MAX-PLANCK-INSTITUT FÜR DEMOGRAFISCHE FORSCHUNG

MAX PLANCK INSTITUTE FOR DEMOGRAPHIC RESEARCH

New functionality in blockops, a Mata library for efficient operations on block matrices

Daniel C. Schneider

German Stata Conference, June 19, 2026
Munich, IBZ

blockops: Two Purposes

1. by-matblock operations
   - apply a function to (all) individual blocks of a block matrix
     => more readable, simpler code
     => fewer bugs
     => faster coding
2. multiplication of large sparse matrices with known zero blocks
   - matrix multiplication is a very costly operation
     => increase speed
     => save memory
   - to a smaller extent may even be applicable to inversion

Preliminaries: Pointer Variables ("pointers") in Mata

```
: m = J(3,3,3)
: p = &m
: p
  0x1bc7f2e0
: *p
[symmetric]
       1   2   3
    +--------------+
  1 |  3           |
  2 |  3   3       |
  3 |  3   3   3   |
    +--------------+
```

```
: *p = 27
: p = 0x1bc7dfc0
: *p
  27

: pmat = (&J(2,2,1), &J(2,5,2) \
          &J(3,2,3), &J(3,5,4) )
: pmat
                  1              2
    +-----------------------------+
  1 |  0x156b1800   0x155befc0    |
  2 |  0x155a6a90   0x155abcb0    |
    +-----------------------------+
```

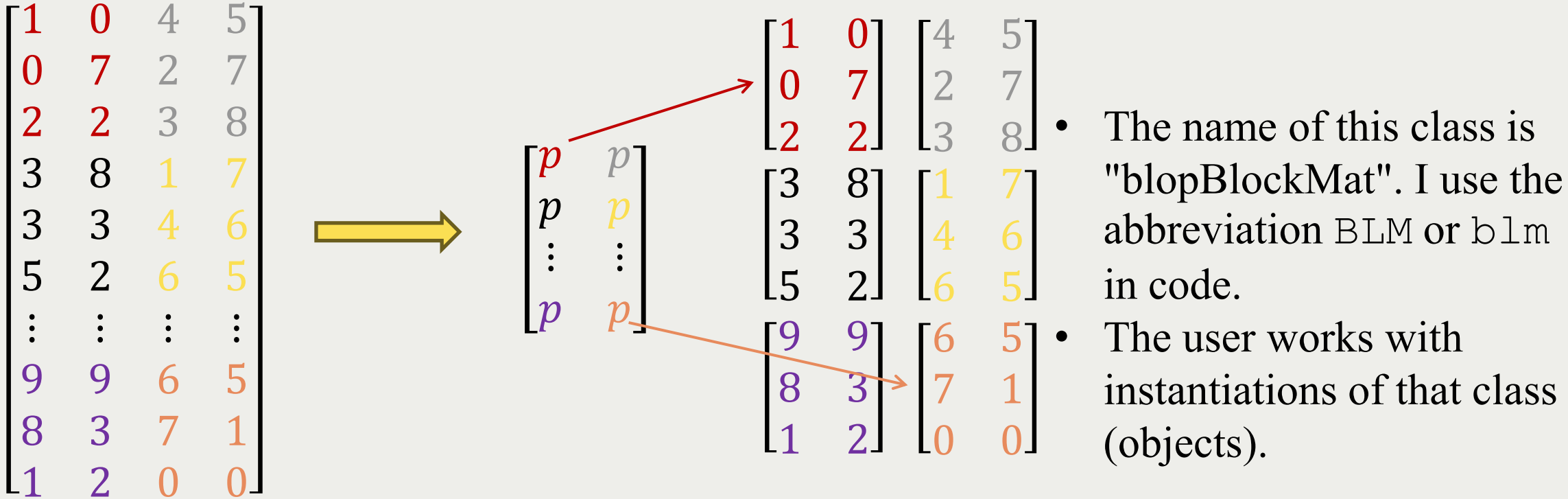
Preliminaries: Objects in Mata

```
: mat = rowshape(1::24, 4)
: mat
        1    2    3    4    5    6
    +-------------------------------+
  1 |   1    2    3    4    5    6  |
  2 |   7    8    9   10   11   12  |
  3 |  13   14   15   16   17   18  |
  4 |  19   20   21   22   23   24  |
    +-------------------------------+
: blm = blopBlockMat(1)
: blm._init(mat, 2, (1, 2, 5))
: blm.list()

  block matrix:

  1 |  2   3    4 |  5   6
  7 |  8   9   10 | 11  12
 ---+-------------+---------
 13 | 14  15   16 | 17  18
 19 | 20  21   22 | 23  24
```

```
: // blm.numibl // exists, but error
: blm.numibl()
  2

: pblm = &blm
: pblm->_repartition(1, 2)
: pblm->list()

block matrix:

 1   2   3 |  4   5   6
 7   8   9 | 10  11  12
13  14  15 | 16  17  18
19  20  21 | 22  23  24
```

Basic Idea

- matrix with logically related "regular" blocks
- separate blocks, and add a matrix of pointers, with pointers to each block

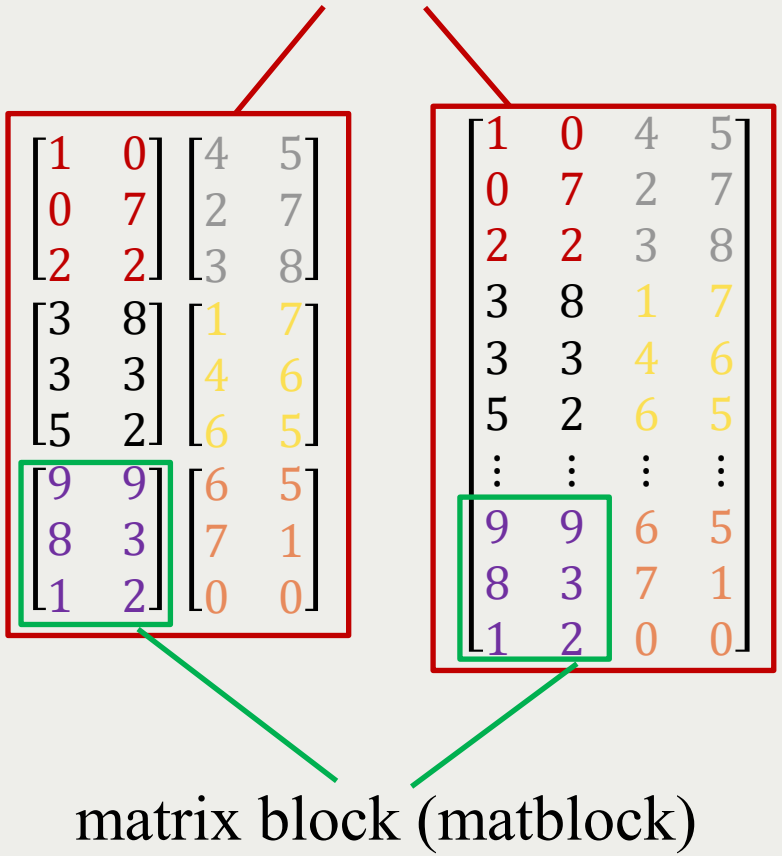


- The name of this class is "blopBlockMat". I use the abbreviation `BLM` or `blm` in code.
- The user works with instantiations of that class (objects).

Basic Idea

terminology:

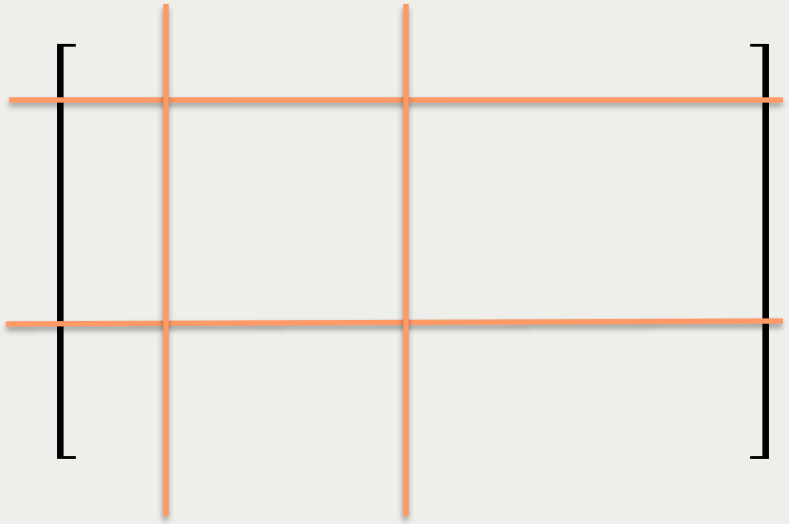
block matrix (blockmat)



matrix block (matblock)

"regular blocks":
global row and column separation

allowed:

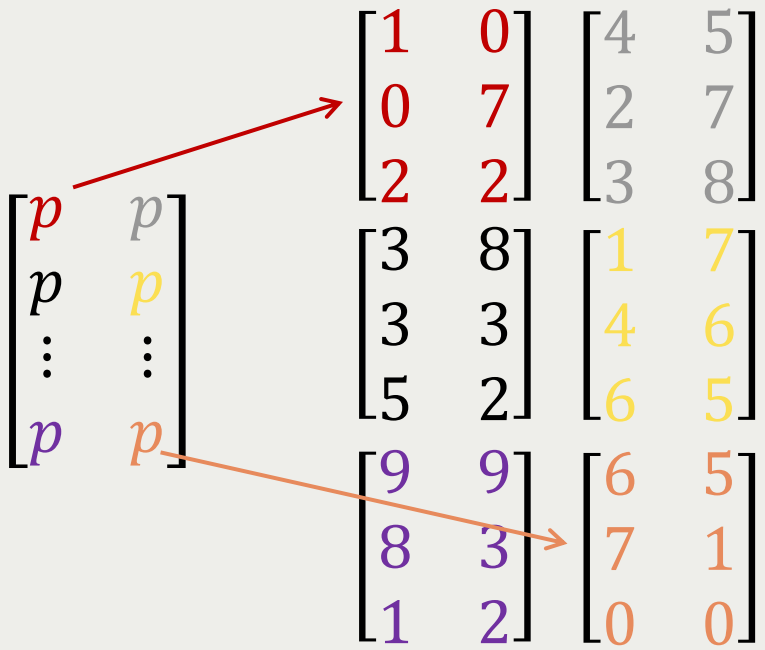


partially allowed:

Basic Idea

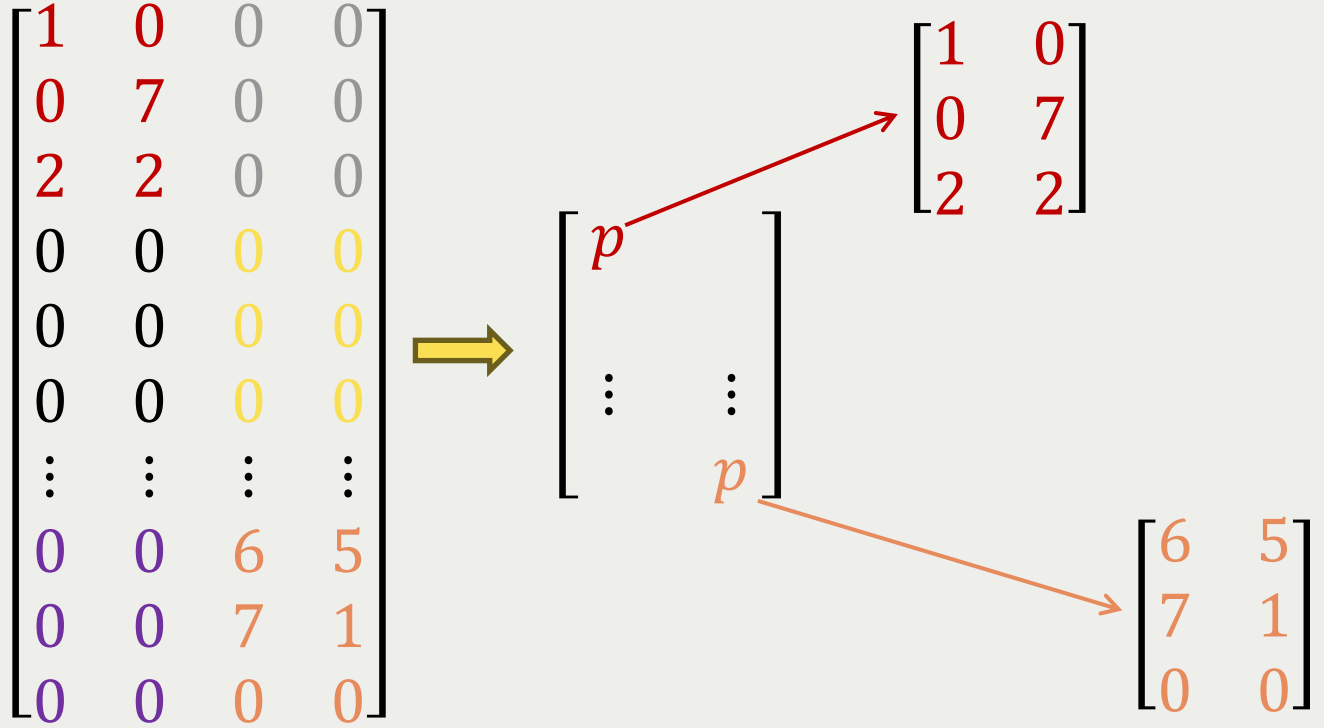
1) by-matblock ops:

apply function passed to object
to each matrix block separately



2) sparse matrix multiplication:

null pointers stand for
all-zero matrix blocks



I call this version of a blopBlockMat a "nullmat"

Functionality / Methods

```
class blopBlockMat {
    [...]
    // ****** SETTINGS AND INFO ******
    `TMT'  set()
    void   list()
    void   describe()

    // ****** OBJECT MANAGEMENT ******
    void   _init()      // 4 different init methods
    void   _pinit()
    void   _Jinit()
    void   _stinit()

    void    reset()
    void   _repartition()
    void   _copy()
    void   _fill()

    // ****** MAIN OPERATIONS ******
    void   _opin()      // ops in/within matblocks
    void   _opover()    // ops over/across matblocks
```

```
    `RMT'   rejoin() // piece blocks back together

    void   _subsetin()
    void   _subsetover()
    void   _assignin()
    void   _assignover()
    void   _assigndiag()
    void   _concatin()
    void   _concatover()

    void   _transpose()
    void   _multr()
    void   _multl()
    void   _add()
    void   _subtract()
    void   _power()

    void   _multc()
    void   _divc()
    void   _addc()
    void   _subtractc()
    void   _powerc()
}
```

Functionality / Methods

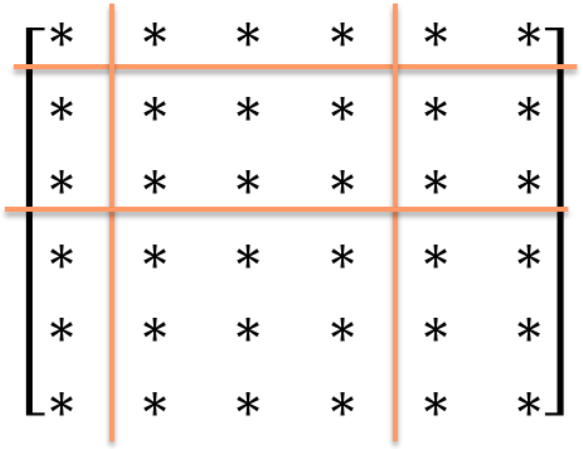
Many methods come in two variants:

```
class blopBlockMat {
    [...]
    // *** MAIN OPERATIONS ***
    void   _opin()
    void   _opover()
    [...]
}
```

```
class blopBlockMat {
    [...]
    // *** MAIN OPERATIONS ***
    void   _opin()
    `pBLM'  opin()
    void   _opover()
    `pBLM'  opover()
    [...]
}
```

Underscore methods: modify object, return nothing
Non-underscore methods: modify and return pointer to object
This enables **method chaining**.

Examples: By-Matblock Ops: In / Within Blocks



```
blm._init(mat,(1,2,4),(1,2,5))
```

- max of top row:

```
blm.subsetin(1,.)->opin("max")
>   ->rejoin()
```

- sort by 1. column:

```
blm._opin("sort",1)
```

- trace:

```
blm._opin("trace")
```

trace, manually:

```
blm.set("dimserror", 0)
blm.opin("diagonal")->opin("runningsum")
>   ->opin("sort",-1)->_subsetin(1,1)
```

- max of last row:

```
`RVE' lastrow(`RMT' submat) {
      return(submat[rows(submat),.])
}
blm.opin("lastrow")->_opin("max")
```

Examples: By-Matblock Ops: In / Within Blocks: $(X'X)^{-1}X'y$

```
// OLS BY LEVEL OF A VARIABLE
. sysuse auto, clear
. gen const = 1
. keep if !inlist(rep78, 1, .)
. sort rep78
.
. mata :
: X = blopBlockMat(1)
: y = blopBlockMat(1)
: X._stinit(., ("const", "price",
"weight"), ., "rep78", 1)
: X.list("blockdims")

matrix block dimensions:
   (08,02)
   (30,02)
   (18,02)
   (11,02)

: y._stinit(., ("mpg"), ., "rep78")
```

```
: XXinv = X.copy()->opin("cross")->opin("invsym")

: Xy = X.copy()->opin("cross", y)

: b = XXinv->opin("cross", Xy)->opin("transposeonly")
: b->list()

block matrix:

 44.9531   -.000105   -.007514
------------------------------
  34.052    .000014   -.004459
------------------------------
 34.9185    .000228   -.005099
------------------------------
 78.6478    .001252   -.025266

: end
```

Examples: By-Matblock Ops: Over / Across Blocks

e.g. 1 x 4 block matrix: 

- stack blocks: `blm.init(mat,1,4)->opover("vec")`

  compare: `colshape(rowshape(mat',4)[.,(1,4,7,2,5,8,3,6,9)],3))`
  (submats 3x3)

  `blm._init(mat,1,4)`

- reverse block order: `blm._subsetover(1,(4..1))`
- custom block order: `blm._subsetover(1,(3,1,4,2))`
- delete blocks: `blm._subsetover(1,(1,3))`
- duplicate blocks: `blm._subsetover(1,(1,1)#(1..4))`
- make block-diag: `blm._opover("diagonal")`

Examples: By-Matblock Ops: Age-State Markov Models

	p_{11}	p_{12}	p_{13}	p_{21}	p_{22}	p_{23}
50	.	.	.	.	.	.
60	0.95	0.04	0.01	0.35	0.61	0.04
70	0.93	0.06	0.01	0.25	0.68	0.06
80	0.86	0.10	0.04	0.15	0.74	0.12
90	0.67	0.18	0.16	0.06	0.69	0.25
100	.	.	1	.	.	1

Markov transition matrix

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{P}_{50} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{P}_{60} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{P}_{90} & \mathbf{0} \\ \boldsymbol{p}_{50} & \boldsymbol{p}_{60} & \cdots & \boldsymbol{p}_{90} & \mathbf{1} \end{bmatrix}$$

```
: mat = (0.95, 0.04, [...] )
: blm_tmp = blopBlockMat(1)
: blm_top = blopBlockMat(1)
: blm_bot = blopBlockMat(1)

: blm_tmp.init(mat[1::4, (1,4,2,5)], 4, 1)
>     ->_opin("rowshape", 2)

: blm_top.Jinit(J(2,2,0), 5, 5)
>     ->_assigndiag(blm_tmp, -1)

: blm_bot._init(mat[., (3,6)], 5, 1)
: blm_bot.opin("colshape", 1)->_transpose()

: blm_top.concatover(blm_bot, "bottom")->list()
   0     0 |    0     0 |    0     0 |    0     0 |    0     0
   0     0 |    0     0 |    0     0 |    0     0 |    0     0
-----------+-----------+-----------+-----------+-----------
0.95  0.35 |    0     0 |    0     0 |    0     0 |    0     0
0.04  0.61 |    0     0 |    0     0 |    0     0 |    0     0
-----------+-----------+-----------+-----------+-----------
   0     0 | 0.93  0.25 |    0     0 |    0     0 |    0     0
   0     0 | 0.06  0.68 |    0     0 |    0     0 |    0     0
-----------+-----------+-----------+-----------+-----------
   0     0 |    0     0 | 0.86  0.15 |    0     0 |    0     0
   0     0 |    0     0 | 0.10  0.74 |    0     0 |    0     0
-----------+-----------+-----------+-----------+-----------
   0     0 |    0     0 |    0     0 | 0.67  0.06 |    0     0
   0     0 |    0     0 |    0     0 | 0.18  0.69 |    0     0
-----------+-----------+-----------+-----------+-----------
0.01  0.04 | 0.01  0.06 | 0.04  0.12 | 0.16  0.25 |    1     1
```

Examples: nullmats: Age-State Markov Models

$$\boldsymbol{U} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \boldsymbol{U}_2 & \mathbf{0} & \mathbf{0} & & \mathbf{0} \\ 0 & \boldsymbol{U}_3 & \mathbf{0} & & \mathbf{0} \\ \vdots & & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{U}_{\bar{a}-1} & \mathbf{0} \end{bmatrix}$$

$$\boldsymbol{F} = (\boldsymbol{I} - \boldsymbol{U})^{-1} = \sum_{a=0}^{\bar{a}-2} \boldsymbol{U}^a = \begin{bmatrix} \boldsymbol{I} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{U}_2 & \boldsymbol{I} & & \mathbf{0} & \mathbf{0} \\ \boldsymbol{U}_3 \cdot \boldsymbol{U}_2 & \boldsymbol{U}_3 & & \mathbf{0} & \mathbf{0} \\ \vdots & & \ddots & & \vdots \\ \boldsymbol{U}_{\bar{a}-1} \cdot \ldots \cdot \boldsymbol{U}_2 & \boldsymbol{U}_{\bar{a}-1} \cdot \ldots \cdot \boldsymbol{U}_3 & \cdots & \boldsymbol{U}_{\bar{a}-1} & \boldsymbol{I} \end{bmatrix}$$

Examples: nullmats: IRFs of (S)VARs (Time Series)

Stata [TS] manual entry for **`irf create`**, Methods and Formulas

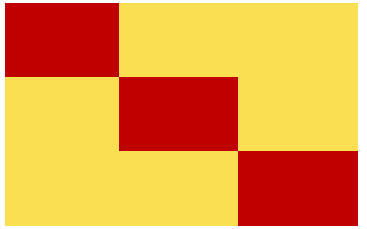
$$\mathbf{G}_i = \sum_{m=0}^{i-1} \mathbf{J}(\widehat{\mathbf{M}}')^{(i-1-m)} \otimes \widehat{\mathbf{\Phi}}_m$$

$$\mathbf{J} = (\mathbf{I}_K, \mathbf{0}_K, \ldots, \mathbf{0}_K)$$

$$\widehat{\mathbf{M}} = \begin{bmatrix} \widehat{\mathbf{A}}_1 & \widehat{\mathbf{A}}_2 & \cdots & \widehat{\mathbf{A}}_{p-1} & \widehat{\mathbf{A}}_p \\ \mathbf{I}_K & \mathbf{0}_K & \cdots & \mathbf{0}_K & \mathbf{0}_K \\ \mathbf{0}_K & \mathbf{I}_K & & \mathbf{0}_K & \mathbf{0}_K \\ \vdots & & \ddots & \vdots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \cdots & \mathbf{I}_K & \mathbf{0}_K \end{bmatrix}$$

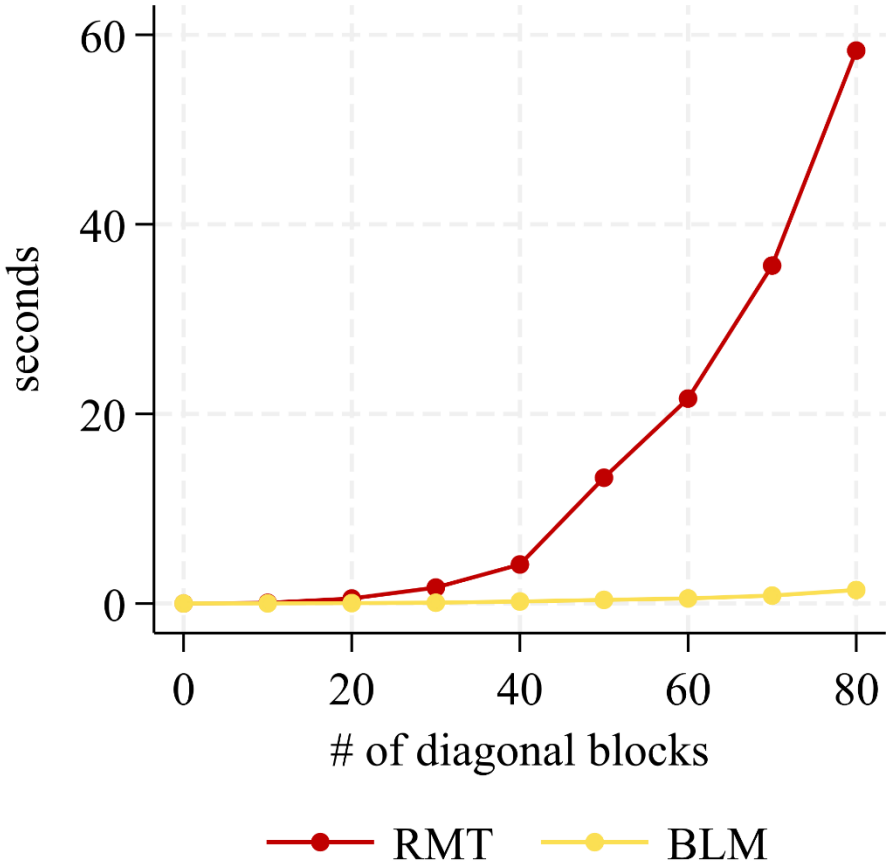
Examples: nullmats: Speed Comparison

self-multiplication of a block-diagonal matrix: increase # of diagonal blocks



10 multiplications
block size 50 x 50

# of diagonal blocks	size of entire block matrix	time ratio BLM / RMT
10	500 x 500	0.113
20	1000 x 1000	0.075
30	1500 x 1500	0.049
40	2000 x 2000	0.050
50	2500 x 2500	0.028
60	3000 x 3000	0.024
70	3500 x 3500	0.023
80	4000 x 4000	0.024



Settings

- `blm.s_nullmat` = 0/1/2/3:
  - 0: nullmats are not used
  - 1: "lazy": if a calculation results in a nullmat, it is kept and not converted to an all-zero matblock
  - 2: "eager": after an operation, all matblocks are actively searched for all-zero matrices and, if found, converted to nullmats.
  - 3: [to long to explain here]
  - default = 0
- `blm.s_dimserror` = 0/1: enforces conformability of matblocks (default: 1)
- `blm.s_interactive` = 0/1: the internal state of the object remains intact after errors (default=0)

Miscellaneous Remarks

- Not publicly available yet
- Colon operations: C-conformability with blopBlockMats (not covered in talk)
- No string operations
- opin(): By-matblock calls to built-in functions, library functions, or user-defined functions
  - Some built-in functions may be omitted from blockops. Here the user can write a wrapper to it and pass the name of the wrapper.
- opover(): Expand set of functions
- opin(): Passing information about each submatrix:
  - blockmatset class? Would hold two or more blopBlockMat objects.
  - Alternative: BLMs passed to `blm.opin()` are passed block-by-block.
  - User can always write a custom function.

Thank you

schneider@demogr.mpg.de