Price discovery in high resolution: computational appendix

*Joel Hasbrouck**

October 1, 2018

*Department of Finance
Stern School of Business, NYU
44 West 4th St.
New York, NY 10012
jhasbrou@stern.nyu.edu

Abstract

Estimating and forecasting long-lag high-resolution multivariate time series models pose computational challenges. This appendix describes several techniques used in the paper. Most are based on standard polynomial distributed lag (PDL) and sparse matrix methods. I discuss these in simple univariate and bivariate settings, along with extensions relevant to the problem at hand.

This document includes Mathematica code at the end. The Matlab software uses additional conventions and will be described in a separate document. There are two versions of this appendix: a pdf document (with a 'pdf' extension) and a Mathematica notebook (with an 'nb' extension). The latter is executable in Mathematica.

A review of polynomial distributed lags (PDLs)

For bivariate time series $\{x_t, y_t\}$, consider a standard linear lagged regression model:

$$y_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + \ldots + \beta_{\mathcal{L}} x_{t-\mathcal{L}} + e_t \tag{1}$$

If $x_t = y_t$ this is an autoregression. The data sample is $\{x_t, y_t : t = 1, \ldots, \mathcal{T}\}$. The values can be arranged for estimation as $y = X\beta + e$, where $y = \begin{pmatrix} y_1 & y_2, & \ldots & y_{\mathcal{T}} \end{pmatrix}^{\mathrm{T}}$, e is a $\mathcal{T} \times 1$ vector of errors, and X is the lagged data matrix, that is a $\mathcal{T} \times \mathcal{L}$ matrix of the lagged xs:

$$X = \begin{pmatrix} x_0 & x_{-1} & x_{-2} & \cdots & x_{-\mathcal{L}+1} \\ x_1 & x_0 & x_{-1} & \ldots & x_{-\mathcal{L}} \\ & \vdots & & & \\ x_t & x_{t-1} & x_{t-2} & \cdots & x_{t-\mathcal{L}+1} \\ & \vdots & & & \\ x_{\mathcal{T}-1} & x_{\mathcal{T}-1} & x_{\mathcal{T}-1} & \cdots & x_{\mathcal{T}-\mathcal{L}+1} \end{pmatrix}$$

Here, values of x_t for $t < 1$ can be handled in various ways. If $Ex = 0$ we can set the pre-sample values to zero. Alternatively, we can discard rows that contain any pre-sample values; y is adjusted to be conformable. The OLS estimates of β are $\hat{\beta} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} y$.

In the autoregressive case, y may be forecast subsequent to an initial disturbance e_0 by recursion. The forecasts, denoted y_t^*, are:

$$\begin{aligned} y_1^* &= e_0 \\ y_2^* &= \beta_1 y_1^* \\ y_3^* &= \beta_1 y_2^* + \beta_2 y_1^* \\ &\vdots \end{aligned} \tag{2}$$

$$y_t^* = \beta_1 y_{t-1}^* + \beta_2 y_{t-2}^* + \ldots + \beta_{\mathcal{L}} y_{t-\mathcal{L}}^*$$

This is the same as the lagged regression model, but since we are forecasting, we are recursively calculating the system assuming that all disturbances, except e_0, are zero.

In the present applications, both the sample size $\mathcal{T}$ and the number of parameters $\mathcal{L}$ are large: $\mathcal{T} \approx 10^9$ and $\mathcal{L} \approx 10^6$. This makes direct estimation and forecasting difficult at best and infeasible at worst. The following material describes how use of polynomials and sparsity can help.

The first device we employ is due to Almon (1965), who suggests constraining the β_i to lie on polynomials in i. In the quadratic case, for example, $\beta_i = \gamma_0 + \gamma_1 i + \gamma_2 i^2$. In matrix notation, this can be expressed as $\beta = D\gamma$, where $\gamma = (\gamma_0 \;\; \gamma_1 \;\; \gamma_2)^{\mathrm{T}}$ and D is the $(\mathcal{L} \times 3)$ design matrix:

$$D = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ & \vdots & \\ 1 & \mathcal{L} & \mathcal{L}^2 \end{pmatrix} \tag{3}$$

Substituting into the regression equation gives

$$y = X\beta + e = XD\gamma + e$$

The first term on the right hand side can be grouped as $X(D\gamma)$, which suggests a smoothing of the model parameters (the β coefficients). Grouping as $(XD)\,\gamma$ suggests a transformation or smoothing of the data. This distinction, though, is purely a matter of perspective. The mathematics are the same.

A PDL can greatly reduce the dimensionality of the estimation. The transformed data matrix, XD is $\mathcal{T} \times 3$, and the γ may be estimated directly as

$$\hat{\gamma} = (D^{\mathrm{T}} X^{\mathrm{T}} X D)^{-1} D^{\mathrm{T}} X^{\mathrm{T}} y \tag{4}$$

This effectively reduces the size of the cross-product matrix: $X^{\mathrm{T}} X$ is $\mathcal{L} \times \mathcal{L}$, but $D^{\mathrm{T}} X^{\mathrm{T}} X D$ is 3×3.

PDLs can also simplify the forecasting computations. Direct application of (2) requires $\mathcal{L}$ multiplications and $\mathcal{L}$ additions at each step. Using the PDL representation for the coefficients,

$$\begin{aligned} y_t &= (\gamma_0 + \gamma_1 + \gamma_2)\, y_{t-1} + (\gamma_0 + 2\,\gamma_1 + 4\,\gamma_2)\, y_{t-2} + \cdots + \left(\gamma_0 + \mathcal{L}\gamma_1 + \mathcal{L}^2\,\gamma_2\right) y_{t-\mathcal{L}} \\ &= \gamma_0 (y_{t-1} + y_{t-2} + y_{t-3} + \cdots + y_{t-\mathcal{L}}) + \\ &\quad \gamma_1 (y_{t-1} + 2\,y_{t-2} + 3\,y_{t-3} + \cdots + \mathcal{L}\mathrm{y}_{t-\mathcal{L}}) + \gamma_2 \left(y_{t-1} + 4\,y_{t-2} + 9\,y_{t-3} + \cdots + \mathcal{L}^2\,y_{t-\mathcal{L}}\right) \end{aligned} \tag{5}$$

Now define three state variables that will carry the quantities in the parentheses:

$$S(t,\, d) = \sum_{i=1}^{\mathcal{L}} i^d\, y_{t-i}, \;\; \text{for } d = 0,\, 1,\, 2$$

To update $S(t, d) \rightarrow S(t+1, d)$, we first compute $y_t = \sum_{d=0}^{2} \gamma_d\, S(t, d)$. Then to take $S(t, 0) \rightarrow S(t+1, 0)$, note that $y_{t-\mathcal{L}}$ leaves the sum and y_t enters:

$$S(t+1,\, 0) = S(t,\, 0) + y_t - y_{t-\mathcal{L}} \tag{6}$$

Similarly, by direct examination of the leaving and entering terms for higher orders:

$$\begin{aligned} S(t+1,1) &= S(t,1) + S(t+1,0) - \mathcal{L} y_{t-\mathcal{L}} \\ &= S(t,1) + S(t,0) + y_t - (\mathcal{L}+1)\, y_{t-\mathcal{L}} \\ S(t+1,2) &= S(t,2) + 2\,S(t+1,1) - S(t+1,0) - \mathcal{L}^2\, y_{t-\mathcal{L}} \\ &= S(t,2) + 2\,S(t,1) + S(t,0) + y_t - (1+\mathcal{L})^2\, y_{t-\mathcal{L}} \end{aligned} \tag{7}$$

It is still necessary to store all lagged values $y_t, \ldots, y_{t-\mathcal{L}}$, but the update calculations are more efficient. (In the Matlab code, see the **stateUpdate** method in the **polynom** class.)

Additional notes on PDLs

The design matrix involves multiple polynomials covering different lagged intervals. These polynomials may be of different orders. Consider, for example, the design matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 & 0 \\ 1 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 & 4 \\ 0 & 0 & 0 & 1 & 3 & 9 \end{pmatrix}$$

This specifies that: $\beta_1, \ldots, \beta_4$ lie on a linear segment; $\beta_5 = \beta_6 = \beta_7$; and, β_8 through β_{10} lie on a quadratic segment. Linear restrictions on the γ can be imposed to ensure that the segments join up. Splines can be used to give smoothness (continuity of derivatives, see Smith (1979)). Although PDLs are usually introduced using the direct polynomial forms used above, most estimation software packages use alternative representations that aim to control collinearity and errors of numerical precision (Cooper (1972)).

Sparsity

Application of PDL's greatly reduces the size of the cross product matrix, from $\mathcal{L} \times \mathcal{L}$ to (in the quadratic case) 3×3. To arrive at that point in the direct way, though, we must compute XD where X is $\mathcal{T} \times \mathcal{L}$ and D is $\mathcal{L} \times 3$. This is still a formidable computation, so we take advantage of the sparsity of the data. Specifically, if y and x are $\mathcal{T}$-vectors of price changes, it may well be that the number of non-zero elements is on the order of $10^{-6}\,\mathcal{T}$. This is sparsity in the usual sense, wherein a matrix is assumed to be primarily zeros, and only the non-zero elements (and their locations) are stored. Notationally, we write $x = \{t_k, v_k \text{ for } k = 1, \ldots, K\}$ where t_k and v_k denote the time and value of the k^{th} element. Implicitly, the K nonzero elements are embedded in a vector of size $\mathcal{T}$, which has zeros (except at the t_k positions). Direct calculation of $D^{\mathrm{T}} X^{\mathrm{T}} X D$ requires on the order of $\mathcal{T}^2\, \mathcal{L}^4$ multiplications. By taking advantage of sparsity, this can be brought down to the order of K^2.

Define $\mathcal{I}(k)$ as a $\mathcal{T} \times \mathcal{L}$ matrix of zeros, with an $\mathcal{L} \times \mathcal{L}$ identity matrix located with its first element in row k and column 1. With this definition, the lagged data matrix may be written as

$$X = \sum_{k=1}^{K} v_k\, \mathcal{I}(t_k + 1)$$

where the unit offset in $t_k + 1$ imposes a lag relative to the original series. It is understood that any terms with row index larger than $\mathcal{T}$ are discarded.

For example, suppose that with $\mathcal{T} = 12$, there are three non-zero values of x: $x_1 = 4$, $x_3 = 2$, and $x_7 = 5$. Specified as a sparse matrix, $t = \{1, 3, 7\}$ and $v = \{4, 2, 5\}$ The lagged data matrix for $\mathcal{L} = 4$ is $X = 4\,\mathcal{I}(2) + 2\,\mathcal{I}(4) + 5\,\mathcal{I}(8)$. Expressed in full form:

$$X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 0 & 2 & 0 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \\ 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{pmatrix} = 4 \times \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + 2 \times \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + 5 \times \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

It is initially easier to illustrate the calculations when the data are assumed to satisfy a non-overlap condition such that the t_k are sufficiently removed from one another that $t_k - t_{k-1} \geq \mathcal{L}$ for $k = 2, \ldots, K$. In this case, the blocks of lagged variables are distinct. The example given above does not satisfy this condition. (Rows 4 and 5 contain overlapped lagged values.) A lagged data matrix that *does* satisfy the non-overlap condition can be constructed for nonzero values $x_1 = 4$, $x_4 = 2$, $x_7 = 5$, with $\mathcal{T} = 10$ and $\mathcal{L} = 3$:

$$X = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \\ 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{pmatrix} = 4 \times \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + 2 \times \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + 5 \times \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The cross product required for the OLS estimates is then:

$$\begin{aligned} D^{\mathrm{T}} X^{\mathrm{T}} X D &= D^{\mathrm{T}} \left(\sum_k v_k\, \mathcal{I}(t_k + 1)^{\mathrm{T}} \right) \left(\sum_j v_j\, \mathcal{I}(t_j + 1) \right) D \\ &= \left(\sum_k v_k\, D^{\mathrm{T}}\, \mathcal{I}(t_k + 1)^{\mathrm{T}} \right) \left(\sum_j v_j\, \mathcal{I}(t_j + 1) D \right) \\ &= \sum_{k,j} v_j\, v_k\, D^{\mathrm{T}}\, \mathcal{I}(t_k + 1)^{\mathrm{T}}\, \mathcal{I}(t_j + 1) D \\ &= \sum_k v_k^2\, D^{\mathrm{T}}\, \mathcal{I}(t_k + 1)^{\mathrm{T}}\, \mathcal{I}(t_k + 1) D \end{aligned} \tag{8}$$

The last equality relies on the non-overlap condition. Since $\mathcal{I}(t_k + 1)^{\mathrm{T}}\, \mathcal{I}(t_k + 1)$ is an identity matrix of order $\mathcal{L}$, the inner term of the last summation is equal to the product $D^{\mathrm{T}} D$, is a constant 3×3 matrix. Since we only need to compute it once, direct calculation is straightforward. Of course, in this case, it might be pointed out that with the non-overlap condition, we could view the problem as consisting of K disjoint samples. Cumulating over these samples is the obvious thing to do, with no particular need to embed the data in the length-$\mathcal{T}$ time frame.

We now drop the non-overlap assumption. We also consider a more general bivariate cross product situation similar to those that arise in multivariate VARs. Suppose that both x and y are sparse time series. The lagged data matrix corresponding to x is X, a $\mathcal{T}\times\mathcal{L}_x$ matrix, where $\mathcal{L}$ denotes the number of lags on x. Similarly, the lagged data matrix for y is Y, a $\mathcal{T}\times L_y$ matrix. The corresponding PDL design matrices are D_x and D_y. D_x has $\mathcal{L}_x$ rows, and D_y has $\mathcal{L}_y$ rows. The orders of the polynomials (equal to the numbers of columns) may also differ. The desired cross product matrix is $D_x{}^{\mathrm{T}} X^{\mathrm{T}}\, Y D_y$. The sparse representation of x involves time and value vectors t_k and v_k for $k = 1, \; ... K \ll \mathcal{T}$; that of y has time and value vectors s_j and u_j for $j = 1, \; ..., J \ll \mathcal{T}$. The lagged data matrices are represented as:

$$X = \sum_{k=1}^{K} v_k\, \mathcal{I}(t_k + 1) \text{ and } Y = \sum_{j=1}^{J} u_j\, \mathcal{J}(s_j + 1)$$

where $\mathcal{I}(t)$ is (as above) a $\mathcal{T}\times\mathcal{L}_x$ matrix of zeros, with an identity matrix of order $\mathcal{L}_x$ positioned with its leading element in the first column of row t, and $\mathcal{J}(s)$ is similarly defined as a $\mathcal{T}\times\mathcal{L}_y$ matrix of zeros, with an identity matrix of order $\mathcal{L}_y$ positioned at s. With these definitions,

$$\begin{aligned} D_x{}^{\mathrm{T}} X^{\mathrm{T}}\, Y D_y &= D_x{}^{\mathrm{T}} \left(\sum_{k=1}^{K} v_k\, \mathcal{I}(t_k + 1) \right)^{\mathrm{T}} \left(\sum_{j=1}^{J} u_j\, \mathcal{J}(s_j + 1) \right) D_y \\ &= \sum_{k=1}^{K} \sum_{j=1}^{J} v_k\, u_j\, D_x{}^{\mathrm{T}}\, \mathcal{I}(t_k + 1)^{\mathrm{T}}\, \mathcal{J}(s_j + 1)\, D_y \end{aligned}$$

For convenience, define $\mathcal{I}\mathcal{J}(t, s) \equiv \mathcal{I}(t)^{\mathrm{T}}\, \mathcal{J}(s)$, a matrix of size $\mathcal{L}_x\times\mathcal{L}_y$, and with elements are to zero or one. $\mathcal{I}\mathcal{J}(t, s)$ indicates the overlap between the two lagged structures. Suppose, for example, that with $\mathcal{T} = 10$, $\mathcal{L}_x = 3, \; \mathcal{L}_y = 4, \; t_1 = 2,$ and $s_1 = 3$. Then

$$\mathcal{I}(2) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \mathcal{J}(3) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \text{ and } \mathcal{I}\mathcal{J}(2, 3) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

These representations can greatly simplify the cross product calculation. Initially suppose that each of the PDL design matrices contains only a constant term: $D_x = (\,1 \;\; 1 \;\; 1\,)^{\mathrm{T}}$ and $D_y = (\,1 \;\; 1 \;\; 1 \;\; 1\,)^{\mathrm{T}}$. Then the contribution to the cross product is

$$v_1\, u_1 (\,1 \;\; 1 \;\; 1\,) \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = v_1\, u_1 \times 2$$

If the PDL design matrices contain constant and linear terms,

$$D_x{}^{\mathrm{T}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \text{and}\, D_y{}^{\mathrm{T}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix},$$

and the contribution to the cross product is:

$$v_1\, u_1 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} = v_1\, u_1 \begin{pmatrix} 2 & 3 \\ 5 & 8 \end{pmatrix}$$

With quadratic PDL design matrices the cross product contribution is

$$v_1\, u_1 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix} = v_1\, u_1 \begin{pmatrix} 2 & 3 & 5 \\ 5 & 8 & 14 \\ 13 & 22 & 40 \end{pmatrix}$$

In general, the entries in the last matrix are double partial sums of offset finite geometric series, for which compact closed-form expressions are readily available. (It is not necessary to construct and multiply the matrices on the left hand sides of these expressions. See the Mathematica section on lagged polynomial cross products of sparse vectors.) Taking advantage of these closed-form expressions, define $f(t, s, \mathcal{L}_x, \mathcal{L}_y, d_x, d_y) = D_x{}^{\mathrm{T}}\, \mathcal{I}(t)\, \mathcal{J}(s)\, D_y$, where D_x is the PDL design matrix for x, with maximum degree d_x and lag length $\mathcal{L}_x$. D_y is defined similarly.

$$D_x{}^{\mathrm{T}}\, X^{\mathrm{T}}\, Y D_y = \sum_{k=1}^{K} \sum_{j=1}^{J} v_k\, u_j\, f(t_k + 1, s_j + 1, \mathcal{L}_x, \mathcal{L}_y, d_x, d_y)$$

Direct computation of the left-hand side involves something on the order of $\mathcal{L}^2{}_x \times \mathcal{T}^2 \times \mathcal{L}^2{}_y$ operations. Evaluation of f involves (at most) several hundred operations. The double sum is only over the nonzero values of x and y. If we sort the entries in the sparse matrices by time (of the nonzero elements), we can realize greater efficiencies. For a given k, the values of j that make nonzero contributions to the crossproduct (that is, $\mathcal{I}\mathcal{J}(t_k + 1, s_j + 1) \neq 0$) can be determined before the start of the inner loop.

References

Almon, Shirley, 1965, The Distributed Lag Between Capital Appropriations and Expenditures, *Econometrica* 33, 178-196.

Cooper, J. Phillip, 1972, Two Approaches to Polynomial Distributed Lags Estimation: An Expository Note and Comment, The American Statistician 26, 32-35.

Smith, Patricia L., 1979, Splines as a useful and convenient statistical tool, *American Statistician* 33, 57-62.

Mathematica Code

Polynomial update verifications

This section may be evaluated.

```
Clear[y, S, t, n, d];
```

The PDL state variables are as follows. **t** is the current time index; **n** is the max lag; **d** is the degree.

```
S[t_: t, n_: n, d_: 0] := Sum[y_{t-i} i^d, {i, 1, n}]
```

```
{S[t, n, #] & /@ {0, 1, 2}} // TableForm
```

$$\sum_{i=1}^{n} y_{-i+t} \qquad \sum_{i=1}^{n} i\; y_{-i+t} \qquad \sum_{i=1}^{n} i^2\; y_{-i+t}$$

Verify the degree 0 update expression for various values of **n**:

```
Table[Evaluate[S[t + 1, n, 0] == (S[t, n, 0] + y_t - y_{t-n})], {n, 5, 20, 5}] //
 Simplify
```

```
{True, True, True, True}
```

Verify the degree 1 update expression for various values of **n**.

```
Table[S[t + 1, n, 1] == (S[t, n, 1] + S[t + 1, n, 0] - n y_{t-n}), {n, 5, 20, 5}]
```

```
{True, True, True, True}
```

Alternatively, the expression used in the Matlab code is:

```
(S[t, n, 1] + (S[t, n, 0] + y_t - y_{t-n}) - n y_{t-n}) // Simplify
```

$$y_t - (1 + n)\, y_{-n+t} + \sum_{i=1}^{n} y_{-i+t} + \sum_{i=1}^{n} i\, y_{-i+t}$$

```
Table[S[t + 1, n, 1] == (S[t, n, 1] + (S[t, n, 0] + y_t) - (n + 1) y_{t-n}),
 {n, 5, 20, 5}]
```

```
{True, True, True, True}
```

Verify the degree 2 update expression for various values of **n**.

```
Table[S[t + 1, n, 2] == S[t, n, 2] + 2 S[t + 1, n, 1] - S[t + 1, n, 0] - n^2 y_{t-n},
  {n, 5, 20, 5}] // Simplify
```

```
{True, True, True, True}
```

Alternatively, the expression used in the Matlab code is:

```
S[t, n, 2] + 2 (S[t, n, 1] + (S[t, n, 0] + y_t) - (n + 1) y_{t-n}) -
  (S[t, n, 0] + y_t - y_{t-n}) - n^2 y_{t-n} // Simplify
```

$$y_t - (1 + n)^2\, y_{-n+t} + \sum_{i=1}^{n} y_{-i+t} + 2 \sum_{i=1}^{n} i\, y_{-i+t} + \sum_{i=1}^{n} i^2\, y_{-i+t}$$

```
Table[
 S[t + 1, n, 2] == S[t, n, 2] + 2 (S[t, n, 1] + (S[t, n, 0] + y_t) - (n + 1) y_{t-n}) -
   (S[t, n, 0] + y_t - y_{t-n}) - n^2 y_{t-n}, {n, 5, 20, 5}] // Simplify
```

```
{True, True, True, True}
```

Lagged polynomial cross products of sparse vectors

Matrix construction

```
Clear[X, Y, 𝒟, tSum];
```

$\mathcal{D}$ denotes the coefficient (design) matrix for polynomials. (“D” is reserved in Mathematica for derivatives.)

```
𝒟[n_, d_] := Table[i^dd, {dd, 0, d}, {i, 1, n}]^T;
𝒟[4, 2] // MatrixForm
```

```
1 1  1
1 2  4
1 3  9
1 4 16
```

X[i,t] is a matrix with $\mathcal{T}$ rows, **n** columns and an identity matrix starting in row **i**. It represents a lag matrix, where a variable that appears in position *i*, 1 also appears in position $i-1$, 2, position $i-2$, 3, ... position $i-n+1$, *n*. (The Mathematica **Block** construct is used to localize the assignment $\mathcal{T}$=**8**.)

```
X[i_, n_] := Module[{x},
   x = Table[0, {𝒯}, {n}];
   If[i + n - 1 > 𝒯, Print["X[i,n]: i+n-1>𝒯 (i=", i, "; n=", n,
     "; 𝒯=", 𝒯, ")"];
    Return[Null]];
   x[[i ;; i + n - 1]] = IdentityMatrix[n];
   x];
Block[{𝒯 = 8}, X[3, 4]] // MatrixForm
```

```
0 0 0 0
0 0 0 0
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
0 0 0 0
```

Define *Y* similarly:

```
Y[i_, n_] := X[i, n]
```

and

```
Ones := Table[{1}, {𝒯}];
Block[{𝒯 = 8}, Ones] // MatrixForm
```

```
1
1
1
1
1
1
1
1
```

Matrices used in text examples:

```
Block[{𝒯 = 12, ℒ = 4}, x1 = X[2, ℒ];
 x2 = X[4, ℒ];
 x3 = X[8, ℒ];
 r = 4 * x1 + 2 * x2 + 5 * x3;
 MatrixForm /@ {x1, x2, x3, r}]
```

```
 ⎛0 0 0 0⎞   ⎛0 0 0 0⎞   ⎛0 0 0 0⎞   ⎛0 0 0 0⎞
 ⎜1 0 0 0⎟   ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜4 0 0 0⎟
 ⎜0 1 0 0⎟   ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜0 4 0 0⎟
 ⎜0 0 1 0⎟   ⎜1 0 0 0⎟   ⎜0 0 0 0⎟   ⎜2 0 4 0⎟
 ⎜0 0 0 1⎟   ⎜0 1 0 0⎟   ⎜0 0 0 0⎟   ⎜0 2 0 4⎟
 ⎜0 0 0 0⎟   ⎜0 0 1 0⎟   ⎜0 0 0 0⎟   ⎜0 0 2 0⎟
{⎜0 0 0 0⎟ , ⎜0 0 0 1⎟ , ⎜0 0 0 0⎟ , ⎜0 0 0 2⎟}
 ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜1 0 0 0⎟   ⎜5 0 0 0⎟
 ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜0 1 0 0⎟   ⎜0 5 0 0⎟
 ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜0 0 1 0⎟   ⎜0 0 5 0⎟
 ⎜0 0 0 0⎟   ⎜0 0 0 0⎟   ⎜0 0 0 1⎟   ⎜0 0 0 5⎟
 ⎝0 0 0 0⎠   ⎝0 0 0 0⎠   ⎝0 0 0 0⎠   ⎝0 0 0 0⎠
```

```
Block[{𝒯 = 10, ℒ = 3}, x1 = X[2, ℒ];
 x2 = X[5, ℒ];
 x3 = X[8, ℒ];
 r = 4 * x1 + 2 * x2 + 5 * x3;
 MatrixForm /@ {r, x1, x2, x3}]
```

```
 ⎛0 0 0⎞   ⎛0 0 0⎞   ⎛0 0 0⎞   ⎛0 0 0⎞
 ⎜4 0 0⎟   ⎜1 0 0⎟   ⎜0 0 0⎟   ⎜0 0 0⎟
 ⎜0 4 0⎟   ⎜0 1 0⎟   ⎜0 0 0⎟   ⎜0 0 0⎟
 ⎜0 0 4⎟   ⎜0 0 1⎟   ⎜0 0 0⎟   ⎜0 0 0⎟
 ⎜2 0 0⎟   ⎜0 0 0⎟   ⎜1 0 0⎟   ⎜0 0 0⎟
{⎜0 2 0⎟ , ⎜0 0 0⎟ , ⎜0 1 0⎟ , ⎜0 0 0⎟}
 ⎜0 0 2⎟   ⎜0 0 0⎟   ⎜0 0 1⎟   ⎜0 0 0⎟
 ⎜5 0 0⎟   ⎜0 0 0⎟   ⎜0 0 0⎟   ⎜1 0 0⎟
 ⎜0 5 0⎟   ⎜0 0 0⎟   ⎜0 0 0⎟   ⎜0 1 0⎟
 ⎝0 0 5⎠   ⎝0 0 0⎠   ⎝0 0 0⎠   ⎝0 0 1⎠
```

```
Block[{𝒯 = 10}, x = X[2, 3];
 y = X[3, 4];
 xty = xᵀ.y;
 MatrixForm /@ {x, y, xty}]
```

$$\left\{\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\right\}$$

```
                                           ⎛0 0 0 0⎞
MatrixForm /@ {𝒟[3, 1], 𝒟[4, 1], 𝒟[3, 1]ᵀ.⎜1 0 0 0⎟.𝒟[4, 1]}
                                           ⎝0 1 0 0⎠
```

$$\left\{\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}, \begin{pmatrix} 2 & 3 \\ 5 & 8 \end{pmatrix}\right\}$$

```
                                           ⎛0 0 0 0⎞
MatrixForm /@ {𝒟[3, 2], 𝒟[4, 2], 𝒟[3, 2]ᵀ.⎜1 0 0 0⎟.𝒟[4, 2]}
                                           ⎝0 1 0 0⎠
```

$$\left\{\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix}, \begin{pmatrix} 2 & 3 & 5 \\ 5 & 8 & 14 \\ 13 & 22 & 40 \end{pmatrix}\right\}$$

Continuation

The result matrix is $\mathcal{D}_x{}^{\mathrm{T}}.X^{\mathrm{T}}.Y.\mathcal{D}_y$. The following depicts the pieces and final result in the case of fixed values for $\mathcal{T}$, **ℒx**, and **ℒy**; for **X[5,ℒx]** and **Y[j,ℒy]** where **j** goes from 2 to 9.

```
Block[{𝒯 = 12, ℒx = 4, ℒy = 3},
Table[{j,
      X[5, ℒx] // MatrixForm,
      Y[j, ℒy] // MatrixForm,
      X[5, ℒx]ᵀ.Y[j, ℒy] // MatrixForm,
      𝒟[ℒx, 2]ᵀ.X[5, ℒx]ᵀ.X[j, ℒy].𝒟[ℒy, 2] // MatrixForm},
    {j, 2, 9}] // Transpose // TableForm] // Style[#, FontSize → 9] &
```

```
2           3           4           5           6           7           8           9
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0     1 0 0 0
0 1 0 0     0 1 0 0     0 1 0 0     0 1 0 0     0 1 0 0     0 1 0 0     0 1 0 0     0 1 0 0
0 0 1 0     0 0 1 0     0 0 1 0     0 0 1 0     0 0 1 0     0 0 1 0     0 0 1 0     0 0 1 0
0 0 0 1     0 0 0 1     0 0 0 1     0 0 0 1     0 0 0 1     0 0 0 1     0 0 0 1     0 0 0 1
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0
0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0     0 0 0 0

0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0
1 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0
0 1 0       1 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0
0 0 1       0 1 0       1 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0
0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0
0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 1       0 1 0       1 0 0
0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 1       0 1 0
0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 1
0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0       0 0 0

0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0       0 0 0
0 0 0       0 0 0       0 0 0       0 0 0       0 0 1       0 1 0       1 0 0       0 0 0

0 0 0       1 3 9       2  5 13      3  6 14     3  6  14     2  3  5     1  1  1    0 0 0
0 0 0       1 3 9       3  8 22      6 14 36     9 20  50     7 11 19     4  4  4    0 0 0
0 0 0       1 3 9       5 14 40     14 36 98    29 70 184    25 41 73    16 16 16    0 0 0
```

Depending on the relative offset between the X and Y matrices, the X^TY product is a diagonal band of ones that moves from top/right to bottom/left.

Here is an alternative calculation. **i** indicates the location of the identity matrix in **X**, and **j** indicates the location for **Y**.

```
CP0[i_, j_, ℒx_, ℒy_] := Module[{k, ka, kb, z},
   z = Table[0, {3}, {3}];
   k = j - i; (*offset*)
   ka = Max[1, k + 1];
   kb = Min[ℒx, ℒy + k];
   If[ka > kb, Return[z]];
   Table[Sum[ii^d1 (ii - k)^d2, {ii, ka, kb}], {d1, 0, 2}, {d2, 0, 2}]];
Table[{j, CP0[5, j, 4, 3] // MatrixForm}, {j, 2, 9}] // Transpose //
  TableForm // Style[#, FontSize → 10] &
```

```
2           3           4           5            6            7           8           9
0 0 0       1 3 9       2  5 13      3  6 14      3  6  14     2  3  5     1  1  1    0 0 0
0 0 0       1 3 9       3  8 22      6 14 36      9 20  50     7 11 19     4  4  4    0 0 0
0 0 0       1 3 9       5 14 40     14 36 98     29 70 184    25 41 73    16 16 16    0 0 0
```

In **CP0** the geometric sums in the matrix given by

```
Table[Sum[ii^d1 (ii - k)^d2, {ii, ka, kb}], {d1, 0, 2}, {d2, 0, 2}]
```

are evaluated in full form (with the number of iterations approximately equal to `Min[ℒx, ℒy]`). It is very useful to have closed-form representations for these sums. Below is a grid with rows indexed by **`d1`** and columns indexed by **`d2`**.

```
Clear[i, k, ka, kb, ia, ib];

t = Table[Sum[i^d1 (i - k)^d2, {i, ia, ib}], {d1, 0, 2}, {d2, 0, 2}];

Grid[t, Frame → All, ItemStyle → {FontSize → 12}, ItemSize → 13,
  Alignment → {Center, Center}]
```

$1 - ia + ib$	$-\frac{1}{2}(-1 + ia - ib)(ia + ib - 2k)$	$-\frac{1}{6}(-1 + ia - ib)(-ia + 2ia^2 + ib + 2\,ia\,ib + 2ib^2 - 6\,ia\,k - 6\,ib\,k + 6k^2)$
$-\frac{1}{2}(-1 + ia - ib)(ia + ib)$	$-\frac{1}{6}(-1 + ia - ib)(-ia + 2ia^2 + ib + 2\,ia\,ib + 2ib^2 - 3\,ia\,k - 3\,ib\,k)$	$-\frac{1}{12}(-1 + ia - ib)(-3ia^2 + 3ia^3 + 3ia^2\,ib + 3ib^2 + 3\,ia\,ib^2 + 3ib^3 + 4\,ia\,k - 8ia^2\,k - 4\,ib\,k - 8\,ia\,ib\,k - 8ib^2\,k + 6\,ia\,k^2 + 6\,ib\,k^2)$
$-\frac{1}{6}(-1 + ia - ib)(-ia + 2ia^2 + ib + 2\,ia\,ib + 2ib^2)$	$-\frac{1}{12}(-1 + ia - ib)(-3ia^2 + 3ia^3 + 3ia^2\,ib + 3ib^2 + 3\,ia\,ib^2 + 3ib^3 + 2\,ia\,k - 4ia^2\,k - 2\,ib\,k - 4\,ia\,ib\,k - 4ib^2\,k)$	$-\frac{1}{30}(-1 + ia - ib)(ia + ia^2 - 9ia^3 + 6ia^4 - ib - 2\,ia\,ib - 3ia^2\,ib + 6ia^3\,ib + ib^2 + 3\,ia\,ib^2 + 6ia^2\,ib^2 + 9ib^3 + 6\,ia\,ib^3 + 6ib^4 + 15ia^2\,k - 15ia^3\,k - 15ia^2\,ib\,k - 15ib^2\,k - 15\,ia\,ib^2\,k - 15ib^3\,k - 5\,ia\,k^2 + 10ia^2\,k^2 + 5\,ib\,k^2 + 10\,ia\,ib\,k^2 + 10ib^2\,k^2)$

(In the Matlab code, these calculations are implemented in the **`pSum`** method of the **`polynom`** class.)

Modifications and extensions used in Matlab code

Adjusting the lagged data matrix to conform to `[firstValid, lastValid]`

In the Matlab programing, the data vectors are implicitly defined on $i = 1, \ldots, \mathcal{T}$. The lagged data matrix X is assumed to have $\mathcal{T}$ rows. (In the VAR applications, $\mathcal{T}$ is the number of time units per day, $3600 \times 24 = 86, 400$ if the time units are seconds, and 86,400,000 if the time units are milliseconds.) Due to lags and incomplete data, though, not all rows of X are valid. The valid range is indicated as $0 < \mathit{firstValid} < \mathit{lastValid} \leq \mathcal{T}$. All routines are adjusted to confine calculations to these rows. This section uses the following conventions.

- **`i`** is the original position in the sparse data vector corresponding to x (i.e., at lag zero).
- **`a`** is an offset relative to **`i`**, where the PDL starts.
- Similarly, **`j`** and **`b`** are the original position for y and the offset for the y PDL.

`vTrim` removes all rows from **`X`** that are outside of `[firstValid, lastValid]`.

```
vTrim[X_] := Module[{d},
   d = Dimensions[X];
   If[Not[MatrixQ[X]] || lastValid > d[[1]] || firstValid < 1 ||
     firstValid > lastValid,
    Print["vTrim error."]; Return[Null]];
   X[[firstValid ;; lastValid]]];
Block[{𝒯 = 10, firstValid = 2, lastValid = 8},
 MatrixForm /@ {X[3, 4], vTrim[X[3, 4]]}]
```

```
 (0 0 0 0)
 (0 0 0 0)
 (1 0 0 0)    (0 0 0 0)
 (0 1 0 0)    (1 0 0 0)
 (0 0 1 0)    (0 1 0 0)
{(0 0 0 1) ,  (0 0 1 0)}
 (0 0 0 0)    (0 0 0 1)
 (0 0 0 0)    (0 0 0 0)
 (0 0 0 0)    (0 0 0 0)
 (0 0 0 0)
```

Here is a cross product calculation that uses `vTrim` to adjust the range.

```
Block[{𝒯 = 20, firstValid = 8, lastValid = 10, j = 10, a = 0, b = 0,
   ℒx = 4, ℒy = 3},
Table[{
      i,
      (*X[i+a,ℒx]//MatrixForm,
      Y[j+b,ℒy]//MatrixForm,
      vTrim[X[i+a,ℒx]]//MatrixForm,
      vTrim[Y[j+b,ℒy]]//MatrixForm,*)
      X[i + a, ℒx]ᵀ.Y[j + b, ℒy] // MatrixForm,
      vTrim[ X[i + a, ℒx] ]ᵀ.vTrim[ Y[j + b, ℒy] ] // MatrixForm,
      𝒟[ℒx, 2]ᵀ.vTrim[X[i + a, ℒx]]ᵀ.vTrim[Y[j + b, ℒy]].𝒟[ℒy, 2] //
       MatrixForm}, {i, 6, 13}] // Transpose // TableForm] //
 Style[#, FontSize → 9] &
```

```
6          7             8          9          10         11         12         13
(0 0 0)    (0 0 0)       (0 0 0)    (0 0 0)    (1 0 0)    (0 1 0)    (0 0 1)    (0 0 0)
(0 0 0)    (0 0 0)       (0 0 0)    (1 0 0)    (0 1 0)    (0 0 1)    (0 0 0)    (0 0 0)
(0 0 0)    (0 0 0)       (1 0 0)    (0 1 0)    (0 0 1)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (1 0 0)       (0 1 0)    (0 0 1)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (0 0 0)       (0 0 0)    (0 0 0)    (1 0 0)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (0 0 0)       (0 0 0)    (1 0 0)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (0 0 0)       (1 0 0)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (1 0 0)       (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    ( 1  1  1)    (1 1 1)    (1 1 1)    (1 1 1)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    ( 4  4  4)    (3 3 3)    (2 2 2)    (1 1 1)    (0 0 0)    (0 0 0)    (0 0 0)
(0 0 0)    (16 16 16)    (9 9 9)    (4 4 4)    (1 1 1)    (0 0 0)    (0 0 0)    (0 0 0)
```

The code used in Matlab adjusts the starting and ending values of the loop to conform to firstValid and lastValid:

```
CP1[i_, j_, a_, b_, n_, m_, dx_, dy_] := Module[{k, ka, kb, z},
   z = Table[0, {dx + 1}, {dy + 1}];
   k = (j + b) - (i + a); (*offset*)
   ka = Max[1, k + 1, firstValid - (i + a) + 1];
   kb = Min[n, m + k, lastValid - (j + b) + 1 + k];
   If[ka > kb, Return[z]];
   Table[Sum[ii^d1 (ii - k)^d2, {ii, ka, kb}], {d1, 0, dx}, {d2, 0, dy}]];
Block[{𝒯 = 20, firstValid = 8, lastValid = 10, j = 10, a = 0, b = 0,
   ℒx = 4, ℒy = 3},
  Table[{i, CP1[i, j, a, b, ℒx, ℒy, 2, 2] // MatrixForm}, {i, j - 4, j + 3}] //
    Transpose // TableForm] // Style[#, FontSize → 9] &
```

```
6           7              8           9           10          11          12          13
⎛0  0  0⎞   ⎛ 1   1   1 ⎞  ⎛1  1  1⎞   ⎛1  1  1⎞   ⎛1  1  1⎞   ⎛0  0  0⎞   ⎛0  0  0⎞   ⎛0  0  0⎞
⎜0  0  0⎟   ⎜ 4   4   4 ⎟  ⎜3  3  3⎟   ⎜2  2  2⎟   ⎜1  1  1⎟   ⎜0  0  0⎟   ⎜0  0  0⎟   ⎜0  0  0⎟
⎝0  0  0⎠   ⎝16  16  16 ⎠  ⎝9  9  9⎠   ⎝4  4  4⎠   ⎝1  1  1⎠   ⎝0  0  0⎠   ⎝0  0  0⎠   ⎝0  0  0⎠
```

Verify equivalence of calculation methods for a range of values ...

```
kStep = 0;
Block[{𝒯 = 30},
  Do[
   Block[{firstValid = kfv, lastValid = klv, i = ki, j = kj, a = kka,
     b = kkb, dx = 2, dy = 1, n = 4, m = 1},
    kStep = kStep + 1;
    If[Mod[kStep, 2500] == 0 , Print[kStep, " ", kfv, " ", klv,
      " ", ki, " ", kj, " ", kka]];
    If[kStep > 15 000, Break[]];
    z1 = CP1[i, j, a, b, n, m, dx, dy];
    z2 = 𝒟[n, dx]^T.vTrim[X[i + a, n]]^T.vTrim[X[j + b, m]].𝒟[m, dy];
    (*z2=123;*)
    If[z1 === z2, Continue[],
     Print["Whoops"] Print[z1 // MatrixForm, z2 // MatrixForm];
     Break[];];],
   {kfv, 5, 15, 5}, {klv, kfv + 10, 𝒯 - 5, 5}, {ki, Max[1, kfv - 5], 𝒯 - 8},
   {kj, Max[1, kfv - 5], 𝒯 - 8, 2}, {kka, 0, 2}, {kkb, 0, 2}]];
```

```
2500 5 20 4 5 2

5000 5 25 7 11 1

7500 10 20 16 21 0

10 000 15 25 18 20 0
```

Non-sparse data series

The sparse data vectors discussed to this point have all been sparse (like price changes). A price discovery VECM will also often include a constant term and/or a cointegration vector. These require special handling.

Constant terms

Many models will include constant terms. In the data matrix, this is usually represented as a column of ones. The constant crossed with itself is simply the number of observations, N. We might have $N = \mathcal{T}$, the length of the implicit data matrix. More generally, though, $N = lastValid - firstValid + 1$, where *firstValid* and *lastValid* are (as described earlier) pointers in X.

The cross product of the constant vector and a sparse data vector is the sum of the nonzero values.

Letting ι_l denote a column vector of l ones, the cross product of the constant vector and a quadratic PDL on a sparse data vector is:

$$D_x^{\mathrm{T}} X^{\mathrm{T}} \iota_{\mathcal{T}} = D_x^{\mathrm{T}} \left(\sum_{k=1}^{K} v_k \, \mathcal{I}(t_k + 1) \right)^{\mathrm{T}} \iota_{\mathcal{T}} = \sum_{k=1}^{K} v_k \, D_x^{\mathrm{T}} \, \mathcal{I}(t_k + 1)^{\mathrm{T}} \iota_{\mathcal{T}} = \sum_{k=1}^{K} v_k \, D_x^{\mathrm{T}} \iota_{\mathcal{L}}$$

where $\mathcal{L}$ is the length of the PDL. The $D_x^{\mathrm{T}} \iota_{\mathcal{L}}$ term is the sum (across columns) of the elements of $\mathcal{D}_x^{\mathrm{T}}$.

```
Block[{𝒯 = 10, ℒx = 4}, {MatrixForm[#] & /@ {
     X[5, ℒx],
     X[5, ℒx]ᵀ.Ones,
     𝒟[ℒx, 2]ᵀ,
     𝒟[ℒx, 2]ᵀ.X[5, ℒx]ᵀ.Ones}}] // TableForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 10 \\ 30 \end{pmatrix}$$

Sparse levels

Cointegrating vectors are not sparse in the usual sense (of having a small number of nonzero values). The difference between two price series $\Delta_t = p_{1\,t} - p_{2\,t}$ is not sparse, even if the price changes of component prices are sparse. Δ_t is sparse in a different sense, sometimes called piecewise sparse or level sparse. Its entries consist of constant levels that persist in time until one of component prices changes. The cross product calculations are adjusted accordingly.

Matlab notes

The Matlab code has two sparse matrix classes. **`spd`** matrices, used for price differences, are sparse in the usual sense. **`spl`** matrices are level sparse.

A VECM analysis involves four types of data series:

- constant vectors
- **`spd`** vectors (like price changes)
- **`spl`** vectors (like cointegrating vectors)
- **`spd`** vectors operated on by PDLs

The **`Crossproduct`** class has static methods (functions) to compute crossproducts involving all pairwise combinations of these types.