MIDAS in gretl

Allin Cottrell, Jack Lucchetti

September 19, 2017

These notes describes the state of play with regard to MIDAS (Mixed Data Sampling—see Ghysels *et al.*, 2004; Ghysels, 2015; Armesto *et al.*, 2010) in gretl as of the 2017a release (March 2017). The points covered in these notes are not yet elaborated in the formal gretl documentation, but the relevant new commands and functions are described in the current *Gretl Command Reference*.

1 Handling data of more than one frequency

The essential move in MIDAS is estimation of models including one or more independent variables observed at a higher frequency than the dependent variable. Since a gretl dataset formally handles only a single data frequency it may seem that we have a problem. However, we have adopted a straightforward solution: a higher frequency series x_H is represented by a set of m series, each holding the value of x_H in a sub-period of the "base" (lower-frequency) period (where m is the ratio of the higher frequency to the lower).

This is most easily understood by means of an example. Suppose our base frequency is quarterly and we wish to include a monthly series in the analysis. Then a relevant fragment of the gretl dataset might look as shown in Table 1. Here, `gdpc96` is a quarterly series while `indpro` is monthly, so $m = 12/4 = 3$ and the per-month values of `indpro` are identified by the suffix `_m`n, $n = 3, 2, 1$.

	gdpc96	indpro_m3	indpro_m2	indpro_m1
1947:1	1934.47	14.3650	14.2811	14.1973
1947:2	1932.28	14.3091	14.3091	14.2532
1947:3	1930.31	14.4209	14.3091	14.2253
1947:4	1960.70	14.8121	14.7562	14.5606
1948:1	1989.54	14.7563	14.9240	14.8960
1948:2	2021.85	15.2313	15.0357	14.7842

Table 1: A slice of MIDAS data

To recover the actual monthly time series for `indpro` one must read the three relevant series right-to-left by row. At first glance this may seem perverse, but in fact it is the most convenient setup for MIDAS analysis. In such models, the high-frequency variables are represented by lists of lags, and of course in econometrics it is standard to give the most recent lag first $(x_{t-1}, x_{t-2}, \ldots)$.

One can construct such a dataset manually from "raw" sources using hansl's matrix-handling methods or the `join` command (see Appendix A for illustrations), but we have added native

support for the common cases shown below.

base frequency	*higher frequency*
annual	quarterly or monthly
quarterly	monthly or daily
monthly	daily

The examples below mostly pertain to the case of quarterly plus monthly data. Appendix B has details on our handling of daily data.

The new native methods support creation of a MIDAS-ready dataset in either of two ways: by selective importation of series from a database, or by creating two datasets of different frequencies then merging them.

Importation from a database

Here's a simple example, in which we draw from the `fedstl` (St Louis Fed) database which is supplied in the gretl distribution:

```
clear
open fedstl.bin
data gdpc96
data indpro --compact=spread
store gdp_indpro.gdt
```

Since `gdpc96` is a quarterly series, its importation via the `data` command establishes a quarterly dataset. Then the MIDAS work is done by the option `--compact=spread` for the second invocation of `data`. This "spreads" the series `indpro` — which is monthly at source — into three quarterly series, exactly as shown in Table 1.

Merging two datasets

In this case we consider an Excel file provided by Eric Ghysels in his MIDAS Matlab Toolbox,[1] namely `mydata.xlsx`. This contains quarterly real GDP in Sheet1 and monthly non-farm payroll employment in Sheet2. A hansl script to build a MIDAS-style file named `gdp_midas.gdt` is shown in Listing 1.

Note that both series are simply named `VALUE` in the source file, so we use gretl's `rename` command to set distinct and meaningful names. The heavy lifting is done here by the line

```
dataset compact 4 spread
```

which tells gretl to compact an entire dataset (in this case, as it happens, just containing one series) to quarterly frequency using the "spread" method. Once this is done, it is straightforward to `append` the compacted data to the quarterly GDP dataset.

We will put this dataset to use in subsequent sections. Note that it is provided in the gretl package (`gdp_midas`, which you can find under the Gretl tab in the practice datafiles window in the gretl GUI).

[1] See http://www.unc.edu/~eghysels/ for links.

Listing 1: Building a gretl MIDAS dataset via merger

```
# sheet 2 contains monthly employment data
open MIDASv2.0/mydata.xlsx --sheet=2
rename VALUE payems
dataset compact 4 spread
# limit to the sample range of the GDP data
smpl 1947:1 2011:2
setinfo payems_m3 --description="Non-farm payroll employment, month 3 of quarter"
setinfo payems_m2 --description="Non-farm payroll employment, month 2 of quarter"
setinfo payems_m1 --description="Non-farm payroll employment, month 1 of quarter"
store payroll_midas.gdt

# sheet 1 contains quarterly GDP data
open MIDASv2.0/mydata.xlsx --sheet=1
rename VALUE qgdp
setinfo qgdp --description="Real quarterly US GDP"
append payroll_midas.gdt
store gdp_midas.gdt
```

2 The notion of a "MIDAS list"

In the following two sections we'll describe functions that (rather easily) do the right thing if you wish to create lists of lags or first differences of high-frequency series. However, we should first be clear about the correct domain for such functions, since they could produce the most diabolical mash-up of your data if applied to the wrong sort of list argument—for instance, a regular list containing distinct series, all observed at the "base frequency" of the dataset.

So let us define a **MIDAS list**: this is a list of m series holding per-period values of a *single* high-frequency series, arranged in the order of most recent first, as illustrated above. Given the dataset shown in Table 1, an example of a correctly formulated MIDAS list would be

```
list INDPRO = indpro_m3 indpro_m2 indpro_m1
```

Or, since the monthly observations are already in the required order, we could define the list by means of a "wildcard":

```
list INDPRO = indpro_m*
```

Having created such a list, one can use the `setinfo` command to tell gretl that it's a *bona fide* MIDAS list:

```
setinfo INDPRO --midas
```

This will spare you some warnings that gretl would otherwise emit when you call some of the functions described below. This step should not be necessary, however, if the series in question are the product of a `compact` operation with the `spread` parameter.

Inspecting high-frequency data

The layout of high-frequency data shown in Table 1 is convenient for running regressions, but not very convenient for inspecting and checking such data. We therefore provide some methods for displaying MIDAS data at their "natural" frequency. Figure 1 shows the gretl main window with the `gdp_midas` dataset loaded, along with the menu that pops up if you right-click with the `payems` series highlighted. The items "Display values" and "Time series plot" show the data on their original monthly calendar, while the "Display components" item shows the three component series on a quarterly calendar, as in Table 1.

These methods are also available via the command line. For example, the commands

```
list PAYEMS = payems_*
print PAYEMS --byobs --midas
hfplot PAYEMS --with-lines --output=display
```

produce a monthly printout of the payroll employment data, followed by a monthly time-series plot. (See section 7 for more on `hfplot`.)

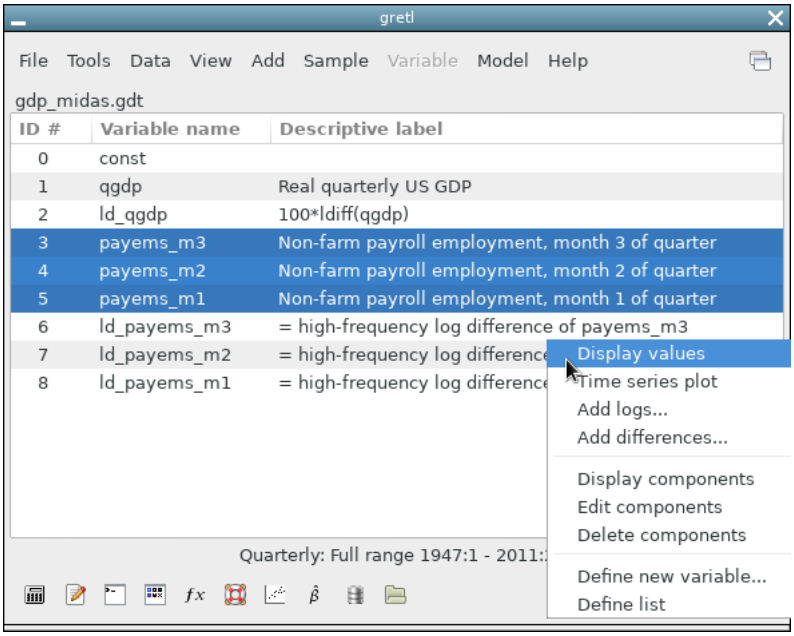

Figure 1: MIDAS data menu

3 High-frequency lag lists

A basic requirement of MIDAS is the creation of lists of high-frequency lags for use on the right-hand side of a regression specification. This is possible, but not very convenient, using the long-standing behavior of gretl's `lags` function; it is made easier by a dedicated variant of that function described below.

For illustration we'll consider an example presented in Ghysels' Matlab implementation of MIDAS: this uses 9 monthly lags of payroll employment, starting at lag 3, in a model for quarterly GDP. The estimation period for this model starts in 1985Q1. At this observation, the stipulation that we start at lag 3 means that the first (most recent) lag is employment for

October 1984,[2] and the 9-lag window means that we need to include monthly lags back to February 1984. Let the per-month employment series be called `x_m3`, `x_m2` and `x_m1`, and let (quarterly) lags be represented by `(-1)`, `(-2)` and so on. Then the terms we want are (reading left-to-right by row):

```
      .          .       x_m1(-1)
x_m3(-2)   x_m2(-2)   x_m1(-2)
x_m3(-3)   x_m2(-3)   x_m1(-3)
x_m3(-4)   x_m2(-4)       .
```

We could construct such a list in gretl using the following standard syntax. (Note that the third argument of 1 to `lags` below tells gretl that we want the terms ordered "by lag" rather than "by variable"; this is required to respect the order of the terms shown above.)

```
list X = x_m*
# create lags for 4 quarters, "by lag"
list XL = lags(4,X,1)
# convert the list to a matrix
matrix tmp = XL
# trim off the first two elements, and the last
tmp = tmp[3:11]
# and convert back to a list
XL = tmp
```

However, the following specialized syntax is more convenient:

```
list X = x_m*
setinfo X --midas
# create high-frequency lags 3 to 11
list XL = hflags(3, 11, X)
```

In the case of `hflags` the length of the list given as the third argument defines the "compaction ratio" ($m = 3$ in this example); we can (in fact, must) specify the lags we want in high-frequency terms; and ordering of the generated series by lag is automatic.

Word to the wise: do not use `hflags` on anything other than a **MIDAS list** as defined in section 2, unless perhaps you have some special project in mind and really know what you are doing.

Leads and nowcasting

Before leaving the topic of lags, it is worth commenting on the question of leads and so-called "nowcasting"—that is, prediction of the current value of a lower-frequency variable before its measurement becomes available.

In a regular dataset where all series are of the same frequency, lag 1 means the observation from the previous period, lag 0 is equivalent to the current observation, and lag -1 (or lead 1) is the observation for the next period into the relative future.

When considering high-frequency lags in the MIDAS context, however, there is no uniquely determined high-frequency sub-period which is temporally coincident with a given low-frequency

[2] That is what Ghysels means, but see the sub-section on "Leads and nowcasting" below for a possible ambiguity in this regard.

period. The placement of high-frequency lag 0 therefore has to be a matter of convention. Unfortunately, there are two incompatible conventions in currently available MIDAS software, as follows.

- High-frequency lag 0 corresponds to the *first* sub-period within the current low-frequency period. This is what we find in Eric Ghysels' MIDAS Matlab Toolbox; it's also clearly stated and explained in Armesto *et al.* (2010).
- High-frequency lag 0 corresponds to the *last* sub-period in the current low-frequency period. This convention is employed in the `midasr` package for `R`.[3]

Consider, for example, the quarterly/monthly case. In Matlab, high-frequency (HF) lag 0 is the first month of the current quarter, HF lag 1 is the last month of the prior quarter, and so on. In `midasr`, however, HF lag 0 is the last month of the current quarter, HF lag 1 the middle month of the quarter, and HF lag 3 is the first one to take you "back in time" relative to the start of the current quarter, namely to the last month of the prior quarter.

In gretl we have chosen to employ the first of these conventions. So Lag 1 points to the most recent sub-period in the previous base-frequency period, lag 0 points to the first sub-period in the current period, and lag -1 to the second sub-period within the current period. Continuing with the quarterly/monthly case, monthly observations for lags 0 and -1 are likely to become available before a measurement for the quarterly variable is published (possibly also a monthly value for lag -2). The first "truly future" lead does not occur until lag -3.

The `hflags` function supports negative lags. Suppose one wanted to use 9 lags of a high-frequency variable, $-1, 0, 1, \ldots, 7$, for nowcasting. Given a suitable MIDAS list, `X`, the following would do the job:

```
list XLnow = hflags(-1, 7, X)
```

This means that one could generate a forecast for the current low-frequency period (which is not yet completed and for which no observation is available) using data from two sub-periods into the low-frequency period (e.g. the first two months of the quarter).

4 High-frequency first differences

When working with non-stationary data one may wish to take first differences, and in the MIDAS context that probably means high-frequency differences of the high-frequency data. Note that the ordinary gretl functions `diff` and `ldiff` will *not* do what is wanted for series such as `indpro`, as shown in Table 1: these functions will give per-month *quarterly* differences of the data (month 3 of the current quarter minus month 3 of the previous quarter, and so on).

To get the desired result one could create the differences before compacting the high-frequency data but this may not be convenient, and it's not compatible with the method of constructing a MIDAS dataset shown in section 1. The alternative is to employ the specialized differencing function `hfdiff`. This takes one required argument, a **MIDAS list** as defined in section 2. A second, optional argument is a scalar multiplier (with default value 1.0); this permits scaling the output series by a constant. There's also an `hfldiff` function for creating high-frequency log differences; this has the same syntax as `hfdiff`.

So for example, the following creates a list of high-frequency percentage changes (100 times log-difference) then a list of high-frequency lags of the changes.

[3] See http://cran.r-project.org/web/packages/midasr/, and for documentation https://github.com/mpiktas/midasr-user-guide/raw/master/midasr-user-guide.pdf.

```
list X = indpro_*
setinfo X --midas
list dX = hfldiff(X, 100)
list dXL = hflags(3, 11, dX)
```

If you only need the series in the list dXL, however, you can nest these two function calls:

```
list dXL = hflags(3, 11, hfldiff(X, 100))
```

5 Parsimonious parameterizations

The simplest MIDAS regression specification—known as "unrestricted MIDAS" or U-MIDAS—simply includes p lags of a high-frequency regressor, each with its own parameter to be estimated (which can be done via OLS). It is more common, however, to enforce parsimony by making the individual coefficients on lagged high-frequency terms a function of a relatively small number of hyperparameters. This presents a couple of computational questions: how to calculate the per-lag coefficients given the values of the hyperparameters, and how best to estimate the value of the hyperparameters?

Hansl is functional enough to allow a savvy user to address these questions from scratch, but of course it's helpful to have some built-in high-level functionality. At present gretl can handle natively four commonly used parameterizations: normalized exponential Almon, normalized beta (with or without a zero last coefficient) and plain (non-normalized) Almon polynomial. The Almon variants take one or more parameters (two being a common choice), and the beta takes either two or three parameters.[4] All are handled by the functions `mweights` and `mgradient`. These functions work as follows.

- `mweights` takes three arguments: the number of lags required (p), the k-vector of hyperparameters (θ), and an integer code or string indicating the method (see Table 2). It returns a p-vector containing the coefficients.
- `mgradient` takes three arguments, just like `mweights`. However, this function returns a $p \times k$ matrix holding the (analytical) gradient of the p coefficients or weights with respect to the k elements of θ.

Parameterization	*code*	*string*
Normalized exponential Almon	1	`"nealmon"`
Normalized beta, zero last lag	2	`"beta0"`
Normalized beta, non-zero last lag	3	`"betan"`
Almon polynomial	4	`"almonp"`

Table 2: MIDAS parameterizations

An additional function is provided for convenience: it is named `mlincomb` and it combines `mweights` with the long-standing `lincomb` function, which takes a list (of series) argument followed by a vector of coefficients and produces a series result, namely a linear combination of the elements of the list. If we have a suitable list `L` available, we can do, for example,

```
series foo = mlincomb(L, theta, "beta0")
```

[4] Two is the standard case; see Appendix C for details.

This is equivalent to

```
series foo = lincomb(L, mweights(nelem(L), theta, "beta0"))
```

but saves a little typing and some CPU cycles.

6 Estimating MIDAS models

Gretl offers a dedicated command, `midasreg`, for estimation of MIDAS models. (There's a corresponding item, MIDAS, under the Time series section of the Model menu in the gretl GUI.) We begin by discussing that, then move on to possibilities for defining your own estimator.

The syntax of `midasreg` looks like this:

`midasreg` *depvar xlist* ; *midas-terms* [*options*]

The *depvar* slot takes the name (or series ID number) of the dependent variable, and *xlist* is the list of regressors that are observed at the same frequency as the dependent variable; this list may contain lags of the dependent variable. The *midas-terms* slot accepts one or more specification(s) for high-frequency terms. Each of these specifications must conform to one or other of the following patterns:

1. `mds(`*mlist*`,` *minlag*`,` *maxlag*`,` *type*`,` *theta*`)`
2. `mdsl(`*llist*`,` *type*`,` *theta*`)`

In case 1 *mlist* must be a **MIDAS list**, as defined above, which contains a full set of per-period series (but no lags). Lags will be generated automatically, governed by the *minlag* and *maxlag* (integer) arguments, which may be given as numerical values or the names of predefined scalar variables. The integer (or string) *type* argument represents the type of parameterization; in addition to the values 1 to 4 defined in Table 2 a value of 0 (or the string `"umidas"`) indicates unrestricted MIDAS.

In case 2 *llist* is assumed to be a list that already contains the required set of high-frequency lags—as may be obtained via the `hflags` function described in section 3—hence *minlag* and *maxlag* are not wanted.

The final *theta* argument is optional in most cases (implying an automatic initialization of the hyperparameters). If this argument is given it must take one of the following forms:

1. The name of a matrix (vector) holding initial values for the hyperparameters, or a simple expression which defines a matrix using scalars, such as `{1, 5}`.
2. The keyword `null`, indicating that an automatic initialization should be used (as happens when this argument is omitted).
3. An integer value (in numerical form), indicating how many hyperparameters should be used (which again calls for automatic initialization).

The third of these forms is required if you want automatic initialization in the Almon polynomial case, since we need to know how many terms you wish to include. (In the normalized exponential Almon case we default to the usual two hyperparameters if *theta* is omitted or given as `null`.)

The `midasreg` syntax allows the user to specify multiple high-frequency predictors, if wanted: these can have different lag specifications, different parameterizations and/or different frequencies.

The options accepted by `midasreg` include `--quiet` (suppress printed output), `--verbose` (show detail of iterations, if applicable) and `--robust` (use a HAC estimator of the Newey–West type in computing standard errors). Two additional specialized options are described below.

Examples of usage

Suppose we have a dependent variable named `dy` and a MIDAS list named `dX`, and we wish to run a MIDAS regression using one lag of the dependent variable and high-frequency lags 1 to 10 of the series in `dX`. The following will produce U-MIDAS estimates:

```
midasreg dy const dy(-1) ; mds(dX, 1, 10, 0)
```

The next lines will produce estimates for the normalized exponential Almon parameterization with two coefficients, both initialized to zero:

```
midasreg dy const dy(-1) ; mds(dX, 1, 10, "nealmon", {0,0})
```

In the examples above, the required lags will be added to the dataset automatically then deleted after use. If you are estimating several models using a single set of MIDAS lags it is more efficient to create the lags once and use the `mdsl` specifier. For example, the following estimates three variant parameterizations (exponential Almon, beta with zero last lag, and beta with non-zero last lag) on the same data:

```
list dXL = hflags(1, 10, dX)
midasreg dy 0 dy(-1) ; mdsl(dXL, "nealmon", {0,0})
midasreg dy 0 dy(-1) ; mdsl(dXL, "beta0", {1,5})
midasreg dy 0 dy(-1) ; mdsl(dXL, "betan", {1,1,0})
```

Any additional MIDAS terms should be separated by spaces, as in

```
midasreg dy const dy(-1) ; mds(dX,1,9,1,theta1) mds(Z,1,6,3,theta2)
```

Replication exercise

We give a substantive illustration of `midasreg` in Listing 2. This replicates the first practical example discussed by Ghysels in the user's guide titled *MIDAS Matlab Toolbox*,[5] The dependent variable is the quarterly log-difference of real GDP, named `dy` in our script. The independent variables are the first lag of `dy` and monthly lags 3 to 11 of the monthly log-difference of non-farm payroll employment (named `dXL` in our script). Formally, the model may be written as

$$y_t = \alpha + \beta y_{t-1} + \gamma W(x_{\tau-3}, x_{\tau-4}, \ldots, x_{\tau-11}; \theta) + \varepsilon_t$$

where $W(\cdot)$ is the weighting function associated with a given MIDAS specification, θ is a vector of hyperparameters, and τ represents "high-frequency time." In the case of the non-normalized Almon polynomial the γ coefficient is identically 1.0 and is omitted; and in the U-MIDAS case the model comes down to

$$y_t = \alpha + \beta y_{t-1} + \sum_{i=1}^{9} \delta_i x_{\tau-i-2} + \varepsilon_t$$

[5] See Ghysels (2015). This document announces itself as Version 2.0 of the guide and is dated November 1, 2015. The example we're looking at appears on pages 24–26; the associated Matlab code can be found in the program `appADLMIDAS1.m`.

The script exercises all five of the parameterizations mentioned above,[6] and in each case the results of 9 pseudo-out-of-sample forecasts are recorded so that their Root Mean Square Errors can be compared.

The data file used in the replication, `gdp_midas.gdt`, was contructed as described in section 1 (and as noted there, it is included in the current gretl package). Part of the output from the replication script is shown in Listing 3. The y coefficient is labeled `HF_slope` in the gretl output.

For reference, output from Matlab (version R2016a for Linux) is available at http://gretl.sourceforge.net/midas/matlab_output.txt. For the most part (in respect of regression coefficients and auxiliary statistics such as R^2 and forecast RMSEs), gretl's output agrees with that of Matlab to the extent that one can reasonably expect on nonlinear problems — that is, to at least 4 significant digits in all but a few instances.[7] Standard errors are not quite so close across the two programs, particularly for the hyperparameters of the beta and exponential Almon functions. We show these in Table 3.

	2-param beta		3-param beta		Exp Almon	
	Matlab	gretl	Matlab	gretl	Matlab	gretl
const	0.135	0.140	0.143	0.146	0.135	0.140
dy(-1)	0.116	0.118	0.116	0.119	0.116	0.119
HF slope	0.559	0.575	0.566	0.582	0.562	0.575
θ_1	0.067	0.106	0.022	0.027	2.695	6.263
θ_2	9.662	17.140	1.884	2.934	0.586	1.655
θ_3			0.022	0.027		

Table 3: Comparison of standard errors from MIDAS regressions

Differences of this order are not unexpected, however, when different methods are used to calculate the covariance matrix for a nonlinear regression. The Matlab standard errors are based on a numerical approximation to the Hessian at convergence, while those produced by gretl are based on a Gauss–Newton Regression, as discussed and recommended in Davidson and MacKinnon (2004, chapter 6).

Underlying methods

The `midasreg` command calls one of several possible estimation methods in the background, depending on the MIDAS specification(s). As shown in Listing 3, this is flagged in a line of output immediately preceding the "`Dependent variable`" line. If the only specification type is U-MIDAS, the method is OLS. Otherwise it is one of three variants of Nonlinear Least Squares.

- Levenberg–Marquardt. This is the back-end for gretl's `nls` command.
- L-BFGS-B with conditional OLS. L-BFGS is a "limited memory" version of the BFGS optimizer and the trailing "-B" means that it supports bounds on the parameters, which is useful for reasons given below.
- Golden Section search with conditional OLS. This is a line search method, used only when there is a just a single hyperparameter to estimate.

[6] The Matlab program includes an additional parameterization not supported by gretl, namely a step-function.

[7] Nonlinear results, even for a given software package, are subject to slight variation depending on the compiler used and the exact versions of supporting numerical libraries.

Listing 2: Script to replicate results given by Ghysels

```
set verbose off
open gdp_midas.gdt --quiet

# form the dependent variable
series dy = 100 * ldiff(qgdp)
# form list of high-frequency lagged log differences
list X = payems*
list dXL = hflags(3, 11, hfldiff(X, 100))
# initialize matrix to collect forecasts
matrix FC = {}

# estimation sample
smpl 1985:1 2009:1

print "=== unrestricted MIDAS (umidas) ==="
midasreg dy 0 dy(-1) ; mdsl(dXL, 0)
fcast --out-of-sample --static --quiet
FC ~= $fcast

print "=== normalized beta with zero last lag (beta0) ==="
midasreg dy 0 dy(-1) ; mdsl(dXL, 2, {1,5})
fcast --out-of-sample --static --quiet
FC ~= $fcast

print "=== normalized beta, non-zero last lag (betan) ==="
midasreg dy 0 dy(-1) ; mdsl(dXL, 3, {1,1,0})
fcast --out-of-sample --static --quiet
FC ~= $fcast

print "=== normalized exponential Almon (nealmon) ==="
midasreg dy 0 dy(-1) ; mdsl(dXL, 1, {0,0})
fcast --out-of-sample --static --quiet
FC ~= $fcast

print "=== Almon polynomial (almonp) ==="
midasreg dy 0 dy(-1) ; mdsl(dXL, 4, 4)
fcast --out-of-sample --static --quiet
FC ~= $fcast

smpl 2009:2 2011:2
matrix my = {dy}
print "Forecast RMSEs:"
printf "  umidas  %.4f\n", fcstats(my, FC[,1])[2]
printf "  beta0   %.4f\n", fcstats(my, FC[,2])[2]
printf "  betan   %.4f\n", fcstats(my, FC[,3])[2]
printf "  nealmon %.4f\n", fcstats(my, FC[,4])[2]
printf "  almonp  %.4f\n", fcstats(my, FC[,5])[2]
```

Listing 3: Replication of Ghysels' results, partial output

```
=== normalized beta, non-zero last lag (betan) ===
Model 3: MIDAS (NLS), using observations 1985:1-2009:1 (T = 97)
Using L-BFGS-B with conditional OLS
Dependent variable: dy

              estimate    std. error   t-ratio   p-value
  ---------------------------------------------------------
  const       0.748578    0.146404      5.113    1.74e-06 ***
  dy_1        0.248055    0.118903      2.086    0.0398   **

        MIDAS list dXL, high-frequency lags 3 to 11

  HF_slope    1.72167     0.582076      2.958    0.0039   ***
  Beta1       0.998501    0.0269479    37.05     1.10e-56 ***
  Beta2       2.95148     2.93404       1.006    0.3171
  Beta3      -0.0743143   0.0271273    -2.739    0.0074   ***

Sum squared resid    28.78262   S.E. of regression   0.562399
R-squared            0.356376   Adjusted R-squared   0.321012
Log-likelihood      -78.71248   Akaike criterion     169.4250
Schwarz criterion    184.8732   Hannan-Quinn         175.6715

=== Almon polynomial (almonp) ===
Model 5: MIDAS (NLS), using observations 1985:1-2009:1 (T = 97)
Using Levenberg-Marquardt algorithm
Dependent variable: dy

              estimate    std. error   t-ratio   p-value
  ---------------------------------------------------------
  const       0.741403    0.146433      5.063    2.14e-06 ***
  dy_1        0.255099    0.119139      2.141    0.0349   **

        MIDAS list dXL, high-frequency lags 3 to 11

  Almon0      1.06035     1.53491       0.6908   0.4914
  Almon1      0.193615    1.30812       0.1480   0.8827
  Almon2     -0.140466    0.299446     -0.4691   0.6401
  Almon3      0.0116034   0.0198686     0.5840   0.5607

Sum squared resid    28.66623   S.E. of regression   0.561261
R-squared            0.358979   Adjusted R-squared   0.323758
Log-likelihood      -78.51596   Akaike criterion     169.0319
Schwarz criterion    184.4802   Hannan-Quinn         175.2784

Forecast RMSEs:
  umidas  0.5424
  beta0   0.5650
  betan   0.5210
  nealmon 0.5642
  almonp  0.5329
```

Levenberg–Marquardt is the default NLS method, but if the MIDAS specifications include any of the beta variants or normalized exponential Almon we switch to L-BFGS-B, *unless* the user gives the `--levenberg` option. The ability to set bounds on the hyperparameters via L-BFGS-B is helpful, first because the beta parameters (other than the third one, if applicable) must be non-negative but also because one is liable to run into numerical problems (in calculating the weights and/or gradient) if their values become too extreme. For example, we have found it useful to place bounds of -2 and $+2$ on the exponential Almon parameters.

Here's what we mean by "conditional OLS" in the context of L-BFGS-B and line search: the search algorithm itself is only responsible for optimizing the MIDAS hyperparameters, and when the algorithm calls for calculation of the sum of squared residuals given a certain hyperparameter vector we optimize the remaining parameters (coefficients on base-frequency regressors, slopes with respect to MIDAS terms) via OLS.

Other specialized options

We mentioned above the standard options that are supported by the `midasreg` command, plus the `--levenberg` option; here we explain two additional options, `--clamp-beta` and `--breaktest`.

First, a case can be made for a variant of the normalized beta parameterization that is even more parsimonious than those discussed above: we take as a basis the two-parameter case (which implies a zero coefficient on the last lag) and "clamp" the first parameter, θ_1, at 1.0; the second parameter is then optimized, subject to the constraint $\theta_2 \geq 1.0$. This variant allows for a wide range of patterns of declining weights while arguably avoiding over-fitting of the weighting function to the estimation sample. Although it is bound to fit somewhat less well than the more general beta specifications in-sample it may be more effective in out-of-sample forecasting (Ghysels and Qian, 2016). This is supported under `midasreg` as follows: you specify the two-parameter beta option (*type* = 2, see section 5) but add the option flag `--clamp-beta`. At present this option is valid only if the regression contains a single MIDAS specification.

Second, the `--breaktest` option can be used to carry out the Quandt Likelihood Ratio (QLR) test for a structural break at the stage of running the final Gauss–Newton regression (to check for convergence and calculate the covariance matrix of the parameter estimates). This can be a useful aid to diagnosis, since non-homogeneity of the data over the estimation period can lead to numerical problems in nonlinear estimation, besides compromising the forecasting capacity of the resulting equation. For example, when this option is given with the command to estimate the "`BetaNZ`" model shown in Listing 3, the following result is appended to the standard output:

```
QLR test for structural break -
  Null hypothesis: no structural break
  Test statistic: chi-square(6) = 35.1745 at observation 2005:2
  with asymptotic p-value = 0.000127727
```

Despite the strong evidence for a structural break, in this case the nonlinear estimator appears to converge successfully, but one might wonder if a shorter estimation period could provide better out-of-sample forecasts.

Defining your own MIDAS estimator

As explained above, the `midasreg` command is in effect a "wrapper" for various underlying methods. Some users may wish to undo the wrapping. (This would be required if you wish to

introduce any nonlinearity other than that associated with the stock MIDAS parameterizations, or to define your own MIDAS parameterization).

Anyone with ambitions in this direction will presumably be quite familiar with the commands and functions available in hansl, gretl's scripting language, so we will not say much here beyond presenting a couple of examples. First we show how the `nls` command can be used, along with the MIDAS-related functions described in section 5, to estimate a model with the exponential Almon specification.

```
open gdp_midas.gdt --quiet
series dy = 100 * ldiff(qgdp)
series dy1 = dy(-1)
list X = payems*
list dXL = hflags(3, 11, hfldiff(X, 100))

smpl 1985:1 2009:1

# initialization via OLS
series mdX = mean(dXL)
ols dy 0 dy1 mdX --quiet
matrix b = $coeff | {0,0}'
scalar p = nelem(dXL)

# convenience matrix for computing gradient
matrix mdXL = {dXL}

# normalized exponential Almon via nls
nls dy = b[1] + b[2]*dy1 + b[3]*mdx
  series mdx = mlincomb(dXL, b[4:], 1)
  matrix grad = mgradient(p, b[4:], 1)
  deriv b = {const, dy1, mdx} ~ (b[3] * mdXL * grad)
  param_names "const dy(-1) HF_slope Almon1 Almon2"
end nls
```

Listing 4 presents a more ambitious example: we use `GSSmin` (Golden Section minimizer) to estimate a MIDAS model with the "one-parameter beta" specification (that is, the two-parameter beta with θ_1 clamped at 1). Note that while the function named `beta1_SSR` is specialized to the given parameterization, `midas_GNR` is a fairly general means of calculating the Gauss–Newton regression for an ADL(1) MIDAS model, and it could be generalized further without much difficulty.

7 MIDAS-related plots

In the context of MIDAS analysis one may wish to produce time-series plots which show high- and low-frequency data in correct registration (as in Figures 1 and 2 in Armesto *et al.*, 2010). This can be done using the `hfplot` command, which has the following syntax:

`hfplot` *midas-list* `[;` *lflist* `]` *options*

The required argument is a MIDAS list, as defined above. Optionally, one or more lower-frequency series (*lflist*) can be added to the plot following a semicolon. Supported options are `--with-lines`, `--time-series` and `--output`. These have the same effects as with the gretl's `gnuplot` command.

An example based on Figure 1 in Armesto *et al.* (2010) is shown in Listing 5 and Figure 2.

Listing 4: Manual MIDAS: one-parameter beta specification

```
set verbose off

function scalar beta1_SSR (scalar th2, const series y,
                           const series x, list L)
  matrix theta = {1, th2}
  series mdx = mlincomb(L, theta, 2)
  # run OLS conditional on theta
  ols y 0 x mdx --quiet
  return $ess
end function

function matrix midas_GNR (const matrix theta, const series y,
                           const series x, list L, int type)
  # Gauss-Newton regression
  series mdx = mlincomb(L, theta, type)
  ols y 0 x mdx --quiet
  matrix b = $coeff
  matrix u = {$uhat}
  matrix mgrad = mgradient(nelem(L), theta, type)
  matrix M = {const, x, mdx} ~ (b[3] * {L} * mgrad)
  matrix V
  set svd on # in case of strong collinearity
  mols(u, M, null, &V)
  return (b | theta) ~ sqrt(diag(V))
end function

/* main */

open gdp_midas.gdt --quiet

series dy = 100 * ldiff(qgdp)
series dy1 = dy(-1)
list dX = ld_payem*
list dXL = hflags(3, 11, dX)

# estimation sample
smpl 1985:1 2009:1

matrix b = {0, 1.01, 100}
# use Golden Section minimizer
SSR = GSSmin(b, beta1_SSR(b[1], dy, dy1, dXL), 1.0e-6)
printf "SSR (GSS) = %.15g\n", SSR
matrix theta = {1, b[1]}' # column vector needed
matrix bse = midas_GNR(theta, dy, dy1, dXL, 2)
bse[4,2] = $nan # mask std error of clamped coefficient
modprint bse "const dy(-1) HF_slope Beta1 Beta2"
```

Listing 5: Replication of a plot from Armesto et al

```
open gdp_midas.gdt

# form and label the dependent variable
series dy = log(qgdp/qgdp(-1))*400
setinfo dy --graph-name="GDP"

# form list of annualized HF differences
list X = payems*
list dX = hfldiff(X, 1200)
setinfo dX --graph-name="Payroll Employment"

smpl 1980:1 2009:1
hfplot dX ; dy --with-lines --time-series --output=display
```

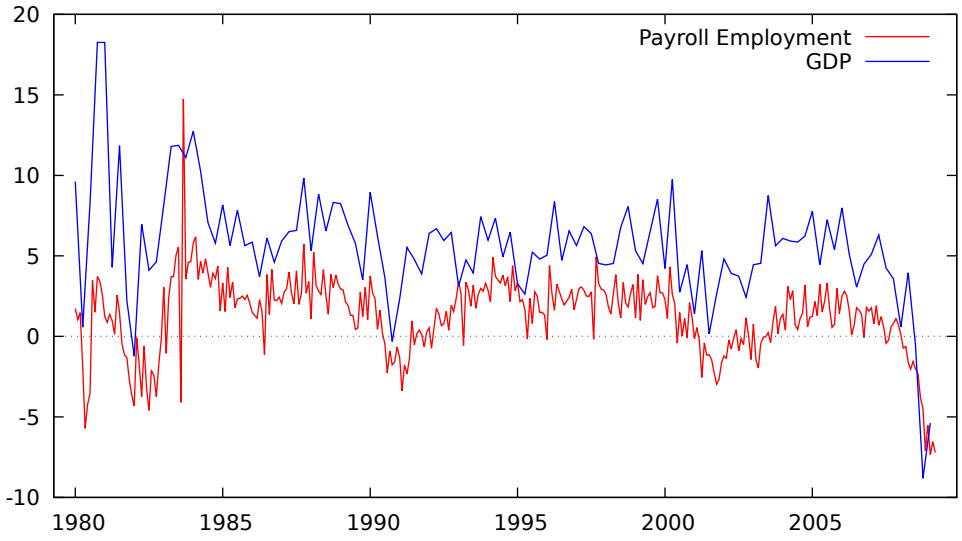

Figure 2: Quarterly GDP and monthly Payroll Employment, annualized percentage changes

Another sort of plot which may be useful shows the "gross" coefficients on the lags of the high-frequency series in a MIDAS regression—that is, the normalized weights multiplied by the HF_slope coefficient. After estimation of a MIDAS model in the gretl GUI this is available via the item MIDAS coefficients under the Graphs menu in the model window. It is also easily generated via script, since the $model bundle that becomes available following the midasreg command contains a matrix, midas_coeffs, holding these coefficients. So the following is sufficient to display the plot:

```
matrix m = $model.midas_coeffs
gnuplot --matrix=m --with-lp --fit=none --output=display \
  { set title "MIDAS coefficients"; set ylabel ''; }
```

Caveat: this feature is at present available only for models with a single MIDAS specification.

References

Armesto, M. T., K. Engemann and M. Owyang (2010) 'Forecasting with mixed frequencies', *Federal Reserve Bank of St. Louis Review* 92(6): 521–536. URL http://research.stlouisfed.org/publications/review/10/11/Armesto.pdf.

Davidson, R. and J. G. MacKinnon (2004) *Econometric Theory and Methods*, New York: Oxford University Press.

Ghysels, E. (2015) 'MIDAS Matlab Toolbox'. University of North Carolina, Chapel Hill. URL http://www.unc.edu/~eghysels/papers/MIDAS_Usersguide_V1.0.pdf.

Ghysels, E. and H. Qian (2016) 'Estimating MIDAS regressions via OLS with polynomial parameter profiling'. University of North Carolina, Chapel Hill, and MathWorks. URL http://dx.doi.org/10.2139/ssrn.2837798.

Ghysels, E., P. Santa-Clara and R. Valkanov (2004) 'The MIDAS touch: Mixed data sampling regression models'. Série Scientifique, CIRANO, Montréal. URL http://www.cirano.qc.ca/files/publications/2004s-20.pdf.

Appendix A: alternative MIDAS data methods

Importation via a column vector

Listing 6 illustrates how one can construct via hansl a "midas list" from a matrix (column vector) holding data of a higher frequency than the given dataset. In practice one would probably read high frequency data from file using the `mread` function, but here we just construct an artificial sequential vector.

Note the check in the `high_freq_list` function: we determine the current sample size, `T`, and insist that the input matrix is suitably dimensioned, with a single column of length equal to `T` times the compaction factor (here 3, for monthly to quarterly).

Listing 6: Create a midas list from a matrix

```
function list high_freq_list (const matrix x, int compfac, string vname)
  list ret = null
  scalar T = $nobs
  if rows(x) != compfac*T || cols(x) != 1
     funcerr "Invalid x matrix"
  endif
  matrix m = mreverse(mshape(x, compfac, T))'
  loop i=1..compfac --quiet
    scalar k = compfac + 1 - i
    ret += genseries(sprintf("%s%d", vname, k), m[,i])
  endloop
  setinfo ret --midas
  return ret
end function

# construct a little "quarterly" dataset
nulldata 12
setobs 4 1980:1

# generate "monthly" data, 1,2,...,36
matrix x = seq(1,3*$nobs)'
print x
# turn into midas list
list H = high_freq_list(x, 3, "test_m")
print H --byobs
```

The final command in the script should produce

```
            test_m3      test_m2      test_m1

1980:1            3            2            1
1980:2            6            5            4
1980:3            9            8            7
...
```

This functionality is available in the built-in function `hflist`, which has the same signature as the hansl prototype above.

Importation via `join`

Listing 7 illustrates how `join` can be used to pull higher-frequency data (in this example, quarterly) into a lower frequency (annual) dataset. The example is artificial, in that we have no actual annual data in view; consider it just as "proof of concept."

We begin by opening the Area-Wide Model (`AWM`) quarterly dataset and writing three series to a CSV file: the year, the quarter, and the associated value of `STN` (short-term interest rate).

We then establish an annual dataset with the same temporal span as the `AWM` data, and use `join` to create four series from the CSV file, `Eurorate_q4` to `Eurorate_q1`, matching rows via `join`'s `tkey` apparatus and filtering on the respective quarters. Partial output is shown beneath the script.

At this point we'd be ready to append an actual annual series that's of interest as a dependent variable, and estimate a model in which the quarterly interest-rate series figure as high-frequency independent variables.

Listing 7: Create a MIDAS dataset via `join`

```
open AWM --quiet
series yr = $obsmajor
series qtr = $obsminor
# STN = short-term interest rate
store @dotdir/tmp.csv yr qtr STN

# 116 quarters = 29 annual observations
nulldata 29
# Annual data starting in 1970
setobs 1 1970
list ER = null
loop for (q=4; q>0; q--) --quiet
  vname = sprintf("Eurorate_q%d", q)
  flt = sprintf("qtr==%d", q)
  join "@dotdir/tmp.csv" @vname --data=STN --tkey=",%YQ%q" --filter="@flt"
  ER += @vname
endloop
setinfo ER --midas
print ER -o
```

```
      Eurorate_q4  Eurorate_q3  Eurorate_q2  Eurorate_q1

1970       7.1892       7.5600       7.8705       7.9032
1971       5.9811       6.2442       5.8679       6.2773
1972       6.3387       4.6302       4.6883       4.9361
1973      10.9720       9.9925       8.1424       6.7541
1974      10.6747      11.3925      11.0187      11.1175
...
```

Appendix B: daily data

Daily data (commonly financial-market data) are often used in practical applications of the MIDAS methodology. It's therefore important that gretl support use of such data, but there are special issues arising from the fact that the number of days in a month, quarter or year is not a constant. Here we describe the state of things as of September 19, 2017.

It seems to us that it's necessary to stipulate a fixed, conventional number of days per lower-frequency period (that is, in practice, per month or quarter, since for the moment we're ignoring the week as a basic temporal unit and we're not yet attempting to support the combination of annual and daily data). But matters are further complicated by the fact that daily data come in (at least) three sorts: 5 days per week (as in financial-market data), 6-day (some commercial data which skip Sunday) and 7-day.

That said, we currently support—via `compact=spread`, as described in section 1—the following conversions:

- Daily to monthly: If the daily data are 5-days per week, we impose 22 days per month. This is the median, and also the mode, of weekdays per month, although some months have as few as 20 weekdays and some have 23. If the daily data are 6-day we impose 26 days per month, and in the 7-day case, 30 days per month.
- Daily to quarterly: In this case the stipulated days per quarter are simply 3 times the days-per-month values specified above.

So, given a daily dataset, you can say

```
dataset compact 12 spread
```

to convert MIDAS-wise to monthly (or substitute `4` for `12` for a quarterly target). And this is supposed to work whether the number of days per week is 5, 6 or 7.

That leaves the question of how we handle cases where the actual number of days in the calendar month or quarter falls short of, or exceeds, the stipulated number. We'll talk this through with reference to the conversion of 5-day daily data to monthly; all other cases are essentially the same, *mutatis mutandis*.[8]

We start at "day 1," namely the first relevant daily date within the calendar period (so the first weekday, with 5-day data). From that point on we fill up to 22 slots with relevant daily observations (including, not skipping, `NA`s due to holidays or whatever). If at the end we have daily observations left over, we ignore them. If we're short we fill the empty slots with the arithmetic mean of the valid, used observations;[9] and we fill in any missing values in the same way.

This means that lags 1 to 22 of 5-day daily data in a monthly dataset are always observations from days within the prior month (or in some cases "padding" that substitutes for such observations); lag 23 takes you back to the most recent day in the month before that.

Clearly, we *could* get a good deal fancier in our handling of daily data: for example, letting the user determine the number of days per month or quarter, and/or offering more elaborate means of filling in missing and non-existent daily values. It's not clear that this would be worthwhile, but it's open to discussion.

A little daily-to-monthly example is shown in Listing 8 and Figure 3. The example exercises the `hfplot` command (see section 7).

[8] Or should be! We're not ready to guarantee that just yet.

[9] This is the procedure followed in some example programs in the MIDAS Matlab Toolbox.

Listing 8: Monthly plus daily data

```
# open a daily dataset
open djclose.gdt

# spread the data to monthly
dataset compact 12 spread
list DJ = djc*

# import an actual monthly series
open fedstl.bin
data indpro

# high-frequency plot
hfplot DJ ; indpro --with-lines --output=daily.pdf \
 {set key top left;}
```

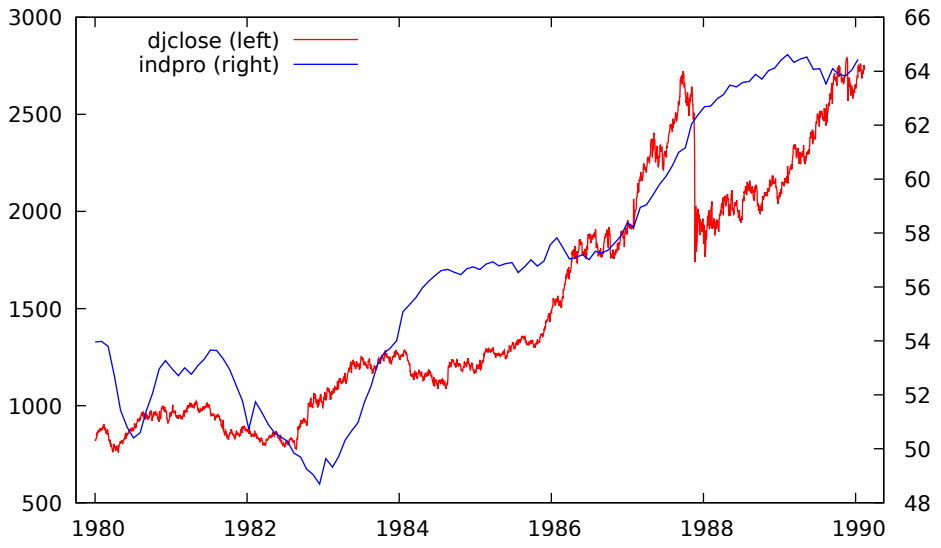

Figure 3: Monthly industrial production and daily Dow Jones close

Appendix C: parameterization functions

Here we give some more detail of the MIDAS parameterizations supported by gretl.

In general the normalized coefficient or weight i ($i = 1, \ldots, p$) is given by

$$w_i = \frac{f(i, \theta)}{\sum_{i=1}^{p} f(i, \theta)} \tag{1}$$

such that the coefficients sum to unity.

In the **normalized exponential Almon** case with k parameters the function $f(\cdot)$ is

$$f(i, \theta) = \exp\left(\sum_{j=1}^{k} \theta_j i^j\right) \tag{2}$$

So in the usual two-parameter case we have

$$w_i = \frac{\exp\left(\theta_1 i + \theta_2 i^2\right)}{\sum_{i=1}^{p} \exp\left(\theta_1 i + \theta_2 i^2\right)}$$

and equal weighting is obtained when $\theta_1 = \theta_2 = 0$.

In the standard, two-parameter **normalized beta** case we have

$$f(i, \theta) = (i^- / p^-)^{\theta_1 - 1} \cdot (1 - i^- / p^-)^{\theta_2 - 1} \tag{3}$$

where $p^- = p - 1$, and $i^- = i - 1$ except at the end-points, $i = 1$ and $i = p$, where we add and subtract, respectively, machine epsilon to avoid numerical problems. This formulation constrains the coefficient on the last lag to be zero—provided that the weights are declining at higher lags, a condition that is ensured if θ_2 is greater than θ_1 by a sufficient margin. The special case of $\theta_1 = \theta_2 = 1$ yields equal weights at all lags. A third parameter can be used to allow a non-zero final weight, even in the case of declining weights. Let w_i denote the normalized weight obtained by using (3) in (1). Then the modified variant with additional parameter θ_3 can be written as

$$w_i^{(3)} = \frac{w_i + \theta_3}{1 + p\theta_3}$$

That is, we add θ_3 to each weight then renormalize so that the $w_i^{(3)}$ values again sum to unity.

In Eric Ghysels' Matlab code the two beta variants are labeled "normalized beta density with a zero last lag" and "normalized beta density with a non-zero last lag" respectively. Note that while the two basic beta parameters must be positive, the third additive parameter may be positive, negative or zero.

In the case of the plain **Almon polynomial**, coefficient i is given by

$$w_i = \sum_{j=1}^{k} \theta_j i^{j-1}$$

Note that no normalization is applied in this case, so no additional coefficient should be placed before the MIDAS lags term in the context of a regression.

Analytical gradients

Here we set out the expressions for the analytical gradients produced by the `mgradient` function, and also used internally by the `midasreg` command. In these expressions $f(i,\theta)$ should be understood as referring back to the specific forms noted above for the exponential Almon and beta distributions.

For the normalized exponential Almon case, the gradient is

$$\begin{aligned}\frac{dw_i}{d\theta_j} &= \frac{f(i,\theta)i^j}{\sum_i f(i,\theta)} - \frac{f(i,\theta)}{\left[\sum_i f(i,\theta)\right]^2}\sum_i\left[f(i,\theta)i^j\right] \\ &= w_i\left(i^j - \frac{\sum_i\left[f(i,\theta)i^j\right]}{\sum_i f(i,\theta)}\right)\end{aligned}$$

For the two-parameter normalized beta case it is

$$\begin{aligned}\frac{dw_i}{d\theta_1} &= \frac{f(i,\theta)\log(i^-/p^-)}{\sum_i f(i,\theta)} - \frac{f(i,\theta)}{\left[\sum_i f(i,\theta)\right]^2}\sum_i\left[f(i,\theta)\log(i^-/p^-)\right] \\ &= w_i\left(\log(i^-/p^-) - \frac{\sum_i\left[f(i,\theta)\log(i^-/p^-)\right]}{\sum_i f(i,\theta)}\right) \\ \frac{dw_i}{d\theta_2} &= \frac{f(i,\theta)\log(1-i^-/p^-)}{\sum_i f(i,\theta)} - \frac{f(i,\theta)}{\left[\sum_i f(i,\theta)\right]^2}\sum_i\left[f(i,\theta)\log(1-i^-/p^-)\right] \\ &= w_i\left(\log(1-i^-/p^-) - \frac{\sum_i\left[f(i,\theta)\log(1-i^-/p^-)\right]}{\sum_i f(i,\theta)}\right)\end{aligned}$$

And for the three-parameter beta, we have

$$\begin{aligned}\frac{dw_i^{(3)}}{d\theta_1} &= \frac{1}{1+p\theta_3}\frac{dw_i}{d\theta_1} \\ \frac{dw_i^{(3)}}{d\theta_2} &= \frac{1}{1+p\theta_3}\frac{dw_i}{d\theta_2} \\ \frac{dw_i^{(3)}}{d\theta_3} &= \frac{1}{1+p\theta_3} - \frac{p(w_i+\theta_3)}{(1+p\theta_3)^2}\end{aligned}$$

For the (non-normalized) Almon polynomial the gradient is simply

$$\frac{dw_i}{d\theta_j} = i^{j-1}$$