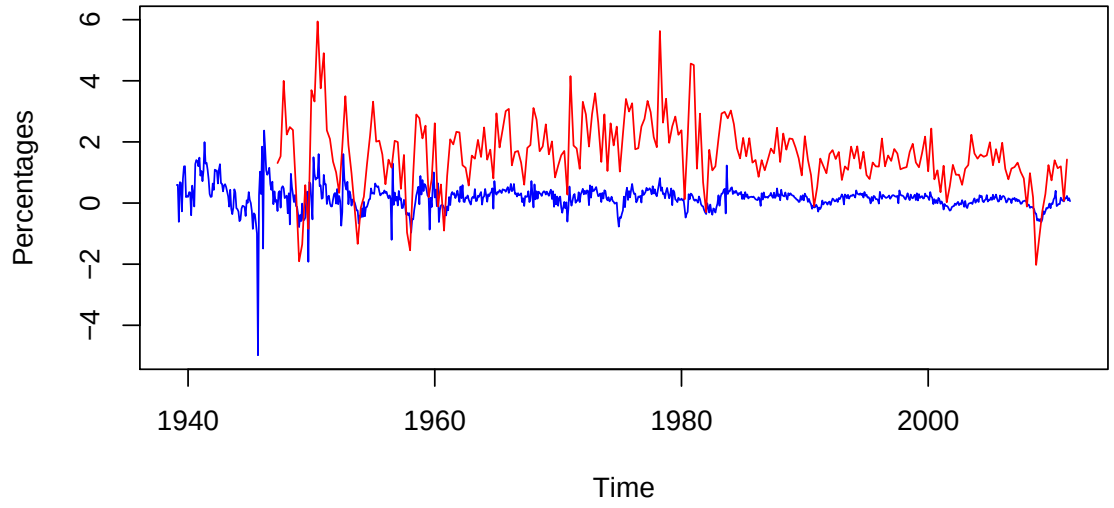

Figure 3: Quatertly GDP and Monthly Non-Farm Payroll Employment Growth Rate

```
xx <- window(nx, start = c(1985, 1), end = c(2009, 3))
yy <- window(ny, start = c(1985, 1), end = c(2009, 1))
beta0 <- midas_r(yy ~ mls(yy, 1, 1) + mls(xx, 3:11, 3, nbeta),
    start = list(xx = c(1.7, 1, 5)))
coef(beta0)

## (Intercept)          yy         xx1         xx2         xx3
##      0.8348      0.1054      2.5795      1.0188     13.1287

betan <- midas_r(yy ~ mls(yy, 1, 1) + mls(xx, 3:11, 3, nbetaMT),
    start = list(xx = c(2, 1, 5, 0)))
coef(betan)

## (Intercept)          yy         xx1         xx2         xx3         xx4
##     0.93868     0.06977     2.24646     0.98693     1.46542    -0.09264

um <- midas_r(yy ~ mls(yy, 1, 1) + mls(xx, 3:11, 3), start = NULL)
coef(um)

## (Intercept)          yy         xx1         xx2         xx3         xx4
##     0.93459     0.08329     1.97711     0.92310     0.44524    -0.21101
##         xx5         xx6         xx7         xx8         xx9
##     0.20813     1.20043    -0.50631    -0.67365    -1.24858
```

We proceed now with forecast evaluation.

```
fulldata <- list(xx=window(nx,start=c(1985,1),end=c(2011,6)),
                 yy=window(ny,start=c(1985,1),end=c(2011,2)))
insample <- 1:length(yy)
outsample <- (1:length(fulldata$yy))[-insample]

avgf<-average_forecast(list(beta0,betan,um),
                       data=fulldata,
                       insample=insample,
                       outsample=outsample)
sqrt(avgf$accuracy$individual$MSE.out.of.sample)

## [1] 0.5412 0.5002 0.4623
```

3.2. Forecasting realized volatility

For another demonstration we use package **midasr** to forecast the daily realized volatility. A simple model for forecasting the daily realized volatility was proposed by Corsi (2009). The model, heterogeneous autoregressive model of realized volatility, HAR-RV is defined as

$$RV_{t+1}^{(d)} = c + \beta^{(d)} RV_t^{(d)} + \beta^{(w)} RV_t^{w} + \beta^{(m)} RV_t^{(m)} + w_{t+1},$$

where RV_t is the daily realized volatility and RV_t^w and $RV_t^{(m)}$ are weekly and monthly averages:

$$\begin{aligned} RV_t^{(w)} &= \frac{1}{5}\left(RV_t^{(w)} + RV_{t-1}^{(w)} + ... + RV_{t-4}^{(w)}\right) \\ RV_t^{(m)} &= \frac{1}{20}\left(RV_t^{(w)} + RV_{t-1}^{(w)} + ... + RV_{t-19}^{(w)}\right), \end{aligned}$$

where we assume that week has 5 days, ant the month has 4 weeks. This model is a special case of MIDAS regression:

$$RV_{t+1}^{(d)} = c + \sum_{h=0}^{19} \beta_h RV_{t-h}^{(d)} + w_{t+1},$$

where

$$\beta_h = \begin{cases} \beta^{(d)} + \frac{1}{5}\beta^{(w)} + \frac{1}{20}\beta^{(m)}, & \text{for } h = 0, \\ \frac{1}{5}\beta^{(w)} + \frac{1}{20}\beta^{(m)}, & \text{for } h = 1, .., 4, \\ \frac{1}{20}\beta^{(m)}, & \text{for } h = 5, ..., 19. \end{cases}$$

The corresponding R code is the following

```
harstep <- function(p, d, m) {
    if (d != 20)
        stop("HAR(3)-RV process requires 20 lags")
    out <- rep(0, 20)
    out[1] <- p[1] + p[2]/5 + p[3]/20
    out[2:5] <- p[2]/5 + p[3]/20
    out[6:20] <- p[3]/20
    out
}
```

For empirical demonstration we use the realized variance data on stock indexes provided by Oxford-Man Institute of Quantitative Finance.

We estimate this model for annualized realized volatility of S&P500 index

```
load("data/spx2.RData")
spx2.rvol <- na.omit(100 * sqrt(252 * SPX2))
mh <- midas_r(rv ~ mls(rv, 1:20, 1, harstep), data = list(rv = spx2.rvol),
    start = list(rv = c(1, 1, 1)))
summary(mh)

##
##  Formula rv ~ mls(rv, 1:20, 1, harstep)
##
##  Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.8300     0.3645    2.28  0.02283 *
## rv1           0.3407     0.0446    7.63  3.0e-14 ***
## rv2           0.4113     0.0693    5.93  3.3e-09 ***
## rv3           0.1932     0.0508    3.80  0.00015 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Residual standard error: 5.56 on 3434 degrees of freedom
```

As a comparison we estimate this model with normalized exponential Almon weights

```
mr <- midas_r(rv ~ mls(rv, 1:20, 1, nealmon), data = list(rv = spx2.rvol),
    start = list(rv = c(0, 0, 0)), user.gradient = TRUE)
summary(mr)

##
##  Formula rv ~ mls(rv, 1:20, 1, nealmon)
##
##  Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.89832    0.40084    2.24    0.025 *
```

```
## rv1          0.94344    0.02845   33.16  < 2e-16 ***
## rv2         -0.78211    0.10196   -7.67  2.3e-14 ***
## rv3          0.02960    0.00597    4.96  7.4e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Residual standard error: 5.69 on 3064 degrees of freedom
```

We can test which of these restrictions is a correct one, by using heteroscedasticity and autocorrelation robust weight specification test `hAhr.test`.

```
bm <- modsel(mtb)

## Error:  objet 'mtb' introuvable
```

The HAC robust version of `hAh.test` again does not reject the null hypothesis that the exponential Almon lag spefication is suitable for this data.

```
## Error:  objet 'bm' introuvable
```

We can look into how both models perform at forecasting, by doing a rolling forecast with 1000 observation window. For comparison we also calculate the forecasts for unrestricted AR(20) model.

```
ar20 <- midas_r(rv~mls(rv,1:20,1),data=list(rv=spx2.rvol),start=NULL)
forc <- average_forecast(list(ar20,mh,bm),
                         data=list(rv=spx2.rvol),
                         insample=1:1000,outsample=1001:1100,
                         type="rolling",showprogress=FALSE)

## Error:  objet 'bm' introuvable

forc$accuracy$individual

## Error:  objet 'forc' introuvable
```

We see that exponential Almon lag model slightly outperforms the HAR-RV model and both models outperform AR(20) model.

4. Final remarks

Only a part of the available functionality of the discussed functions of package **midasr** was revealed. As it is usual in R, much more information on the resulting objects than characterized above and all the information on the package-specific functions can be reached using generic functions `objects` and `?`, respectively. Furthermore, in order to save the space, the coding

examples provided above are almost always presented with minimal accompanying output obtained after running the code. The package page contains all the codes and complete output together with some additional illustration of the functionality of the package. Other information with a list of the functions and a number of demonstration codes is accessible using the usual `??midasr`.

5. Appendix

The figure 1 was created using Monte-Carlo simulation. The following DGP was used

$$y_t = 2 + 0.1t + \sum_{h=0}^{16} \beta_h z_{12t-h} + u_t,\ z_\tau \sim N(0, \sigma^2),\ u_t \sim N(0, \sigma^2)$$

The coefficients β_h were chosen to come from normalized exponential Almon polynomial restriction:

```
nealmon(p = c(2, 0.5, -0.1), d = 17)
```

The data for this DGP was generated for low frequency sample sizes 50, 100, 200, 300, 500, 750 and 1000. For each sample size additional out-of-sample data set the size quarter of the in-sample data set was generated. Three MIDAS regression models were estimated using in-sample data set: unrestricted MIDAS, the restriction of the DGP and the incorrect restriction of Almon polynomial. The forecast was calculated using the out-of-sample dataset. The euclidean distance between the model coefficients and the coefficients of DGP was recorded together with mean squared error of the forecast.

This process was repeated 1000 times. The points in the figure are the averages of the replications. Full code can be found in package **midasr** website.

References

Bilinskas, B., and Zemlys, V. (2013). Testing the Functional Constraints on Parameters in Regression Models with Cointegrated Variables of Different Frequency. *Submitted, available upon a request.*

Clements, M.P., and Galvão, A.B. (1991). Macroeconomic Forecasting With Mixed-Frequency Data: Forecasting Output Growth in the United States. *Journal of Business* & Economic Statistics **26(4)**: 546–554.

Foroni, C., Marcellino, M., and Schumacher, C. (2012). U-MIDAS: MIDAS regressions with unrestricted lag polynomials. CEPR Discussion Papers No 8828.

Ghysels, E., Santa-Clara, P., and Valkanov, R. (2002). The MIDAS touch: Mixed data sampling regression models. Working paper, UNC and UCLA.

Ghysels, E., Sinko, A., and Valkanov, R. (2007). MIDAS regressions: Further results and new directions*. *Econometric Reviews* **26**: 53–90.

Kvedaras, V., and Račkauskas, A. (2010). Regression models with variables of different frequencies: The case of a fixed frequency ratio. *Oxford Bulletin of Economics and Statistics* **72**: 600–620.

Kvedaras, V., Bilinskas, B. and Zemlys, V. (2013). Testing the functional constraints on parameters in cointegrated MIDAS regressions. *Work in progress.*

Kvedaras, V., and Zemlys, V. (2012). Testing the functional constraints on parameters in regressions with variables of different frequency. *Economics Letters* **116**: 250-254.

Kvedaras, V., and Zemlys, V. (2013). The statistical content and empirical testing of the MIDAS restrictions *Submitted, available upon a request.*

Ghysels, E. (2013). Matlab Toolbox for Mixed Sampling Frequency Data Analysis using MIDAS Regression Models. Unpublished manuscript. July 25, 2013 (Version 7).

Zeileis, A. (2004). Econometric Computing with HC and HAC Covariance Matrix Estimators. *Journal of Statistical Software* **11**: 1–17.

Affiliation:

Vaidotas Zemlys
Department of Econometric Analysis
Faculty of Mathematics and Informatics
Vilnius University
Naugarduko g. 24, Vilnius, Lithuania
E-mail: vaidotas.zemlys@mif.vu.lt
URL:http://mif.vu.lt/~zemlys/