fc amat Octave package, User's Guide *

François Cuvelier[†]

September 17, 2018

Abstract

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

0 Contents

*Compiled with Octave 4.2.1

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

This work was partially supported by ANR Dedales.

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 gradiants 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 elementwise binary operators and functions can be used, as described in:

https://www.gnu.org/software/octave/doc/v4.2.1/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 matricial products 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, matricial products or more complex as determinants computation, factorizations, solving, ... by only using Octave language. One can refered 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 $m \times n$. In a more condensed way we say that `A` is a $N \times m \times m$ `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 $m \times n$)
`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 $m \times n$ 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 $\mathtt{N}\times 3\times 3$ and $\mathtt{N}\times 3\times 4$ `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 $\mathtt{N}\times 4\times 4$ `amat` objects, and `b` is a $\mathtt{N}\times 4\times 1$ `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: 3.108624e-15
max of E4 elements: 3.497203e-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 18.04.1 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: 4.2.1
#------------------------------------------------------------------------------
# 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:2018/09/16 12:03:17
#nbruns:5
#numpy:       i4          f4          f4
#format:      %d        %.3f        %.3e
#labels:       N   mldivide(s)    Error[0]
          200000       0.359   8.682e-14
          400000       1.238   4.388e-14
          600000       2.253   1.299e-13
          800000       2.886   7.017e-14
         1000000       3.678   8.693e-14
```

N	mtimes (s)	chol (s)	lu (s)	mldivide (s)
200 000	0.435(s)	0.014(s)	0.235(s)	0.299(s)
400 000	1.471(s)	0.040(s)	0.907(s)	0.958(s)
600 000	2.206(s)	0.060(s)	1.351(s)	1.513(s)
800 000	2.940(s)	0.081(s)	1.868(s)	2.036(s)
1 000 000	3.711(s)	0.106(s)	2.358(s)	2.545(s)
5 000 000	18.898(s)	0.926(s)	14.012(s)	16.268(s)
10 000 000	38.702(s)	1.851(s)	28.822(s)	33.950(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 under

Windows 10: with Octave 4.2.x (x in 0,1,2) and Octave 4.4.x (x in 0,1)

Mac OS High Sierra (10.13.4): with Octave 4.2.2 (installed with homebrew)

Ubuntu 18.04 LTS: with Octave 4.2.x (x in 0,1,2) and Octave 4.4.x (x in 0,1) (all compiled from source)

It is not compatible with Octave 4.0.x and previous.

2.1 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 package *fc-tools* 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. Then in a 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 F. Cuvelier <cuvelier@math.univ-paris13.fr>

1- Downloading and extracting the packages
2- Setting packages
Write in ~/Octave/packages/fc-amat-full/fc_bench-0.0.5/configure_loc.m ...
    ...
Write in ~/Octave/packages/fc-amat-full/fc_amat-0.0.2/configure_loc.m ...
3- Using packages :
   ->              fc-tools : 0.0.23
   ->              fc-bench : 0.0.5
   ->               fc-amat : 0.0.2
*** Using instructions
   To use the <fc-amat> package:
   addpath('~/Octave/packages/fc-amat-full/fc_amat-0.0.2')
   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-0.0.2')
>> fc_amat.init()
Using fc_bench[0.0.5] with fc_tools[0.0.23].
fc-amat[0.0.2] package/toolbox is ready to use!
```

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

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

3 Notations

Some typographic conventions are used in the following:

- $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{R}$, $\mathbb{C}$ 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 $\boldsymbol{aAbBcC}$ ....
- 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 $\mathbb{aAbBcC}$ ....
- All arrays of matrices or 3D-arrays or `amat` objects are represented with the bold blackboard font as : $\boldsymbol{\mathbb{M}} \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ or $\boldsymbol{\mathbb{b}}$ a N-by-m-by-n 3D-array. The first alphabetic characters are $\boldsymbol{\mathbb{aAbBcC}}$ ....

We now introduce some notations. Let $\boldsymbol{\mathbb{A}} = (\mathbb{A}_1, \ldots \mathbb{A}_N) \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ be a set of m-by-n matrices.We identify $\boldsymbol{\mathbb{A}}$ 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 `A` $\in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ has a matricial property 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 an N-by-n-by-m zeros `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)=
   2   1   7   3
   6   1  10   9
  Z(2)=
   2   1   7   3
   6   1  10   9
  ...

  Z(29)=
   2   1   7   3
   6   1  10   9
  Z(30)=
   2   1   7   3
   6   1  10   9
```

4.2 Particular generators

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

- `fc_amat.zeros` , generates an zeros `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 a zeros `N`-by-`m`-by-`n` `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)=
   0.162300  0.103277  0.322580
   0.342284  0.073000  0.315786
  matrix(2)=
   0.96422  0.10184  0.82498
   0.35289  0.50874  0.40917
  ...

  matrix(49)=
   0.268077  0.653604  0.082544
   0.601922  0.277525  0.358532
  matrix(50)=
   0.211777  0.919503  0.917542
   0.835133  0.032007  0.885713
```

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)=
  -0.81786 -0.22079  2.00258
   0.32105  1.19696 -0.18866
  matrix(2)=
   0.86434  0.61677  0.36906
   0.24202 -1.76201  2.20900
   ...

  matrix(49)=
  -0.84092 -0.88737  0.45599
  -0.33046  0.86163  2.65158
  matrix(50)=
  -1.15195  0.83447  0.73462
  -0.92570  1.81504  1.34278
```

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)=
   0   3  -1
   0  -2  -3
  matrix(2)=
  -5   5   3
  -3   1   2
  ...

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

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)=
   0.26372  0.68964
   0.68964  0.51194
  matrix(2)=
   0.65715  0.13424
   0.13424  0.85783
  ...

  matrix(49)=
   0.68823  0.53033
   0.53033  0.47122
  matrix(50)=
   0.063727  0.674280
   0.674280  0.999737
```

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)=
   0.73970  0.46537
   0.46537  1.34101
  matrix(2)=
  -0.10097  0.67055
   0.67055 -0.70101
  ...

  matrix(49)=
  -1.9316  1.6903
   1.6903 -1.3888
  matrix(50)=
  -0.073314  0.578709
   0.578709  0.607454
```

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 pseudorandom integer values drawn from the discrete uniform distribution on 1:Imax

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

pseudorandom 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   4
   4   4
  matrix(2)=
  -3   5
   5   1
  ...

  matrix(99)=
  -1   5
   5  -1
  matrix(100)=
   1   2
   2   1
```

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)=
   0.54655 + 0.86540i 0.67790 + 0.46746i
   0.67790 - 0.46746i 0.34255 + 0.11337i
  matrix(2)=
   0.85087 + 0.47965i 0.78095 - 0.75229i
   0.78095 + 0.75229i 0.16567 + 0.54543i
  ...

  matrix(49)=
   0.00118 - 0.14317i 0.06166 - 0.89734i
   0.06166 + 0.89734i 0.92888 + 0.16207i
  matrix(50)=
   0.18015 - 0.13837i 0.98998 + 0.48780i
   0.98998 - 0.48780i 0.75252 - 0.13614i
```

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)=
   0.14069 + 0.66559i 2.06276 - 0.82348i
   2.06276 + 0.82348i -1.32926 + 0.51524i
  matrix(2)=
  -0.75148 + 0.21483i -2.08023 - 0.93843i
  -2.08023 + 0.93843i -0.80646 + 1.80087i
  ...

  matrix(49)=
   0.73147 - 0.13050i -1.24562 - 0.43580i
  -1.24562 + 0.43580i 0.61212 + 0.80425i
  matrix(50)=
  -0.32677 - 0.86111i -0.22713 - 0.30154i
  -0.22713 + 0.30154i -0.81991 + 0.11632i
```

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)=
   3 - 3i  2 - 3i
   2 + 3i  0 + 3i
  matrix(2)=
   2 + 1i -3 + 2i
  -3 - 2i  5 + 4i
  ...

  matrix(99)=
   2 - 1i  5 + 0i
   5 - 0i -4 - 1i
  matrix(100)=
  -1 - 1i -3 + 5i
  -3 - 5i  1 - 3i
```

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  2.86685  0.00000
   0.00000  0.00000  1.11560
   0.00000  0.00000  0.00000
  Z(2)=
   0.00000  1.72859  0.00000
   0.00000  0.00000  4.74901
   0.00000  0.00000  0.00000
  ...

  Z(49)=
   0.00000  1.80371  0.00000
   0.00000  0.00000  1.52506
   0.00000  0.00000  0.00000
  Z(50)=
   0.00000  4.56119  0.00000
   0.00000  0.00000  0.66912
   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
   1.18609  0.00000  0.00000
   0.00000  3.37777  0.00000
  Z(2)=
   0.00000  0.00000  0.00000
   5.77589  0.00000  0.00000
   0.00000  4.33309  0.00000
  ...

  Z(49)=
   0.00000  0.00000  0.00000
   4.35512  0.00000  0.00000
   0.00000  4.27270  0.00000
  Z(50)=
   0.00000  0.00000  0.00000
   3.33080  0.00000  0.00000
   0.00000  3.48486  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   4
   0   0   0
  Z(2)=
   0   1   0
   0   0  -5
   0   0   0
  ...

  Z(49)=
   0   4   0
   0   0  -4
   0   0   0
  Z(50)=
   0  -1   0
   0   0  -1
   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)=
   0.47822  1.72849  0.00000
   0.63760  1.84145  4.56624
   4.93940  3.45513  1.60681
  Z(2)=
   0.02586  3.92066  0.00000
   4.58894  2.93345  1.97577
   3.63245  0.86811  2.24580
  ...

  Z(49)=
   0.31409  3.53731  0.00000
   1.61026  2.41480  4.85458
   4.44273  3.97222  1.22138
  Z(50)=
   1.06532  0.43160  0.00000
   2.68678  1.36116  1.58018
   3.31072  0.20934  3.18497
```

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
   3.65835  0.00000  0.00000
   3.37358  5.00059  0.00000
  Z(2)=
   0.00000  0.00000  0.00000
   4.55979  0.00000  0.00000
   5.39378  3.26381  0.00000
  ...

  Z(49)=
   0.00000  0.00000  0.00000
   2.47294  0.00000  0.00000
   4.36854  3.63698  0.00000
  Z(50)=
   0.00000  0.00000  0.00000
   3.81025  0.00000  0.00000
   2.21459  4.03886  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)=
  -2  -1   0
  -4  -3   4
  -2   3   5
  Z(2)=
   5   0   0
   1  -5   1
   2  -2  -5
  ...

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

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)=
   2.21046  0.72052  4.63522
   1.83625  0.60721  3.17250
   0.00000  2.12039  3.11659
  Z(2)=
   2.85120  0.71303  3.37125
   2.61537  1.29075  3.11909
   0.00000  3.01080  4.61724
  ...

  Z(49)=
   3.37238  4.62451  1.62760
   0.51618  3.49586  4.15363
   0.00000  4.16738  1.62380
  Z(50)=
   3.71503  0.50063  0.51944
   3.19787  1.51166  4.26385
   0.00000  3.84443  0.40601
```

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)=
   5.96065  4.13985  4.71808
   4.24137  4.37960  5.84421
   0.00000  3.70477  2.56892
  Z(2)=
   3.96341  4.81129  4.28604
   4.67596  3.05944  3.10928
   0.00000  4.61160  3.57968
  ...

  Z(49)=
   4.83169  3.41658  4.09875
   3.50863  4.12481  2.95405
   0.00000  4.72311  3.88201
  Z(50)=
   3.86921  3.03941  4.33878
   4.96630  4.26620  4.32492
   0.00000  3.70896  3.95123
```

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  -5   2
   0   0  -1
   0   0   0
  Z(2)=
   0   4   0
   0   0   4
   0   0   0
  ...

  Z(49)=
   0  -5   0
   0   0   5
   0   0   0
  Z(50)=
   0  -3  -4
   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)=
  -0.77789 - 2.40801i 0.78479 + 0.07216i 0.66875 - 0.55415i
  -0.64979 + 0.22981i -1.98375 - 1.88303i -0.82007 - 0.75423i
  -0.78963 + 0.83161i -0.52728 - 0.52738i -1.33215 - 2.23131i
  Z(2)=
  -1.71212 + 0.80777i 0.30256 + 0.44065i 0.25464 + 0.26410i
   0.90300 + 0.54097i 0.71601 + 2.22596i 0.23838 - 0.58021i
  -0.68331 - 0.51406i 0.69914 + 0.27192i -2.47954 - 0.57956i
  ...

  Z(49)=
   0.419992 + 1.277553i -0.054503 + 0.007714i -0.045537 - 0.310801i
   0.363618 + 0.899288i -2.265441 - 0.640170i -0.750250 + 0.201043i
  -0.237873 + 0.650659i 0.072127 - 0.156984i -1.779876 + 0.411884i
  Z(50)=
   2.51934 - 0.58922i 0.20618 - 0.96974i 0.38060 + 0.86251i
  -0.44007 + 0.77774i -1.30830 - 1.88104i 0.14156 - 0.30670i
  -0.39681 - 0.88653i -0.34475 + 0.36923i -2.45033 + 0.31948i
```

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)=
  -15.2931   4.4226   5.3910
    3.7043  13.9892   4.6007
    4.4133   6.5237 -16.8454
  Z(2)=
  -17.3435   4.5242   7.1804
    4.1595 -14.2908   5.8489
    5.6384   6.4579 -16.9438
  ...

  Z(49)=
   14.4832   4.8193   4.8182
    5.4831 -14.9574   3.8421
    3.4436   6.8317  15.6339
  Z(50)=
  -14.4360   4.9359   5.3349
    4.5212 -12.3521   3.6805
    5.8326   3.7637 -14.2649
```

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 pseudorandom 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 pseudorandom 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)=
    7 - 14i   5 -  5i  -1 -  3i
    1 +  0i   8 -  7i   2 +  0i
    3 +  4i   0 -  5i   5 - 16i
  Z(2)=
    3 + 14i  -4 -  5i   0 +  3i
    3 +  2i -12 +  9i   0 +  5i
   -3 +  5i  -2 -  2i  -7 - 14i
  ...

  Z(49)=
   -5 - 17i  -1 +  3i   4 +  4i
   -3 +  5i   9 + 11i   4 +  0i
   -3 +  3i  -1 +  5i   8 + 11i
  Z(50)=
   -3 -  6i   0 +  0i   2 -  2i
    1 -  1i   7 - 12i   4 -  5i
    2 -  2i   5 +  4i  15 -  7i
```

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)=
   1.141497 -1.290412 -0.077571
  -1.290412  4.260648 -1.687598
  -0.077571 -1.687598  1.334730
  Z(2)=
   1.88598  0.33419  0.71086
   0.33419  0.82415  1.21344
   0.71086  1.21344  2.94938
  ...

  Z(49)=
   2.96378  2.14503 -1.33325
   2.14503  3.24436  0.22837
  -1.33325  0.22837  2.33649
  Z(50)=
   1.8766 -1.0419  1.5994
  -1.0419  3.1462 -2.0009
   1.5994 -2.0009  2.0516
```

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)=
   279.4370   -2.1602   10.5344
    -2.1602  284.9448 -110.8492
    10.5344 -110.8492  251.4429
  Z(2)=
   247.214   27.702 -133.508
    27.702  203.016   18.694
  -133.508   18.694  227.247
  ...

  Z(49)=
   347.46 -136.83 -151.77
  -136.83  291.60 -108.89
  -151.77 -108.89  272.01
  Z(50)=
   391.30 -171.34 -154.67
  -171.34  348.26 -149.93
  -154.67 -149.93  301.86
```

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 generated 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 pseudorandom 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 pseudorandom 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)=
    51    51   -24
    51   161     0
   -24     0    90
  Z(2)=
   139   -83    41
   -83   125   -41
    41   -41    83
  ...

  Z(49)=
   222    67    16
    67    78    54
    16    54   147
  Z(50)=
   53  -36   -2
  -36   48   36
   -2   36   65
```

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-0.0.2/+fc_amat/+random/randherpd.m'
warning: called from
    randherpd01 at line 1 column 2
    fctoto at line 5 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)=
   10.3329 + 0.0000i   4.3007 - 0.5762i  -2.5107 - 0.4735i
    4.3007 + 0.5762i   6.9518 + 0.0000i   2.8317 + 1.4094i
   -2.5107 + 0.4735i   2.8317 - 1.4094i  12.6671 + 0.0000i
  Z(2)=
   9.8903 + 0.0000i 4.3848 + 2.6237i -4.7093 - 1.0134i
   4.3848 - 2.6237i 6.6922 + 0.0000i -3.2425 + 0.4698i
  -4.7093 + 1.0134i -3.2425 - 0.4698i 6.2729 + 0.0000i
  ...

  Z(49)=
   4.94334 + 0.00000i -0.83129 - 1.24896i 0.56506 + 0.71350i
  -0.83129 + 1.24896i 8.52441 + 0.00000i -3.79866 - 3.75527i
   0.56506 - 0.71350i -3.79866 + 3.75527i 9.06173 + 0.00000i
  Z(50)=
   5.02406 + 0.00000i 1.18740 - 0.25416i 0.49865 - 0.52385i
   1.18740 + 0.25416i 5.31599 + 0.00000i -0.24566 + 3.18233i
   0.49865 + 0.52385i -0.24566 - 3.18233i 5.99775 + 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)=
   581.958 +  0.000i   68.757 - 103.033i 105.407 -  2.020i
    68.757 + 103.033i 596.524 +  0.000i  -75.792 + 87.991i
   105.407 +  2.020i  -75.792 - 87.991i  536.923 +  0.000i
  Z(2)=
   497.205 +  0.000i   28.042 - 228.251i -65.933 - 254.062i
    28.042 + 228.251i 411.650 +  0.000i  -17.217 - 31.192i
   -65.933 + 254.062i -17.217 + 31.192i  546.283 +  0.000i
  ...

  Z(49)=
   559.897 +  0.000i  302.830 - 226.861i 117.887 - 38.401i
   302.830 + 226.861i 718.091 +  0.000i   66.436 + 144.098i
   117.887 + 38.401i   66.436 - 144.098i 535.676 +  0.000i
  Z(50)=
   645.792 +  0.000i  -44.151 +  6.523i -213.225 + 190.178i
   -44.151 -  6.523i  466.913 +  0.000i  -85.153 + 141.892i
  -213.225 - 190.178i -85.153 - 141.892i 631.652 +  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)=
   293 +  0i   108 + 36i    37 +  14i
   108 - 36i   318 +  0i    52 + 116i
    37 - 14i    52 - 116i  327 +   0i
  Z(2)=
   268 +  0i   -73 + 84i    37 + 142i
   -73 - 84i   127 +  0i    81 -  36i
    37 - 142i   81 + 36i   235 +   0i
  ...

  Z(49)=
   281 +  0i    45 - 101i   94 + 122i
    45 + 101i  296 +  0i    60 + 145i
    94 - 122i   60 - 145i  413 +   0i
  Z(50)=
   208 +  0i  -139 +  8i   -90 +   5i
  -139 -  8i   503 +  0i   -38 - 210i
   -90 -  5i   -38 + 210i  292 +   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 ∀`i` ∈ `1:length(I)`, ∀`j` ∈ `1:length(J)`, ∀ `k` ∈ `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)` ← `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)` ← `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 ∀`i` ∈ `1:p`, ∀`j` ∈ `1:q`, ∀ `k` ∈ `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)` ← `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)` × `length(J)` matrix, then ∀`k` ∈ `1:length(K)` the `K(k)`-th matrix of `A` is set to `B` , i.e. ∀`i` ∈ `1:length(I)`, ∀`j` ∈ `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 redimensioned 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 an other input data. We define four kinds of element by element arithmetic binary operations when $\mathbb{A}$ is the left operand.

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

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

   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

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

   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

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

   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

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

   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)=
  -0.10661 -0.29965 -0.45368
  -0.44280 -0.15818 -0.45084
  D(2)=
  -0.10661 -0.29965 -0.45368
  -0.44280 -0.15818 -0.45084
  ...

  D(99)=
  -0.10661 -0.29965 -0.45368
  -0.44280 -0.15818 -0.45084
  D(100)=
  -0.10661 -0.29965 -0.45368
  -0.44280 -0.15818 -0.45084
```

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)=
  0  1  0
  0  0  0
  D(2)=
  0  1  1
  0  0  0
  ...

  D(99)=
  1  1  1
  1  1  1
  D(100)=
  1  0  1
  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 whose the 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 ∀`i` ∈ `[1:m]`, ∀`j` ∈ `[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 ∀`i` ∈

[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 whose the 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 ∃`i` ∈ `[1:m]`, ∃`j` ∈ `[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 ∃`i` ∈ `[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 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 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] = min (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] = 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)=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] = min (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')
```

Output

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

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))`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)=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)=
   9   8   1
   5   6   2
  X(2)=
   6   2   7
   9   6   6
  ...

  X(9)=
   1   8   7
   9   5   6
  X(10)=
   6   2   7
   3   6   7
W=min(X) ->
   1   2   1
   1   2   2
W1=min(X,[],1) ->
W1 is a 10x1x3 amat[double] object
  W1(1)=
   5
   6
   1
  W1(2)=
   6
   2
   6
  ...

  W1(9)=
   1
   5
   6
  W1(10)=
   3
   2
   7
```

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:

$$\texttt{Z= alpha*X + beta*Y} \quad (5)$$

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,

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

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 Matricial product

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

$$Z= X*Y \tag{6}$$

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

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

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

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

- 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 fonction of very small values of `d` .

N	mtimes
200 000	0.434(s)
400 000	1.476(s)
600 000	2.220(s)
800 000	2.980(s)
1 000 000	3.713(s)
5 000 000	19.379(s)
10 000 000	38.721(s)

Table 2: Computational times in seconds of `mtimes(X,Y)` (`X*Y` matricial products) 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 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]`)

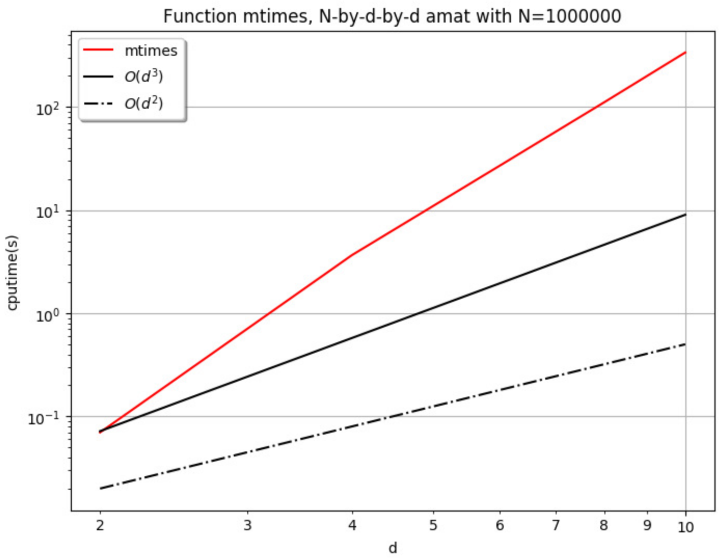

Figure 1: Computational times in seconds of `mtimes(X,Y)` or `X*Y` (matricial products) where `X` and `Y` are `N`-by-`d`-by-`d` `amat` objects.

- `'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 18.04.1 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: 4.2.1
#------------------------------------------------------------------------------
# 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:2018/09/16 16:34:43
#nbruns:5
#numpy:       i4         f4
#format:      %d       %.3f
#labels:       N   mtimes(s)
          200000      0.380
          400000      1.307
          600000      1.961
          800000      2.596
         1000000      3.251
```

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:2018/09/16 16:35:51
#nbruns:5
#numpy:       i4         f4
#format:      %d       %.3f
#labels:       N   mtimes(s)
          200000      0.516
          400000      1.783
          600000      2.659
          800000      3.551
         1000000      4.422
```

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 an 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 an 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)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   6.9389e-17 + 0.0000e+00i 1.1102e-16 - 5.5511e-17i
  -1.1102e-16 + 5.5511e-17i 1.1102e-16 - 5.5511e-17i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 2.7756e-17i
  -2.2204e-16 + 0.0000e+00i
  E(2)=
 Columns 1 and 2:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   2.2204e-16 - 4.4409e-16i 0.0000e+00 + 0.0000e+00i
  -2.2204e-16 + 4.4409e-16i 1.3878e-17 + 0.0000e+00i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i
   4.1633e-17 + 0.0000e+00i
  ...

  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:

   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
  -5.5511e-17 + 0.0000e+00i 0.0000e+00 - 2.2204e-16i
   0.0000e+00 - 1.3878e-17i 0.0000e+00 + 0.0000e+00i

 Column 3:

   0.0000e+00 + 0.0000e+00i
   6.9389e-18 + 0.0000e+00i
   0.0000e+00 + 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 fonction of very small values of `d` .

N	d=2	d=4	d=6	d=8	d=10
200 000	0.038(s)	0.222(s)	1.485(s)	5.834(s)	13.947(s)
400 000	0.081(s)	0.713(s)	3.944(s)	11.673(s)	27.611(s)
600 000	0.114(s)	1.360(s)	5.868(s)	17.439(s)	41.832(s)
800 000	0.155(s)	1.822(s)	7.862(s)	23.513(s)	56.893(s)
1 000 000	0.192(s)	2.278(s)	10.029(s)	29.951(s)	70.819(s)

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

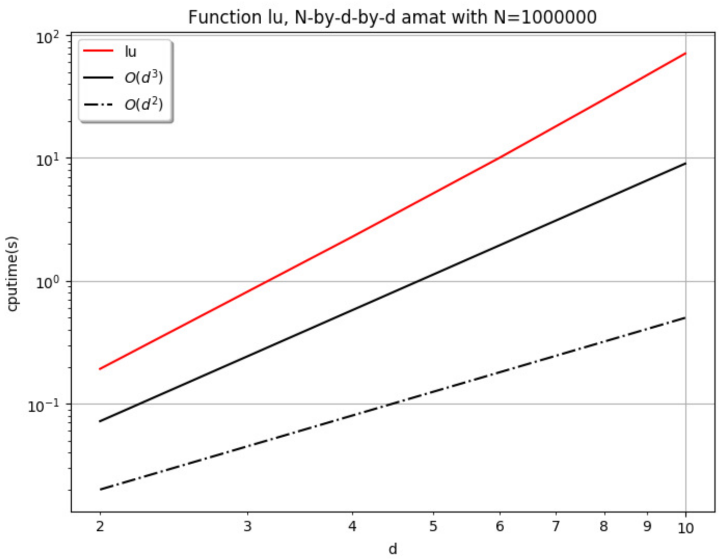

Figure 2: Computational times in seconds of of `[L,U,P]=lu(A)` where `A` is a `N`-by-`d`-by-`d` `amat` object.

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 **??** 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 18.04.1 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: 4.2.1
#----------------------------------------------------------------------------
# input parameter is :
# -> amat[double] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
#----------------------------------------------------------------------------
#date:2018/09/16 19:04:32
#nbruns:5
#numpy:       i4      f4           f4
#format:      %d    %.3f         %.3e
#labels:       N   lu(s)     Error[0]
          200000   0.248    1.554e-15
          400000   0.907    1.776e-15
          600000   1.361    1.749e-15
          800000   1.926    1.554e-15
         1000000   2.370    1.887e-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:2018/09/16 19:05:41
#nbruns:5
#numpy:       i4      f4           f4
#format:      %d    %.3f         %.3e
#labels:       N   lu(s)     Error[0]
          200000   0.168    9.842e-07
          400000   0.400    9.873e-07
          600000   0.646    1.243e-06
          800000   0.901    1.160e-06
         1000000   1.145    9.848e-07
```

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:

  -1.7764e-15 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 0.0000e+00i 0.0000e+00 + 0.0000e+00i
   0.0000e+00 + 2.2204e-16i 0.0000e+00 + 1.1102e-16i

 Column 3:

   0.0000e+00 - 2.2204e-16i
   0.0000e+00 - 1.1102e-16i
   0.0000e+00 + 0.0000e+00i
  E(2)=
 Columns 1 and 2:

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

 Column 3:

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

  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:

  -1.7764e-15 + 0.0000e+00i 0.0000e+00 - 4.4409e-16i
   0.0000e+00 + 4.4409e-16i 0.0000e+00 + 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 + 0.0000e+00i
   0.0000e+00 + 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)	0.045(s)	0.088(s)	0.150(s)
400 000	0.008(s)	0.033(s)	0.093(s)	0.180(s)	0.305(s)
600 000	0.011(s)	0.060(s)	0.142(s)	0.277(s)	0.467(s)
800 000	0.016(s)	0.082(s)	0.196(s)	0.380(s)	0.650(s)
1 000 000	0.020(s)	0.105(s)	0.260(s)	0.510(s)	0.872(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.

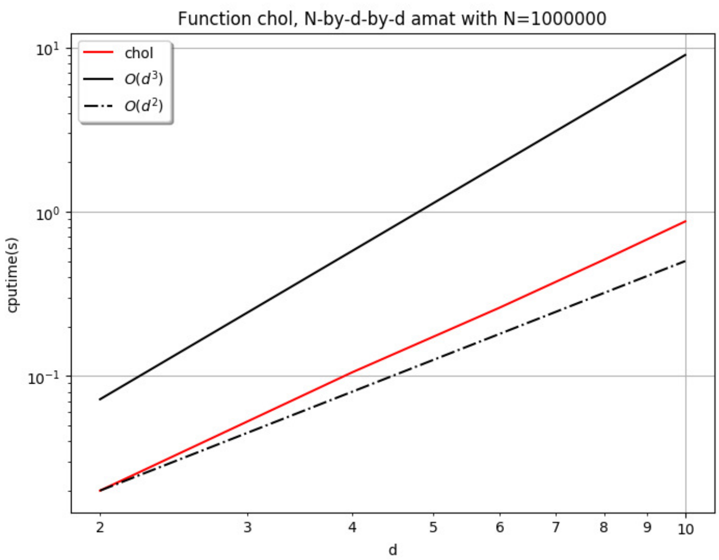

Figure 3: Computational times in seconds of of `B=chol(A)` where `A` is a `N`-by-`d`-by-`d` symmetric positive definite `amat` object.

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 **??** 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 18.04.1 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: 4.2.1
#---------------------------------------------------------------------------
# Benchmarking command: @(A) chol(A,'lower');
#---------------------------------------------------------------------------
# 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)
#---------------------------------------------------------------------------
#date:2018/09/16 20:28:43
#nbruns:5
#numpy:       i4       f4          f4
#format:      %d     %.3f        %.3e
#labels:       N   chol(s)    Error[0]
          200000    0.014   2.842e-14
          400000    0.039   2.132e-14
          600000    0.059   2.842e-14
          800000    0.081   3.197e-14
         1000000    0.104   3.020e-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

```
#---------------------------------------------------------------------------
# Benchmarking command: @(A) chol(A,'upper');
#---------------------------------------------------------------------------
# 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)
#---------------------------------------------------------------------------
#date:2018/09/16 20:29:11
#nbruns:5
#numpy:       i4       f4          f4
#format:      %d     %.3f        %.3e
#labels:       N   chol(s)    Error[0]
          200000    0.036   1.156e-05
          400000    0.081   1.156e-05
          600000    0.123   1.189e-05
          800000    0.179   1.168e-05
         1000000    0.221   1.240e-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 factorizations 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)=
   1.7902e-15
  E(2)=
   4.4409e-16
  ...

  E(99)=
   1.9860e-15
  E(100)=
   8.8818e-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 fonction of very small values of d .

N	d=2	d=4	d=6	d=8	d=10
200 000	0.041(s)	0.218(s)	1.141(s)	5.767(s)	13.764(s)
400 000	0.083(s)	0.555(s)	3.851(s)	11.568(s)	27.466(s)
600 000	0.130(s)	1.338(s)	5.809(s)	17.406(s)	41.530(s)
800 000	0.164(s)	1.782(s)	7.895(s)	23.667(s)	55.688(s)
1 000 000	0.195(s)	2.257(s)	10.090(s)	29.682(s)	70.091(s)

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

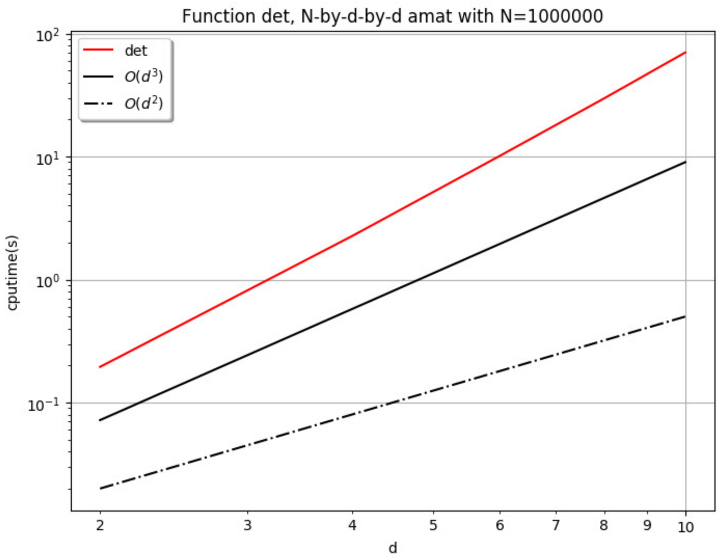

Figure 4: Computational times in seconds of of B=det(A) where A is a N-by-d-by-d amat object.

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 **??** 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 18.04.1 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: 4.2.1
#------------------------------------------------------------------------------
# Benchmarking command: @(A) det(A,'select','lu');
#------------------------------------------------------------------------------
# input parameter for N=200000 is :
# -> amat[double] with (N,nr,nc)=(200000,4,4), size=[200000 4      4]
#------------------------------------------------------------------------------
#date:2018/09/16 21:38:42
#nbruns:5
#numpy:       i4      f4
#format:      %d    %.3f
#labels:       N   det(s)
          200000   0.272
          400000   0.922
          600000   1.397
          800000   1.890
         1000000   2.315
```

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

```
#--------------------------------------------------------------------------
# Benchmarking command: @(A) det(A,'select','lu');
#--------------------------------------------------------------------------
# input parameter for N=200000 is :
# -> amat[complex single] with (N,nr,nc)=(200000,3,3), size=[200000 3      3]
#--------------------------------------------------------------------------
#date:2018/09/16 21:39:31
#nbruns:5
#numpy:       i4      f4
#format:      %d    %.3f
#labels:       N   det(s)
          200000   0.178
          400000   0.383
          600000   0.655
          800000   0.888
         1000000   1.198
```

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 an 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 in a similar way.

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: 1.1130e-14
X2 is a 100x3x1 amat[double] object
E2 is a 100x3x1 amat[double] object
  E2(1)=
   5.5511e-16
   0.0000e+00
   0.0000e+00
  E2(2)=
  -3.3307e-15
   0.0000e+00
   0.0000e+00
  ...

  E2(99)=
   0
   0
   0
  E2(100)=
   2.2204e-16
   0.0000e+00
   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 fonction of very small values of `d`.

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

N	solvetriu	Error
200 000	0.009(s)	$5.5510e-15$
400 000	0.022(s)	$7.0780e-15$
600 000	0.028(s)	$9.9920e-15$
800 000	0.037(s)	$6.9940e-15$
1 000 000	0.070(s)	$1.3540e-14$
5 000 000	0.559(s)	$9.4370e-15$
10 000 000	1.238(s)	$1.2770e-14$

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.

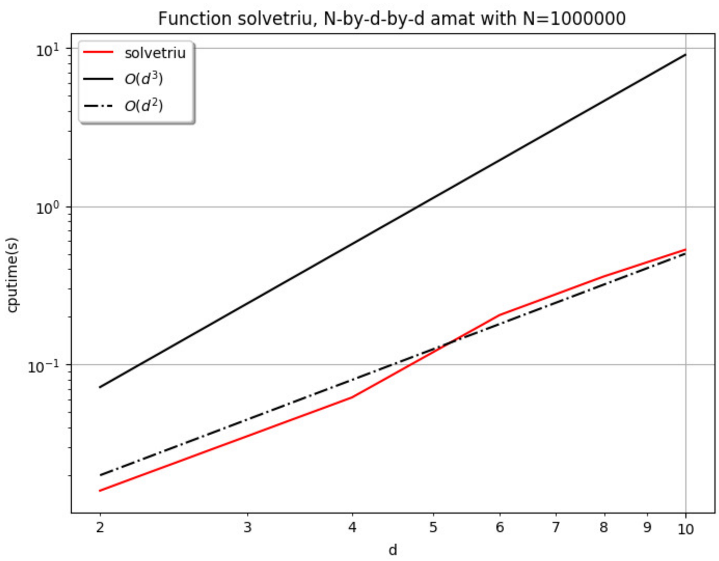

Figure 5: Computational times in seconds of of `X=solvetriu(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object.

```
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 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 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 18.04.1 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: 4.2.1
#--------------------------------------------------------------------------
# Benchmarking command: @(A,B) solvetriu(A,B);
#--------------------------------------------------------------------------
# 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))
#--------------------------------------------------------------------------
#date:2018/09/16 21:42:24
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d          %.3f        %.3e
#labels:       N   solvetriu(s)    Error[0]
          200000         0.101   1.345e-14
          400000         0.603   1.630e-14
          600000         0.906   1.461e-14
          800000         1.214   1.345e-14
         1000000         1.518   1.960e-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

```
#--------------------------------------------------------------------------
# Benchmarking command: @(A,B) solvetriu(A,B);
#--------------------------------------------------------------------------
# 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))
#--------------------------------------------------------------------------
#date:2018/09/16 21:43:14
#nbruns:5
#numpy:       i4           f4          f4
#format:      %d          %.3f        %.3e
#labels:       N   solvetriu(s)    Error[0]
          200000         0.024   1.820e-06
          400000         0.047   1.916e-06
          600000         0.074   3.014e-06
          800000         0.118   2.521e-06
         1000000         0.145   3.549e-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

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

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: 2.0539e-15
X2 is a 100x3x1 amat[double] object
E2 is a 100x3x1 amat[double] object
  E2(1)=
   0.0000e+00
   1.1102e-16
   0.0000e+00
  E2(2)=
   0.0000e+00
   0.0000e+00
  -3.3307e-16
  ...

  E2(99)=
   0.0000e+00
   0.0000e+00
   1.1102e-16
  E2(100)=
   0.0000e+00
  -4.4409e-16
   3.3307e-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 fonction of very small values of `d` .

N	d=2	d=4	d=6	d=8	d=10
200 000	0.062(s)	0.267(s)	1.704(s)	6.014(s)	14.253(s)
400 000	0.111(s)	0.987(s)	4.056(s)	11.968(s)	28.020(s)
600 000	0.152(s)	1.466(s)	6.072(s)	18.099(s)	42.765(s)
800 000	0.209(s)	2.029(s)	8.305(s)	24.580(s)	59.244(s)
1 000 000	0.269(s)	2.587(s)	10.486(s)	32.035(s)	74.451(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.

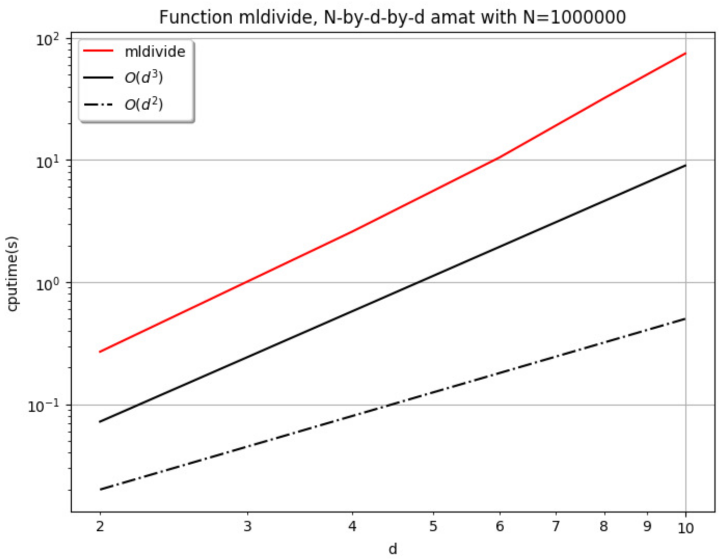

Figure 6: Computational times in seconds of of `X=mldivide(A,B)` where `A` is a `N`-by-`d`-by-`d` `amat` object and `B` is a `N`-by-`d`-by-`1` `amat` object.

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 18.04.1 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: 4.2.1
#--------------------------------------------------------------------------
# 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:2018/09/16 22:58:28
#nbruns:5
#numpy:       i4          f4          f4
#format:      %d        %.3f        %.3e
#labels:       N  mldivide(s)   Error[0]
          200000       0.969   1.404e-14
          400000       4.398   1.898e-14
          600000       6.712   2.165e-14
          800000       8.911   2.687e-14
         1000000      11.195   2.381e-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:2018/09/16 23:02:32
#nbruns:5
#numpy:       i4          f4          f4
#format:      %d        %.3f        %.3e
#labels:       N  mldivide(s)   Error[0]
          200000       0.243   2.109e-06
          400000       0.511   2.302e-06
          600000       0.921   4.968e-06
          800000       1.282   2.580e-06
         1000000       1.568   2.706e-06
```

8 References

[1] François Cuvelier. Efficient algorithms to perform linear algebra operations on 3d arrays in vector languages. Technical report, LAGA - Institut Galilée - Paris 13 University, 2018.

[2] Francois Cuvelier. fc-bench: Octave package for benckmarking. http://www.math.univ-paris13.fr/~cuvelier/software/Octave/fc-bench.html, 2018.