amat Octave package, User's Guide*

version 0.1.2

François Cuvelier†

February 16, 2020

Abstract

This object-oriented Octave package allows to efficiently extend some linear algebra operations on array of matrices (with same size) as matrix product, determinant, factorization, solving, ...

0 Contents

*LaTeX manual, revision 0.1.2, compiled with Octave 5.2.0, and packages `fc-amat`[0.1.2], `fc-tools`[0.0.30], `fc-bench`[0.1.2]

†LAGA, UMR 7539, CNRS, Université Paris 13 - Sorbonne Paris Cité, Université Paris 8, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr.

This work was supported by the ANR project DEDALES under grant ANR-14-CE23-0005.

Initially the fc amat Octave package was created to be used with finite elements codes for computing volumes and gradiants of barycentric coordinates on each mesh elements. The volume of mesh element can be computed with the determinant of a matrix depending on the coordinates of the mesh element vertices. The gradients of the barycentric coordinates of a mesh element are solutions of linear systems. So we want to be able to do efficiently these operations on a very large number (few millions?) of very small matrices with same order (order less than 10?). In Octave, all theses matrices can be stored as a N-by-m-by-m 3D-array. Currently, with Octave from version 4.0.3 (and Matlab from release R2017a) only element-wise binary operators and functions can be used, as described in:

https://www.gnu.org/software/octave/doc/v5.2.0/Broadcasting.html

For example, the sum of a m-by-n matrix with all the N matrices in a N-by-m-by-n 3D-array can be performed as follows:

```
A=rand(m,n);      % generate a m-by-n matrix (n>1)
B=randn(N,m,n); % generate a N-by-m-by-n 3D-array
C=reshape(A,[1,m,n])+B; % generate "A+B" 3D-array
```

Unfortunately, simple operation as matrix product between a m-by-n matrix and all the N matrices in a N-by-n-by-p 3D-array or between all the N matrices of two 3D-arrays with sizes N-by-m-by-n and N-by-n-by-p are not implemented yet.

The purpose of this package is to give efficient operators and functions acting on amat object (array of matrices) to perform operations like sums, matrix product or more complex as determinants computation, factorization, solving, ... by only using Octave language. One can referred to [1] for more details, tests and benchmarks.

In the first section, the fc amat package is quickly presented. Thereafter, its installation process is described.

1 Presentation

The amat object provided in the fc amat package represents an array of matrices of the same order. All the following functions return an amat object with N matrices whose order is n × m or d × d:

amat(N,m,n)	constructor with all matrices to zeros
fc_amat.zeros(N,m,n)	same as amat(N,m,n)
fc_amat.ones(N,m,n)	matrices of 1
fc_amat.eye(N,d)	identity matrices
fc_amat.random.randn(N,m,n)	normally distributed random elements
fc_amat.random.randnsym(N,d)	randomized symmetric matrices
fc_amat.random.randnher(N,d)	randomized hermitian matrices
fc_amat.random.randntril(N,d)	randomized lower triangular matrices
fc_amat.random.randntriu(N,d)	randomized upper triangular matrices

...

The complete list of constructor and generating functions is given in section 4.

Let A be an amat object with N matrices whose order are m × n. In a

more condensed way we say that `A` is a `N` × `m` × `m` `amat` object. One can easily manipulate and edit its content by using indexing. Here is a small part of the offered possibilities. These are detailed in section 5.

`A(k,i,j)`	return element (`i`, `j`) of the `k`-th matrix
`A(k)`	return the `k` -th matrix (order `m` × `n`)
`A(i,j)`	return elements (`i`, `j`) of all the matrices as an `N`-by-1-by-1 `amat`
`A(k,i,j)=c`	assign `c` scalar value to element (`i`, `j`) of the `k`-th matrix
`A(i,j)=c`	assign `c` value to elements (`i`, `j`) of all the matrices
`A(k)=B`	assign the `m` × `n` matrix `B` to the `k`-th matrix

...

It should be noted that resizing objects can happen when one of the indices is larger than the corresponding dimension. In Listing 1, some examples are provided.

```
A=fc_amat.random.randn(100,3,4);% A: 100-by-3-by-4 amat
B=randn(3,4);
A(10)=B; % B assign to the 10-th matrix
A(20:25)=B; % the matrices 20 to 25 are set to B
A(30:2:36)=0; % the matrices 30,32,34 and 36 are set to 0
A(120)=1; % now A is a 120-by-3-by-4 amat ...
A(1,2)=0; % elements (1,2) of all the matrices are set to 0
A(2:3,3)=1; % elements (2,3) and (3,3) of all the matrices are set to 1
A(4,5)=1;  % now A is a 120-by-4-by-5 amat ...
A(5,1,2)=pi;  % element (1,2) of the 5-th matrix is set to pi
A(10:15,1,2)=1; % element (1,2) of the matrices 10 to 15 are set to 1
A(130,6,7)=1; % now A is a 130-by-6-by-7 amat ...
```

Listing 1: Assigments with `amat` object

The `amat` class is provided with the usual elementary operations:

- `+` , `-` , `.*` , `./` , `.\` , , `.^` . (Arithmetic opertors)
- `==` , `>=` , `>` , `<=` , `<` , `~=` . (Relational opertors)
- `&` , `|` , `~` , `xor` , `all` , `any` . (Logical opertors)

These are detailed in section 6. In Listing 2, some examples are provided.

```
A=fc_amat.ones(100,3,4);% A: 100-by-3-by-4 amat
B=fc_amat.random.randn(100,3,4);% B: 100-by-3-by-4 amat
C=randn(3,4);
D1=-A+1;
D2=B.^2-A/2;
D3=-2*A.*C;
```

Listing 2: Element by elements operations with `amat` object

Matricial products can also be done between `amat` objects or between an `amat` object and a matrix if their dimensions are compatible. For this operation the operator `*` can be used. In Listing 3, some examples are provided.

Listing 3: : matricial products with `amat` object

```
A=fc_amat.ones(100,3,4);%   100-by-3-by-4
info(A)
B=fc_amat.random.randn(100,4,2);% 100-by-4-by-2
info(B)
C=randn(4,5);
D1=A*B; % 100-by-3-by-2
info(D1)
D2=A*C; % 100-by-3-by-5
info(D2)
```

Output

```
A is a 100x3x4 amat[double] object
B is a 100x4x2 amat[double] object
D1 is a 100x3x2 amat[double] object
D2 is a 100x3x5 amat[double] object
```

Some usual mathematical functions as `cos`, `sin`, `exp`, `sqrt`, `abs`, `max`, ... are available for `amat` objects. One can refered to section 7 for more details.

Other operations such as determinants computation (`det` method), LU factorization with partial pivot (`lu` method), Cholesky factorization (`chol` method), solving linear systems (`mldivide` method or `\` operator) are also implemented for `amat` objects and described in section 8. In Listing 4, some examples using these functions are given.

Thereafter in Listing 5, the benchmark function `fc_amat.benchs.mldivide` is used to obtain cputimes of the `X=mldivide(A,b)` command where `A` and `b` are respectively `N` $\times 3 \times 3$ and `N` $\times 3 \times 4$ `amat` objects. The provided error is computed by taking the maximum of the infinity norms of all the matrices in the error `amat` object `E=A*X-b` obtained by `max(norm(E))`.

Finally, in Table 1 benchmark functions `fc_amat.benchs.mtimes`, `fc_amat.benchs.lu`, `fc_amat.benchs.chol` and `fc_amat.benchs.mldivide` are respectively used to get cputimes of the `X=mtimes(A,B)`, `[L,U,P]=lu(A)`, `R=chol(A)` and `X=mldivide(A,b)` where `A` and `B` are `N` $\times 4 \times 4$ `amat` objects, and `b` is a `N` $\times 4 \times 1$ `amat` object.

Listing 4: : Linear algebra with amat object

```
% Generate 100-by-4-by-4 amat object symmetric positive definite ...
    matrices:
A=fc_amat.random.randnsympd(100,4);
% determinants computation:
D=det(A); % D: 100-by-1-by-1 amat object,  det(A(k))=D(k),     for all k
% LU factorizations:
[L,U,P]=lu(A);
E1=abs(L*U-P*A);
fprintf('max of E1 elements: %.6e\n',max(E1(:)))
% Cholesky factorizations:
R=chol(A);
E2=abs(R'*R-A);
fprintf('max of E2 elements: %.6e\n',max(E2(:)))
% Solving linear systems:
b=ones(4,1); % RHS
X=A\b;    % X: 100-by-4-by-1, X(k)=A(k)\b,     for all k
E3=abs(A*X-b);
fprintf('max of E3 elements: %.6e\n',max(E3(:)))
B=fc_amat.random.randn(100,4,1); % RHS
Y=A\B;    % Y: 100-by-4-by-1, Y(k)=A(k)\B(k), for all k
E4=abs(A*Y-B);
fprintf('max of E4 elements: %.6e\n',max(E4(:)))
whos
```

Output

```
max of E1 elements: 7.105427e-15
max of E2 elements: 7.105427e-15
max of E3 elements: 7.105427e-15
max of E4 elements: 9.769963e-15
Variables in the current scope:

  Attr Name           Size                 Bytes Class
  ==== ====           ====                 ===== =====
       A              1x1                      0 amat
       B              1x1                      0 amat
       D              1x1                      0 amat
       E1             1x1                      0 amat
       E2             1x1                      0 amat
       E3             1x1                      0 amat
       E4             1x1                      0 amat
       L              1x1                      0 amat
       P              1x1                      0 amat
       R              1x1                      0 amat
       SaveOptions    1x6                     25 cell
       U              1x1                      0 amat
       X              1x1                      0 amat
       Y              1x1                      0 amat
       b              4x1                     32 double

Total is 23 elements using 57 bytes
```

Listing 5: : Computational times of the `X=mldivide(A,b)` command where `A` and `b` are respectively $N \times 3 \times 3$ and $N \times 3 \times 4$ `amat` objects by using the benchmark function `fc_amat.benchs.mldivide`

```
LN=10^5*[2:2:10];
fc_amat.benchs.mldivide(LN,'d',3,'n',4,'nbruns',5)
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[double] with (N,m,n)=(N,3,3)
#    containing strictly diagonally dominant matrices
# 2nd parameter is :
# -> amat[double] with (N,nr,nc)=(200000,3,4), size=[200000 3      4]
# Error function: @(X)max(norm(A*X-B))
#------------------------------------------------------------------------------
#date:2020/02/16 07:41:49
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d         %.3f        %.3e
#labels:       N   mldivide(s)    Error[0]
          200000        0.391   4.477e-14
          400000        1.476   1.053e-13
          600000        2.091   5.539e-14
          800000        3.084   6.757e-14
         1000000        3.881   8.921e-14
```

N	`mtimes` (s)	`chol` (s)	`lu` (s)	`mldivide` (s)
200 000	0.230(s)	0.014(s)	0.261(s)	0.327(s)
400 000	1.524(s)	0.040(s)	0.943(s)	1.065(s)
600 000	2.296(s)	0.061(s)	1.537(s)	1.676(s)
800 000	3.070(s)	0.082(s)	2.070(s)	2.244(s)
1 000 000	3.858(s)	0.106(s)	2.527(s)	2.764(s)
5 000 000	20.247(s)	0.960(s)	16.125(s)	18.693(s)
10 000 000	40.457(s)	1.929(s)	33.192(s)	38.470(s)

Table 1: Computational times in seconds of `mtimes(A,B)` (i.e. `A*B`), `lu(A)` , `chol(A)` and `mldivide(A,b)` (i.e. `A\b`) with `A` and `B` $N \times 4 \times 4$ `amat` objects and `b` n $N \times 4 \times 1$ `amat` object.

2 Installation

This package was tested on various OS with Octave releases:

Operating system	4.4.0	4.4.1	5.1.0	5.2.0
CentOS 7.7.1908	✓	✓	✓	✓
Debian 9.11	✓	✓	✓	✓
Fedora 29	✓	✓	✓	✓
OpenSUSE Leap 15.0	✓	✓	✓	✓
Ubuntu 18.04.3 LTS	✓	✓	✓	✓
MacOS High Sierra 10.13.6	✓	✓		
MacOS Mojave 10.14.4	✓	✓		
MacOS Catalina 10.15.2	✓	✓		
Windows 10 (1909)	✓	✓	✓	✓

It is not compatible with Octave releases prior to 4.2.0. Here are the links used to install the Octave releases tested:

- **Linux :** sources from `https://www.gnu.org/software/octave/`;
- **MacOS :** binaries from `http://octave-app.org/Download.html`;
- **Windows :** binaries from `https://www.gnu.org/software/octave/`.

2.0.1 Automatic installation, all in one (recommended)

For this method, one just has to get/download the install file

ofc_amat_install.m

or to get it on the dedicated web page. Thereafter, one runs it under Octave. This script downloads, extracts and configures the *fc-amat* and the required packages (*fc-tools* and *fc-bench*) in the current directory.

For example, to install this package in `~/Octave/packages` directory, one has to copy the file ofc_amat_install.m in the `~/Octave/packages` directory by using previous link. For example, in a Linux terminal, we can do:

```
cd ~/Octave/packages
HTTP=http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave
wget $HTTP/fc-amat/0.1.2/ofc_amat_install.m
```

Then in an Octave terminal run the following commands:

```
>> cd ~/Octave/packages
>> ofc_amat_install
```

The optional `'dir'` option can be used to specify installation directory:

```
ofc_amat_install('dir',dirname)
```

where `dirname` is the installation directory (string).
This is the output of the `ofc_amat_install` command on a Linux computer:

```
Parts of the <fc-amat> Octave package.
Copyright (C) 2018-2020 F. Cuvelier

1- Downloading and extracting the packages
2- Setting the <fc-amat> package
Write in ~/Octave/packages/fc-amat-full/fc_amat-0.1.2/configure_loc.m ...
3- Using packages :
   ->             fc-tools : 0.0.30
   ->             fc-bench : 0.1.2
 with              fc-amat : 0.1.2
*** Using instructions
   To use the <fc-amat> package:
   addpath('~/Octave/packages/fc-amat-full/fc_amat-0.1.2')
   fc_amat.init()

   See ~/Octave/packages/ofc_amat_set.m
```

The complete package (i.e. with all the other needed packages) is stored in the directory `~/Octave/packages/fc-amat-full` and, for each Octave session, one have to set the package by:

```
>> addpath('~/Octave/packages/fc-amat-full/fc-amat-0.1.2')
>> fc_amat.init()
```

If it's the first time the `fc_amat.init()` function is used, then its output is

```
Try to use default parameters!
 Use fc_tools.configure to configure.
Write in ...
     /home/cuvelier/tmp/fc-amat-full/fc_tools-0.0.30/configure_loc.m ...
Try to use default parameters!
 Use fc_bench.configure to configure.
Write in ...
     /home/cuvelier/tmp/fc-amat-full/fc_bench-0.1.2/configure_loc.m ...
Using fc_amat[0.1.2] with fc_tools[0.0.30], fc_bench[0.1.2].
```

Otherwise, the output of the `fc_amat.init()` function is

```
Using fc_amat[0.1.2] with fc_tools[0.0.30], fc_bench[0.1.2].
```

For **uninstalling**, one just has to delete the directory

`~/Octave/packages/fc-amat-full`

2.0.2 Manual installation

- Download one of the **full archives** (see web page) which contains all the needed toolboxes (*fc-amat*, *fc-tools* and *fc-bench*).
- Extract the archive in a folder.
- Set Octave path by adding path of needed packages.

For example under Linux, to install this package in `~/Octave/packages` directory, one can download fc-amat-0.1.2-full.tar.gz and extract it in the `~/Octave/packages` directory:

```
HTTP=http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave
wget $HTTP/fc-amat/0.1.2/fc-amat-0.1.2-full.tar.gz
tar zxf fc-amat-0.1.2-full.tar.gz -C ~/Octave/packages
```

For each Octave session, one has to set the package by adding path of all packages:

```
>> warning('off','Octave:shadowed-function');more off
>> addpath('~/Octave/packages/fc-amat-0.1.2/fc_amat-0.1.2')
>> addpath('~/Octave/packages/fc-amat-0.1.2/fc_tools-0.0.30')
>> addpath('~/Octave/packages/fc-amat-0.1.2/fc_bench-0.1.2')
```

3 Notations

Some typographic conventions are used in the following:

- $\mathbb{Z}$, $\mathtt{N}$, $\mathbb{R}$, $\mathbb{C}$ are respectively the set of integers, positive integers, reals and complex numbers. $\mathbb{K}$ is either $\mathbb{R}$ or $\mathbb{C}$.
- All vectors or 1D-arrays are represented in bold: $\boldsymbol{v} \in \mathbb{R}^n$ or $\boldsymbol{X}$ a 1D-array.The first alphabetic characters are $\boldsymbol{aAbBcC}\ldots$.
- All matrices or 2D-arrays are represented with the blackboard font as: $\mathbb{M} \in \mathcal{M}_{m,n}(\mathbb{K})$ or $\mathbb{b}$ a m-by-n 2D-array. The first alphabetic characters are $\mathbb{aAbBcC}\ldots$.
- All arrays of matrices or 3D-arrays or `amat` objects are represented with the bold blackboard font as: $\mathbf{\mathbb{M}} \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ or $\mathbf{\mathbb{b}}$ a N-by-m-by-n 3D-array. The first alphabetic characters are $\mathbf{\mathbb{aAbBcC}}\ldots$.

We now introduce some notations. Let $\mathbf{\mathbb{A}} = (\mathbb{A}_1, \ldots \mathbb{A}_N) \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ be a set of m-by-n matrices.We identify $\mathbf{\mathbb{A}}$ as a N-by-m-by-n `amat` object and we said that the `amat` object `A` is in $(\mathcal{M}_{m,n}(\mathbb{K}))^N$. The `k`-th matrix of `A` is `A(k)` and the `(` i,j) entry of the `k`-th matrix of `A` is `A(k,i,j)` .

Thereafter, we said that an `amat` object $\mathtt{A} \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ has a property of matrix if all its matrices have this property. For example, `A` is a symmetrical `amat` object if all its matrices are symmetrical.

4 Constructor and generators

We give properties of the `amat` class :

Properties of `amat` class

`nr`	:	number of rows
`nc`	:	number of columns
`N`	:	number of matrices (`nr`-by-`nc`)
`values`	:	`N`-by-`nr`-by-`nc` array which contains all the matrices

4.1 Constructor

Syntaxe

```
X=amat(N,nr,nc)
X=amat(T)
X=amat(N,A)
X=amat(...,classname)
```

Description

`X=amat(N,n,m)` returns a `N`-by-`n`-by-`m` `amat` object where all its elements are set to 0.

`X=amat(T)` when `T` is a `N`-by-`n`-by-`m` array, returns the `N`-by-`n`-by-`m` `amat` object set to `T`.
When `T` is a `N`-by-`n`-by-`m` `amat` object, returns a `N`-by-`n`-by-`m` zero `amat` object.

`X=amat(N,A)` with `A` a `n`-by-`m` matrix, return the `N`-by-`n`-by-`m` `amat` object where all its matrices are set to the matrix `A`.

`X=amat(...,classname)` returns an `amat` object with values of class `classname`.

In Listing 6, some examples are provided.

Listing 6: : `amat` constructors

```
X=amat(100,3,4);              % X: 100-by-3-by-4 amat
info(X)
W=amat(X);                    % W: 100-by-3-by-4 amat
info(W)
T=randn(200,2,3);             % T: 200-by-2-by-3 array
Y=amat(T);                    % Y: 200-by-2-by-3 amat
info(Y)
A=randi(10,[2,4],'int32');% A: 2-by-4 int32 matrix
Z=amat(30,A,'int64');         % Z: 30-by-2-by-4 int64 amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x4 amat[double] object
W is a 100x3x4 amat[double] object
Y is a 200x2x3 amat[double] object
Print Z amat object :
Z is a 30x2x4 amat[int64] object
  Z(1)=
   6   9  10  10
   5   9  10   4
  Z(2)=
   6   9  10  10
   5   9  10   4
  ...

  Z(29)=
   6   9  10  10
   5   9  10   4
  Z(30)=
   6   9  10  10
   5   9  10   4
```

4.2 Particular generators

There is the list of functions which generate some particular `amat` objects:

- `fc_amat.zeros` , generates an zero `amat` object,
- `fc_amat.ones` , generates an `amat` object of one's,
- `fc_amat.eye` , generates an `amat` object of identity matrices.

4.2.1 `fc_amat.zeros` function

Syntaxe

```
X=fc_amat.zeros(N,m,n)
X=fc_amat.zeros([N,m,n])
X=fc_amat.zeros([N,d])
X=fc_amat.zeros(...,classname)
```

Description

`X=fc_amat.zeros(N,m,n)` return an `N`-by-`m`-by-`n` zero `amat` object.

`X=fc_amat.zeros([N,m,n])` same as `X=fc_amat.zeros(N,m,n)`

`X=fc_amat.zeros(N,d)` same as `X=fc_amat.zeros(N,d,d)`

`X=fc_amat.zeros(...,classname)` returns an `amat` object with values of class `classname`

In Listing 7, some examples are provided.

Listing 7: : examples of `fc_amat.zeros` function usage

```
X=fc_amat.zeros(100,2,4);                % X: 100-by-2-by-4 amat
Y=fc_amat.zeros(200,3);                  % Y: 100-by-3-by-3 amat
Z=fc_amat.zeros([50,2,3],'single');      % Y: 100-by-2-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
Z
```

Output

```
List current variables :
Variables in the current scope:

  Attr Name           Size                Bytes Class
  ==== ====           ====                ===== =====
       SaveOptions    1x6                    25 cell
       X              1x1                     0 amat
       Y              1x1                     0 amat
       Z              1x1                     0 amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z =

 is a 50x2x3 amat[single] object
  matrix(1)=
   0   0   0
   0   0   0
  matrix(2)=
   0   0   0
   0   0   0
  ...

  matrix(49)=
   0   0   0
   0   0   0
  matrix(50)=
   0   0   0
   0   0   0
```

4.2.2 `fc_amat.ones` function

Syntaxe

```
X=fc_amat.ones(N,m,n)
X=fc_amat.ones([N,m,n])
X=fc_amat.ones(N,d)
X=fc_amat.ones(...,classname)
```

Description

`X=fc_amat.ones(N,m,n)` return a N-by-m-by-n `amat` object of ones.

`X=fc_amat.ones([N,m,n])` same as `X=fc_amat.ones(N,m,n)`

`X=fc_amat.ones(N,d)` same as `X=fc_amat.ones(N,d,d)`

`X=fc_amat.ones(...,classname)` returns an `amat` object with values of class `classname`

In Listing 7, some examples are provided.

Listing 8: : examples of `fc_amat.ones` function usage

```
X=fc_amat.ones(100,2,4);                % X: 100-by-2-by-4 amat
Y=fc_amat.ones(200,3);                  % Y: 200-by-3-by-3 amat
Z=fc_amat.ones([50,2,3],'single');      % Y: 50-by-2-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
Z
```

Output

```
List current variables :
Variables in the current scope:

  Attr Name           Size                Bytes Class
  ==== ====           ====                ===== =====
       SaveOptions    1x6                    25 cell
       X              1x1                     0 amat
       Y              1x1                     0 amat
       Z              1x1                     0 amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z =

 is a 50x2x3 amat[single] object
  matrix(1)=
   1   1   1
   1   1   1
  matrix(2)=
   1   1   1
   1   1   1
  ...

  matrix(49)=
   1   1   1
   1   1   1
  matrix(50)=
   1   1   1
   1   1   1
```

4.2.3 `fc_amat.eye` function

Syntaxe

```
X=fc_amat.eye(N,d)
X=fc_amat.eye(N,m,n)
X=fc_amat.eye([N,m,n])
X=fc_amat.eye(...,classname)
```

Description

`X=fc_amat.eye(N,d)` return a `N`-by-`d`-by-`d` `amat` object whose all its matrices are the `d`-by-`d` identity matrix.

`X=fc_amat.eye(N,m,n)` return a `N`-by-`m`-by-`n` `amat` object whose all its matrices are the `m`-by-`n` matrix with one's on the diagonal and zeros elsewhere.

`X=fc_amat.eye([N,m,n])` same as `X=fc_amat.eye(N,m,n)`

`X=fc_amat.eye(...,classname)` returns an `amat` object with values of class `classname`

In Listing 7, some examples are provided.

Listing 9: : examples of `fc_amat.eye` function usage

```
X=fc_amat.eye(100,2,4);          % X: 100-by-2-by-4 amat
Y=fc_amat.eye(200,3,'int32'); % Y: 200-by-3-by-3 int32 amat
Z=fc_amat.eye([50,2,3]);         % Z: 50-by-2-by-3 amat
disp('List current variables :')
whos
disp('Print Y amat object :')
Y
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Y amat object :
Y =

 is a 200x3x3 amat[int32] object
  matrix(1)=
  1  0  0
  0  1  0
  0  0  1
  matrix(2)=
  1  0  0
  0  1  0
  0  0  1
  ...

  matrix(199)=
  1  0  0
  0  1  0
  0  0  1
  matrix(200)=
  1  0  0
  0  1  0
  0  0  1
```

4.3 Random generators

There is the list of functions which generate some `amat` objects with random elements. They all belong to the namespace `fc_amat.random` :

- `rand` , `randn` , `randi` random elements,
- `randsym` , `randnsym` , `randisym` random **symmetric** matrices,
- `randsym` , `randnsym` , `randisym` random **Hermitian** matrices,
- `randdiag` , `randndiag` , `randidiag` random **diagonal** matrices,
- `randtril` , `randntril` , `randitril` random **lower triangular** matrices,

- `randtriu` , `randntriu` , `randitriu` random **upper triangular** matrices,
- `randsdd` , `randnsdd` , `randisdd` random **stricly diagonally dominant** matrices,
- `randsympd` , `randnsympd` , `randisympd` random **symmetric positive definite** matrices,
- `randherpd` , `randnherpd` , `randiherpd` random **Hermitian positive definite** matrices.

4.3.1 `fc_amat.random.rand` function

The `fc_amat.random.rand` function return an `amat` object with random elements uniformly distributed on the interval $]0,1[$.

Syntaxe

```
X=fc_amat.random.rand(N,m,n)
X=fc_amat.random.rand([N,m,n])
X=fc_amat.random.rand(N,d)
X=fc_amat.random.rand(...,classname)
```

Description

`X=fc_amat.random.rand(N,m,n)` return a `N`-by-`m`-by-`n` `amat` object with random elements uniformly distributed on the interval $]0,1[$.

`X=fc_amat.random.rand([N,m,n])` same as `X=fc_amat.random.rand(N,m,n)`

`X=fc_amat.random.rand(N,d)` same as `X=fc_amat.random.rand(N,d,d)`

`X=fc_amat.random.rand(...,classname)` returns an `amat` object with values of class `classname` . `classname` could be `'single'` or `'double'` (default).

In Listing 10, some examples are provided.

Listing 10: : examples of `fc_amat.random.rand` function usage

```
X=fc_amat.random.rand(100,2,4);                % X: 100-by-2-by-4 amat
Y=fc_amat.random.rand(200,3);                  % Y: 200-by-3-by-3 amat
Z=fc_amat.random.rand([50,2,3],'single');      % Y: 50-by-2-by-3 single ...
    amat
disp('List current variables :')
whos
disp('Print Z amat object :')
Z
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat
        Z              1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z =

 is a 50x2x3 amat[single] object
  matrix(1)=
   0.83439   0.34903   0.80812
   0.36462   0.80937   0.11879
  matrix(2)=
   0.93437   0.44776   0.15967
   0.58540   0.51862   0.39244
  ...

  matrix(49)=
   0.51687   0.42974   0.41078
   0.94184   0.46520   0.54993
  matrix(50)=
   0.35404   0.18784   0.75419
   0.83725   0.33357   0.83256
```

4.3.2 `fc_amat.random.randn` function

The `fc_amat.random.randn` function return an `amat` object with normally distributed random elements having zero mean and variance one.

Syntaxe

```
X=fc_amat.random.randn(N,m,n)
X=fc_amat.random.randn([N,m,n])
X=fc_amat.random.randn(N,d)
X=fc_amat.random.randn(...,classname)
```

Description

`X=fc_amat.random.randn(N,m,n)`

returns a `N`-by-`m`-by-`n` `amat` object with normally distributed random elements having zero mean and variance one.

`X=fc_amat.random.randn([N,m,n])`

same as `X=fc_amat.random.randn(N,m,n)`

```
X=fc_amat.random.randn(N,d)
```

same as `X=fc_amat.random.randn(N,d,d)`

```
X=fc_amat.random.randn(...,classname)
```

returns an `amat` object with values of class `classname`. `classname` could be `'single'` or `'double'` (default).

In Listing 10, some examples are provided.

Listing 11: : examples of `fc_amat.random.randn` function usage

```
X=fc_amat.random.randn(100,2,4);                % X: 100-by-2-by-4 amat
Y=fc_amat.random.randn(200,3);                  % Y: 200-by-3-by-3 amat
Z=fc_amat.random.randn([50,2,3],'single');      % Y: 50-by-2-by-3 single ...
    amat
disp('List current variables :')
whos
disp('Print Z amat object :')
Z
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat
        Z              1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z =

 is a 50x2x3 amat[single] object
  matrix(1)=
  -0.90816 -0.54618  0.50288
   1.37383 -0.71557 -2.80470
  matrix(2)=
   2.218531 -0.171473  0.972514
  -0.319126 -0.063963  0.502081
  ...

  matrix(49)=
  -1.29457 -1.40515  1.58949
   0.87341  0.64191 -1.28766
  matrix(50)=
   0.379296  0.012471 -0.482323
  -0.482702 -0.707150  0.047960
```

4.3.3 `fc_amat.random.randi` function

The function `fc_amat.random.randi` return an `amat` object whose elements are random integers.

Syntaxe

```
X=fc_amat.random.randi(Imax,N,m,n)
X=fc_amat.random.randi(Imax,[N,m,n])
X=fc_amat.random.randi(Imax,N,d)
```

```
X=fc_amat.random.randi([Imin,Imax],...)
X=fc_amat.random.randi(...,classname)
```

Description

`X=fc_amat.random.randi(Imax,N,m,n)`

returns a `N`-by-`m`-by-`n` `amat` object containing pseudorandom integer values drawn from the discrete uniform distribution on `1:Imax`.

`X=fc_amat.random.randi(Imax,[N,m,n])`

same as `X=fc_amat.random.randi(Imax,N,m,n)`

`X=fc_amat.random.randi(Imax,N,d)`

same as `X=fc_amat.random.randi(Imax,N,d,d)`

`X=fc_amat.random.randi([Imin,Imax],...)`

returns an `amat` object containing integer values drawn from the discrete uniform distribution on `Imin:Imax`.

`X=fc_amat.random.randi(...,classname)`

returns an `amat` object with values of class `classname`. Accepted `classname` strings are those of the `randi` Matlab function. Default is `'double'`.

In Listing 10, some examples are provided.

Listing 12: : examples of `fc_amat.random.randi` function usage

```
X=fc_amat.random.randi(10,100,2,4);                % X: 100-by-2-by-4 amat
Y=fc_amat.random.randi(15,200,3);                  % Y: 200-by-3-by-3 amat
Z=fc_amat.random.randi([-5,5],[50,2,3],'int32'); % Z: 50-by-2-by-3 ...
    int32 amat
disp('List current variables:')
whos
disp('Print Z amat object:')
Z
```

Output

```
List current variables:
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat
        Z              1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object:
Z =

 is a 50x2x3 amat[int32] object
  matrix(1)=
  -4   3   3
   1   1  -3
  matrix(2)=
  -3  -5   5
   0  -5   5
  ...

  matrix(49)=
   1   3  -2
  -2   0  -2
  matrix(50)=
  -5   2   0
   1   1  -1
```

4.3.4 `fc_amat.random.randsym` function

The `fc_amat.random.randsym` function return an `amat` object whose matrices are symmetric with random elements uniformly distributed on the interval $]0, 1[$.

Syntaxe

```
X=fc_amat.random.randsym(N,d)
X=fc_amat.random.randsym(N,d,'class',value)
```

Description

`X=fc_amat.random.randsym(N,d)`

return a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric with random elements uniformly distributed on the interval $]0, 1[$.

`X=fc_amat.random.randsym(N,d,'class',classname)`

returns an `amat` object with values of class `classname`. `classname` could be `'single'` or `'double'` (default).

In Listing 13, some examples are provided.

Listing 13: : examples of `fc_amat.random.randsym` function usage

```
X=fc_amat.random.randsym(100,3);                    % X: 100-by-3-by-3 amat
Y=fc_amat.random.randsym(50,2,'class','single'); % Y: 50-by-2-by-2 ...
    single amat
disp('List current variables :')
whos
disp('Print Y amat object :')
Y
```

Output

```
List current variables :
Variables in the current scope:

  Attr Name           Size                Bytes Class
  ==== ====           ====                ===== =====
       SaveOptions    1x6                    25 cell
       X              1x1                     0 amat
       Y              1x1                     0 amat

Total is 8 elements using 25 bytes

Print Y amat object :
Y =

 is a 50x2x2 amat[single] object
  matrix(1)=
   0.23768  0.37225
   0.37225  0.76504
  matrix(2)=
   0.28976  0.15546
   0.15546  0.89015
  ...

  matrix(49)=
   0.46575  0.13877
   0.13877  0.31999
  matrix(50)=
   0.966122  0.082660
   0.082660  0.958567
```

4.3.5 `fc_amat.random.randnsym` function

The `fc_amat.random.randnsym` function return an `amat` object whose matrices are symmetric with normally distributed random elements having zero mean and variance one.

Syntaxe

```
X=fc_amat.random.randnsym(N,d)
X=fc_amat.random.randnsym(N,d,'class',value)
```

Description

`X=fc_amat.random.randnsym(N,d)`

return a N-by-d-by-d `amat` object whose matrices are symmetric normally distributed random elements having zero mean and variance one.

`X=fc_amat.random.randnsym(N,d,'class',classname)`

returns an `amat` object with values of class `classname`. `classname`

could be `'single'` or `'double'` (default).

In Listing 14, some examples are provided.

Listing 14: : examples of `fc_amat.random.randnsym` function usage

```
X=fc_amat.random.randnsym(100,3);                    % X: 100-by-3-by-3 ...
    amat
Y=fc_amat.random.randnsym(50,2,'class','single'); % Y: 50-by-2-by-2 ...
    single amat
disp('List current variables :')
whos
disp('Print Y amat object :')
Y
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name          Size                Bytes Class
   ==== ====          ====                ===== =====
        SaveOptions   1x6                    25 cell
        X             1x1                     0 amat
        Y             1x1                     0 amat

Total is 8 elements using 25 bytes

Print Y amat object :
Y =

 is a 50x2x2 amat[single] object
  matrix(1)=
  -1.3961 -1.2095
  -1.2095  1.8918
  matrix(2)=
   0.56166 -0.86104
  -0.86104 -1.05533
  ...

  matrix(49)=
   0.0050567 -0.4670636
  -0.4670636 -1.0188835
  matrix(50)=
   0.22045  1.25065
   1.25065 -1.52897
```

4.3.6 `fc_amat.random.randisym` function

The `fc_amat.random.randisym` function return an `amat` object whose matrices are symmetric with random integers values.

Syntaxe

```
X=fc_amat.random.randisym(Imax,N,d)
X=fc_amat.random.randisym([Imin,Imax],...)
X=fc_amat.random.randisym(...,'class',classname)
```

Description

`X=fc_amat.random.randisym(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric pseudo random integer values drawn from the discrete uniform distribution on `1:Imax`

`X=fc_amat.random.randisym([Imin,Imax], ...)`

pseudo random integer values are drawn from the discrete uniform distribution on `Imin:Imax`

`X=fc_amat.random.randisym(...,'class',classname)`

returns an `amat` object with values of class `classname` . Accepted `classname` strings are those of the `randi` Matlab function. Default is `'double'` .

In Listing 15, some examples are provided.

Listing 15: : examples of `fc_amat.random.randisym` function usage

```
X=fc_amat.random.randisym(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randisym([-5,5],100,2,'class','single');
% Y: 50-by-2-by-2 single amat
disp('List␣current␣variables␣:')
whos
disp('Print␣Y␣amat␣object␣:')
Y
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat

Total is 8 elements using 25 bytes

Print Y amat object :
Y =

 is a 100x2x2 amat[single] object
  matrix(1)=
  -4  -5
  -5  -5
  matrix(2)=
  -4  -4
  -4   3
  ...

  matrix(99)=
  -3  -4
  -4   1
  matrix(100)=
  -4   2
   2   4
```

4.3.7 `fc_amat.random.randher` function

The `fc_amat.random.randher` function return an `amat` object whose matrices are hermitian with random real part elements uniformly distributed on the interval $]0, 1[$ and imaginary part elements uniformly distributed on the interval $]-1, 1[$.

Syntaxe

```
X=fc_amat.random.randher(N,d)
X=fc_amat.random.randher(...,'class',value)
```

Description

`X=fc_amat.random.randher(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric with random elements uniformly distributed on the interval $]0,1[$.

`X=fc_amat.random.randher(...,'class',classname)`

returns an `amat` object with values of class `classname`. `classname` could be `'single'` or `'double'` (default).

In Listing 16, some examples are provided.

Listing 16: : examples of `fc_amat.random.randher` function usage

```
X=fc_amat.random.randher(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randher(50,2,'class','single');
% Y: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Y amat object :')
Y
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat

Total is 8 elements using 25 bytes

Print Y amat object :
Y =

 is a 50x2x2 amat[complex single] object
  matrix(1)=
   0.99150 - 0.30193i 0.42763 + 0.73866i
   0.42763 - 0.73866i 0.60391 + 0.70163i
  matrix(2)=
   0.025895 - 0.515599i 0.949283 + 0.047682i
   0.949283 - 0.047682i 0.581856 + 0.609545i
  ...

  matrix(49)=
   0.50040 - 0.95036i 0.25870 + 0.44602i
   0.25870 - 0.44602i 0.95662 + 0.89598i
  matrix(50)=
   0.02089 - 0.70954i 0.04883 - 0.64010i
   0.04883 + 0.64010i 0.09396 + 0.65342i
```

4.3.8 `fc_amat.random.randnher` function

The `fc_amat.random.randnher` function return an `amat` object whose matrices are hermitian with normally distributed random real and imaginary part elements having zero mean and variance one.

Syntaxe

```
X=fc_amat.random.randnher(N,d)
X=fc_amat.random.randnher(...,'class',value)
```

Description

`X=fc_amat.random.randnher(N,d)`

returns a N-by-d-by-d `amat` object whose matrices are Hermitian normally distributed random elements having zero mean and variance one.

`X=fc_amat.random.randnher(...,'class',classname)`

returns an `amat` object with values of class `classname`. `classname` could be `'single'` or `'double'` (default).

In Listing 17, some examples are provided.

Listing 17: : examples of `fc_amat.random.randnher` function usage

```
X=fc_amat.random.randnher(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randnher(50,2,'class','single');
% Y: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Y amat object :')
Y
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat

Total is 8 elements using 25 bytes

Print Y amat object :
Y =

 is a 50x2x2 amat[complex single] object
  matrix(1)=
   0.03756 + 1.57262i 0.27277 + 1.18677i
   0.27277 - 1.18677i 2.52557 + 0.57663i
  matrix(2)=
   0.67704 + 0.95820i -1.00240 + 1.27612i
  -1.00240 - 1.27612i -0.52705 - 1.51491i
  ...

  matrix(49)=
  -0.91310 - 1.17757i 0.40275 + 1.64343i
   0.40275 - 1.64343i 0.89764 + 0.32499i
  matrix(50)=
  -0.134212 - 1.600914i 0.138198 - 0.033163i
   0.138198 + 0.033163i 0.079685 - 0.599938i
```

4.3.9 `fc_amat.random.randiher` function

The `fc_amat.random.randiher` function return an `amat` object whose matrices are Hermitian with random integers values.

Syntaxe

```
X=fc_amat.random.randiher(Imax,N,d)
X=fc_amat.random.randiher([Imin,Imax],...)
X=fc_amat.random.randiher(...,'class',classname)
```

Description

`X=fc_amat.random.randiher(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are Hermitian where real and imaginay part values are respectively drawn from the discrete uniform distribution on `1:Imax` and the discrete uniform distribution on `1:Imax` times a random sign.

`X=fc_amat.random.randiher([Imin,Imax], ...)`

pseudorandom integer values are drawn from the discrete uniform distribution on `Imin:Imax`

`X=fc_amat.random.randiher(...,'class',classname)`

returns an `amat` object with values of class `classname` . Accepted `classname` strings are those of the `randi` Matlab function. Default is `'double'` .

In Listing 18, some examples are provided.

Listing 18: : examples of `fc_amat.random.randiher` function usage

```
X=fc_amat.random.randiher(10,100,3);  % X: 100-by-3-by-3 amat
info(X)
Y=fc_amat.random.randiher([-5,5],100,2,'class','single');
% Y: 50-by-2-by-2 single amat
disp('Print␣Y␣amat␣object␣:')
Y
```

Output

```
X is a 100x3x3 amat[complex double] object
Print Y amat object :
Y =

 is a 100x2x2 amat[complex single] object
  matrix(1)=
   5 + 0i  5 + 3i
   5 - 3i -1 - 5i
  matrix(2)=
   0 - 1i -5 + 2i
  -5 - 2i -1 - 1i
  ...

  matrix(99)=
   3 - 5i -5 + 2i
  -5 - 2i -4 - 2i
  matrix(100)=
   3 + 2i -1 + 5i
  -1 - 5i -4 - 4i
```

4.3.10 `fc_amat.random.randdiag` function

The `fc_amat.random.randdiag` function return an `amat` object whose matrices are diagonal with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

Syntaxe

```
X=fc_amat.random.randdiag(N,d)
X=fc_amat.random.randdiag(...,key,value)
```

Description

`X=fc_amat.random.randdiag(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are diagonal with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

`X=fc_amat.random.randdiag(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the diagonal matrices elements are also drawn from the uniform distribution on the interval $]a, b[$. (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'a'` , to set a (lower bound of the interval) value (0 by default).
- `'b'` , to set b (upper bound of the interval) value (1 by default).

In Listing 19, some examples are provided.

Listing 19: : examples of `fc_amat.random.randdiag` function usage

```
X=fc_amat.random.randdiag(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randdiag(200,3,'nc',4,'complex',true,'a',-1);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randdiag(50,3,'class','single','k',1,'b',5);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x3 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0.00000  4.45636  0.00000
   0.00000  0.00000  1.89847
   0.00000  0.00000  0.00000
  Z(2)=
   0.00000  1.02896  0.00000
   0.00000  0.00000  1.47020
   0.00000  0.00000  0.00000
  ...

  Z(49)=
   0.00000  1.73700  0.00000
   0.00000  0.00000  4.85011
   0.00000  0.00000  0.00000
  Z(50)=
   0.00000  3.70307  0.00000
   0.00000  0.00000  4.90340
   0.00000  0.00000  0.00000
```

4.3.11 `fc_amat.random.randndiag` function

The `fc_amat.random.randndiag` function return an `amat` object whose matrices are diagonal with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

Syntaxe

```
X=fc_amat.random.randndiag(N,d)
X=fc_amat.random.randndiag(...,key,value)
```

Description

`X=fc_amat.random.randndiag(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are diagonal with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

`X=fc_amat.random.randndiag(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the diagonal matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)

- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'mean'` , to set mean of the normal distribution (0 by default).
- `'sigma'` , to set standard deviation of the normal distribution (1 by default).

In Listing 20, some examples are provided.

Listing 20: : examples of `fc_amat.random.randndiag` function usage

```
X=fc_amat.random.randndiag(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randndiag(200,3,'nc',4,'complex',true,'sigma',5);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randndiag(50,3,'class','single','k',-1,'mean',4);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x3 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0.00000  0.00000  0.00000
   4.41041  0.00000  0.00000
   0.00000  3.10479  0.00000
  Z(2)=
   0.00000  0.00000  0.00000
   3.58672  0.00000  0.00000
   0.00000  4.70365  0.00000
  ...

  Z(49)=
   0.00000  0.00000  0.00000
   3.50369  0.00000  0.00000
   0.00000  3.27365  0.00000
  Z(50)=
   0.00000  0.00000  0.00000
   3.97868  0.00000  0.00000
   0.00000  4.60571  0.00000
```

4.3.12 `fc_amat.random.randidiag` function

The `fc_amat.random.randidiag` function return an `amat` object whose matrices are diagonal and non zeros elements are random integers

Syntaxe

```
X=fc_amat.random.randidiag(Imax,N,d)
X=fc_amat.random.randidiag([Imin,Imax],...)
X=fc_amat.random.randidiag(...,key,value)
```

Description

`X=fc_amat.random.randidiag(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are diagonal and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `1:Imax` .

`X=fc_amat.random.randidiag([Imin,Imax],N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are diagonal and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `Imin:Imax` .

`X=fc_amat.random.randidiag(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the diagonal matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value are those of the `randi` Matlab function. Default is `'double'` .
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .

In Listing 21, some examples are provided.

Listing 21: : examples of `fc_amat.random.randidiag` function usage

```
X=fc_amat.random.randidiag(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randidiag(8,200,3,'nc',4,'complex',true);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randidiag([-5,5],50,3,'class','single','k',1);
% Z: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

  Attr Name           Size                 Bytes Class
  ==== ====           ====                 ===== =====
       SaveOptions    1x6                     25 cell
       X              1x1                      0 amat
       Y              1x1                      0 amat
       Z              1x1                      0 amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0   4   0
   0   0  -3
   0   0   0
  Z(2)=
   0  -5   0
   0   0  -3
   0   0   0
  ...

  Z(49)=
   0   1   0
   0   0   0
   0   0   0
  Z(50)=
   0   5   0
   0   0  -4
   0   0   0
```

4.3.13 `fc_amat.random.randtril` function

The `fc_amat.random.randtril` function return an `amat` object whose matrices are lower triangular with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

Syntaxe

```
X=fc_amat.random.randtril(N,d)
X=fc_amat.random.randtril(...,key,value)
```

Description

`X=fc_amat.random.randtril(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are lower triangular with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

```
X=fc_amat.random.randtril(...,key,value)
```

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the lower triangular matrices elements are also drawn from the uniform distribution on the interval $]a, b[$. (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'a'` , to set a (lower bound of the interval) value (0 by default).
- `'b'` , to set b (upper bound of the interval) value (1 by default).

In Listing 22, some examples are provided.

Listing 22: : examples of `fc_amat.random.randtril` function usage

```
X=fc_amat.random.randtril(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randtril(200,3,'nc',4,'complex',true,'a',-1);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randtril(50,3,'class','single','k',1,'b',5);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x4 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0.76513  3.68083  0.00000
   2.15876  1.45784  3.44434
   3.13170  1.73051  4.26784
  Z(2)=
   0.88066  4.65178  0.00000
   0.87706  4.27085  2.20639
   2.89094  3.52727  3.13777
  ...

  Z(49)=
   4.46879  2.52914  0.00000
   1.05125  4.08994  4.64593
   2.08514  1.99479  1.37625
  Z(50)=
   1.33470  3.66791  0.00000
   2.08195  2.08667  1.49593
   0.82091  4.08619  2.97380
```

4.3.14 `fc_amat.random.randntril` function

The `fc_amat.random.randntril` function return an `amat` object whose matrices are lower triangular with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

Syntaxe

```
X=fc_amat.random.randntril(N,d)
X=fc_amat.random.randntril(...,key,value)
```

Description

`X=fc_amat.random.randntril(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are lower triangular with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

`X=fc_amat.random.randntril(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the lower triangular matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'mean'` , to set mean of the normal distribution (0 by default).
- `'sigma'` , to set standard deviation of the normal distribution (1 by default).

In Listing 23, some examples are provided.

Listing 23: : examples of `fc_amat.random.randntril` function usage

```
X=fc_amat.random.randntril(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randntril(200,3,'nc',4,'complex',true,'sigma',5);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randntril(50,3,'class','single','k',-1,'mean',4);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x4 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0.00000  0.00000  0.00000
   4.77031  0.00000  0.00000
   2.92923  5.07623  0.00000
  Z(2)=
   0.00000  0.00000  0.00000
   3.45716  0.00000  0.00000
   4.33151  3.01048  0.00000
  ...

  Z(49)=
   0.00000  0.00000  0.00000
   3.44813  0.00000  0.00000
   3.09175  2.99986  0.00000
  Z(50)=
   0.00000  0.00000  0.00000
   3.88062  0.00000  0.00000
   4.98268  3.51951  0.00000
```

4.3.15 `fc_amat.random.randitril` function

The `fc_amat.random.randitril` function return an `amat` object whose matrices are lower triangular and non zeros elements are random integers

Syntaxe

```
X=fc_amat.random.randitril(Imax,N,d)
X=fc_amat.random.randitril([Imin,Imax],...)
X=fc_amat.random.randitril(...,key,value)
```

Description

`X=fc_amat.random.randitril(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are lower triangular and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `1:Imax` .

`X=fc_amat.random.randitril([Imin,Imax],N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are lower triangular and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `Imin:Imax` .

`X=fc_amat.random.randitril(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the lower triangular matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value are those of the `randi` Matlab function. Default is `'double'` .
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .

In Listing 24, some examples are provided.

Listing 24: : examples of `fc_amat.random.randitril` function usage

```
X=fc_amat.random.randitril(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randitril(8,200,3,'nc',4,'complex',true);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randitril([-5,5],50,3,'class','single','k',1);
% Z: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes Class
   ==== ====           ====                  ===== =====
        SaveOptions    1x6                      25 cell
        X              1x1                       0 amat
        Y              1x1                       0 amat
        Z              1x1                       0 amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
  -4   4   0
  -4   3   0
  -4   4  -2
  Z(2)=
   3  -1   0
   4  -3   4
  -4  -4   4
  ...

  Z(49)=
   5  -2   0
  -2  -1   0
  -1  -5  -3
  Z(50)=
   4  -5   0
   3   0  -1
  -5   2   5
```

4.3.16 `fc_amat.random.randtriu` function

The `fc_amat.random.randtriu` function return an `amat` object whose matrices are upper triangular with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

Syntaxe

```
X=fc_amat.random.randtriu(N,d)
X=fc_amat.random.randtriu(...,key,value)
```

Description

`X=fc_amat.random.randtriu(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are diagonal with non zeros elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

`X=fc_amat.random.randtriu(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the upper triangular matrices elements are also drawn from the uniform distribution on the interval $]a, b[$. (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'a'` , to set a (lower bound of the interval) value (0 by default).
- `'b'` , to set b (upper bound of the interval) value (1 by default).

In Listing 25, some examples are provided.

Listing 25: : examples of `fc_amat.random.randtriu` function usage

```
X=fc_amat.random.randtriu(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randtriu(200,3,'nc',4,'complex',true,'a',-1);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randtriu(50,3,'class','single','k',-1,'b',5);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x4 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   2.52783  1.51558  3.60382
   4.27325  2.57585  1.99681
   0.00000  4.09008  4.24505
  Z(2)=
   0.87431  4.26757  4.35941
   0.30258  2.15214  4.84750
   0.00000  1.68569  0.71271
  ...

  Z(49)=
   4.83694  4.18473  4.60748
   2.75617  3.29259  3.57546
   0.00000  4.11499  1.05780
  Z(50)=
   3.55765  1.86919  1.34360
   3.69565  3.06679  4.86493
   0.00000  1.97281  0.70751
```

4.3.17 `fc_amat.random.randntriu` function

The `fc_amat.random.randntriu` function return an `amat` object whose matrices are upper triangular with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

Syntaxe

```
X=fc_amat.random.randntriu(N,d)
X=fc_amat.random.randntriu(...,key,value)
```

Description

`X=fc_amat.random.randntriu(N,d)`

returns a N-by-d-by-d `amat` object whose matrices are upper triangular with non zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

`X=fc_amat.random.randntriu(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the upper triangular matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)

- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .
- `'mean'` , to set mean of the normal distribution (0 by default).
- `'sigma'` , to set standard deviation of the normal distribution (1 by default).

In Listing 26, some examples are provided.

Listing 26: : examples of `fc_amat.random.randntriu` function usage

```
X=fc_amat.random.randntriu(100,3);
info(X) % X: 100-by-3-by-3 amat
Y=fc_amat.random.randntriu(200,3,'nc',4,'complex',true,'sigma',5);
info(Y) % Y: 200-by-3-by-4 amat
Z=fc_amat.random.randntriu(50,3,'class','single','k',-1,'mean',4);
% Z: 50-by-3-by-3 single amat
disp('Print␣Z␣amat␣object␣:')
disp(Z)
```

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x4 amat[complex double] object
Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   3.18578  3.15301  3.98081
   4.27241  3.71358  4.96857
   0.00000  4.05525  4.29779
  Z(2)=
   3.18073  4.51928  3.58377
   2.22726  2.94344  5.91858
   0.00000  3.31299  3.80498
  ...

  Z(49)=
   3.19077  3.27238  3.71265
   3.28618  5.35572  3.73057
   0.00000  4.33164  4.04673
  Z(50)=
   3.13021  4.39424  4.30541
   4.65391  4.42561  3.22127
   0.00000  2.50469  4.57964
```

4.3.18 `fc_amat.random.randitriu` function

The `fc_amat.random.randitriu` function return an `amat` object whose matrices are upper triangular and non zeros elements are random integers

Syntaxe

```
X=fc_amat.random.randitriu(Imax,N,d)
X=fc_amat.random.randitriu([Imin,Imax],...)
X=fc_amat.random.randitriu(...,key,value)
```

Description

`X=fc_amat.random.randitriu(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are upper triangular and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `1:Imax` .

`X=fc_amat.random.randitriu([Imin,Imax],N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are upper triangular and non zeros elements are pseudorandom integer drawn from the discrete uniform distribution on `Imin:Imax` .

`X=fc_amat.random.randitriu(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the upper triangular matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value are those of the `randi` Matlab function. Default is `'double'` .
- `'nc'` , number of columns of the matrices (default: `d`)
- `'k'` , offset of `k` diagonals above or below the main diagonal; above for positive `k` and below for negative `k` .

In Listing 27, some examples are provided.

Listing 27: : examples of `fc_amat.random.randitriu` function usage

```
X=fc_amat.random.randitriu(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randitriu(8,200,3,'nc',4,'complex',true);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randitriu([-5,5],50,3,'class','single','k',1);
% Z: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0   4   3
   0   0   5
   0   0   0
  Z(2)=
   0  -5  -4
   0   0  -4
   0   0   0
  ...

  Z(49)=
   0  -1   0
   0   0  -5
   0   0   0
  Z(50)=
   0  -1   2
   0   0  -2
   0   0   0
```

4.3.19 `fc_amat.random.randsdd` function

The `fc_amat.random.randsdd` function return an `amat` object whose matrices are strictly diagonally dominant with non-diagonal elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

Syntaxe

```
X=fc_amat.random.randsdd(N,d)
X=fc_amat.random.randsdd(...,key,value)
```

Description

`X=fc_amat.random.randsdd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant with non-diagonal elements drawn from the uniform distribution on the interval $]a, b[=]0, 1[$.

```
X=fc_amat.random.randsdd(...,key,value)
```

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts elements are also drawn from the uniform distribution on the interval $]a, b[=]0, 1[$. (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'a'` , to set a (lower bound of the interval) value (0 by default).
- `'b'` , to set b (upper bound of the interval) value (1 by default).

In Listing 28, some examples are provided.

Listing 28: : examples of `fc_amat.random.randsdd` function usage

```
X=fc_amat.random.randsdd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randsdd(200,3,'a',-2,'b',2);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randsdd(50,3,'complex',true,'a',-1,'class','single');
% Z: 50-by-3-by-3 single amat
disp('List␣current␣variables␣:')
whos
disp('Print␣Z␣amat␣object␣:')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name          Size                Bytes  Class
   ==== ====          ====                =====  =====
        SaveOptions   1x6                    25  cell
        X             1x1                     0  amat
        Y             1x1                     0  amat
        Z             1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[complex single] object
  Z(1)=
   1.82130 + 2.44173i 0.92301 - 0.83918i 0.64689 + 0.48471i
  -0.75664 + 0.58265i 1.69748 + 2.03362i -0.70986 - 0.65586i
   0.70379 - 0.38836i -0.23540 + 0.38262i -1.62677 + 1.73656i
  Z(2)=
   0.94083 + 2.82209i -0.69923 - 0.90343i 0.61767 - 0.58252i
   0.77167 + 0.44795i -2.57718 - 0.65203i 0.31135 + 0.55797i
   0.05727 - 0.80282i -0.14683 - 0.19362i 1.33822 - 1.42426i
  ...

  Z(49)=
  -2.55436 - 1.71435i 0.58283 + 0.96068i -0.71920 - 0.63674i
  -0.91747 - 0.31792i -2.50032 + 1.69249i -0.59664 - 0.97891i
  -0.11547 + 0.03776i 0.31316 + 0.11007i 0.89640 + 0.71681i
  Z(50)=
  -1.656294 + 1.969870i 0.651576 + 0.465400i -0.048147 - 0.885714i
  -0.206838 + 0.008966i 1.931039 + 0.537902i 0.928770 + 0.768489i
  -0.104545 - 0.658196i -0.211330 + 0.064402i 1.066469 + 1.247006i
```

4.3.20 `fc_amat.random.randnsdd` function

The `fc_amat.random.randnsdd` function return an `amat` object whose matrices are strictly diagonally dominant with non-diagonal elements drawn from the normal distribution having zero mean and unit standard deviation.

Syntaxe

```
X=fc_amat.random.randnsdd(N,d)
X=fc_amat.random.randnsdd(...,key,value)
```

Description

`X=fc_amat.random.randnsdd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant with non-diagonal elements drawn from the normal distribution having zero mean and unit standard deviation.

`X=fc_amat.random.randnsdd(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'` , if value is `true` the `amat` object is complex and the imaginary parts of the upper triangular matrices elements are also drawn from the normal distribution having zero mean and unit standard deviation (default `false` i.e real `amat` object)
- `'class'` , to set `amat` object data type; value could be `'single'` or `'double'` (default).
- `'mean'` , to set mean of the normal distribution (0 by default).
- `'sigma'` , to set standard deviation of the normal distribution (1 by default).

In Listing 29, some examples are provided.

Listing 29: : examples of `fc_amat.random.randnsdd` function usage

```
X=fc_amat.random.randnsdd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randnsdd(200,3,'complex',true,'sigma',5);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randnsdd(50,3,'class','single','mean',5);
% Z: 50-by-3-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
  -16.2419   4.8539   6.4797
    5.2433 -15.3493   5.1978
    5.1973   5.4701 -15.7322
  Z(2)=
  -15.4242   5.0299   4.2395
    5.2160 -16.7547   5.8927
    5.6284   4.7385 -14.2700
  ...

  Z(49)=
  -16.3509   4.9734   6.2882
    4.6682  15.5086   4.9399
    3.9419   6.1464 -14.2336
  Z(50)=
   13.7269   4.9052   3.4033
    4.1906 -13.5413   4.7456
    4.6441   5.0364 -15.6661
```

4.3.21 `fc_amat.random.randisdd` function

The `fc_amat.random.randisdd` function return an `amat` object whose matrices are strictly diagonally dominant with random integers

Syntaxe

```
X=fc_amat.random.randisdd(Imax,N,d)
X=fc_amat.random.randisdd([Imin,Imax],...)
X=fc_amat.random.randisdd(...,key,value)
```

Description

`X=fc_amat.random.randisdd(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on `1:Imax` .

`X=fc_amat.random.randisdd([Imin,Imax],N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on `Imin:Imax`.

`X=fc_amat.random.randisdd(...,key,value)`

Some optional key/value pairs arguments are available with keys:

- `'complex'`, if value is `true` the `amat` object is complex and the imaginary parts of the non-diagonal elements are also drawn from the discrete uniform distribution (default `false` i.e real `amat` object).
- `'class'`, to set `amat` object data type; value are those of the `randi` Matlab function. Default is `'double'`.

In Listing 30, some examples are provided.

Listing 30: : examples of `fc_amat.random.randisdd` function usage

```
X=fc_amat.random.randisdd(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randisdd(8,200,3,'class','single');
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randisdd([-5,5],50,3,'class','single','complex',true);
% Z: 50-by-2-by-2 single amat
disp('List␣current␣variables␣:')
whos
disp('Print␣Z␣amat␣object␣:')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat
        Z              1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[complex single] object
  Z(1)=
   -6 + 9i    0 - 1i    1 - 5i
    5 - 4i   14 + 10i    1 - 2i
    1 + 2i    1 - 4i    4 + 11i
  Z(2)=
    4 - 18i   4 + 2i   -5 - 3i
    5 - 5i  -12 + 12i   3 + 2i
   -2 + 1i    4 - 2i   10 + 11i
  ...

  Z(49)=
   11 + 5i   -2 - 2i    2 + 5i
    3 - 5i   13 + 10i  -4 + 4i
    5 + 5i   -4 - 4i  -11 + 11i
  Z(50)=
   -8 - 9i   -2 - 1i   -1 - 2i
   -2 + 4i   17 + 7i    0 + 4i
    0 - 2i    5 + 0i   -2 - 13i
```

4.3.22 `fc_amat.random.randsympd` function

The `fc_amat.random.randsympd` function return an `amat` object whose matrices are symmetric positive definite. This object is generated by using `randsdd` function from `fc_amat.random` namespace.

Syntaxe

```
X=fc_amat.random.randsympd(N,d)
X=fc_amat.random.randsympd(...,key,value)
```

Description

`X=fc_amat.random.randsympd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric positive definite.

`X=fc_amat.random.randsympd(...,key,value)`

Optional key/value pairs arguments are those of the `fc_amat.random.randnsdd` function except for `'complex'` key which is forced to `false`. Keys can be:

- `'class'`, to set `amat` object data type; value can be `'single'` or `'double'` (default).
- `'a'`, to set a (lower bound of the interval) value (0 by default).
- `'b'`, to set b (upper bound of the interval) value (1 by default).

In Listing 31, some examples are provided.

Listing 31: : examples of `fc_amat.random.randsympd` function usage

```
X=fc_amat.random.randsympd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randsympd(200,3,'a',-2,'b',2);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randsympd(50,3,'a',-1,'class','single');
% Z: 50-by-3-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   0.79832 -0.39764 -1.07966
  -0.39764  4.77610 -3.75622
  -1.07966 -3.75622  5.83655
  Z(2)=
   1.95787  0.37171 -0.10843
   0.37171  2.20853  0.42402
  -0.10843  0.42402  0.96431
  ...

  Z(49)=
   2.7371 -2.2359 -1.9445
  -2.2359  5.9287  2.9052
  -1.9445  2.9052  6.8980
  Z(50)=
   1.85455 -0.79858 -0.41873
  -0.79858  2.46552  0.53532
  -0.41873  0.53532  0.96403
```

4.3.23 `fc_amat.random.randnsympd` function

The `fc_amat.random.randnsympd` function return an `amat` object whose matrices are symmetric positive definite. This object is generated by using `fc_amat.random.randnsdd` function.

Syntaxe

```
X=fc_amat.random.randnsympd(N,d)
X=fc_amat.random.randnsympd(...,key,value)
```

Description

`X=fc_amat.random.randnsympd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric positive definite.

```
X=fc_amat.random.randnsympd(...,key,value)
```

Optional key/value pairs arguments are those of the `fc_amat.random.randnsdd` function except for `'complex'` key which is forced to `false`. Keys can be:

- `'class'`, to set `amat` object data type; value can be `'single'` or `'double'` (default).
- `'mean'`, to set mean of the normal distribution (0 by default).
- `'sigma'`, to set standard deviation of the normal distribution (1 by default).

In Listing 32, some examples are provided.

Listing 32: : examples of `fc_amat.random.randnsympd` function usage

```
X=fc_amat.random.randnsympd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randnsympd(200,3,'sigma',5);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randnsympd(50,3,'class','single','mean',5);
% Z: 50-by-3-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
   351.80 -113.39 -212.95
  -113.39  256.65 -125.42
  -212.95 -125.42  377.58
  Z(2)=
   323.023   15.048 -134.546
    15.048  281.507   38.280
  -134.546   38.280  248.126
  ...

  Z(49)=
   311.6880   48.8840   -8.2757
    48.8840  360.1663  180.0631
    -8.2757  180.0631  252.3960
  Z(50)=
   284.30713    0.41471   20.96068
     0.41471  321.52509 -109.41101
    20.96068 -109.41101  228.93466
```

4.3.24 `fc_amat.random.randisympd` function

The `fc_amat.random.randisympd` function return an `amat` object whose matrices are symmetric positive definite with random integers. This object is gen-

erated by using `randisympd` function from `fc_amat.random` namespace.

Syntaxe

```
X=fc_amat.random.randisympd(Imax,N,d)
X=fc_amat.random.randisympd([Imin,Imax],...)
X=fc_amat.random.randisympd(...,key,value)
```

Description

`X=fc_amat.random.randisympd(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on `1:Imax` .

`X=fc_amat.random.randisympd([Imin,Imax],N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on `Imin:Imax` .

`X=fc_amat.random.randisympd(...,key,value)`

Optional key/value pairs arguments are those of the `randisdd` function except for `'complex'` key which is forced to `false` and `'class'` key which can only be `'single'` or `'double'` . Keys can be:

- `'class'` , to set `amat` object data type; value can be `'single'` or `'double'` (default).

In Listing 33, some examples are provided.

Listing 33: : examples of `fc_amat.random.randisympd` function usage

```
X=fc_amat.random.randisympd(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randisympd(8,200,3,'class','single');
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randisympd([-5,5],50,3,'class','single');
% Z: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes  Class
   ==== ====           ====                =====  =====
        SaveOptions    1x6                    25  cell
        X              1x1                     0  amat
        Y              1x1                     0  amat
        Z              1x1                     0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[single] object
  Z(1)=
    45    27   -39
    27    97   -63
   -39   -63   115
  Z(2)=
    41    -6    17
    -6   126    12
    17    12   173
  ...

  Z(49)=
   206    44   100
    44   125    21
   100    21   115
  Z(50)=
   110    24     5
    24   108   -74
     5   -74   129
```

4.3.25 `fc_amat.random.randherpd` function

The `fc_amat.random.randherpd` function return an `amat` object whose matrices are hermitian positive definite. This object is generated by using `randsdd` function from `fc_amat.random` namespace.

Syntaxe

```
X=fc_amat.random.randherpd(N,d)
X=fc_amat.random.randherpd(...,key,value)
```

Description

`X=fc_amat.random.randherpd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are symmetric positive definite.

`X=fc_amat.random.randherpd(...,key,value)`

Optional key/value pairs arguments are those of the `fc_amat.random.randnsdd` function except for `'complex'` key which is forced to `true`. keys can be:

- `'class'`, to set `amat` object data type; value can be `'single'` or `'double'` (default).
- `'a'`, to set a (lower bound of the interval) value (0 by default).
- `'b'`, to set b (upper bound of the interval) value (1 by default).

In Listing 34, some examples are provided.

Listing 34: : examples of `fc_amat.random.randherpd` function usage

```
X=fc_amat.random.randherpd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randherpd(200,3,'a',-2,'b',2);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randherpd(50,3,'a',-1,'class','single');
% Z: 50-by-3-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
warning: function name 'randsympd' does not agree with function filename ...
    '/home/cuvelier/Travail/Recherch/Matlab/fc-config/build/tmpdir/packages/fc_amat-0.1.2/+fc_amat/+random/randherpd.m'
warning: called from
   randherpd01 at line 1 column 2
   fctoto at line 4 column 2
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[complex single] object
  Z(1)=
   2.66537 + 0.00000i -0.77662 + 2.69829i 1.30669 + 0.83546i
  -0.77662 - 2.69829i 4.40465 + 0.00000i -0.06135 - 2.50612i
   1.30669 - 0.83546i -0.06135 + 2.50612i 3.78700 + 0.00000i
  Z(2)=
   2.90871 + 0.00000i -0.08088 - 2.35498i -0.27468 + 1.71907i
  -0.08088 + 2.35498i 7.78436 + 0.00000i -1.14395 - 4.60380i
  -0.27468 - 1.71907i -1.14395 + 4.60380i 6.97446 + 0.00000i
  ...

  Z(49)=
   7.75465 + 0.00000i -2.76181 - 2.69206i 2.41451 - 1.09117i
  -2.76181 + 2.69206i 7.92448 + 0.00000i -0.24347 + 1.93767i
   2.41451 + 1.09117i -0.24347 - 1.93767i 7.71438 + 0.00000i
  Z(50)=
   7.94404 + 0.00000i -1.40789 - 4.52225i 0.75344 + 3.25722i
  -1.40789 + 4.52225i 8.04111 + 0.00000i -1.54385 + 0.95836i
   0.75344 - 3.25722i -1.54385 - 0.95836i 4.53781 + 0.00000i
```

4.3.26 `fc_amat.random.randnherpd` function

The `fc_amat.random.randnherpd` function return an `amat` object whose matrices are hermitian positive definite. This object is generated by using `randnsdd` function from `fc_amat.random` namespace.

Syntaxe

```
X=fc_amat.random.randnherpd(N,d)
X=fc_amat.random.randnherpd(...,key,value)
```

Description

`X=fc_amat.random.randnherpd(N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are Hermitian positive definite.

`X=fc_amat.random.randnherpd(...,key,value)`

Optional key/value pairs arguments are those of the `randnsdd` function except for `'complex'` key which is forced to `true`. Keys can be:

- `'class'`, to set `amat` object data type; value can be `'single'` or `'double'` (default).
- `'mean'`, to set mean of the normal distribution (0 by default).
- `'sigma'`, to set standard deviation of the normal distribution (1 by default).

In Listing 35, some examples are provided.

Listing 35: : examples of `fc_amat.random.randnherpd` function usage

```
X=fc_amat.random.randnherpd(100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randnherpd(200,3,'sigma',5);
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randnherpd(50,3,'class','single','mean',5);
% Z: 50-by-3-by-3 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        Y              1x1                       0  amat
        Z              1x1                       0  amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[complex single] object
  Z(1)=
   582.185 +  0.000i  306.119 + 154.592i -10.545 + 85.487i
   306.119 - 154.592i 573.796 +  0.000i  122.437 - 85.941i
   -10.545 - 85.487i  122.437 + 85.941i  519.305 +  0.000i
  Z(2)=
   506.100 +  0.000i  -17.513 - 188.091i -164.563 - 96.504i
   -17.513 + 188.091i 567.228 +  0.000i  -21.353 + 47.878i
  -164.563 + 96.504i  -21.353 - 47.878i  456.978 +  0.000i
  ...

  Z(49)=
   580.325 +  0.000i -112.770 + 122.868i -149.556 - 255.373i
  -112.770 - 122.868i 558.757 +  0.000i  -98.095 - 275.457i
  -149.556 + 255.373i -98.095 + 275.457i 685.978 +  0.000i
  Z(50)=
   512.873 +  0.000i  -90.324 - 258.985i  40.898 + 30.949i
   -90.324 + 258.985i 652.527 +  0.000i  135.560 + 263.580i
    40.898 - 30.949i  135.560 - 263.580i 506.772 +  0.000i
```

4.3.27 `fc_amat.random.randiherpd` function

The `fc_amat.random.randiherpd` function return an `amat` object whose matrices are Hermitian positive definite with random integers. This object is generated by using `randiherpd` function from `fc_amat.random` namespace.

Syntaxe

```
X=fc_amat.random.randiherpd(Imax,N,d)
X=fc_amat.random.randiherpd([Imin,Imax],...)
X=fc_amat.random.randiherpd(...,key,value)
```

Description

`X=fc_amat.random.randiherpd(Imax,N,d)`

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally

dominant and non-diagonal elements are pseudorandom integer drawn from the discrete uniform distribution on `1:Imax`.

```
X=fc_amat.random.randiherpd([Imin,Imax],N,d)
```

returns a `N`-by-`d`-by-`d` `amat` object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudorandom integer drawn from the discrete uniform distribution on `Imin:Imax`.

```
X=fc_amat.random.randiherpd(...,key,value)
```

Optional key/value pairs arguments are those of the `randisdd` function except for `'complex'` key which is forced to `true` and `'class'` key which can only be `'single'` or `'double'`. Keys can be:

- `'class'`, to set `amat` object data type; value can be `'single'` or `'double'` (default).

In Listing 36, some examples are provided.

Listing 36: : examples of `fc_amat.random.randiherpd` function usage

```
X=fc_amat.random.randiherpd(10,100,3);
% X: 100-by-3-by-3 amat
Y=fc_amat.random.randiherpd(8,200,3,'class','single');
% Y: 200-by-3-by-4 amat
Z=fc_amat.random.randiherpd([-5,5],50,3,'class','single');
% Z: 50-by-2-by-2 single amat
disp('List current variables :')
whos
disp('Print Z amat object :')
disp(Z,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                Bytes Class
   ==== ====           ====                ===== =====
        SaveOptions    1x6                    25 cell
        X              1x1                     0 amat
        Y              1x1                     0 amat
        Z              1x1                     0 amat

Total is 9 elements using 25 bytes

Print Z amat object :
Z is a 50x3x3 amat[complex single] object
  Z(1)=
   288 +  0i    1 - 88i  -79 + 62i
     1 + 88i  356 +  0i  -15 + 139i
   -79 - 62i  -15 - 139i 213 +  0i
  Z(2)=
   139 +  0i   46 - 55i   22 - 24i
    46 + 55i  186 +  0i   30 - 30i
    22 + 24i   30 + 30i  173 +  0i
  ...

  Z(49)=
   254 +  0i  -46 - 105i  78 - 19i
   -46 + 105i 337 +  0i  205 - 44i
    78 + 19i  205 + 44i  373 +  0i
  Z(50)=
   375 +  0i   80 + 23i   26 + 87i
    80 - 23i  451 +  0i -180 - 22i
    26 - 87i -180 + 22i  376 +  0i
```

5 Indexing

5.1 Subscripted reference

Let `A` be a `N`-by-`m`-by-`n` `amat` object.

5.1.1 `A(K,I,J)`

- With `K`, `I`, `J` three 1D-arrays of indices, a `length(K)`-by-`length(I)`-by-`length(J)` `amat` object is returned where $\forall$`i` $\in$ `1:length(I)`, $\forall$`j` $\in$ `1:length(J)`, $\forall$ `k` $\in$ `1:length(K)` the element (`i`, `j`) of its `k`-th matrix is the element (`I(i)`, `J(j)`)) of `K(k)`-th matrix of `A`, i.e. with `B` denoting the output `amat` object:

  `B(k,i,j)` $\leftarrow$ `A( k, I(i), J(j))`.

  If `length(K)==1`, then the returned object is a `length(I)`-by-`length(J)` matrix such that

  `B(i,j)` $\leftarrow$ `A( k, I(i), J(j))`.

- *(experimental)* With `K`, `I`, `J` three `M`-by-`p`-by-`q` `amat` object a `M`-by-`p`-by-`q` `amat` object is returned where $\forall$`i` $\in$ `1:p`, $\forall$`j` $\in$ `1:q`, $\forall$ `k` $\in$ `1:M` the element (`i`, `j`) of its `k`-th matrix is the element (`I(k,i,j)`, `J(k,i,j)`) of `K(k,i,j)`-th matrix of `A`, i.e. with `B` denoting the output `amat` object:

  `B(k,i,j)` $\leftarrow$ `A( K(k,i,j), I(k,i,j), J(k,i,j))`.

The commands `A(K,I,:)` and `A(K,I,1:end)` are equivalent to `A(K,I,1:n)`.
The commands `A(K,:,J)` and `A(K,:,J)` are equivalent to `A(K,:,1:n)`.
The commands `A(:,I,J)` and `A(1:end,I,J)` are equivalent to `A(1:N,I,J)`.
The commands `A(K,:,:)` and `A(K,1:end,1:end)` are equivalent to `A(K,1:m,1:m)`.
...

5.1.2 `A(K)`

Identically to `A(K,:,:)`.

5.1.3 `A(I,J)`

Identically to `A(:,I,J)`.
In Listing 37, some examples are provided.

Listing 37: : examples of `subsref` method usage

```
N=100;m=2;n=3;
X=fc_amat.random.randi(9,[N,m,n]);
A=X(1,2,2); % A is a scalar
B=X([2,end-1],1:2,[1,3]);
info(B)
C=X(1); % C is a m-by-n matrix
D=X(1:10);
info(D)
E=X(1,2);
info(E)
F=X(1,[1,3]);
info(F)
p=2;q=2;
K=fc_amat.ones(N,p,q).*[1:N]';
I=fc_amat.random.randi(m,[N,p,q]);
J=fc_amat.random.randi(n,[N,p,q]);
sK=1:2:N;
G=X(K(sK),I(sK),J(sK));
info(G)
H=X(I,J);
info(H)
disp('List␣of␣some␣variables␣:')
whos A C sK
```

Output

```
B is a 2x2x2 amat[double] object
D is a 10x2x3 amat[double] object
E is a 100x1x1 amat[double] object
F is a 100x1x2 amat[double] object
G is a 50x2x2 amat[double] object
H is a 100x2x2 amat[double] object
List of some variables :
Variables in the current scope:

   Attr Name      Size                Bytes  Class
   ==== ====      ====                =====  =====
        A         1x1                     8  double
        C         2x3                    48  double
        sK        1x50                   24  double

Total is 57 elements using 80 bytes
```

5.2 Subscripted assignment

Let `A` be a `N`-by-`m`-by-`n` `amat` object.

5.2.1 `A(K,I,J)=B`

- `I`, `J` and `K` are scalars indices, `B` must be a scalar and it is assigned to element `(I,J)` of the `K`-th matrix of `A`.
- `I`, `J` and `K` are 1D-arrays of indices. Then three cases are possible
  - `B` is a scalar, then

    $$\texttt{A(k,i,j)=B},\quad \forall \texttt{i} \in \texttt{I}, \forall \texttt{j} \in \texttt{J}, \forall \texttt{k} \in \texttt{K}.$$

  - `B` is a `length(I)` $\times$ `length(J)` matrix, then $\forall \texttt{k} \in \texttt{1:length(K)}$ the `K(k)`-th matrix of `A` is set to `B`, i.e. $\forall \texttt{i} \in \texttt{1:length(I)}$, $\forall \texttt{j} \in \texttt{1:length(J)}$,

    $$\texttt{A(K(k),I(i),J(j))=B(i,j)}.$$

– `B` is a `length(K)`-by-`length(I)`-by-`length(J)` `amat` object then $\forall$ `k` $\in$ `1:length(K)` the `K(k)`-th matrix of `A` is set to `k`-th matrix of `B`, i.e. $\forall$`i` $\in$ `1:length(I)`, $\forall$`j` $\in$ `1:length(J)`,

`A(K(k),I(i),J(j))=B(k,i,j)`.

- `I`, `J` and `K` are `M`-by-`p`-by-`q` `amat` objects of indices

  Then three cases are possible

  – `B` is a scalar, then $\forall$`i` $\in$ `1:p`, $\forall$`j` $\in$ `1:q`, $\forall$`k` $\in$ `1:M`

  `A(K(k,i,j),I(k,i,j),J(k,i,j))=B`

  – *(experimental)* `B` is a `M`-by-`p`-by-`q` `amat` object then $\forall$`i` $\in$ `1:p`, $\forall$`j` $\in$ `1:q`, $\forall$`k` $\in$ `1:M`

  `A(K(k,i,j),I(k,i,j),J(k,i,j))=B(k,i,j)`

If `max(I)>m`, `max(J)>n` or `max(K)>N` then before assignment `A` is reshaped to fit the new size by setting 0 for missing elements.

5.2.2 `A(K)=B`

Identically to the equivalent commands `A(K,1:m,1:n)=B` or `A(K,:,:)=B` or `A(K,1:end,1:end)=B`

5.2.3 `A(I,J)=B`

If `B` is a scalar or a matrix or an `amat` object, this command is equivalent to one of these commands `A(1:N,I,J)=B` or `A(:,I,J)=B` or `A(1:end,I,J)=B`. If `B` is a `N`-by-1 array then $\forall$`k` $\in$ `1:N`, $\forall$`i` $\in$ `1:length(I)`, $\forall$`j` $\in$ `1:length(J)`,

`A(k,I(i),J(j))=B(k)`.

In Listing 38, some examples are provided.

Listing 38: : examples of `subsasgn` method usage

```
N=100;m=3;n=2;
X=fc_amat.ones(N,m,n,'int32');
X(2,1,2)=3;
X([2,N],1:2,[1,3])=2;
X(1)=-1;
X([2,N])=0;
X(3,3)=1:N;
disp('Print␣X␣amat␣object␣:')
X
```

Output

```
Print X amat object :
X =

 is a 100x3x3 amat[int32] object
  matrix(1)=
  -1  -1  -1
  -1  -1  -1
  -1  -1   1
  matrix(2)=
  0  0  0
  0  0  0
  0  0  2
  ...

  matrix(99)=
   1   1   0
   1   1   0
   1   1  99
  matrix(100)=
    0    0    0
    0    0    0
    0    0  100
```

6 Elementary operations

6.1 Arithmetic operations

The implemented element by element arithmetic operators/methods for `amat` objects are:

- `+` / `plus` , addition
- `+` / `uplus` , unary plus
- `-` / `minus` , subtraction
- `-` / `uminus` , unary minus
- `.*` / `times` , element-wise multiplication
- `./ rdivide` , element-by-element right division
- `.\` / `ldivide` , element-by-element left division

Let $\mathbb{A} \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$, (i.e. a N-by-m-by-n `amat` object) we now explain how a generic binary operator, denoted by $\bigotimes$, act between $\mathbb{A}$ and another input data. We define four kinds of element by element arithmetic binary operations when $\mathbb{A}$ is the left operand.

1. Let $\mathbb{B} \in \left(\mathcal{M}_{m,n}(\mathbb{K})\right)^N$, we have

$$\mathbb{A} \bigotimes \mathbb{B} \overset{\text{def}}{=} \mathbb{C} \in \left(\mathcal{M}_{m,n}(\mathbb{K})\right)^N \tag{1}$$

where $\forall k \in [\![1, N]\!]$

$$\mathbb{C}_k(i,j) = \mathbb{A}_k(i,j) \bigotimes \mathbb{B}_k(i,j), \quad \forall i \in [\![1,m]\!],\ \forall j \in [\![1,n]\!].$$

2. Let $\mathbb{B} \in \mathcal{M}_{m,n}(\mathbb{K})$, we have

$$\mathbb{A} \bigotimes \mathbb{B} \overset{\text{def}}{=} \mathbb{C} \in \left(\mathcal{M}_{m,n}(\mathbb{K})\right)^N \tag{2}$$

where $\forall k \in [\![1, N]\!]$

$$\mathbb{C}_k(i,j) = \mathbb{A}_k(i,j) \bigotimes \mathbb{B}(i,j), \quad \forall i \in [\![1,m]\!],\ \forall j \in [\![1,n]\!].$$

3. Let $\boldsymbol{B} \in \mathbb{K}^N$, (i.e. a N-by-1 array) we have

$$\mathbb{A} \bigotimes \boldsymbol{B} \overset{\text{def}}{=} \mathbb{C} \in \left(\mathcal{M}_{m,n}(\mathbb{K})\right)^N \tag{3}$$

where $\forall k \in [\![1, N]\!]$

$$\mathbb{C}_k(i,j) = \mathbb{A}_k(i,j) \bigotimes \boldsymbol{B}(k), \quad \forall i \in [\![1,m]\!],\ \forall j \in [\![1,n]\!].$$

4. Let $B \in \mathbb{K}$, we have

$$\mathbb{A} \bigotimes B \overset{\text{def}}{=} \mathbb{C} \in \left(\mathcal{M}_{m,n}(\mathbb{K})\right)^N \tag{4}$$

where $\forall k \in [\![1, N]\!]$

$$\mathbb{C}_k(i,j) = \mathbb{A}_k(i,j) \bigotimes B, \quad \forall i \in [\![1,m]\!],\ \forall j \in [\![1,n]\!].$$

When $\mathbb{A}$ is the right operand element by element binary operations can be easily deduced.

In Listing 39, some examples are provided.

Listing 39: : examples of element by element operations

```
N=100; m=2;n=3;
X=fc_amat.ones(N,m,n);
A=X+2;
B=[1:N]'.*X;
M=rand(m,n);
C=M-X;
D=C./(2.*X);
disp('List␣current␣variables␣:')
whos
disp('Print␣D␣amat␣object␣:')
disp(D,'n',2)
```

Output

```
List current variables :
Variables in the current scope:

   Attr Name           Size                  Bytes  Class
   ==== ====           ====                  =====  =====
        A              1x1                       0  amat
        B              1x1                       0  amat
        C              1x1                       0  amat
        D              1x1                       0  amat
        M              2x3                      48  double
        N              1x1                       8  double
        SaveOptions    1x6                      25  cell
        X              1x1                       0  amat
        m              1x1                       8  double
        n              1x1                       8  double

Total is 20 elements using 97 bytes

Print D amat object :
D is a 100x2x3 amat[double] object
  D(1)=
  -0.012798 -0.375869 -0.019882
  -0.440048 -0.490302 -0.162417
  D(2)=
  -0.012798 -0.375869 -0.019882
  -0.440048 -0.490302 -0.162417
  ...

  D(99)=
  -0.012798 -0.375869 -0.019882
  -0.440048 -0.490302 -0.162417
  D(100)=
  -0.012798 -0.375869 -0.019882
  -0.440048 -0.490302 -0.162417
```

6.2 Relational operators

The implemented element by element relational operators/methods for `amat` objects are:

- `==` / `eq` , equality
- `>=` / `ge` , greater than or equal
- `>` / `gt` , greater than
- `<=` / `le` , less than or equal
- `<` / `lt` , less than
- `~=` / `ne` , inequality

With these binary operators, four kind element by element operations occur. They are the same as those described for the *element by element arithmetic operations*, Section 6.1, and given by (1) to (4) except that the output differs: it is a **logical** `amat` object.

In Listing 40, some examples are provided.

Listing 40: : examples of relational operators

```
N=100; m=2;n=3;
X=fc_amat.random.randn(N,m,n);
Y=randn(m,n);
Z=randn(N,1);
W=fc_amat.random.randn(N,m,n);
A= X>=0;
info(A)
B= X<Y;
info(B)
C= X==Z;
info(C)
D= X~=W;
disp(D)
```

Output

```
A is a 100x2x3 amat[logical] object
B is a 100x2x3 amat[logical] object
C is a 100x2x3 amat[logical] object
D is a 100x2x3 amat[logical] object
  D(1)=
  1  1  1
  1  1  1
  D(2)=
  1  1  1
  1  1  1
  ...

  D(99)=
  1  1  1
  1  1  1
  D(100)=
  1  1  1
  1  1  1
```

6.3 Logical operations

The implemented logical operators/methods for `amat` objects are:

- `&` / `and` , logical and
- `|` / `or` , logical or
- `~` / `not` , logical not
- `xor` , logical xor
- `all` , ...
- `any` , ...

With the binary operators `and` , `or` , and `xor` four kind element by element operations occur. They are the same as those described for the *element by element arithmetic operations*, section 6.1, and given by (1) to (4) except that the output differs: it is a **logical** `amat` object.

In Listing 41, some examples are provided.

Listing 41: : examples of relational operators

```
N=100; m=2;n=3;
X=( fc_amat.random.randi([-2,2],N,m,n) >=0 );
y=( randi([-2,2],m,n) <0 );
w=( randi([-2,2],N,1) <=1 );
A= X & y;
info(A)
B= X | w;
info(B)
C= ~B;
info(C)
D= xor(X,C);
disp(D)
```

Output

```
A is a 100x2x3 amat[logical] object
B is a 100x2x3 amat[logical] object
C is a 100x2x3 amat[logical] object
D is a 100x2x3 amat[logical] object
  D(1)=
  1  1  1
  1  1  0
  D(2)=
  1  1  1
  1  1  1
  ...

  D(99)=
  1  0  1
  0  1  0
  D(100)=
  1  1  0
  1  0  0
```

6.3.1 `all` method

Let `X` be a `N`-by-`m`-by-`n` `amat` object. The `all` method of `X` return a `N`-by-`1`-by-`1` logical `amat` object such that its k-th element (`1`-by-`1` matrix) is `true` (logical 1) if all elements of the k-th matrix of `X` are all nonzero.

Syntaxe

```
B=all(X)
B=all(X,dim)
```

Description

`B=all(X)`

return a `N`-by-`1`-by-`1` logical `amat` object such that `B(k,1,1)` is one (logical `true`) if $\forall$`i` $\in$ `[1:m]`, $\forall$`j` $\in$ `[1:n]`, `A(k,i,j)` is nonzero. Otherwise `B(k,1,1)` is zero (logical `false`).

`B=all(X,dim)`

- `dim=1` , along rows of matrices of `X`. Returns a `N`-by-`1`-by-`n` logical `amat` object such that `B(k,1,j)` is one (logical `true`) if $\forall$`i` $\in$

[1:m], A(k,i,j) is nonzero. Otherwise, B(k,1,j) is zero (logical false).

- dim=2 , along columns of matrices of X. Returns a N-by-m-by-1 logical amat object such that B(k,i,1) is one (logical true) if ∀j ∈ [1:n], A(k,i,j) is nonzero. Otherwise, B(k,i,1) is zero (logical false).
- dim=3 , (default value) , along rows and columns of matrices of X. Returns a N-by-1-by-1 logical amat object such that B(k,1,1) is one (logical true) if ∀i ∈ [1:m], ∀j ∈ [1:n], A(k,i,j) is nonzero. Otherwise, B(k,1,1) is zero (logical false).
- dim=0 , along matrices index of X. Returns return a m-by-n logical matrix such that B(i,j) is one (logical true) if ∀k ∈ [1:N], A(k,i,j) is nonzero. Otherwise, B(i,j) is zero (logical false).

In Listing 42, some examples are provided.

Listing 42: : examples of all function usage

```
X=fc_amat.random.rand(100,2,3);
info(X)
A=all(X>0); info(A)
B=all(X>0,1); info(B)
C=all(X>0,2); info(C)
D=all(X>0,0);
fprintf('␣D␣is␣\n␣');disp(D)
E=all(all(X>0),0);
fprintf('␣E␣is␣\n␣');disp(E)
```

Output

```
X is a 100x2x3 amat[double] object
A is a 100x1x1 amat[logical] object
B is a 100x1x3 amat[logical] object
C is a 100x2x1 amat[logical] object
 D is
   1  1  1
  1  1  1
 E is
 1
```

6.3.2 any method

Let X be a N-by-m-by-n amat object. The any method of X return a N-by-1-by-1 logical amat object such that its k-th element (1-by-1 matrix) is true (logical 1) if any of the elements of the k-th matrix of X is nonzero.

Syntaxe

```
B=any(X)
B=any(X,dim)
```

Description

`B=any(X)`

return a `N`-by-`1`-by-`1` logical `amat` object such that `B(k,1,1)` is one (logical `true`) if $\exists$`i` $\in$ `[1:m]`, $\exists$`j` $\in$ `[1:n]`, `A(k,i,j)` is nonzero.

`B=any(X,dim)`

- `dim=1` , along rows of matrices of `X`. Returns a `N`-by-`1`-by-`n` logical `amat` object such that `B(k,1,j)` is one (logical `true`) if $\exists$`i` $\in$ `[1:m]`, `A(k,i,j)` is nonzero. Otherwise, `B(k,1,j)` is zero (logical `false`).
- `dim=2` , along columns of matrices of `X`. Returns a `N`-by-`m`-by-`1` logical `amat` object such that `B(k,i,1)` is one (logical `true`) if $\exists$`j` $\in$ `[1:n]`, `A(k,i,j)` is nonzero. Otherwise, `B(k,i,1)` is zero (logical `false`).
- `dim=3` , (default value) , along rows and columns of matrices of `X`. Returns a `N`-by-`1`-by-`1` logical `amat` object such that `B(k,1,1)` is one (logical `true`) if $\exists$`i` $\in$ `[1:m]`, $\exists$`j` $\in$ `[1:n]`, `A(k,i,j)` is nonzero. Otherwise, `B(k,1,1)` is zero (logical `false`).
- `dim=0` , along matrices index of `X`. Returns return a `m`-by-`n` logical matrix such that `B(i,j)` is one (logical `true`) if $\exists$`k` $\in$ `[1:N]`, `A(k,i,j)` is nonzero. Otherwise, `B(i,j)` is zero (logical `false`).

In Listing 43, some examples are provided.

Listing 43: : examples of `fc_amat.random.randher` function usage

```
X=fc_amat.random.rand(100,2,3);
info(X)
A=any(X>0); info(A)
B=any(X>0,1); info(B)
C=any(X>0,2); info(C)
D=any(X>0,0);
fprintf('␣D␣is␣\n␣');disp(D)
E=any(any(X>0),0);
fprintf('␣E␣is␣\n␣');disp(E)
```

Output

```
X is a 100x2x3 amat[double] object
A is a 100x1x1 amat[logical] object
B is a 100x1x3 amat[logical] object
C is a 100x2x1 amat[logical] object
 D is
   1  1  1
  1  1  1
 E is
 1
```

7 Elementary mathematical functions

A lot of elementary mathematical functions can be used with `amat` objects. In Listing 44, some examples are provided and complete lists are given thereafter.

Listing 44: : examples of elementary mathematical functions

```
A=fc_amat.random.randiher(10,100,3);
info(A);
X=cos(A);
info(X);
Y=sin(A);
info(Y) ;
Z=X.^2+Y.^2;
disp('Print␣Z␣amat␣object␣:')
Z
```

Output

```
A is a 100x3x3 amat[complex double] object
X is a 100x3x3 amat[complex double] object
Y is a 100x3x3 amat[complex double] object
Print Z amat object :
Z =

 is a 100x3x3 amat[complex double] object
  matrix(1)=
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 - 0.00000i
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
  matrix(2)=
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 - 0.00000i 1.00000 + 0.00000i 1.00000 - 0.00000i
  ...

  matrix(99)=
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 - 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
  matrix(100)=
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
   1.00000 + 0.00000i 1.00000 - 0.00000i 1.00000 + 0.00000i
   1.00000 + 0.00000i 1.00000 + 0.00000i 1.00000 + 0.00000i
```

7.1 trigonometric functions

- `sin` , `asin` , `sind` , `asind` , `sinh` , `asinh` for sine functions
- `cos` , `acos` , `cosd` , `acosd` , `cosh` , `acosh` for cosine functions
- `tan` , `atan` , `tand` , `atand` , `tanh` , `atanh` , `atan2` , `atan2d` for tangent functions
- `csc` , `acsc` , `cscd` , `acscd` , `csch` , `acsch` for cosecant functions
- `sec` , `asec` , `secd` , `asecd` , `sech` , `asech` for secant functions
- `cot` , `acot` , `cotd` , `acotd` , `coth` , `acoth` for cotangent functions
- `hypot` , square root of the sum of the squares
- `deg2rad` , `rad2deg` for convert functions

7.2 Exponents and Logarithms

- `exp` , exponential function
- `expm1` , exponential function minus one
- `log` , natural logarithm
- `reallog` , real-valued natural logarithm
- `log1p` , compute `log(1+x)`
- `log10` , base-10 logarithm
- `log2` , base-2 logarithm
- `pow2` , base-2 power
- `nextpow2` , exponent of next higher power of 2
- `realpow` , real-valued power
- `sqrt` , square root
- `realsqrt` , real-valued square root
- `cbrt` , cube root
- `cbrtsqrt` , real-valued cube root
- `nthroot` , real (non-complex) n-th root

7.3 Complex Arithmetic

- `abs` , magnitude
- `arg` , `angle` , argument
- `conj` , complex conjugate
- `imag` , imaginary part
- `real` , real part

7.4 Utility methods

- `ceil` , round toward positive infinity
- `fix` , round toward zero
- `floor` , round toward negative infinity
- `round` , Round to the nearest integer

7.4.1 `max` method

Let `X` be a `N`-by-`m`-by-`n` `amat` object. The `max` method of `X` return its maximum values.

Syntaxe

```
W = max (X)
W = max (X, [], DIM)
W = max (X, Y)
[W, I] = max (X)
[W, I] = max (X, [], DIM)
[W, I, J] = max (X, [], 3)
```

Description

`W=max(X)`

return a `m`-by-`n` matrix such that `W(i,j)` is the maximum value of `X(:,i,j)`

`W = max (X, [], dim)`

- `dim=0` , along the number of matrices of `X`. Same as `W = max (X)`.
- `dim=1` , along rows of matrices of `X`. Returns a `N`-by-`1`-by-`n` `amat` object such that `W(k,1,j)` is the maximum value of `X(k,:,j)` .
- `dim=2` , along columns of matrices of `X`. Returns a `N`-by-`m`-by-`1` `amat` object such that `W(k,i,1)` is the maximum value of `X(k,i,:)` .
- `dim=3` , along rows and columns of matrices of `X`. Returns a `N`-by-`1`-by-`1` `amat` object such that `W(k,1,1)` is the maximum value of `X(k,:,:)` .

`W = max (X, Y)`

Returns a `N`-by-`m`-by-`n` `amat` object such that

- `W(k,i,j)=max(X(k,i,j),Y(k,i,j))` if `Y` is a `N`-by-`m`-by-`n` `amat` object,
- `W(k,i,j)=max(X(k,i,j),Y(i,j))` if `Y` is a `m`-by-`n` matrix,
- `W(k,i,j)=max(X(k,i,j),Y(k))` if `Y` is a `N`-by-`1` or `1`-by-`N` array,
- `W(k,i,j)=max(X(k,i,j),Y)` if `Y` is a scalar.

`[W, K] = max (X)`

Returns two `m`-by-`n` matrices such that

`W(i,j)=max(X(:,i,j))` and `W(i,j)=X(K(i,j),i,j)`

```
[W, Idx] = max (X, [], DIM)
```

- if `DIM=0` , command is equivalent to `[W, Idx] = max (X)`,
- if `DIM=1` , returns two `N`-by-`1`-by-`n amat` objects such that

  `W(k,1,j)=max(X(k,:,j))`and`W(k,1,j)=X(K,Idx(k,1,j),j)`,
- if `DIM=2` , returns two `N`-by-`m`-by-`1 amat` objects such that

  `W(k,i,1)=max(X(k,i,:))` and `W(k,i,1)=X(K,i,Idx(k,i,1))`.

```
[W, I, J] = max (X, [], 3)
```

returns three `N`-by-`1`-by-`1 amat` objects such that

`W(k,1,1)=max(X(k,:,:))` and `W(k,1,1)=X(K,I(k,1,1),J(k,1,1))`.

In Listing 45, some examples are provided.

Listing 45: : examples of `fc_amat.random.randher` function usage

```
N=3;m=2;n=3;
X=fc_amat.random.randi(9,[N,m,n]);
Y=fc_amat.random.randi(9,[N,m,n]);
disp(X)
W=max(X);
fprintf('W=max(X)␣->\n')
disp(W)
W1=max(X,[],1);
fprintf('W1=max(X,[],1)␣->␣\n')
disp(W1)
```

Output

```
X is a 3x2x3 amat[double] object
  X(1)=
   5   1   4
   1   1   8
  X(2)=
   2   1   6
   5   1   9
  X(3)=
   9   6   9
   7   9   9
W=max(X) ->
   9   6   9
   7   9   9
W1=max(X,[],1) ->
W1 is a 3x1x3 amat[double] object
  W1(1)=
   5
   1
   8
  W1(2)=
   5
   1
   9
  W1(3)=
   9
   9
   9
```

7.4.2 `min` method

Let `X` be a `N`-by-`m`-by-`n amat` object. The `min` method of `X` return its minimum values.

Syntaxe

```
W = min (X)
W = min (X, [], DIM)
W = min (X, Y)
[W, I] = min (X)
[W, I] = min (X, [], DIM)
[W, I, J] = min (X, [], 3)
```

Description

`W=min(X)`

return a m-by-n matrix such that `W(i,j)` is the minimum value of `X(:,i,j)`

`W = min (X, [], dim)`

- `dim=0` , along the number of matrices of `X`. Same as `W = min (X)`.
- `dim=1` , along rows of matrices of `X`. Returns a N-by-1-by-n `amat` object such that `W(k,1,j)` is the minimum value of `X(k,:,j)` .
- `dim=2` , along columns of matrices of `X`. Returns a N-by-m-by-1 `amat` object such that `W(k,i,1)` is the minimum value of `X(k,i,:)` .
- `dim=3` , along rows and columns of matrices of `X`. Returns a N-by-1-by-1 `amat` object such that `W(k,1,1)` is the minimum value of `X(k,:,:)` .

`W = min (X, Y)`

Returns a N-by-m-by-n `amat` object such that

- `W(k,i,j)=min(X(k,i,j),Y(k,i,j))` if `Y` is a N-by-m-by-n `amat` object,
- `W(k,i,j)=min(X(k,i,j),Y(i,j))` if `Y` is a m-by-n matrix,
- `W(k,i,j)=min(X(k,i,j),Y(k))` if `Y` is a N-by-1 or 1-by-N array,
- `W(k,i,j)=min(X(k,i,j),Y)` if `Y` is a scalar.

`[W, K] = min (X)`

Returns two m-by-n matrices such that

`W(i,j)=min(X(:,i,j))` and `W(i,j)=X(K(i,j),i,j)`

`[W, Idx] = min (X, [], DIM)`

- if `DIM=0` , command is equivalent to `[W, Idx] = min (X)`,

- if DIM=1 , returns two N-by-1-by-n amat objects such that

  W(k,1,j)=min(X(k,:,j))andW(k,1,j)=X(K,Idx(k,1,j),j),

- if DIM=2 , returns two N-by-m-by-1 amat objects such that

  W(k,i,1)=min(X(k,i,:)) and W(k,i,1)=X(K,i,Idx(k,i,1)).

```
[W, I, J] = min (X, [], 3)
```

returns three N-by-1-by-1 amat objects such that

W(k,1,1)=min(X(k,:,:)) and W(k,1,1)=X(K,I(k,1,1),J(k,1,1)).

In Listing 46, some examples are provided.

Listing 46: : examples of fc_amat.random.randher function usage

```
N=10;m=2;n=3;
X=fc_amat.random.randi(9,[N,m,n]);
disp(X)
W=min(X);
fprintf('W=min(X)␣->\n')
disp(W)
W1=min(X,[],1);
fprintf('W1=min(X,[],1)␣->␣\n')
disp(W1)
```

Output

```
X is a 10x2x3 amat[double] object
  X(1)=
   7   2   9
   3   9   1
  X(2)=
   3   7   8
   7   4   6
  ...

  X(9)=
   5   1   2
   9   6   7
  X(10)=
   1   6   6
   7   2   2
W=min(X) ->
   1   1   1
   2   1   1
W1=min(X,[],1) ->
W1 is a 10x1x3 amat[double] object
  W1(1)=
   3
   2
   1
  W1(2)=
   3
   4
   6
  ...

  W1(9)=
   5
   1
   2
  W1(10)=
   1
   2
   2
```

8 Linear algebra

8.1 Linear combination

. Let `X` be a `N`-by-`m`-by-`n` `amat` object, `alpha` and `beta` two scalars. We define four kinds of linear combinations for the Octave instruction:

```
Z= alpha*X + beta*Y
```
(5)

where `Z` be also a `N`-by-`m`-by-`n` `amat` object, and we have $\forall$`k` $\in$ `1:N`, $\forall$`i` $\in$ `1:m`, $\forall$`j` $\in$ `1:n`,

```
Z(k,i,j)= alpha*X(k,i,j)
```

$$+ \begin{cases} \texttt{beta*Y(k,i,j)} & \text{if } \texttt{Y} \text{ is a } \texttt{N}\text{-by-}\texttt{m}\text{-by-}\texttt{n } \texttt{amat} \text{ object} \\ \texttt{beta*Y(i,j)} & \text{if } \texttt{Y} \text{ is a } \texttt{m}\text{-by-}\texttt{n} \text{ matrix} \\ \texttt{beta*Y(i,j)} & \text{if } \texttt{Y} \text{ is a scalar} \\ \texttt{beta*Y(k)} & \text{if } \texttt{Y} \text{ is a } \texttt{N}\text{-by-}\texttt{1} \text{ array} \end{cases}$$

In Listing 47, some examples are provided.

Listing 47: : examples of linear combinations

```
N=100;m=2;n=3;
X=fc_amat.random.randi(9,[N,m,n]);
info(X)
Y=fc_amat.random.randi(9,[N,m,n]);
info(Y)
A=3*X-2*Y;
info(A)
Y2=randi(9,[m,n]);
B=2*Y2-4*X;
info(B)
C=3*X-1;
info(C)
Y3=randi(9,[N,1]);
D=3*Y3-X;
info(D)
```

Output

```
X is a 100x2x3 amat[double] object
Y is a 100x2x3 amat[double] object
A is a 100x2x3 amat[double] object
B is a 100x2x3 amat[double] object
C is a 100x2x3 amat[double] object
D is a 100x2x3 amat[double] object
```

8.2 Matrix product

We define (and extend) matricial products for `amat` objects by using operator `*` (i.e. `mtimes` method)

```
Z= X*Y
```
(6)

where `X` and/or `Y` are `amat` objects. Explanations on programming techniques can be found in [1].

We choose to only described this operator when the left operand `X` is a `N`-by-`m`-by-`n` `amat` object. We can easily deduced results when `X` is not an `amat` object and `Y` is an `amat` object.

- With `Y` a `N`-by-`n`-by-`p` `amat` object (compatible dimensions), instruction (6) defines `Z` as a `N`-by-`m`-by-`p` `amat` object and is equivalent to the `N` matricial products

$$\texttt{Z(k)= X(k)*Y(k)} \ , \quad \forall \texttt{k} \in \texttt{1:N}$$

  i.e. $\forall \texttt{i} \in \texttt{1:m}$, $\forall \texttt{j} \in \texttt{1:p}$,

$$\texttt{Z(k,i,j)=} \sum_{\texttt{r}=1}^{\texttt{n}} \texttt{X(k,i,r)*Y(k,r,j)}, \quad \forall \texttt{k} \in \texttt{1:N}.$$

- With `Y` a `n`-by-`p` matrix (compatible dimensions), instruction (6) defines `Z` as a `N`-by-`m`-by-`p` `amat` object and is equivalent to the `N` matricial products

$$\texttt{Z(k)= X(k)*Y} \ , \quad \forall \texttt{k} \in \texttt{1:N}$$

  i.e. $\forall \texttt{i} \in \texttt{1:m}$, $\forall \texttt{j} \in \texttt{1:p}$,

$$\texttt{Z(k,i,j)=} \sum_{\texttt{r}=1}^{\texttt{n}} \texttt{X(k,i,r)*Y(r,j)}, \quad \forall \texttt{k} \in \texttt{1:N}.$$

- With `Y` a `N`-by-`1` 1D-array, instruction (6) defines `Z` as a `N`-by-`m`-by-`n` `amat` object and we have

$$\texttt{Z(k)= X(k)*Y(k)} \ , \quad \forall \texttt{k} \in \texttt{1:N}$$

  i.e. $\forall \texttt{i} \in \texttt{1:m}$, $\forall \texttt{j} \in \texttt{1:n}$,

$$\texttt{Z(k,i,j)= X(k,i,j)*Y(k)}, \quad \forall \texttt{k} \in \texttt{1:N}.$$

- With `Y` a scalar, instruction (6) defines `Z` as a `N`-by-`m`-by-`n` `amat` object and we have

$$\texttt{Z(k)= X(k)*Y} \ , \quad \forall \texttt{k} \in \texttt{1:N}$$

  i.e. $\forall \texttt{i} \in \texttt{1:m}$, $\forall \texttt{j} \in \texttt{1:n}$,

$$\texttt{Z(k,i,j)= X(k,i,j)*Y} \ , \quad \forall \texttt{k} \in \texttt{1:N}.$$

In Listing 47, some examples are provided.

Listing 48: : examples of matricial products

```
N=100;m=2;n=4;p=3;
X=fc_amat.random.randi(9,[N,m,n]);
info(X)
Y=fc_amat.random.randi(9,[N,n,p]);
info(Y)
A=X*Y; % <- matricial products
info(A)
X2=randi(9,[m,n]);
B=X2*Y;% <- matricial products
info(B)
Y2=randi(9,[n,p]);
C=X*Y2;% <- matricial products
info(C)
T=C(1)-X(1)*Y2;
fprintf('T is\n')
disp(T)
```

Output

```
X is a 100x2x4 amat[double] object
Y is a 100x4x3 amat[double] object
A is a 100x2x3 amat[double] object
B is a 100x2x3 amat[double] object
C is a 100x2x3 amat[double] object
T is
   0   0   0
   0   0   0
```

8.2.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.mtimes` can be used and is described in Section 8.2.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `X` and `Y` be `N`-by-`d`-by-`d` `amat` objects, in Table 2 computational times in seconds of `mtimes(X,Y)` (`X*Y` matricial products) are given. In Figure 1, computational times in seconds for a given `N` are represented in function of very small values of `d` .

N	mtimes
200 000	0.231(s)
400 000	1.532(s)
600 000	2.295(s)
800 000	3.059(s)
1 000 000	3.835(s)
5 000 000	20.355(s)
10 000 000	40.554(s)

Table 2: Computational times in seconds of `mtimes(X,Y)` (`X*Y` matrix product) where `X` and `Y` are `N`-by-`d`-by-`d` `amat` objects.

8.2.2 Benchmark function

The function `fc_amat.benchs.mtimes` measures performance of matricial products of `amat` objects done by `mtimes(X,Y)` or `X*Y` command. At least one of the inputs must be an `amat` object. When running this function the matrices

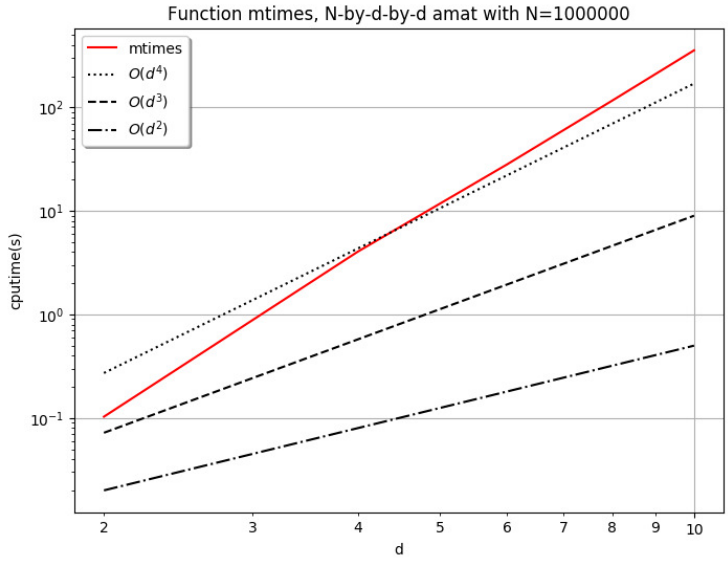

Figure 1: Computational times in seconds of `mtimes(X,Y)` or `X*Y` (matrix product) where `X` and `Y` are `N`-by-`d`-by-`d` `amat` objects.

orders are fixed and only the number `N` of matrices contained in `amat` objects varies and it is given by a list of values `LN` .

Syntaxe

```
fc_amat.benchs.mtimes(LN)
fc_amat.benchs.mtimes(LN,key,value,...)
```

Description

`fc_amat.benchs.mtimes(LN)`

runs a benchmark of the `mtimes` method of the `amat` class between two `N`-by-2-by-2 `amat` objects for all `N` in `LN` .

`fc_amat.benchs.mtimes(LN,key,value,...)`

Optional key/value pairs arguments are available. `key` can be one of the following strings

- `'d'` , left and right matrices dimension (default `value` is `[2,2]`)
- `'type'` , to set type of left and right operands. `value` is either `'amat'` (`amat` object), `'mat'` (matrix), `'array1d'` (`N`-by-`1` 1D-array) or `'scalar'` (default `value` is `'amat'`).
- `'class'` , to set classname of left and right operands. Value can be `'double'` (default), `'single'` , `'int32'` ,...

- `'complex'` , if `true` left and right operands are complex (default `value` is `false`).
- `'ld'` , same as `'d'` but only for left operand.
- `'rd'` , same as `'d'` but only for right operand.
- `'ltype'` same as `'type'` but only for left operand.
- `'rtype'` same as `'type'` but only for right operand.
- `'lclass'` same as `'class'` but only for left operand.
- `'rclass'` same as `'class'` but only for right operand.
- `'lcomplex'` same as `'complex'` but only for left operand.
- `'rcomplex'` same as `'complex'` but only for right operand.

In Listings 49 and 50 two examples with outputs are provided.

Listing 49: : Benchmarking `mtimes(X,Y)` with `X` a 3-by-4 matrix and `Y` a N-by-4-by-5 `amat` object

```
LN=10^5*[2:2:10];
fc_amat.benchs.mtimes(LN,'ltype','mat','ld',[3,4],'rd',[4,5]);
```

Output

```
#-------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#-------------------------------------------------------------------------------
# 1st parameter is :
# -> matrix[double] with (m,n)=(3,4), size=[3 4]
# 2nd parameter is :
# -> amat[double] with (N,nr,nc)=(200000,4,5), size=[200000 4      5]
#-------------------------------------------------------------------------------
#date:2020/02/16 12:27:51
#nbruns:5
#numpy:       i4         f4
#format:      %d       %.3f
#labels:       N   mtimes(s)
          200000      0.391
          400000      1.364
          600000      2.068
          800000      2.735
         1000000      3.440
```

Listing 50: : Benchmarking `mtimes(X,Y)` where `X` and `Y` are `N`-by-4-by-4 `amat` object with `complex single` values.

```
LN=10^5*[2:2:10];
fc_amat.benchs.mtimes(LN,'d',[4,4],'complex',true,'class','single', ...
    'info',false);
```

Output

```
#-------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[complex single] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
# 2nd parameter is :
# -> amat[complex single] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
#-------------------------------------------------------------------------------
#date:2020/02/16 12:29:02
#nbruns:5
#numpy:       i4         f4
#format:      %d       %.3f
#labels:       N   mtimes(s)
          200000      0.518
          400000      1.847
          600000      2.749
          800000      3.657
         1000000      4.552
```

8.3 LU Factorization

Let `A` be a `N`-by-`m`-by-`m` `amat` object. The `[L,U,P]=lu(A)` command returns three `N`-by-`m`-by-`m` `amat` objects where `L` , `U` and `P` are respectively a unit lower triangular `amat` , an upper triangular `amat` and a permutation `amat` such that

$$\texttt{P*A=L*U} \text{ or } \texttt{A=P'*L*U} . \tag{7}$$

Here, operator `*` is the `amat` matricial product, i.e.

$$\forall \texttt{k} \in \texttt{1:N}, \quad P(k) * A(k) = L(k) * U(k).$$

Explanations on programming techniques can be found in [1].

Syntaxe Let `A` be a `N`-by-`m`-by-`m` `amat` object.

```
[L,U,P]=lu(A)
[L,U,P]=lu(A,type)
```

Description

`[L,U,P]=lu(A)`

returns three `N`-by-`m`-by-`m` `amat` objects where `L` , `U` and `P` are respectively a unit lower triangular `amat` , an upper triangular `amat` and a permutation `amat` such that

$$\texttt{P*A=L*U} \text{ or } \texttt{A=P'*L*U} . \tag{8}$$

Here operator `*` is the `amat` matricial product, i.e.

$$\forall \texttt{k} \in \texttt{1:N}, \quad \texttt{P(k)*A(k)=L(k)*U(k)}.$$

`[L,U,P]=lu(A,type)`

- If `type` is `'amat'` , then the command is equivalent to `[L,U,P]=lu(A)` .
- If `type` is `'vector'` or `'matrix'` then, returns the permutation information `P` as a `N`-by-`m` matrix instead of an `amat` . If so, the permutation `amat` object can be build with the `fc_amat.permind2amat(P)` command.

In Listing 51, some examples are provided.

Listing 51: : examples of `lu` method usage

```
A=complex(fc_amat.random.randn(100,3,3),fc_amat.random.randn(100,3,3));
info(A)
[L,U,P]=lu(A);
info(L);info(U);info(P);
E=P*A-L*U;
disp(E);
```

Output

```
A is a 100x3x3 amat[complex double] object
L is a 100x3x3 amat[complex double] object
U is a 100x3x3 amat[complex double] object
P is a 100x3x3 amat[double] object
E is a 100x3x3 amat[complex double] object
  E(1)=
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
  E(2)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 2.7756e-17i -2.0817e-17 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 1.1102e-16i
   0.0000e+00 + 6.2450e-17i
  ...

  E(99)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i -1.1102e-16 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 0.0000e+00 - 6.9389e-18i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   1.1102e-16 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i
  E(100)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
  -5.5511e-17 + 2.2204e-16i 2.2204e-16 + 0.0000e+00i
   0.0000e+00 - 1.1102e-16i 0.0000e+00 + 0.0000e+00i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i
   5.5511e-17 + 0.0000e+00i
```

8.3.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.lu` can be used and is described in Section 8.3.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `A` be a `N`-by-`d`-by-`d` `amat` object, in Table 3 computational times in seconds of `[L,U,P]=lu(A)` are given.In Figure 2, computational times in seconds for a given `N` are represented in function of very small values of `d`.

N	d=2	d=4	d=6	d=8	d=10
200 000	0.041(s)	0.239(s)	2.060(s)	6.135(s)	14.542(s)
400 000	0.090(s)	1.000(s)	4.176(s)	12.548(s)	29.736(s)
600 000	0.131(s)	1.496(s)	6.395(s)	19.187(s)	45.172(s)
800 000	0.177(s)	2.077(s)	8.674(s)	25.679(s)	60.499(s)
1 000 000	0.222(s)	2.611(s)	10.959(s)	32.214(s)	75.789(s)

Table 3: Computational times in seconds of `[L,U,P]=lu(A)` where `A` is a `N`-by-`d`-by-`d` `amat` object.

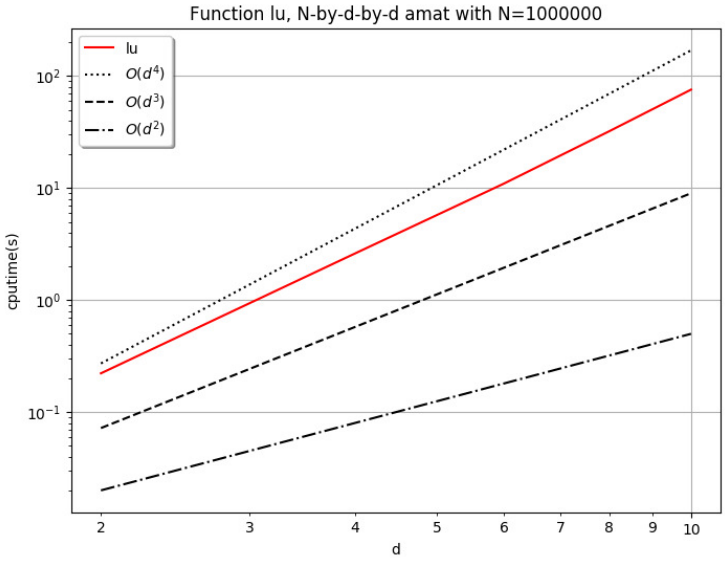

Figure 2: Computational times in seconds of of `[L,U,P]=lu(A)` where `A` is a `N`-by-`d`-by-`d` `amat` object.

8.3.2 Benchmark function

The function `fc_amat.benchs.lu` measures performance of LU factorization `[L,U,P]=lu(A)` where the input `A` is a `N`-by-`d`-by-`d` `amat` object. When running this function the `d` value is fixed, the number `N` varies and it is given by a list of values `LN`.

Syntaxe

```
fc_amat.benchs.lu(LN)
fc_amat.benchs.lu(LN,key,value,...)
```

Description

`fc_amat.benchs.lu(LN)`

runs a benchmark of the `lu` method on a `N`-by-`2`-by-`2` `amat` object for all `N` in `LN`.

`fc_amat.benchs.lu(LN,key,value,...)`

Optional key/value pairs arguments are available and can modify the input `N`-by-`d`-by-`d` `amat` object of the `lu` function. `key` can be one of the following strings

- `'d'`, to set `d` (default `value` is `2`)
- `'class'`, to set classname of the input `amat` object. Value can be `'double'` (default) or `'single'`.
- `'complex'`, if `true` the input `amat` object is complex (default `value` is `false`).

In Listings 52 and 53 two examples with outputs are provided.

Listing 52: : Benchmarking `[L,U,P]=lu(A)` with `A` a `N`-by-`4`-by-`4` matrix `amat` object

```
LN=10^5*[2:2:10];
fc_amat.benchs.lu(LN,'d',4);
```

Output

```
#------------------------------------------------------------------------------
#   computer: cosmos-ubuntu-18-04
#     system: Ubuntu 18.04.4 LTS (x86_64)
#  processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#            (1 procs/14 cores by proc/2 threads by core)
#        RAM: 62.6 Go
#   software: Octave
#    release: 5.2.0
#------------------------------------------------------------------------------
# input parameter is :
# -> amat[double] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
#------------------------------------------------------------------------------
#date:2020/02/16 15:07:33
#nbruns:5
#numpy:       i4     f4          f4
#format:      %d    %.3f        %.3e
#labels:       N   lu(s)    Error[0]
          200000  0.270   1.443e-15
          400000  0.910   1.638e-15
          600000  1.519   1.554e-15
          800000  2.039   1.610e-15
         1000000  2.529   1.554e-15
```

Listing 53: : Benchmarking `[L,U,P]=lu(A)` where `A` is N-by-3-by-3 `amat` object with `complex single` values.

```
LN=10^5*[2:2:10];
fc_amat.benchs.lu(LN,'d',3,'complex',true,'class','single', ...
    'info',false);
```

Output

```
#-------------------------------------------------------------------------------
# input parameter is :
# -> amat[complex single] with (N,nr,nc)=(200000,3,3), size=[200000 3     3]
#-------------------------------------------------------------------------------
#date:2020/02/16 15:08:46
#nbruns:5
#numpy:       i4     f4          f4
#format:      %d   %.3f        %.3e
#labels:       N   lu(s)    Error[0]
          200000  0.170   1.006e-06
          400000  0.405   1.252e-06
          600000  0.749   1.100e-06
          800000  1.014   1.098e-06
         1000000  1.218   1.117e-06
```

8.4 Cholesky Factorization

The `chol(A)` command returns the positive Cholesky factorization of symmetric (or hermitian) positive definite `amat` object `A` as a upper triangular `amat` object with strictly positive diagonal entries. Explanations on programming techniques can be found in [1].

Syntaxe Let `A` be a `N`-by-`d`-by-`d` symmetric (or hermitian) positive definite `amat` object.

```
B=chol(A)
B=chol(A,type)
```

Description

`B=chol(A)`

returns the positive Cholesky factorization of `A` as a `N`-by-`d`-by-`d` upper triangular `amat` object `B` with strictly positive diagonal entries such that

$$\texttt{A=B'*B} \tag{9}$$

Here, operator `*` is the `amat` matricial product, i.e.

$$\forall \texttt{k} \in \texttt{1:N}, \quad \texttt{A(k)=B(k)'*B(k)} \ .$$

`B=chol(A,type)`

- If `type` is `'upper'`, then the command is equivalent to `B=chol(A)`.

- If `type` is `'lower'`, then `B` is a `N`-by-`d`-by-`d` lower triangular `amat` object with strictly positive diagonal entries such that

$$\texttt{A=B*B'} \tag{10}$$

Here, operator `*` is the `amat` matricial product, i.e.

$$\forall \texttt{k} \in \texttt{1:N}, \quad \texttt{A(k)=B(k)*B(k)'}\ .$$

In Listing 54, some examples are provided.

Listing 54: : examples of `chol` method usage

```
A=fc_amat.random.randnherpd(100,3);
info(A)
B=chol(A);
info(B);
E=A-B'*B;
disp(E);
```

Output

```
A is a 100x3x3 amat[complex double] object
B is a 100x3x3 amat[complex double] object
E is a 100x3x3 amat[complex double] object
  E(1)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i -1.7764e-15 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 0.0000e+00 - 4.4409e-16i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 4.4409e-16i
   1.7764e-15 + 0.0000e+00i
  E(2)=
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 - 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   ...

  E(99)=
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
   0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
  E(100)=
 Columns 1 and 2:

  -3.5527e-15 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 1.7764e-15 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 4.4409e-16 - 2.2204e-16i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   4.4409e-16 + 2.2204e-16i
   1.7764e-15 + 0.0000e+00i
```

8.4.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.chol` can be used and is described in Section 8.4.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `A` be a `N`-by-`d`-by-`d` symmetric (or hermitian) positive definite `amat` object, in Table 4 computational times in seconds of `B=chol(A)` are given.In Figure 3, computational times in seconds for a given `N` are represented in fonction of very small values of `d`.

N	d=2	d=4	d=6	d=8	d=10
200 000	0.004(s)	0.014(s)	0.048(s)	0.091(s)	0.156(s)
400 000	0.008(s)	0.040(s)	0.095(s)	0.184(s)	0.313(s)
600 000	0.012(s)	0.061(s)	0.152(s)	0.286(s)	0.491(s)
800 000	0.016(s)	0.084(s)	0.202(s)	0.397(s)	0.679(s)
1 000 000	0.020(s)	0.109(s)	0.264(s)	0.517(s)	0.887(s)

Table 4: Computational times in seconds of `B=chol(A)` where `A` is a `N`-by-`d`-by-`d` symmetric positive definite `amat` object.

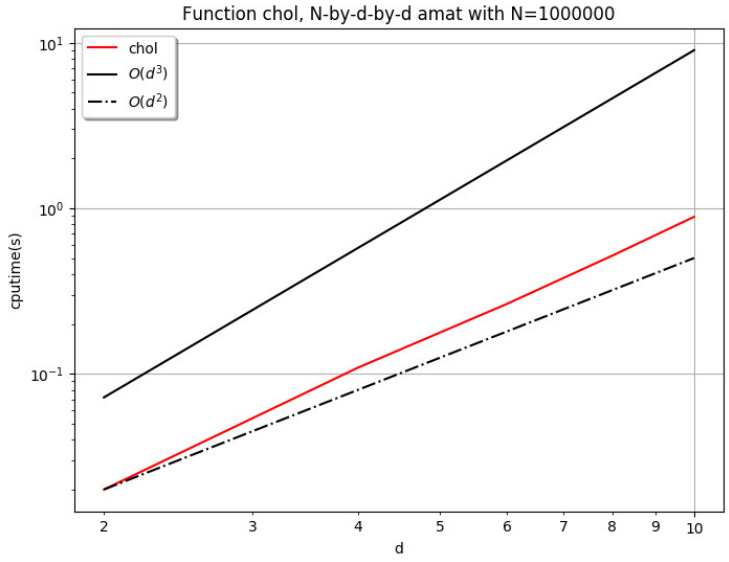

Figure 3: Computational times in seconds of of `B=chol(A)` where `A` is a `N`-by-`d`-by-`d` symmetric positive definite `amat` object.

8.4.2 Benchmark function

The function `fc_amat.benchs.chol` measures performance of Cholesky factorization `B=chol(A)` where the input `A` is a `N`-by-`d`-by-`d` symmetric (or hermitian) positive definite `amat` object. When running this function the `d` value is fixed, the number `N` varies and it is given by a list of values `LN`.

Syntaxe

```
fc_amat.benchs.chol(LN)
fc_amat.benchs.chol(LN,key,value,...)
```

Description

`fc_amat.benchs.chol(LN)`

runs a benchmark of the `chol` method on a `N`-by-`2`-by-`2` symmetric positive definite `amat` object for all `N` in `LN`.

`fc_amat.benchs.chol(LN,key,value,...)`

Optional key/value pairs arguments are available and can modify the input `N`-by-`d`-by-`d` `amat` object of the `chol` function. `key` can be one of the following strings

- `'d'`, to set `d` (default `value` is `2`)
- `'kind'`, to set the kind of the square output `amat` object. If `value` is `'lower'`, then the output is a lower triangular `amat` object with strictly positive diagonal entries. Default `value` is `'upper'`. `d` (default `value` is `2`)
- `'class'`, to set classname of the input `amat` object. Value can be `'double'` (default) or `'single'`.
- `'complex'`, if `true` the input `amat` object is Hermitian positive definite (default `value` is `false`).

In Listings 55 and 56 two examples with outputs are provided.

Listing 55: : Benchmarking `B=chol(A)` with `A` a `N`-by-`4`-by-`4` matrix `amat` object

```
LN=10^5*[2:2:10];
fc_amat.benchs.chol(LN,'d',4,'kind','lower');
```

Output

```
#-------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#-------------------------------------------------------------------------------
# Symmetric Positive Definite matrices
# -> amat[double] with (N,m,n)=(N,4,4)
# Error function: @(X)max(norm(X*X'-A))+all(!istril(X),0)
#-------------------------------------------------------------------------------
# Benchmarking command: @(A) chol(A,'lower');
#-------------------------------------------------------------------------------
#date:2020/02/16 16:37:13
#nbruns:5
#numpy:       i4       f4          f4
#format:      %d     %.3f        %.3e
#labels:       N   chol(s)    Error[0]
          200000    0.014   2.132e-14
          400000    0.040   3.020e-14
          600000    0.061   2.842e-14
          800000    0.083   2.842e-14
         1000000    0.106   3.197e-14
```

Listing 56: : Benchmarking `B=chol(A)` where `A` is `N`-by-`3`-by-`3` `amat` object with `complex single` vacholes.

```
LN=10^5*[2:2:10];
fc_amat.benchs.chol(LN,'d',3,'complex',true,'class','single', ...
    'info',false);
```

Output

```
#-------------------------------------------------------------------------------
# Hermitian Positive Definite matrices
# -> amat[complex single] with (N,m,n)=(N,3,3)
# Error function: @(X)max(norm(X'*X-A))+all(!istriu(X),0)
#-------------------------------------------------------------------------------
# Benchmarking command: @(A) chol(A,'upper');
#-------------------------------------------------------------------------------
#date:2020/02/16 16:37:42
#nbruns:5
#numpy:       i4       f4          f4
#format:      %d     %.3f        %.3e
#labels:       N   chol(s)    Error[0]
          200000    0.037   1.144e-05
          400000    0.077   1.144e-05
          600000    0.128   1.240e-05
          800000    0.187   1.717e-05
         1000000    0.235   1.577e-05
```

8.5 Determinants

The `det(A)` command returns determinants of the matrices of the square `amat` object. Explanations on programming techniques can be found in [1].

Syntaxe Let `A` be a `N`-by-`d`-by-`d` `amat` object.

```
D=det(A)
D=det(A,'select',value)
```

Description

`D=det(A)`

returns determinants of the matrices of `A` as a `N`-by-`1`-by-`1` `amat` object `D` such that

$$\forall k \in 1{:}N, \quad D(k)=\det(A(k)) \ .$$

`D=det(A,'select',value)`

when `value` is

- `'lu'` , uses LU factorizations.
- `'laplace'` , uses vectorized Laplace expansion.
- `'auto'` (default), uses vectorized Laplace expansion for `d<=5` and LU factorization otherwise.

In Listing 57, some examples are provided.

Listing 57: : examples of `det` method usage

```
A=complex(fc_amat.random.randn(100,3),fc_amat.random.randn(100,3));
info(A)
D1=det(A);
info(D1);
D2=det(A,'select','lu');
info(D2);
D3=det(A,'select','laplace');
info(D3);
E=abs(D1-D2)+abs(D1-D3);
disp(E)
```

Output

```
A is a 100x3x3 amat[complex double] object
D1 is a 100x1x1 amat[complex double] object
D2 is a 100x1x1 amat[complex double] object
D3 is a 100x1x1 amat[complex double] object
E is a 100x1x1 amat[double] object
  E(1)=
   2.2888e-16
  E(2)=
   1.3323e-15
  ...

  E(99)=
   1.8310e-15
  E(100)=
   4.4409e-16
```

8.5.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.det` can be used and is described in Section 8.5.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `A` be a `N`-by-`d`-by-`d` `amat` object, in Table 5 computational times in seconds of `B=det(A)` are given.In Figure 4, computational times in seconds for a given `N` are represented in function of very small values of `d`.

N	d=2	d=4	d=6	d=8	d=10
200 000	0.046(s)	0.232(s)	2.060(s)	6.156(s)	14.449(s)
400 000	0.098(s)	0.735(s)	4.093(s)	12.238(s)	28.903(s)
600 000	0.145(s)	1.439(s)	6.152(s)	18.416(s)	43.476(s)
800 000	0.208(s)	1.910(s)	8.611(s)	24.807(s)	58.924(s)
1 000 000	0.233(s)	2.499(s)	10.688(s)	31.698(s)	74.272(s)

Table 5: Computational times in seconds of `B=det(A)` where `A` is a `N`-by-`d`-by-`d` `amat` object.

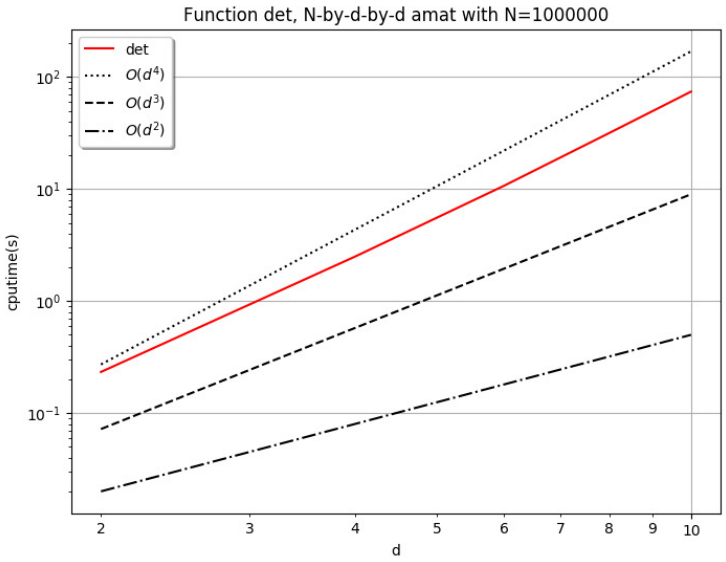

Figure 4: Computational times in seconds of of `B=det(A)` where `A` is a `N`-by-`d`-by-`d` `amat` object.

8.5.2 Benchmark function

The function `fc_amat.benchs.det` measures performance of `B=det(A)` where the input `A` is a `N`-by-`d`-by-`d` `amat` object. When running this function the `d` value is fixed, the number `N` varies and it is given by a list of values `LN`.

Syntaxe

```
fc_amat.benchs.det(LN)
fc_amat.benchs.det(LN,key,value,...)
```

Description

`fc_amat.benchs.det(LN)`

runs a benchmark of the `det` method on a `N`-by-2-by-2 `amat` object for all `N` in `LN`.

`fc_amat.benchs.det(LN,key,value,...)`

Optional key/value pairs arguments are available and can modify the input `N`-by-`d`-by-`d` `amat` object of the `det` function. `key` can be one of the following strings

- `'d'`, to set `d` (default `value` is `2`)
- `'select'`, to set the `'select'` option of the `'det'` function: `value` can be `'lu'` (default), `'laplace'` or `'auto'`.
- `'class'`, to set classname of the input `amat` object. Value can be `'double'` (default) or `'single'`.
- `'complex'`, if `true` the input `amat` object is complex (default `value` is `false`).

In Listings 58 and 59 two examples with outputs are provided.

Listing 58: : Benchmarking `D=det(A)` with `A` a `N`-by-4-by-4 matrix `amat` object

```
LN=10^5*[2:2:10];
fc_amat.benchs.det(LN,'d',4,'select','lu');
```

Output

```
#------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#------------------------------------------------------------------------------
# input parameter for N=200000 is :
# -> amat[double] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
#------------------------------------------------------------------------------
# Benchmarking command: @(A) det(A,'select','lu');
#------------------------------------------------------------------------------
#date:2020/02/16 17:51:52
#nbruns:5
#numpy:       i4      f4
#format:      %d    %.3f
#labels:       N   det(s)
          200000   0.248
          400000   0.925
          600000   1.477
          800000   1.993
         1000000   2.453
```

Listing 59: : Benchmarking `B=det(A)` where `A` is `N`-by-3-by-3 `amat` object with `complex single` vadetes.

```
LN=10^5*[2:2:10];
fc_amat.benchs.det(LN,'d',3,'complex',true,'class','single', ...
    'info',false);
```

Output

```
#-------------------------------------------------------------------------------
# input parameter for N=200000 is :
# -> amat[complex single] with (N,nr,nc)=(200000,3,3), size=[200000 3     3]
#-------------------------------------------------------------------------------
# Benchmarking command: @(A) det(A,'select','lu');
#-------------------------------------------------------------------------------
#date:2020/02/16 17:52:43
#nbruns:5
#numpy:      i4      f4
#format:     %d    %.3f
#labels:      N   det(s)
          200000   0.168
          400000   0.390
          600000   0.690
          800000   1.020
         1000000   1.270
```

8.6 Solving particular linear systems

There are three functions to solve linear systems `A*X=B` where `A` is a particular (regular) `amat` object.

- `X=solvetriu(A,B)` , if `A` is an upper triangular `amat` object.
- `X=solvetril(A,B)` , if `A` is a lower triangular `amat` object.
- `X=solvediag(A,B)` , if `A` is a diagonal `amat` object.

Explanations on programming techniques can be found in [1]. We only describe the `solvetriu` function because the two others are used similarly.

The `X=solvetriu(A,B)` command returns solutions of the linear systems `A*X=B` where `A` is a regular upper triangular `amat` object. If `A` is not upper triangular, then `X` is solution of `triu(A)*X=B` .

Description

`X=solvetriu(A,B)`

The input `A` supposes to be a `N`-by-`d`-by-`d` regular upper triangular `amat` object and `B` is either a `N`-by-`d`-by-`n` `amat` object or a `d`-by-`n` matrix. Then, the ouput `X` is the `N`-by-`d`-by-`n` `amat` object such that

$$\forall \texttt{k} \in \texttt{1:N}, \quad \texttt{A(k)*X(k)=} \begin{cases} \texttt{B(k)} & \text{if } \texttt{B} \text{ is an } \texttt{amat} \text{ object} \\ \texttt{B} & \text{if } \texttt{B} \text{ is a matrix} \end{cases} .$$

In Listing 60, some examples are provided.

Listing 60: : examples of `solvetriu` method usage

```
N=100; d=3;
A=fc_amat.random.randtriu(N,d);
info(A)
B1=fc_amat.random.randn(N,d,4);
info(B1)
X1=solvetriu(A,B1);
info(X1)
fprintf('Max. of errors in Inf. norm: %.4e\n',max(norm(A*X1-B1)))
B2=randn(d,1);
X2=solvetriu(A,B2);
info(X2)
E2=A*X2-B2;
disp(E2)
```

Output

```
A is a 100x3x3 amat[double] object
B1 is a 100x3x4 amat[double] object
X1 is a 100x3x4 amat[double] object
Max. of errors in Inf. norm: 1.0730e-13
X2 is a 100x3x1 amat[double] object
E2 is a 100x3x1 amat[double] object
  E2(1)=
   0.0000e+00
   1.6653e-16
   0.0000e+00
  E2(2)=
   0
   0
   0
  ...

  E2(99)=
  -6.6613e-16
  -1.6653e-16
   0.0000e+00
  E2(100)=
  -2.2204e-16
  -1.6653e-16
   0.0000e+00
```

8.6.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.solvetriu` can be used and is described in Section 8.6.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `A` be a `N`-by-`d`-by-`d` regular triangular upper `amat` object and `B` be a `N`-by-`d`-by-`1` `amat` object. In Table 6 computational times in seconds of `X=solvetriu(A,B)` are given. In Figure 5, computational times in seconds for a given `N` are represented in function of very small values of `d`.

`N`	`solvetriu`	Error
200 000	0.009(s)	$6.9670e-15$
400 000	0.019(s)	$6.4390e-15$
600 000	0.029(s)	$9.2430e-15$
800 000	0.046(s)	$8.6040e-15$
1 000 000	0.060(s)	$8.2160e-15$
5 000 000	0.648(s)	$1.6210e-14$
10 000 000	1.288(s)	$1.3540e-14$

Table 6: Computational times in seconds of `X=solvetriu(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object with `d=4`.

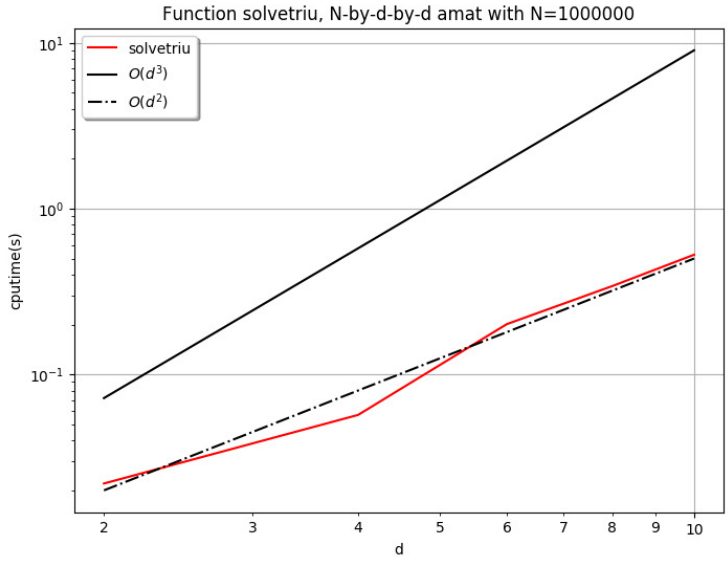

Figure 5: Computational times in seconds of of `X=solvetriu(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object.

8.6.2 Benchmark function

The function `fc_amat.benchs.solvetriu` measures performance of `X=solvetriu(A,B)` where the input `A` is a `N`-by-`d`-by-`d` regular triangular upper `amat` object and `B` is either a `N`-by-`d`-by-`n` `amat` object or a `d`-by-`n` matrix. When running this function the `d` and `n` value are fixed, the number `N` varies and it is given by a list of values `LN`.

Syntaxe

```
fc_amat.benchs.solvetriu(LN)
fc_amat.benchs.solvetriu(LN,key,value,...)
```

Description

`fc_amat.benchs.solvetriu(LN)`

runs a benchmark of the `X=solvetriu(A,B)` command where `A` is a `N`-by-`2`-by-`2` regular triangular upper `amat` object and `B` is a `N`-by-`2`-by-`1` `amat` object for all `N` in `LN`. So, by default `d=2` and `n=1`.

`fc_amat.benchs.solvetriu(LN,key,value,...)`

Optional key/value pairs arguments are available and can modify inputs of the `solvetriu` function. `key` can be one of the following strings

- `'d'`, to set `d` (default `value` is `2`)

- `'n'` , to set `n` (default `value` is `1`)
- `'rhstype'` , to set the kind of `B` : `'amat'` (default) for `amat` object and `'mat'` for matrix
- `'class'` , to set classname of the two inputs. Value can be `'double'` (default) or `'single'` .
- `'complex'` , if `true` the inputs are complex (default `value` is `false`).
- `'a'` , to set the lower bound of the interval of the uniform distribution used to generate input data (default `value` is `0.5`).
- `'b'` , to set `b` the upper bound of the interval of the uniform distribution used to generate input data (default `value` is `2`).

In Listings 61 and 62 two examples with outputs are provided.

Listing 61: : Benchmarking `X=solvetriu(A,B))` with `A` a `N`-by-4-by-4 matrix `amat` object and `B` a `N`-by-4-by-5 matrix `amat` object.

```
LN=10^5*[2:2:10];
fc_amat.benchs.solvetriu(LN,'d',4,'n',5, 'rhstype','mat');
```

Output

```
#-------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#-------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[double] with (N,m,n)=(N,4,4)
#    containing upper triangular matrices
# 2nd parameter is :
# -> matrix[double] with (m,n)=(4,5), size=[4 5]
# Error function: @(X)max(norm(A*X-B))
#-------------------------------------------------------------------------------
# Benchmarking command: @(A,B) solvetriu(A,B);
#-------------------------------------------------------------------------------
#date:2020/02/16 17:55:45
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d         %.3f        %.3e
#labels:       N   solvetriu(s)    Error[0]
          200000         0.109    6.120e-15
          400000         0.541    7.695e-15
          600000         0.966    7.438e-15
          800000         1.293    1.540e-14
         1000000         1.611    1.160e-14
```

Listing 62: : Benchmarking `X=solvetriu(A,B)` where `A` is `N`-by-`3`-by-`3` `amat` object and `B` is `N`-by-`3`-by-`1` `amat` object with both `complex single` values.

```
LN=10^5*[2:2:10];
fc_amat.benchs.solvetriu(LN,'d',3,'complex',true,'class','single', ...
    'info',false);
```

Output

```
#-------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[complex single] with (N,m,n)=(N,3,3)
#    containing upper triangular matrices
# 2nd parameter is :
# -> amat[complex single] with (N,nr,nc)=(200000,3,1), size=[200000 3      1]
# Error function: @(X)max(norm(A*X-B))
#-------------------------------------------------------------------------------
# Benchmarking command: @(A,B) solvetriu(A,B);
#-------------------------------------------------------------------------------
#date:2020/02/16 17:56:35
#nbruns:5
#numpy:       i4            f4          f4
#format:      %d          %.3f        %.3e
#labels:       N   solvetriu(s)    Error[0]
          200000         0.023    1.788e-06
          400000         0.046    2.883e-06
          600000         0.076    2.546e-06
          800000         0.116    3.938e-06
         1000000         0.152    3.278e-06
```

8.7 Solving linear systems

The `X=mldivide(A,B)` or `X=A\B` commands return solutions of the linear systems `A*X=B` where `A` is a regular `amat` object. Explanations on programming techniques can be found in [1].

Description

`X=mldivide(A,B)` or `X=A\B`

The input `A` supposes to be a `N`-by-`d`-by-`d` regular `amat` object and `B` is either a `N`-by-`d`-by-`n` `amat` object or a `d`-by-`n` matrix. Then, the ouput `X` is the `N`-by-`d`-by-`n` `amat` object such that

$$\forall \texttt{k} \in \texttt{1:N}, \quad \texttt{A(k)*X(k)=} \begin{cases} \texttt{B(k)} & \text{if } \texttt{B} \text{ is an } \texttt{amat} \text{ object} \\ \texttt{B} & \text{if } \texttt{B} \text{ is a matrix} \end{cases} .$$

In Listing 63, some examples are provided.

Listing 63: : examples of `mldivide` method operator usage

```
N=100; d=3;
A=fc_amat.random.randnsdd(N,d);
info(A)
B1=fc_amat.random.randn(N,d,4);
info(B1)
X1=mldivide(A,B1); % using function
info(X1)
fprintf('Max. of errors in Inf. norm: %.4e\n',max(norm(A*X1-B1)))
B2=randn(d,1);
X2=A\B2; % using operator
info(X2)
E2=A*X2-B2;
disp(E2)
```

Output

```
A is a 100x3x3 amat[double] object
B1 is a 100x3x4 amat[double] object
X1 is a 100x3x4 amat[double] object
Max. of errors in Inf. norm: 3.2196e-15
X2 is a 100x3x1 amat[double] object
E2 is a 100x3x1 amat[double] object
  E2(1)=
   1.1102e-16
  -1.3878e-17
   1.1102e-16
  E2(2)=
  -2.2204e-16
  -2.6368e-16
   3.3307e-16
  ...

  E2(99)=
   1.1102e-16
   9.7145e-17
  -1.1102e-16
  E2(100)=
   0.0000e+00
   6.9389e-17
   1.1102e-16
```

8.7.1 Efficiency

For benchmarking purpose the function `fc_amat.benchs.mldivide` can be used and is described in Section 8.7.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let `A` be a `N`-by-`d`-by-`d` regular triangular upper `amat` object and `B` be a `N`-by-`d`-by-`1` `amat` object. In Table 7 computational times in seconds of `X=mldivide(A,B)` are given.In Figure 6, computational times in seconds for a given `N` are represented in function of very small values of `d`.

N	d=2	d=4	d=6	d=8	d=10
200 000	0.069(s)	0.297(s)	2.220(s)	6.471(s)	15.181(s)
400 000	0.114(s)	1.049(s)	4.415(s)	12.985(s)	30.606(s)
600 000	0.183(s)	1.619(s)	6.705(s)	20.584(s)	47.922(s)
800 000	0.229(s)	2.252(s)	9.758(s)	27.707(s)	64.999(s)
1 000 000	0.321(s)	2.804(s)	12.194(s)	35.348(s)	81.976(s)

Table 7: Computational times in seconds of `X=mldivide(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object.

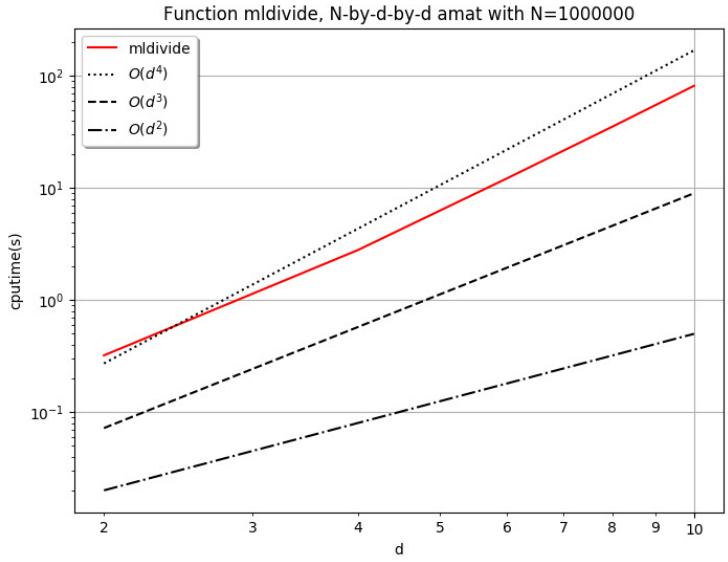

Figure 6: Computational times in seconds of of `X=mldivide(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object.

8.7.2 Benchmark function

The function `fc_amat.benchs.mldivide` measures performance of `X=mldivide(A,B)` where the input `A` is a `N`-by-`d`-by-`d` regular triangular upper `amat` object and `B` is either a `N`-by-`d`-by-`n` `amat` object or a `d`-by-`n` matrix. When running this function the `d` and `n` value are fixed, the number `N` varies and it is given by a list of values `LN`.

Syntaxe

```
fc_amat.benchs.mldivide(LN)
fc_amat.benchs.mldivide(LN,key,value,...)
```

Description

`fc_amat.benchs.mldivide(LN)`

runs a benchmark of the `X=mldivide(A,B)` command where `A` is a `N`-by-`2`-by-`2` regular triangular upper `amat` object and `B` is a `N`-by-`2`-by-`1` `amat` object for all `N` in `LN`. So, by default `d=2` and `n=1`.

`fc_amat.benchs.mldivide(LN,key,value,...)`

Optional key/value pairs arguments are available and can modify inputs of the `mldivide` function. `key` can be one of the following strings

- `'d'`, to set `d` (default `value` is `2`)

- `'n'` , to set `n` (default `value` is `1`)
- `'rhstype'` , to set the kind of `B` : `'amat'` (default) for `amat` object and `'mat'` for matrix
- `'class'` , to set classname of the two inputs. Value can be `'double'` (default) or `'single'` .
- `'complex'` , if `true` the inputs are complex (default `value` is `false`).
- `'a'` , to set the lower bound of the interval of the uniform distribution used to generate input datas (default `value` is `0.5`).
- `'b'` , to set `b` the upper bound of the interval of the uniform distribution used to generate input datas (default `value` is `2`).

In Listings 64 and 65 two examples with outputs are provided.

Listing 64: : Benchmarking `X=mldivide(A,B)` with `A` a `N`-by-4-by-4 matrix `amat` object and `B` a `N`-by-4-by-5 matrix `amat` object.

```
LN=10^5*[2:2:10];
fc_amat.benchs.mldivide(LN,'d',4,'n',5, 'rhstype','mat');
```

Output

```
#-------------------------------------------------------------------------------
#    computer: cosmos-ubuntu-18-04
#      system: Ubuntu 18.04.4 LTS (x86_64)
#   processor: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
#             (1 procs/14 cores by proc/2 threads by core)
#         RAM: 62.6 Go
#    software: Octave
#     release: 5.2.0
#-------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[double] with (N,m,n)=(N,4,4)
#    containing strictly diagonally dominant matrices
# 2nd parameter is :
# -> matrix[double] with (m,n)=(4,5), size=[4 5]
# Error function: @(X)max(norm(A*X-B))
#-------------------------------------------------------------------------------
#date:2020/02/16 19:19:58
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d         %.3f        %.3e
#labels:       N   mldivide(s)    Error[0]
          200000        1.310   2.096e-14
          400000        3.555   1.610e-14
          600000        7.040   1.862e-14
          800000        9.447   7.658e-14
         1000000       11.863   4.619e-14
```

Listing 65: : Benchmarking `X=mldivide(A,B)` where `A` is `N`-by-`3`-by-`3` `amat` object and `B` is `N`-by-`3`-by-`1` `amat` object with both `complex single` values.

```
LN=10^5*[2:2:10];
fc_amat.benchs.mldivide(LN,'d',3,'complex',true,'class','single', ...
    'info',false);
```

Output

```
#--------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[complex single] with (N,m,n)=(N,3,3)
#    containing strictly diagonally dominant matrices
# 2nd parameter is :
# -> amat[complex single] with (N,nr,nc)=(200000,3,1), size=[200000 3      1]
# Error function: @(X)max(norm(A*X-B))
#--------------------------------------------------------------------------------
#date:2020/02/16 19:24:10
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d         %.3f        %.3e
#labels:       N   mldivide(s)    Error[0]
          200000        0.264   2.270e-06
          400000        0.576   2.426e-06
          600000        0.988   2.147e-06
          800000        1.397   2.682e-06
         1000000        1.738   2.588e-06
```

8 References

[1] François Cuvelier. Efficient algorithms to perform linear algebra operations on 3d arrays in vector languages. 2018.

[2] Francois Cuvelier. fc-bench: Octave package for benckmarking. `http://www.math.univ-paris13.fr/~cuvelier/software/Octave/fc-bench.html`, 2018.