MAX-PLANCK-INSTITUT
FÜR DEMOGRAFISCHE
FORSCHUNG

MAX PLANCK INSTITUTE
FOR DEMOGRAPHIC
RESEARCH

blockops:
A new Mata library
for efficient operations on block matrices

Daniel C. Schneider

UK Stata Conference, September 11-12, 2025
London, University of Westminster

Preliminaries

- Code is still work-in-progress and not yet publicly available. Many things will change, even naming.
- 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   |
    +--------------+
```

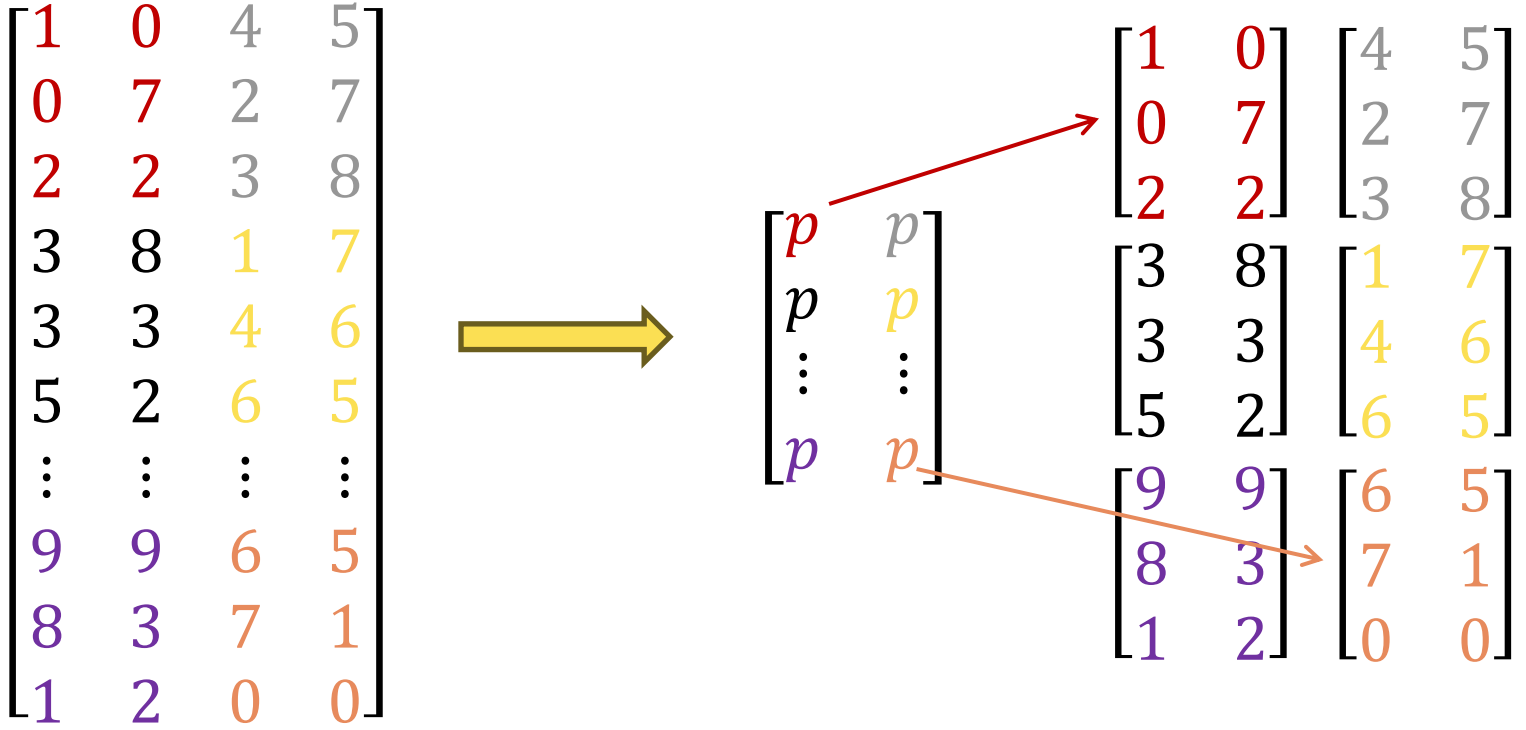
```
: *p = 27
: p = 0x1bc7dfc0
: *p
  27

: real scalar cube(arg1) {
>     return(arg1^3)
> }
: f = &cube()
: (*f)(7)
  343
```

blockops: Two Purposes

1. R apply()-style functionality
   - apply a function to individual blocks of a matrix
     => more readable code
     => fewer bugs
     => faster coding
2. matrix multiplication for special large sparce matrices
   - matrix multiplication is a very costly operation
     => increase speed
     => save memory
   - to a smaller extent may even be applicable to inversion

Basic Idea

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

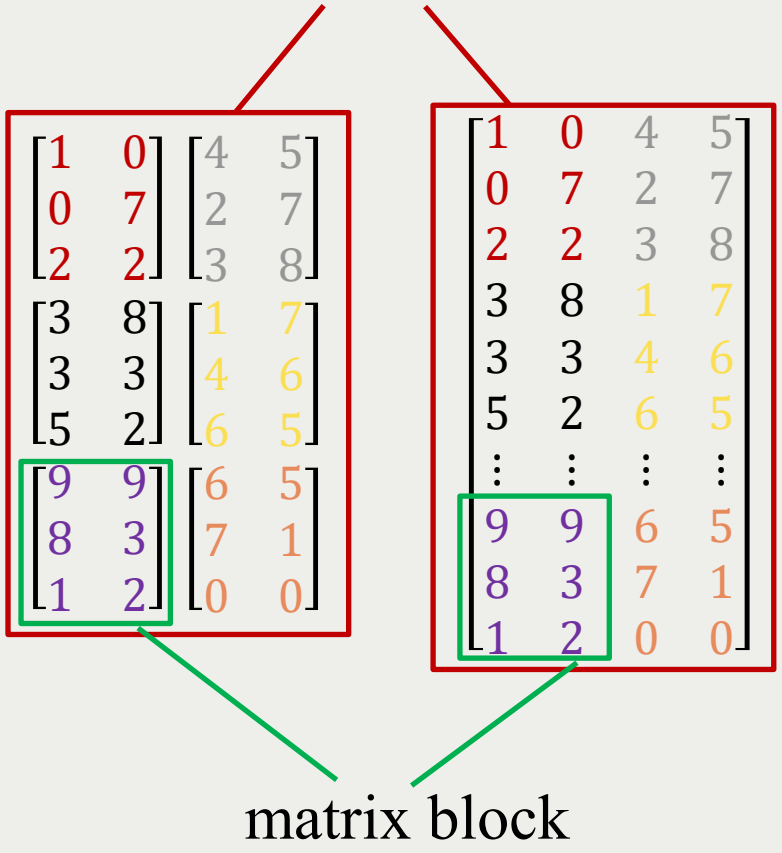


- the current name of this class is "blockopsmat"
- The user works with instantiations of that class (objects).

Basic Idea

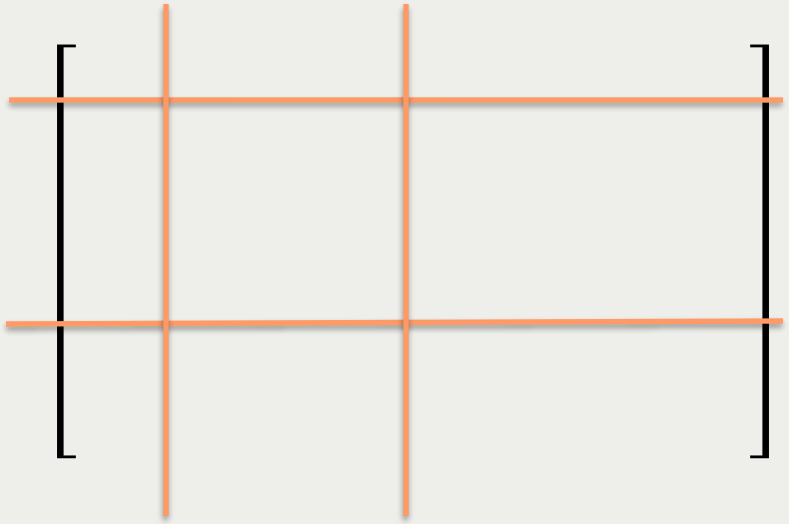
terminology:

block matrix

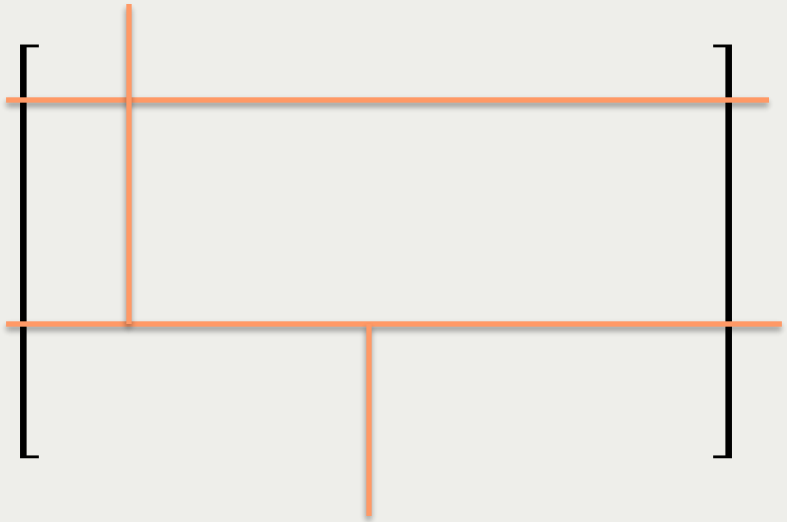


"regular blocks":
global row and column separation

allowed:



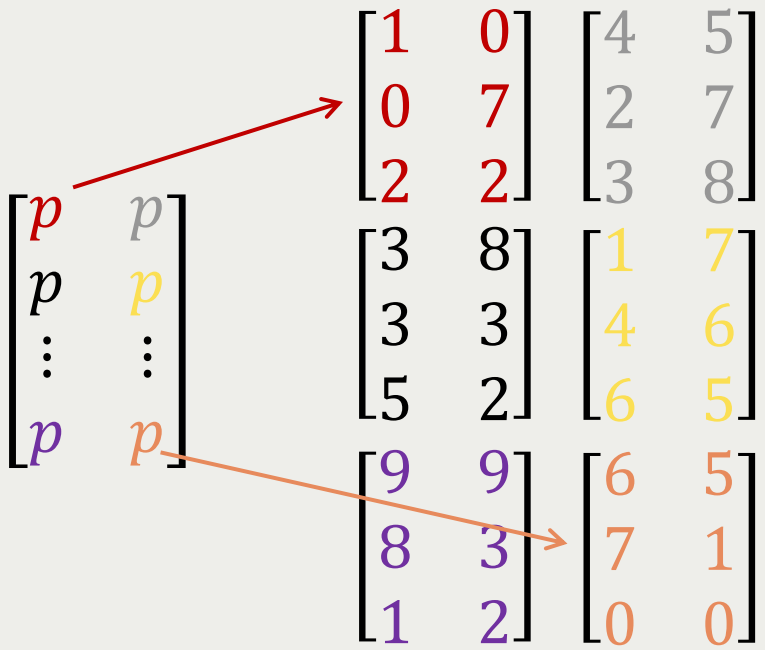
not allowed:



Basic Idea

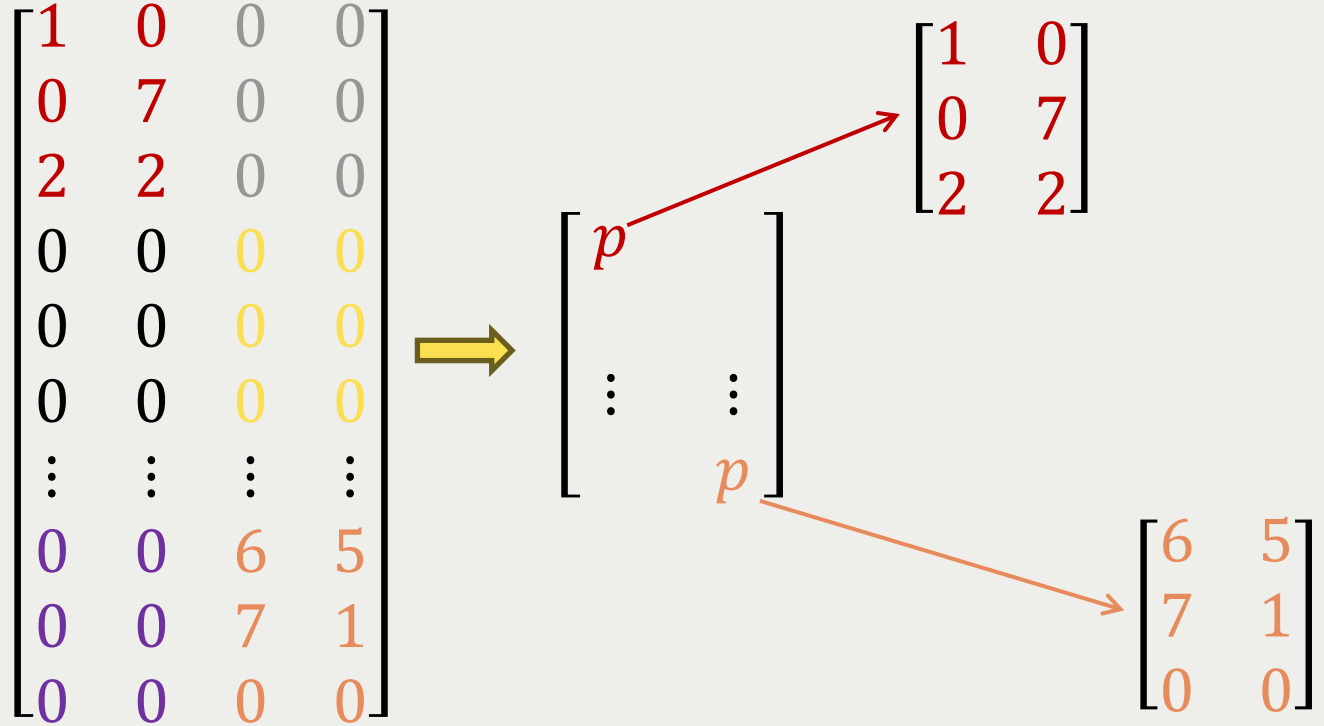
1) apply()-style functionality:

apply function passed to object
to each matrix block separately



2) multiplication:

null pointers stand for matrix
blocks that are all zero



I call this special blockopsmat a "nullmat"

Class Definition

```
class blockopsmat {

    private:
    // information stored in the object
    `RSC' numibl, numjbl
    `RMT' ilen, jlen

    `pMAT_RMT' pmat

    `bool' checkdims_setting
    `bool' dimsok

    public:
    // accessor functions
    `RSC' numibl()
    `RSC' numjbl()
    // [...]
```

Class Definition

```
class blockopsmat {

    // [...]

    // OBJECT MANAGEMENT
    void  new()
    void  reset()
    void  init()            // init(mat, 4), init(mat, 2, 4), init(mat, (1,3), (1,7,9))
    `BLM' copy()

    // CORE FUNCTIONALITY
    void  _opin()
    void  _opover()

    `RMT' rejoin()

    void  _subsetin()
    void  _subsetover()
    void  _repartition()

    // nullmats
    void  _mult()
    void  _add()
    void  _subtract()
```

Class Definition

```
class blockopsmat {

    // [...]

    // DIMENSION CHECKS
    `TMT'  checkdims_set()
    `bool' checkdims()
    void   _blockdims()

    // OBJECT INFO
    void list()
    void describe()
    `bool' equals()      // compares to another BLM
    `bool' identical()

    // NULLMATS
    // [...]

}
```

Class Definition

Core functions come in two variants:

```
class blockopsmat {

    // [...]

    // CORE FUNCTIONALITY
    void  _opin()
    void  _opover()

    void  _subsetin()
    void  _subsetover()
    void  _repartition()
    // [...]

}
```

```
class blockopsmat {

    // [...]

    // CORE FUNCTIONALITY
    void  _opin()
    `BLM' `opin()
    void  _opover()
    `BLM' `opover()
    // [...]

}
```

Underscore functions: modify object

Non-underscore funcs: return new object

This enables **method chaining**.

Examples

```
// DEVIATION FROM GROUP MEANS

: orig
        1
    +-----+
  1 |  2  |
  2 |  5  |
  3 |  8  |
  4 |  1  |
  5 |  6  |
  6 |  0  |
  7 |  3  |
  8 |  9  |
  9 |  4  |
 10 |  3  |
 11 |  9  |
 12 |  6  |
    +-----+
```

```
: blm = blockopsmat(1)
: blm.init(orig, 4, 1)
: blm.opin(&mean()).opin("J",3,1).rejoin() - orig

: blm.opin(&mean()).opin("J",3,1)
>    .subtract(orig).rejoin()

: vec(J(3, 1, mean(colshape(orig, 3)'))) - orig
                1
    +-----------------+
  1 |               3 |
  2 |               0 |
  3 |              -3 |
  4 |   1.333333333   |
  5 |  -3.666666667   |
  6 |   2.333333333   |
  7 |   2.333333333   |
  8 |  -3.666666667   |
  9 |   1.333333333   |
 10 |               3 |
 11 |              -3 |
 12 |               0 |
    +-----------------+
```

```
: blm._opin(&mean())
: blm._opin("J", 3, 1)
: blm._subtract(orig)
: blm.rejoin()
```

Examples

```
// MULTIPLICATION OF BLOCK DIAGONAL
// MATRICES

: orig = I(3) # (1, 9 \ 7, 4)
: orig
       1   2   3   4   5   6
    +-------------------------+
  1 |  1   9   0   0   0   0  |
  2 |  7   4   0   0   0   0  |
  3 |  0   0   1   9   0   0  |
  4 |  0   0   7   4   0   0  |
  5 |  0   0   0   0   1   9  |
  6 |  0   0   0   0   7   4  |
    +-------------------------+
: orig * orig
       1    2    3    4    5    6
    +-----------------------------+
  1 |  64   45    0    0    0    0  |
  2 |  35   79    0    0    0    0  |
  3 |   0    0   64   45    0    0  |
  4 |   0    0   35   79    0    0  |
  5 |   0    0    0    0   64   45  |
  6 |   0    0    0    0   35   79  |
    +-----------------------------+
```

```
: blm.nullmat_init(orig, 3, 3)

: blm.list("blockdims")
matrix block dimensions:
   (2,2) (2,2) (2,2)
   (2,2) (2,2) (2,2)
   (2,2) (2,2) (2,2)

: blm.list("matblocks")
matrix blocks:
[1,1]
       1   2
    +---------+
  1 |  1   9  |
  2 |  7   4  |
    +---------+
[1,2]
  0x0
[1,3]
  0x0
[2,1]
  0x0
[2,2]
       1   2
    +---------+
  1 |  1   9  |
  2 |  7   4  |
    +---------+
[2,3]
  0x0
[3,1]
```

```
: blm.mult(blm).rejoin()
       1    2    3    4    5    6
    +-----------------------------+
  1 |  64   45    0    0    0    0  |
  2 |  35   79    0    0    0    0  |
  3 |   0    0   64   45    0    0  |
  4 |   0    0   35   79    0    0  |
  5 |   0    0    0    0   64   45  |
  6 |   0    0    0    0   35   79  |
    +-----------------------------+
```

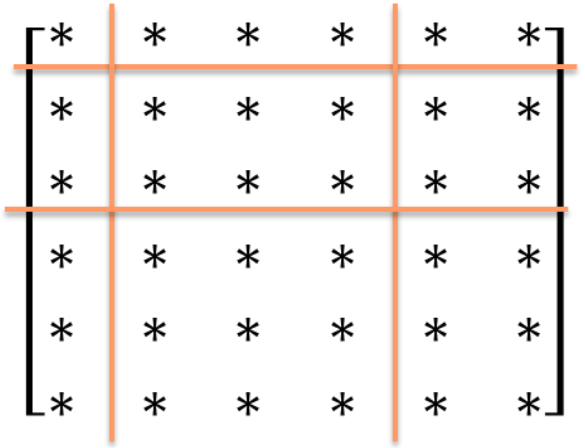
Complications

- conformability of matrix blocks
  - under `checkdims_setting = 1`, conformability is maintained
  - if inconsistency is found, an error occurs
  - the internal state of the object remains intact
  - under `checkdims_setting = 0`, the user can apply any function without conformability checks / errors

Complications

- Mata built-in v. library functions
  - unfortunately, different calling mechanisms
    - built-in: `blm.opin("funcname")`
    - library: `blm.opin(&funcname())`
  - user-written functions are passed like library functions, via a pointer
  - if a built-in function is omitted from blockops, the user can write a wrapper to it and pass a pointer to the wrapper
  - built-in scalar functions:
    e.g. `blm.opin("exp")` works
- code development: avoid digging through long lists of functions
- in-place functions

More Examples



R apply()-style functionality: **in / within blocks**

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

- sort by 1. column:
```
blm.opin(&sort(),1).rejoin()
```
- max of top row:
```
blm.subsetin(1,.).opin(&max()).rejoin()
```
- trace:
```
blm.checkdims_set(0)
blm.opin("diagonal").opin("runningsum")
>  .opin(&sort(),-1).subsetin(1,1).rejoin()
```
- $(X'X)^{-1}$:
```
blm.opin("cross").opin("invsym")
```
- max of last row:
```
`RVE' lastrow(`RMT' submat) {
     return(submat[rows(submat),.])
}
blm.opin(&lastrow()).opin(&max()).rejoin()
```

More Examples

R apply()-style functionality: **over / across blocks**

e.g. 1 x 4 block matrix: 

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

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

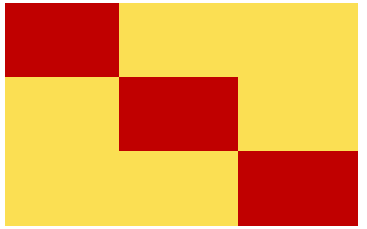
  `blm._init(mat,1,4)`

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

More Examples

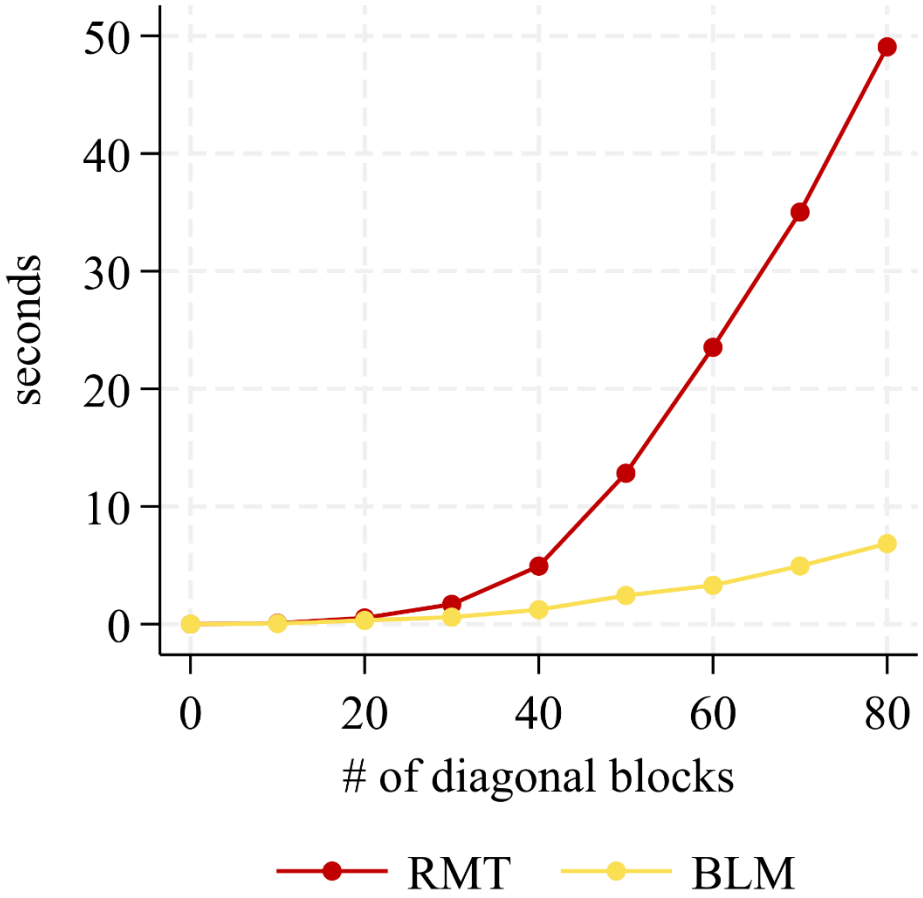
matrix multiplication for special large sparce matrices:

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	0.650
20	1000 x 1000	0.598
30	1500 x 1500	0.352
40	2000 x 2000	0.249
50	2500 x 2500	0.190
60	3000 x 3000	0.140
70	3500 x 3500	0.141
80	4000 x 4000	0.139



Outlook and Extensions

- R apply()-style functionality looks promising
- matrix multiplication for special large sparce matrices needs a lot more work
  - will divide the blockopsmat class into a regular and a nullmat class
- string blockopsmats
- processing information about each submatrix:
  - blockmatset class ? holds two or more blockopsmat objects
  - alternative: BLMs passed to `blm.opin()` are passed on block-by-block

Questions to Be Resolved

- best way to obtain a complete list of built-in Mata functions?
- easy way of making the built-in / library function distinction easier for the user?
- distributing classes: how to avoid trouble for the user if a class definition changes?

Thank you

schneider@demogr.mpg.de