MVAR Classes Demo

Joel Hasbrouck

November 1, 2018

Companion notebook for Price Discovery in High Resolution.

This is a Matlab Live Script to demonstrate and test various classes used in the high-resolution programs:

- `spd` and `spl` are the essential sparse vector classes. `spd` ("sparse price difference") is a standard sparse vector class: only nonzero entries are recorded and stored. `spl` ("sparse price level") has the same properties as `spd`, but price levels are assumed to persist until explicitly changed. That is, an `spl` implicitly describes a piece-wise constant function, such as a price level.
- The `polynom` ("polynomial") class defines the polynomial distributed lag (PDL) structures.
- The `Crossproduct` class holds static methods that compute crossproducts involving `spd`, `spl`, PDL-weighted `spd`, and constant objects.

The calculations implemented in the `polynom` and `Crossproduct` classes are described in the Computational Appendix to the paper.

This script contains tests of the `Crossproduct` methods. For each test, the script computes a direct result (using full lag and PDL design matrices) and a sparse result (using the sparse computations in the `Crossproduct` class). If the maximum absolute difference between the two is nonzero, the program throws an error.

Initializations

```matlab
clc; clear all; close all;
format compact;
addpath('./mClasses','./mFiles')
rng('default')  %   initialize the random number generator
```

The `spd` class

An `spd` object represents a vector of sparse price differences, such as a high-resolution record of bid or ask changes. (Matlab has a general sparse matrix data type, but the present applications involve specialized calculations that are easier to implement in a user-defined class.)

In an `spd` object the implicit indexing goes from 1 to `maxSize`. A 24-hour record at a one-microsecond resolution would have:

```matlab
maxSize = 24*3600*10^6
```

```matlab
maxSize = 8.6400e+10
```

Because most analyses involve price observations over the same time interval and same resolution, it is useful to establish a default `maxSize`. Denoting this default by `T`, it is set by a static method:

```matlab
T=20;  %   for small demonstration
```
To declare an empty spd object:

```matlab
test = spd
```

The properties of an spd:

- `i` and `v` are vectors of indexes and values.
- `1 <= firstValid <= lastValid <= maxSize` are used to restrict the range of indices to those that correspond to valid/usable data. (A stock price vector might have observations over 24 hours, but for some analyses we're only considering 9:30-16:00, for example.)
- The `name` is used to meaningfully identify the series in output and so forth.

All properties are public. They may be accessed directly in code. (Except for setgetMax call, there are no 'set' and 'get' methods.)

The properties can also be initially set in the constructor called with arguments:

```matlab
spd(i,v,firstValid,lastValid,name);
```

The constructor accepts a shortened argument list and empty arguments:

```matlab
test = spd([2 3 18],[10 20 30],[],[],'xyzNBB')
```

The `sim` method populates an spd with random values. For example, to fill an spd at 70% average density:

```matlab
test.sim(0.7);
test
```
To generate the full (nonsparse) counterpart:

```matlab
xc = x.toCol
```

```matlab
xc = 20x1
    0
    81
    15
    43
    0
    92
    0
    0
    0
    80
    ...
```

The `lag` method generates lagged vectors:

```matlab
x.lag(2,0)
```

```matlab
ans =
    spd with properties:
        t: [4 5 6 8 12 13 15 19]
        v: [81 15 43 92 80 96 66 4]
    firstValid: 1
    lastValid: 20
    name: 'xyzNBB(t-2)'
    maxSize: 20
```

To visualize the correspondence between `x` and `x.lag(2)`, put them in column form and concatenate:

```matlab
[x.toCol x.lag(2).toCol]
```

```matlab
ans = 20x2
    0    NaN
    81   NaN
    15    0
    43    81
    0    15
    92    43
    0    0
    0    92
    0    0
    80    0
    ...
```
The default padding is nan; to pad with zeros:

```matlab
[x.toCol x.lag(2,0).toCol]
```

```
ans = 20×2
     0     0
     81    81
     15    15
     43    43
     0     0
     92    92
     0     0
     80    80
     .     .
```

Other methods are documented in the class definition.

**The spl class**

The spl (sparse price level) class is a subclass of spd. An spl represents a piecewise sparse function over 1,..., masSize: values are assumed to persist until superseded.

```matlab
rng(123) % reinitialize the random-number generator
p=spl;
p.sim(.4)
```

The toCol method for the spl class knows that prices persist between changes:

```matlab
pc=p.toCol
```

```
pc = 20×1
    NaN    NaN
    NaN    NaN
    NaN    NaN
    49     49
    40     40
    40     40
    35     35
    .     .
```

Cointegrated terms ("errors") are constructed as differences in spls

```matlab
q=spl;
q.sim(.6);
d=p.splMinus(q);
[p.toCol q.toCol d.toCol]
```
ans = 20×3  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  49  23  26  40  23  17  40  30  10  40  64  -24  35  64  -29  35  64  -29  . .

The polynom class

A polynom object represents a lag polynomial:

```matlab
p = polynom
```

```matlab
p =
    polynom with properties:
        deg: 0
        n: 1
        name: 'poly'
        kOffset: 0
        vNames: {'polyd0'}
```

The properties of a polynom are:

- `deg` the degree of the polynomial
- `n` the length (number of terms)
- `kOffset` The offset for this polynomial within a multiple-polynomial lag structure (see below)
- `name` and `vNames` contain the overall name of the polynomial and the coefficient names.

A polynomial of `deg=2` and `n=5` has the design matrix:

```matlab
p1 = polynom(2,5,'p1'); p1
```

```matlab
p1 =
    polynom with properties:
        deg: 2
        n: 5
        name: 'p1'
        kOffset: 0
        vNames: {'p1d0' 'p1d1' 'p1d2'}
```

```matlab
p1.designMatrix
```

```
ans = 3×5
     1     1     1     1     1
     1     2     3     4     5
```
Lag structures with multiple polynomial segments are defined by `polynom` arrays. In the following code, `pa` is set to a `polynom` array of length 2. There are two segments. The PDL is quadratic over lags 1-3 and linear over lags 4-7.

```plaintext
pa=[polynom(2,3) polynom(1,4,[],3)];
```

The overall lag structure is clearer in the design matrix:

```plaintext
pa.designMatrix
```

```
ans = 5×7
1     1     1     0     0     0     0
1     2     3     0     0     0     0
1     4     9     0     0     0     0
0     0     0     1     1     1     1
0     0     0     1     2     3     4
```

Note that the second `polynom` is offset by `kOffset=3` periods (the last argument in the constructor).

### The Crossproduct class

The Crossproduct class is a container for static methods used compute crossproducts of various types of variables. The methods have the following naming convention. They are all named `Crossproduct.UxV` where `U` and `V` are

- `const` (corresponding to a unit vector).
- `spd`
- `spl`
- `poly` (a PDL applied to an `spd`)

Not all possible pairings are needed. The is a method `Crossproduct.spdxconst`, which computes the crossproduct of `spd x const` (and also, of course, a `const x spd`). Sometimes `U` and `V` need to be swapped, and the result transposed.

For most choices of `U` and `V`, the crossproduct methods allow the calling routine to pass a "shift" argument. In microstructure applications, the `spd` and `spl` vectors are large. They are stored "unlagged". When crossproducts involving lagged values are needed, it is inefficient to generate many `spd`/`spl`s to contain the lags. In the crossproduct methods, the shift arguments do the lagging implicitly.

The code below computes crossproducts "directly" using full matrices and (alternatively) by using sparse methods. The program throws an error if the results don't agree.

### Simple cross products involving spds, spls and constant vectors

Construct and simulate some `spd`s.

```plaintext
T=20;
rng(789);
d1=spd; d1.sim;
```
d2=spd; d2.sim;
d1.toCol'*d2.toCol

ans = 8723

first=1; last=T;

Construct and simulate some spls.

p1=spl; p1.sim; p1c=p1.toCol;
p2=spl; p2.sim; p2c=p2.toCol;
[p1c p2c]

ans = 20×2
   NaN   NaN
   NaN   NaN
   NaN   NaN
   NaN    60
   80     3
   80     3
   56     3
   56     3
   56     3
   40     3
   ... ...

first=5; last=T;
p1c = p1c(first:last);
p2c = p2c(first:last);

Direct computation of cross product:

cpDirect = p1c'*p2c

cpDirect = 24656

... and computation via a crossproduct method:

cpSparse = Crossproduct.splxspl(p1,0,p2,0,first,last)

cpSparse = 24656

**Crossproduct.spdxconst**

The crossproduct here is x'i where x is an spd object and i is a constant (unit) vector.

T=1000;
first=20; last=T-5;
spd.setgetMax(T);
rng('default')
x=spd; x.sim(0.8);
first=10;
\texttt{last=95;}
\texttt{xc = x.toCol;}
\texttt{xc = xc(first:last,:);}
\texttt{cpDirect = sum(xc);}
\texttt{cpSparse = Crossproduct.spdxconst(x,0,first,last);}
\texttt{d = max(abs(cpDirect-cpSparse));}
\texttt{if d~=0; error('spdxconst 1'); end}

By setting the shift argument to 2, we are computing the crossproduct using $[x(t-2)]$

\texttt{xc=x.lag(2).toCol;}
\texttt{xc=xc(first:last,:);}
\texttt{cpDirect=sum(xc);}
\texttt{cpSparse=Crossproduct.spdxconst(x,2,first,last);}
\texttt{d = max(abs(cpDirect-cpSparse));}
\texttt{if d~=0; error('spdxconst 2'); end}

\textbf{Crossproduct.spdxspd}

\texttt{x'y where x and y are spd objects.}

\texttt{y = spd;}
\texttt{y.sim(0.8);}
\texttt{xc = x.toCol;}
\texttt{xc = xc(first:last,:);}
\texttt{yc = y.toCol;}
\texttt{yc = yc(first:last,:);}
\texttt{cpDirect = xc' * yc;}
\texttt{cpSparse = Crossproduct.spdxspd(x,0,y,0,first,last);}
\texttt{d = max(abs(cpDirect-cpSparse));}
\texttt{fprintf('direct: %d; sparse: %d; max abs diff %g\n',cpDirect,cpSparse,d);}

\texttt{direct: 77090; sparse: 77090; max abs diff 0}
\texttt{if d~=0; error('spdxspd 1'); end}

The crossproduct of $x(t-2)$ and $y(t-3)$:

\texttt{xc=x.lag(2).toCol;}
\texttt{xc=xc(first:last,:);}
\texttt{yc=y.lag(3).toCol;}
\texttt{yc=yc(first:last,:);}
\texttt{cpDirect = xc' * yc;}
\texttt{cpSparse = Crossproduct.spdxspd(x,2,y,3,first,last);}
\texttt{d = max(abs(cpDirect-cpSparse));}
\texttt{fprintf('direct: %d; sparse: %d; max abs diff %g\n',cpDirect,cpSparse,d);}

\texttt{direct: 58448; sparse: 58448; max abs diff 0}
\texttt{if d~=0; error('spdxspd 2'); end}
Crossproduct.splxconst

\( x'i \) where \( x \) is an spl object and \( i \) is a (constant) unit vector.

\[
\text{fprintf('Testing splxconst....\n')}
\]

Testing splxconst....

\[
xl = \text{spl};
xl.\text{sim}(0.8);
xc = xl.\text{toCol};
xc = xc(\text{first:}\text{last,:});
\text{cpDirect} = \text{sum}(xc);
\text{cpSparse} = \text{Crossproduct.splxconst}(xl,0,\text{first,}\text{last});
d = \text{max(abs(cpDirect-cpSparse))};
\text{fprintf('direct: \%d; sparse: \%d; max abs diff \%g\n',cpDirect,cpSparse,d)};
\]

direct: 4067; sparse: 4067; max abs diff 0

\textbf{if} d\neq0; \text{error('splxconst 1'); \textbf{end}}

The sum of \( x(t-2) \):

\[
xc=xl.\text{lag}(2).\text{toCol};
xc=xc(\text{first:}\text{last,:});
\text{cpDirect} = \text{sum}(xc);
\text{cpSparse} = \text{Crossproduct.splxconst}(xl,2,\text{first,}\text{last});
d = \text{max(abs(cpDirect-cpSparse))};
\text{fprintf('direct: \%d; sparse: \%d; max abs diff \%g\n',cpDirect,cpSparse,d)};
\]

direct: 3981; sparse: 3981; max abs diff 0

\textbf{if} d\neq0; \text{error('splxconst 2'); \textbf{end}}

Crossproduct.splxspd

\( x'y \) where \( x \) is an spl and \( y \) is an spd

\[
xc = xl.\text{toCol};
xc = xc(\text{first:}\text{last,:});
yc = y.\text{toCol};
yc = yc(\text{first:}\text{last,:});
\text{cpDirect} = xc' * yc;
\text{cpSparse} = \text{Crossproduct.splxspd}(xl,0,y,0,\text{first,}\text{last});
d = \text{max(abs(cpDirect-cpSparse))};
\text{fprintf('direct: \%d; sparse: \%d; max abs diff \%g\n',cpDirect,cpSparse,d)};
\]

direct: 126049; sparse: 126049; max abs diff 0

\textbf{if} d\neq0; \text{error('splxspd 1'); \textbf{end}}
... and with shifts.

```matlab
xc = xl.lag(3).toCol;
xc = xc(first:last,:);
yc = y.lag(4).toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.splxspd(xl,3,y,4,first,last);
d = max(abs(cpDirect-cpSparse));
fprintf('direct: %d; sparse: %d; max abs diff %g\n',cpDirect,cpSparse,d);
```

direct: 100715; sparse: 100715; max abs diff 0

```matlab
if d~=0; error('splxspd 2'); end
```

**Crossproduct.splxspl**

\(x'y\) where x and y are both spl objects.

```matlab
xc = xl.toCol;
xc = xc(first:last,:);
yl = spl;
yl.sim(0.8);
yc = yl.toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.splxspl(xl,0,yl,0,first,last);
d = max(abs(cpDirect-cpSparse));
fprintf('direct: %d; sparse: %d; max abs diff %g\n',cpDirect,cpSparse,d);
```

direct: 218650; sparse: 218650; max abs diff 0

```matlab
if d~=0; error('splxspl 1'); end
```

```matlab
xc = xl.lag(3).toCol;
xc = xc(first:last,:);
yc = yl.lag(4).toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.splxspl(xl,3,yl,4,first,last);
d = max(abs(cpDirect-cpSparse));
fprintf('direct: %d; sparse: %d; max abs diff %g\n',cpDirect,cpSparse,d);
```

direct: 199997; sparse: 199997; max abs diff 0

```matlab
if d~=0; error('splxspl 2'); end
```

**Polynomial crossproduct calculations**
These calculations involve spd and polynom objects. The crossproducts involve $x_m D'$ where $x_m$ is the matrix of lagged values (of $x$, an spd object) and $D$ is the design matrix of a polynomial distributed lag. See the computational appendix.

For extra confidence, some of these tests loop to generate randomly varying test situations

**Crossproduct.polyxconst**

```matlab
rng(789)
T=100; spd.setgetMax(T);
x = spd.*x.sim(.5);
px = horzcat( polynom(2,10,'px1'), polynom(2,5,'px2',3) );
PDL_Design_Matrix=px.designMatrix

PDL_Design_Matrix = 6x10
    1     1     1     1     1     1     1     1     1     1
    1     2     3     4     5     6     7     8     9    10
    1     4     9    16    25    36    49    64    81   100
    0     0     0     1     1     1     1     1     0     0
    0     0     0     1     2     3     4     5     0     0
    0     0     0     1     4     9    16    25    0     0

n = 10;
xc = lagm(x.toCol,n-1) * px.designMatrix';
first = find(~any(isnan(xc),2),1); last=T-5;
xc = xc(first:last,:);
cpDirect = sum(xc);
cpSparse = Crossproduct.polyxconst(x,px,0,first,last);
d = max(abs(cpDirect-cpSparse));
% disp(['direct: ' num2str(cpDirect)]);
% disp(['sparse: ' num2str(cpSparse)]);
fprintf('max abs diff %g
',d);

max abs diff 0

if d~=0; error('polyxconst 1'); end
```

```matlab
xc = lagm(x.lag(2).toCol,n-1,0) * px.designMatrix';
first=12;
xc = xc(first:last,:);
cpDirect = sum(xc);
cpSparse = Crossproduct.polyxconst(x,px,2,first,last);
disp(['direct: ' num2str(cpDirect)]);
direct: 19413 107263 752313 9769 29569 109005
```

```matlab
disp(['sparse: ' num2str(cpSparse)]);
sparse: 19413 107263 752313 9769 29569 109005
```
\[ d = \max(\text{abs}(cpDirect - cpSparse)); \]
\[ \text{fprintf('max abs diff %g
',d);} \]

\[
\text{if } d \neq 0; \text{ error('polyxconst 2'); end }
\]

**Crossproduct.polyxspd**

Test with one-position arrays.

\[
y = \text{spd}(5,2); \\
np = 3; \\
px = \text{polynom}(0,np); \\
xl = \text{lagm}(x.toCol,np-1,0)
\]

\[
xl = \begin{pmatrix}
0 & 0 & 0 \\
13 & 0 & 0 \\
89 & 13 & 0 \\
0 & 89 & 13 \\
72 & 0 & 89 \\
0 & 72 & 0 \\
0 & 0 & 72 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
\]

\[
xld = xl*px.designMatrix'; \\
\text{first} = \text{find}(\neg \text{any(isnan(xld)),2),1); \\
\text{last} = T; \\
yc = y.toCol; \\
\text{[xl xld yc]}
\]

\[
\text{ans} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 \\
13 & 0 & 0 & 13 & 0 \\
89 & 13 & 0 & 102 & 0 \\
0 & 89 & 13 & 102 & 0 \\
72 & 0 & 89 & 161 & 2 \\
0 & 72 & 0 & 72 & 0 \\
0 & 0 & 72 & 72 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

\[
\text{cpDirect} = xld(\text{first}\colon\text{last},:)'*yc(\text{first}\colon\text{last},:)
\]

\[
\text{cpDirect} = 322
\]

\[
\text{cpSparse} = \text{Crossproduct.polyxspd(x,px,0,y,0,1,T)}
\]
\[
\text{cpSparse} = 322
\]

\[
d = \max(\max(\text{abs(cpDirect-cpSparse))));
\text{fprintf('max abs diff %g\n',d);}
\]

max abs diff 0

\[
\text{if } d\neq 0; \text{ error('nonzero diff for polyxspd test 1.1'); end}
\]

\[
\text{px} = \text{horzcat}(\text{polynom}(0,3,'px1'),\text{polynom}(0,2,'px2',3)); \text{ px.designMatrix}
\]

\[
\text{ans} = 2 \times 5
\]

\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

\[
\text{xl} = \text{lagm}(\text{x.toCol},3+2-1,0);
\text{xld} = \text{xl} * \text{px.designMatrix}';
\text{first} = \text{find(}-\text{any(isnan(xld)),2),1);
\text{last} = \text{T};
\text{yc} = \text{y.toCol};
\text{cpDirect=} \text{xld(first:last,:)}'*\text{yc(first:last,:)}
\]

\[
\text{cpDirect} = 2 \times 1
\]

\[
\begin{array}{cccc}
322 & 26 & 26
\end{array}
\]

\[
\text{cpSparse} = \text{Crossproduct.polyxspd}(\text{x},\text{px},0,\text{y},0,1,\text{T})
\]

\[
\text{cpSparse} = 2 \times 1
\]

\[
\begin{array}{cccc}
322 & 26 & 26
\end{array}
\]

\[
d = \max(\max(\text{abs(cpDirect-cpSparse))));
\text{fprintf('max abs diff %g\n',d);}
\]

max abs diff 0

\[
\text{if } d\neq 0; \text{ error('nonzero diff for polyxspd test 1.2'); end}
\]

\[
y = \text{spd([5 7],[2 3])}
\]

\[
y =
\text{spd with properties:}
\]

\[
i: [5 7]
v: [2 3]
\text{firstValid}: 1
\text{lastValid}: 100
\text{name}: 'spd'
\]
yc = y.toCol;

\[
\text{cpDirect} = \text{xld(first:last,:)'}*\text{yc(first:last,:)}
\]

\[
\text{cpDirect} = \begin{bmatrix} 538 \\ 293 \end{bmatrix}
\]

cpSparse = Crossproduct.polyxspd(x, px, 0, y, 0, 1, T)

\[
\text{cpSparse} = \begin{bmatrix} 538 \\ 293 \end{bmatrix}
\]

fprintf('max abs diff %g\n', max(abs(cpDirect-cpSparse), [], 'all'));

max abs diff 0
d = max(max(abs(cpDirect-cpSparse)));

fprintf('max abs diff %g\n', d);

max abs diff 0

if d ~= 0; error('nonzero diff for polyxspd test 1.3'); end

rng(123);
for iSim=1:10
    fprintf('polyxspd random test 1.%d\n', iSim)
    xi = randperm(T, 10);
    xv = randi([10, 20], 1, 10);
    x = spd(xi, xv);
    y = spd(1, 2);
    xl = lagm(x.toCol, 3+2-1, nan);
    xld = xl*px.designMatrix';
    first = find(~any(isnan(xld), 2), 1);
    last = T;
    yc = y.toCol;
    cpDirect = xld(first:last,:)'*yc(first:last,:);
    cpSparse = Crossproduct.polyxspd(x, px, 0, y, 0, first, last);
    d = max(max(abs(cpDirect-cpSparse)));
    % fprintf('max abs diff %g\n', d);
    if d ~= 0; error(sprintf('nonzero diff for polyxspd random test 1.1.%d', iSim)); end
end

polyxspd random test 1.1
polyxspd random test 1.2
polyxspd random test 1.3
polyxspd random test 1.4
polyxspd random test 1.5
polyxspd random test 1.6
polyxspd random test 1.7
T=100; spd.setgetMax(T);
x=spd; x.sim(.5);
y=spd; y.sim(.5);
px=horzcat(polynom(0,3,'px1'),polynom(0,2,'px2',3)); px.designMatrix

ans = 2x5
     1   1   1   0   0
     0   0   0   1   1

first=10; last=T-5;
xc = lagm(x.toCol,3+2-1,0) * px.designMatrix';
xc = xc(first:last,:);
yc = y.toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.polyxspd(x,px,0,y,0,first,last);
d = max(max(abs(cpDirect-cpSparse)));
fprintf('max abs diff %g
',d);

max abs diff 0

if d~=0; error('polyxspd 1'); end

direct:

disp(num2str(cpDirect,'%12d'))

69942
61273

disp('sparse: ')

sparse:

disp(num2str(cpSparse,'%12d'))

69942
\[ d = \max(\max(\text{abs}(\text{cpDirect}-\text{cpSparse}))) \]
\[
\text{fprintf('max abs diff %g
',d);
}
\]

\text{max abs diff 0}

\textbf{if } d \neq 0; \text{ error('polyxspd 2'); end}

\text{... without shifts}

n=10;
px = \text{horzcat( polynom(1,n,'px1'), polynom(2,5,'px2',3) );}
m = 20;
py = \text{horzcat( polynom(2,m,'py1'), polynom(1,10,'py2',10) );}
xc = \text{lagm(x.toCol,n-1) * px.designMatrix'};
xc = xc(first:last,:);
yc = y.toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = \text{Crossproduct.polyxspd(x,px,0,y,0,first,last)};
d = \max(\max(\text{abs}(\text{cpDirect}-\text{cpSparse})))
\%
\text{fprintf('max abs diff %g
',d);
\text{if } d \neq 0; \text{ error('polyxspd 3'); end}

\text{... with shifts}

xc = \text{lagm(x.lag(2).toCol,n-1) * px.designMatrix'};
yc = y.lag(3).toCol;
first = \text{find(~any(isnan([xc yc]),2),1)};
xc = xc(first:last,:);
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = \text{Crossproduct.polyxspd(x,px,2,y,3,first,last)};
d = \max(\max(\text{abs}(\text{cpDirect}-\text{cpSparse})))
\%
\text{fprintf('max abs diff %g
',d);
\text{if } d \neq 0; \text{ error('polyxspd 4'); end}

\textbf{Stress test (large T)}

T=10000;
spd.setgetMax(T);
last=T-100;
xx=spd; xx.sim(.4);
yy=spd; yy.sim(.6);
first = \text{max([yy.i(1),xx.i(1)])}+30

first = 33

px = \text{horzcat( polynom(1,10,'px1'), polynom(2,5,'px2',3) );}
n = 10;
py = \text{horzcat( polynom(2,20,'py1'), polynom(1,10,'py2',10) );}
```matlab
m = 20;
xxc = lagm(xx.toCol,n-1) * px.designMatrix';
xxc = xxc(first:last,:);
yyc = yy.toCol;
yyc = yyc(first:last,:);
cpDirect = xxc' * yyc;
cpSparse = Crossproduct.polyxspd(xx,px,0,yy,0,first,last);
disp('direct:');

direct:
disp(num2str(cpDirect,'%12d'))
    37542975
    206311809
    18863190
    56228893
    205589403
disp('sparse:')
sparse:
disp(num2str(cpSparse,'%12d'))
    37542975
    206311809
    18863190
    56228893
    205589403
d = max(max(abs(cpDirect-cpSparse)));
fprintf('max abs diff %g\n',d);
max abs diff 0
if d~=0; error('polyxspd 5'); end
direct:
disp(num2str(cpDirect,'%12d'))
    37698580
    207338132
    18921436
    56426268
    205929884```

disp('sparse:')
sparse:
disp(num2str(cpSparse,'%12d'))

37698580
207338132
18921436
56426268
205929884

d = max(max(abs(cpDirect-cpSparse)));
fprintf('max abs diff %g\n',d);

max abs diff 0

if d~=0; error('polyxspd 6'); end

with self (as in program)

T=20; spd.setgetMax(T);
x=spd;
x.i=1:T;
x.v=randi([10 50],1,T);
y=x.copy();
y.v=randi([10 50],1,T);
n=3;
px = polynom(0,n,'px',0);
xc = lagm(x.lag(1).toCol,n-1) * px.designMatrix';
first = find(~any(isnan(xc),2),1); last=T;
xc = xc(first:last,:);
yc = y.toCol;
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse=Crossproduct.polyxspd(x,px,1,y,0,first,last)

cpSparse = 41806

d = max(max(abs(cpDirect-cpSparse)));
fprintf('max abs diff %g\n',d);

max abs diff 0

if d~=0; error('polyxspd 7'); end

n=10;
px = horzcat( polynom(1,n,'px1'), polynom(2,5,'px2',3) );
px.designMatrix()

ans = 5x10
   1   1   1   1   1   1   1   1   1   1
   1   2   3   4   5   6   7   8   9  10
xc = lagm(x.lag(1,0).toCol,n-1);
xcp = xc * px.designMatrix';
first = find(~any(isnan(xc),2),1); last = T;
xcp = xcp(first:last,:);
yc = y.toCol;
yc = yc(first:last,:);
cpDirect = xcp' * yc

cpDirect = 5×1
    93609
    513003
    47602
    139719
    502469

cpSparse=Crossproduct.polyxspd(x,px,1,y,0,first,last)

cpSparse = 5×1
    93609
    513003
    47602
    139719
    502469

d = max(max(abs(cpDirect-cpSparse)));
fprintf('max abs diff %g\n',d);

max abs diff 0

if d~=0; error('polyxspd 8'); end

Crossproduct.polyxspl

T=1000; spd.setgetMax(T);
rng(124)
pxA = polynom(1,3,'px1');
pxA_Design_Matrix = pxA.designMatrix

pxA_Design_Matrix = 2×3
    1   1   1
    1   2   3

pxB = horzcat(polynom(1,3,'px1'),polynom(1,4,'px2',3));
pxB_Design_Matrix = pxB.designMatrix

pxB_Design_Matrix = 4×7
    1   1   1   0   0   0   0
    1   2   3   0   0   0   0
    0   0   0   1   1   1   1
    0   0   0   1   2   3   4
for iSim=1:10
    fprintf('polyxspl starting iSim=%d\n',iSim)
    x=spd; x.sim(.5);
    y=spl; y.sim(.4);
    xc = lagm(x.toCol,2,0)*pxA.designMatrix';
    yc=y.toCol;
    first = find(~isnan(yc),1)+10; last=T-10;
    cpDirect=xc(first:last,:)'*yc(first:last,:);
    cpSparse=Crossproduct.polyxspl(x,pxA,0,y,0,first,last);
    d = max(max(abs(cpDirect-cpSparse)));
    if d>0; error(sprintf('polyxspl 1 at iSim=%d',iSim)); end
    xc = lagm(x.toCol,6,0)*pxB.designMatrix';
    cpDirect=xc(first:last,:)'*yc(first:last,:);
    cpSparse=Crossproduct.polyxspl(x,pxB,0,y,0,first,last);
    d = max(max(abs(cpDirect-cpSparse)));
    if d>0; error(sprintf('polyxspl 2 at iSim=%d',iSim)); end
end

polyxspl starting iSim=1
polyxspl starting iSim=2
polyxspl starting iSim=3
polyxspl starting iSim=4
polyxspl starting iSim=5
polyxspl starting iSim=6
polyxspl starting iSim=7
polyxspl starting iSim=8
polyxspl starting iSim=9
polyxspl starting iSim=10

Crossproduct.polyxpoly
With single-point x and y arrays.

T=20; spd.setgetMax(T);
npx = 3;
px = polynom(0,npx);
npy = 2;
py = polynom(0,npy);
x=spd(4,3);
y = spd(5,2);
px=horzcat(polynom(0,3,'px1'),polynom(0,2,'px2',3)); px.designMatrix;
py=horzcat(polynom(0,2,'px1'),polynom(0,3,'px2',2)); py.designMatrix;
xl=lagm(x.toCol,3+2-1,0);
yl=lagm(y.toCol,2+3-1,0);
xld = xl*px.designMatrix';
yld = yl*py.designMatrix';
first = find(~any(isnan([xld yld]),2),1);
last = T;
cpDirect=xld(first:last,:)'*yld(first:last,:);
cpSparse=Crossproduct.polyxpoly(x,px,0,y,py,0,1,T);
\[ d = \max(\text{abs}(\text{cpDirect}-\text{cpSparse}),[],'all'); \]
\[ \text{fprintf('max abs diff %g\n',d);} \]

\[
\text{max abs diff 0}
\]

\[
\text{if } d=0; \text{ error('nonzero diff for polyxpoly test 1.1'); end}
\]

\[
m=10; \quad \% \text{ set to 1 for single-position x and y}
\]
\[
\text{for } i=1:100
\]
\[
\text{fprintf('polyxpoly randomized test 1.1.%d\n','i)}
\]
\[
\times=\text{spd(\text{randperm}(T,m),\text{randi([10 20],1,m))));}
\]
\[
\text{y=spd(\text{randperm}(T,m),\text{randi([10 20],1,m))));}
\]
\[
\text{xl=lagm(x.toCol,3+2-1,0)};
\]
\[
\text{yl=lagm(y.toCol,2+3-1,0)};
\]
\[
\text{xld = xl*px.designMatrix'};
\]
\[
\text{yld = yl*py.designMatrix'};
\]

\[
\text{first = find(~any(\text{isnan([xld yld]),2},1));}
\]
\[
\text{last = T};
\]
\[
\text{cpDirect=xld(\text{first:last,:})'*yld(\text{first:last,:});}
\]
\[
\text{cpSparse=Crossproduct.polyxpoly(x,px,0,y,py,0,1,T);}
\]
\[
\text{d = } \max(\text{abs(cpDirect-cpSparse),[],'all')};
\]
\[
\text{if } d=0; \text{ error('nonzero diff for polyxpoly test 1.1.%d','i); end}
\]
polyxpoly randomized test 1.1.100

Randomized tests

```
T=1000; spd.setgetMax(T); rng(321);
for i=1:100
    x=spd; x.sim(.5);
    y=spd; y.sim(.5);
    xl=lagm(x.toCol,3+2-1);
    yl=lagm(y.toCol,2+3-1);
    xld = xl*px.designMatrix';
    yld = yl*py.designMatrix';
    first = find(~any(isnan([xld yld]),2),1);
    last = T;
    cpDirect=xld(first:last,:)*yld(first:last,:);
    cpSparse=Crossproduct.polyxpoly(x,px,0,y,py,0,first,last);
    d = max(abs(cpDirect-cpSparse),[],'all');
    if d~=0; error('nonzero diff for polyxpoly test 1.2.%d',i); end
end
fprintf('Finished randomized tests of polyxpoly
')
```

Finished randomized tests of polyxpoly

```
T=1000; spd.setgetMax(T); rng(123);
x = spd; x.sim(.4);
y = spd; y.sim(.8);
n=4;
px = polynom(2,n,'px');
xc = lagm(x.toCol,n-1) * px.designMatrix';
m=3;
py = polynom(2,m,'py');
yc = lagm(y.toCol,m-1) * py.designMatrix';
first = find(~any(isnan([xc yc]),2),1);
last = T-10;
xc = xc(first:last,:);
yc = yc(first:last,:);
CPDirect = xc' * yc;
CPSparse = Crossproduct.polyxpoly(x,px,0,y,py,0,first,last);
d = max(max(abs(CPDirect-cPSparse)));
if d~=0; error('polyxpoly 2'); end

... with shifts
```

```
xc = lagm(x.lag(3).toCol,n-1) * px.designMatrix';
yc = lagm(y.lag(1).toCol,m-1) * py.designMatrix';
first = find(~any(isnan([xc yc]),2),1);
last = T-10;
xc = xc(first:last,:);
yc = yc(first:last,:);
```
\begin{verbatim}
cpDirect = xc' * yc;
cpSparse = Crossproduct.polyxpoly(x,px,3,y,py,1,first,last);
d = max(max(abs(cpDirect-cpSparse)));
if d~=0; error('polyxpoly 3'); end

... with arrays

px = horzcat( polynom(2,10,'px1'), polynom(2,5,'px2',3) );
n = 10;
xc = lagm(x.toCol,n-1) * px.designMatrix';
py = horzcat( polynom(2,20,'py1'), polynom(2,10,'py2',10) );
m = 20;
yc = lagm(y.toCol,m-1) * py.designMatrix';
first = find(~any(isnan([xc yc]),2),1);
xc = xc(first:last,:);
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.polyxpoly(x,px,0,y,py,0,first,last);
d = max(max(abs(cpDirect-cpSparse)));
% fprintf('max abs diff %g
',d);
if d~=0; error('polyxpoly 4'); end

... with arrays and shifts

xc = lagm(x.lag(3).toCol,n-1) * px.designMatrix';
yc = lagm(y.lag(1).toCol,m-1) * py.designMatrix';
first = find(~any(isnan([xc yc]),2),1);
xc = xc(first:last,:);
yc = yc(first:last,:);
cpDirect = xc' * yc;
cpSparse = Crossproduct.polyxpoly(x,px,3,y,py,1,first,last);
d = max(max(abs(cpDirect-cpSparse)));
if d~=0; error('polyxpoly 5'); end

End of demos and tests

defprintf("That's all folks!")

That's all folks!
\end{verbatim}