Abstract

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

Contents

1 Presentation 3
2 Installation 7
3 Notations 10

*LaTeX manual, revision 0.1.1, compiled with Octave 5.1.0, and packages fc-amat[0.1.1], fc-tools[0.0.29], fc-bench[0.1.1]
†LAGA, UMR 7539, CNRS, Université Paris 13 - Sorbonne Paris Cité, Université Paris 8, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr.

This work was supported by the ANR project DEDALES under grant ANR-14-CE23-0005.
4 Constructor and generators
   4.1 Constructor ........................................ 11
   4.2 Particular generators .............................. 12
   4.3 Random generators .................................. 15

5 Indexing ........................................ 54
   5.1 Subscripted reference .............................. 54
   5.2 Subscripted assignment ............................ 55

6 Elementary operations ................................ 57
   6.1 Arithmetic operations .............................. 57
   6.2 Relational operators ............................... 59
   6.3 Logical operations ................................... 60

7 Elementary mathematical functions .................... 64
   7.1 Trigonometric functions ............................ 64
   7.2 Exponents and Logarithms .......................... 65
   7.3 Complex Arithmetic ................................. 65
   7.4 Utility methods ..................................... 65

8 Linear algebra ................................... 69
   8.1 Linear combination ................................. 69
   8.2 Matrix product ..................................... 70
   8.3 LU Factorization .................................... 74
   8.4 Cholesky Factorization ............................. 79
   8.5 Determinants ........................................ 83
   8.6 Solving particular linear systems ................. 87
   8.7 Solving linear systems .............................. 91
Initially the Octave package was created to be used with finite elements codes for computing volumes and gradients of barycentric coordinates on each mesh elements. The volume of mesh element can be computed with the determinant of a matrix depending on the coordinates of the mesh element vertices. The gradients of the barycentric coordinates of a mesh element are solutions of linear systems. So we want to be able to do efficiently these operations on a very large number (few millions?) of very small matrices with same order (order less than 10?). In Octave, all these matrices can be stored as a \( N \)-by-\( m \)-by-\( m \) 3D-array. Currently, with Octave from version 4.0.3 (and Matlab from release R2017a) only element-wise binary operators and functions can be used, as described in:

https://www.gnu.org/software/octave/doc/v5.1.0/Broadcasting.html

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

\[
\begin{align*}
A &= \text{rand}(m,n); \quad \text{% generate a } m\text{-by-}n \text{ matrix } (n \geq 1) \\
B &= \text{randn}(N,m,n); \quad \text{% generate a } N\text{-by-}m\text{-by-}n \text{ 3D-array} \\
C &= \text{reshape}(A,[1,m,n])+B; \quad \text{% generate } "A+B" \text{ 3D-array}
\end{align*}
\]

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

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

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

## 1 Presentation

The \amat\ object provided in the \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 \( n \times m \) or \( d \times d \):

\[
\begin{align*}
\amat(N,m,n) & \quad \text{constructor with all matrices to zeros} \\
fc_{\amat}.zeros(N,m,n) & \quad \text{same as } \amat(N,m,n) \\
fc_{\amat}.ones(N,m,n) & \quad \text{matrices of } 1 \\
f_{\amat}.eye(N,d) & \quad \text{identity matrices} \\
f_{\amat}.random.randn(N,m,n) & \quad \text{normally distributed random elements} \\
f_{\amat}.random.randnsym(N,d) & \quad \text{randomized symmetric matrices} \\
f_{\amat}.random.randnher(N,d) & \quad \text{randomized hermitian matrices} \\
f_{\amat}.random.randntril(N,d) & \quad \text{randomized lower triangular matrices} \\
f_{\amat}.random.randntriu(N,d) & \quad \text{randomized upper triangular matrices} \\
\end{align*}
\]

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 \( p_{ij} \) of all the matrices as an \( N \times 1 \times 1 \) 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 \( p_{ij} \) 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.

```matlab
A=fc_amat.random.randn(100,3,4); % A : 100-by-3-by-4 amat
B=randn(3,4);
A(10)=B; % assign to the 10-th matrix
A(20:25)=B; % the matrices 20 to 25 are set to B
A(30)=B; % 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)=2; % elements (2,3) and (3,3) of all the matrices are set to 1
A(1,1)=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: Assignments with amat object

The amat class is provided with the usual elementary operations:

- \( +, -, \cdot, /, \backslash, \wedge. \) (Arithmetic operators)
- \( ==, >=, >, <=, <, ~. \) (Relational operators)
- \( &\;|\;\sim\;\ominus\;\ominus\) (Logical operators)

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

```matlab
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.*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.

```matlab
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+B;
D2=C.*A/2;
D3=-2.*A.*C;
```

Listing 3: Matricial operations with amat object
Some usual mathematical functions as \(\cos\), \(\sin\), \(\exp\), \(\sqrt{\text{r}}\), \(\text{abs}\), \(\text{max}\), ... are available for \(\text{amat}\) objects. One can refer to section 7 for more details.

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

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

Finally, in Table 1 benchmark functions \(\text{fc amat.benchs.mtimes}\), \(\text{fc amat.benchs.lu}\), \(\text{fc amat.benchs.chol}\) and \(\text{fc amat.benchs.mldivide}\) are respectively used to get cputimes of the \(X=\text{mtimes}(A,B)\), \([L,U,P]=\text{lu}(A)\), \(R=\text{chol}(A)\) and \(X=\text{mldivide}(A,b)\) where \(A\) and \(B\) are \(N \times 4 \times 4\) \(\text{amat}\) objects, and \(b\) is a \(N \times 4 \times 1\) \(\text{amat}\) object.
Listing 4: Linear algebra with amat object

% Generate 100-by-4-by-4 amat object symmetric positive definite...

A = fc_amat.random.randnsymd(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
', max(E1(:)));

% Cholesky factorizations:
R = chol(A);
E2 = abs(R'*R-A);

fprintf('max of E2 elements: %.6e
', 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
', max(E3(:)));

B = fc_amat.random.randn(100,4,1);

Y = A\B; % RHS
E4 = abs(A*Y-B);

fprintf('max of E4 elements: %.6e
', max(E4(:)));

whos

Output:

max of E1 elements: 3.552714e-15
max of E2 elements: 7.105427e-15
max of E3 elements: 7.105427e-15
max of E4 elements: 1.199041e-14

Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4x1</td>
<td>32 double</td>
<td></td>
</tr>
</tbody>
</table>

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`

\[
LS=10^5 \times [2:2:10];
fc_amat_benchs.mldivide(LS,'d','3','m','4','numruns',5)
\]

Output

<table>
<thead>
<tr>
<th>N</th>
<th>mtimes (s)</th>
<th>chol (s)</th>
<th>lu (s)</th>
<th>mldivide (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 000</td>
<td>0.546(s)</td>
<td>0.026(s)</td>
<td>0.493(s)</td>
<td>0.072(s)</td>
</tr>
<tr>
<td>400 000</td>
<td>2.558(s)</td>
<td>0.078(s)</td>
<td>1.606(s)</td>
<td>1.636(s)</td>
</tr>
<tr>
<td>600 000</td>
<td>3.717(s)</td>
<td>0.127(s)</td>
<td>2.563(s)</td>
<td>2.750(s)</td>
</tr>
<tr>
<td>800 000</td>
<td>4.973(s)</td>
<td>0.157(s)</td>
<td>3.333(s)</td>
<td>3.802(s)</td>
</tr>
<tr>
<td>1 000 000</td>
<td>6.221(s)</td>
<td>0.219(s)</td>
<td>4.212(s)</td>
<td>4.526(s)</td>
</tr>
<tr>
<td>5 000 000</td>
<td>33.062(s)</td>
<td>1.817(s)</td>
<td>26.460(s)</td>
<td>30.107(s)</td>
</tr>
<tr>
<td>10 000 000</td>
<td>67.444(s)</td>
<td>3.553(s)</td>
<td>53.819(s)</td>
<td>63.394(s)</td>
</tr>
</tbody>
</table>

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

### 2 Installation

This toolbox was tested on various OS and Octave releases:
It is not compatible with Octave 4.0.x and previous.

### 2.0.1 Automatic installation, all in one (recommended)

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

```
ofc_amat_install.m
```

or to get it on the dedicated [web page](http://octave-app.org/Download.html). 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 by using previous link. For example, in a Linux terminal, we can do:

```bash
cd ~/.Octave/packages
HTTP=http://www.math.univ-paris13.fr/~cuvelier/software/codes/Octave
wget $HTTP/fc-amat/0.1.1/ofc_amat_install.m
```

Then in an Octave terminal run the following commands:

```octave
>> 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) 2018-2019 F. Cuvelier

1- Downloading and extracting the packages
2- Setting the `<fc-amat>` package
Write in `/Octave/packages/fc-amat-full/fc_amat-0.1.1/configure_loc.m ...`
3- Using packages :
   ```
   -> fc-tools : 0.0.29
   -> fc-bench : 0.1.1
   with
   fc-amat : 0.1.1
   ```
*** Using instructions
To use the `<fc-amat>` package:
   ```
   addpath('~/Octave/packages/fc-amat-full/fc_amat-0.1.1')
   fc_amat.init()
   ```
   See '~/Octave/packages/fc_amat_set.m'

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

```
>> addpath('~/Octave/packages/fc-amat-full/fc_amat-0.1.1')
>> fc_amat.init()
```

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

```
Try to use default parameters!
Use fc_tools.configure to configure.
Write in ...
/home/cuvelier/tmp/fc-amat-full/fc_tools-0.0.29/configure_loc.m ...
Try to use default parameters!
Use fc_bench.configure to configure.
Write in ...
/home/cuvelier/tmp/fc-amat-full/fc_bench-0.1.1/configure_loc.m ...
Using fc_amat[0.1.1] with fc_tools[0.0.29], fc_bench[0.1.1].
```

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

```
Using fc_amat[0.1.1] with fc_tools[0.0.29], fc_bench[0.1.1].
```

For **uninstalling**, one just has to delete the directory
`~/Octave/packages/fc-amat-full`

### 2.0.2 Manual installation

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

For example under Linux, to install this package in `~/Octave/packages` directory, one can download `fc-amat-0.1.1-full.tar.gz` and extract it in the `~/Octave/packages` directory:
For each Octave session, one has to set the package by adding path of all packages:

```octave
>> warning('off','Octave:shadowed-function'); more off
>> addpath('/Octave/packages/fc-amat-0.1.1/fc_amat-0.1.1/)
>> addpath('/Octave/packages/fc-amat-0.1.1/fc_tools-0.0.29/)
>> addpath('/Octave/packages/fc-amat-0.1.1/fc_bench-0.1.1/)
```

# 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: $v \in \mathbb{R}^n$ or $X$ a 1D-array. The first alphabetic characters are $\mathbb{A}\mathbb{B}\mathbb{C}\ldots$.
- All matrices or 2D-arrays are represented with the blackboard font as: $M \in \mathcal{M}_{m,n}(\mathbb{K})$ or $b$ a $m$-by-$n$ 2D-array. The first alphabetic characters are $\mathbb{A}\mathbb{B}\mathbb{C}\ldots$.
- All arrays of matrices or 3D-arrays or amat objects are represented with the bold blackboard font as: $M \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ or $b$ a $N$-by-$m$-by-$n$ 3D-array. The first alphabetic characters are $\mathbb{A}\mathbb{B}\mathbb{C}\ldots$.

We now introduce some notations. Let $A = (A_1, \ldots, A_N) \in (\mathcal{M}_{m,n}(\mathbb{K}))^N$ be a set of $m$-by-$n$ matrices. We identify $A$ 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 property of matrix if all its matrices have this property. For example, $A$ is a symmetrical amat object if all its matrices are symmetrical.

# 4 Constructor and generators

We give properties of the amat class:

## Properties of amat class

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr</td>
<td>number of rows</td>
</tr>
<tr>
<td>nc</td>
<td>number of columns</td>
</tr>
<tr>
<td>N</td>
<td>number of matrices (nr-by-nc)</td>
</tr>
<tr>
<td>values</td>
<td>N-by-nr-by-nc array which contains all the matrices</td>
</tr>
</tbody>
</table>
### 4.1 Constructor

**Syntaxe**

\[
X = \text{amat}(N, n, m) \\
X = \text{amat}(T) \\
X = \text{amat}(N, A) \\
X = \text{amat}(\ldots, \text{classname})
\]

**Description**

- **\(X = \text{amat}(N, n, m)\)** returns a \(N\)-by-\(n\)-by-\(m\) amat object where all its elements are set to 0.

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

- **\(X = \text{amat}(N, A)\)** 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 \(A\).

- **\(X = \text{amat}(\ldots, \text{classname})\)** returns an amat object with values of class \(\text{classname}\).

In Listing 6, some examples are provided.

#### Listing 6: amat constructors

```matlab
X = amat(100, 3, 4);  \ % X: 100-by-3-by-4 amat
info(X);              \ % amat object info
W = amat(X);          \ % W: 100-by-3-by-4 amat
info(W);              \ % amat object info
T = randn(200, 2, 3); \ % T: 200-by-2-by-3 array
Y = amat(T);          \ % Y: 200-by-2-by-3 amat
info(Y);              \ % amat object info
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(1)=
8 1 2 2
9 1 1 8
Z(2)=
8 1 2 2
9 1 1 8
...
Z(29)=
8 1 2 2
9 1 1 8
Z(30)=
8 1 2 2
9 1 1 8
```
4.2 Particular generators

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

- *fc amat.zeros*, generates an zero amat object,
- *fc amat.ones*, generates an amat object of one’s,
- *fc amat.eye*, generates an amat object of identity matrices.

4.2.1 fc amat.zeros function

Syntaxe

\[
X=\text{fc amat.zeros}(N,m,n) \\
X=\text{fc amat.zeros}([N,m,n]) \\
X=\text{fc amat.zeros}([N,d]) \\
X=\text{fc amat.zeros}(\ldots,\text{classname})
\]

Description

\[
X=\text{fc amat.zeros}(N,m,n) \quad \text{return an } N\text{-by-}m\text{-by-}n \text{ zero amat object.}
\]
\[
X=\text{fc amat.zeros}([N,m,n]) \quad \text{same as } X=\text{fc amat.zeros}(N,m,n)
\]
\[
X=\text{fc amat.zeros}(N,d) \quad \text{same as } X=\text{fc amat.zeros}(N,d,d)
\]
\[
X=\text{fc amat.zeros}(\ldots,\text{classname}) \quad \text{returns an amat object with values of class } \text{classname}
\]

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:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 20 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
  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.

```
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
```

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

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])
```

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
whos
disp('List current variables:')
whos
```

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

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
</tbody>
</table>

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
...
```

```
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 strictly 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

```matlab
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 amat
Z=fc amat.random.rand([50,2,3],'single');  % Z: 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)=
  0.11632  0.18182  0.12407
  0.22650  0.33749  0.41202
  0.28650  0.33749  0.41202
  matrix(2)=
  0.12918  0.09886  0.601748
  0.22184  0.30985  0.26891
  0.28650  0.33749  0.41202
  ...
  matrix(49)=
  0.03805  0.849642  0.06154
  0.28650  0.572586  0.39498
  matrix(50)=
  0.440444  0.995168  0.469574
  0.680158  0.042578  0.916317
```

### 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)`

---

17
\( \texttt{X=fc\_amat.random.randn(N,d)} \)

same as \( \texttt{X=fc\_amat.random.randn(N,d,d)} \)

\( \texttt{X=fc\_amat.random.randn(\ldots,classname)} \)

returns an amat object with values of class \( \texttt{classname} \). \( \texttt{classname} \) could be 'single' or 'double' (default).

In Listing 10, some examples are provided.

```matlab
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 amat
Z=fc_amat.random.randn([50,2,3],"single");
% Y : 50-by-2-by-3 single amat
\texttt{disp(’List\_current\_variables:\’)}
\texttt{disp(’Print\_Z\_amat\_object:\’)}
\texttt{Z}
```

Output

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

\begin{tabular}{cccc}
\textbf{Attr Name} & \textbf{Size} & \textbf{Bytes} & \textbf{Class} \\
\hline
\texttt{SaveOptions} & \texttt{1x6} & \texttt{25} & \texttt{cell} \\
\texttt{X} & \texttt{1x1} & \texttt{0} & \texttt{amat} \\
\texttt{Y} & \texttt{1x1} & \texttt{0} & \texttt{amat} \\
\texttt{Z} & \texttt{1x1} & \texttt{0} & \texttt{amat} \\
\end{tabular}
```

Total is 9 elements using 25 bytes

Print \( Z \) amat object:

\( Z = \)

is a 50x2x3 amat[single] object

\texttt{matrix(1)=}

\begin{verbatim}
1.71262 0.10902 -1.35594
-0.24406 -0.52086 -2.21049
\end{verbatim}

\texttt{matrix(2)=}

\begin{verbatim}
1.1177368 -0.0023704 0.0841556
-0.2556832 -1.2814443 0.1101335
\end{verbatim}

```
Output
```

4.3.3 \texttt{fc\_amat.random.randi} function

The function \texttt{fc\_amat.random.randi} return an amat object whose elements are random integers.

Syntaxe

\( \texttt{X=fc\_amat.random.randi(Imax,N,m,n)} \)
\( \texttt{X=fc\_amat.random.randi(Imax,[N,m,n])} \)
\( \texttt{X=fc\_amat.random.randi(Imax,N,d)} \)
\( \texttt{X=fc\_amat.random.randi([Imin,Imax],\ldots)} \)
\[
X = \text{fc amat.random.randi}(..., \text{classname})
\]

**Description**

- \(X = \text{fc amat.random.randi}(\text{Imax}, N,m,n)\)
  
  returns an \(N\)-by-\(m\)-by-\(n\) amat object containing pseudorandom integer values drawn from the discrete uniform distribution on \(1: \text{Imax}\).

- \(X = \text{fc amat.random.randi}(\text{Imax}, [N,m,n])\)
  
  same as \(X = \text{fc amat.random.randi}(\text{Imax}, N,m,n)\)

- \(X = \text{fc amat.random.randi}(\text{Imax}, N,d)\)
  
  same as \(X = \text{fc amat.random.randi}(\text{Imax}, N,d,d)\)

- \(X = \text{fc amat.random.randi}([\text{Imin,Imax}], ...)\)
  
  returns an amat object containing integer values drawn from the discrete uniform distribution on \(\text{Imin:Imax}\).

- \(X = \text{fc amat.random.randi}(..., \text{classname})\)
  
  returns an amat object with values of class \text{classname}. Accepted \text{classname} strings are those of the \text{randi} Matlab function. Default is ‘double’.

In Listing 10, some examples are provided.
4.3.4 \texttt{fc\_amat.random.randsym} function

The \texttt{fc\_amat.random.randsym} function return an \texttt{amat} object whose matrices are symmetric with random elements uniformly distributed on the interval \([0, 1]\).

**Syntaxe**

\[
X = \texttt{fc\_amat.random.randsym}(N, d) \\
X = \texttt{fc\_amat.random.randsym}(N, d, 'class', value)
\]

**Description**

\[
X = \texttt{fc\_amat.random.randsym}(N, d)
\]

return a \(N\)-by-\(d\)-by-\(d\) \texttt{amat} object whose matrices are symmetric with random elements uniformly distributed on the interval \([0, 1]\).

\[
X = \texttt{fc\_amat.random.randsym}(N, d, 'class', classname)
\]

returns an \texttt{amat} object with values of class \texttt{classname}. \texttt{classname} could be ‘\texttt{single}’ or ‘\texttt{double}’ (default).
In Listing 13, some examples are provided.

```matlab
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/uni2423current/uni2423variables/uni2423:')
whos

disp('Print/uni2423Y/uni2423amat/uni2423object/uni2423:')
Y
```

Output

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 8 elements using 26 bytes

Print Y amat object:

```matlab
Y =
```

is a 50x2x2 amat[single] object

---

### Syntaxe

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

### Description

```matlab
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.

```matlab
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/uni2423current/uni2423variables/uni2423:')
whos
disp('Print/uni2423Y/uni2423amat/uni2423object/uni2423:)
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)=
-0.047627 0.251909
0.251909 0.832787
matrix(2)=
0.767440 .82266
0.822660 .35401
...
matrix(49)=
0.21091 -1.38048
-1.38048 -0.76772
matrix(50)=
0.0086328 -1.4171486
-1.4171486 0.1385266
```

4.3.6 fc_amat.random.randisym function

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

Syntaxe

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

Description

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

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

22
pseudo random integer values are drawn from the discrete uniform distribution on \( \text{Imin:Imax} \)

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 \amat\ function usage

\begin{verbatim}
X=fc_amat.random.randisym([10,100,3]);
  Y=fomat.random.randisym([-5,5],100,2,'class','single');
  disp('Current/Variables:');
  disp('Print/Y/amat/object:');
  Y
\end{verbatim}

Output

\begin{verbatim}
List/Current/Variables:

<table>
<thead>
<tr>
<th>Attr</th>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 8 elements using 25 bytes

Print Y amat object:

Y =

is a 100x2x2 amat[single] object
matrix(1)=
  4  4
  4  3
matrix(2)=
 -1  1
  1  4
  ...
matrix(99)=
  4  4
  4  5
matrix(100)=
-1 -1

4.3.7 \amat\ function

The \amat\ 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
\( \texttt{X=fc\_amat.random.randher(N,d)} \)
\( \texttt{X=fc\_amat.random.randher(...,'class',value)} \)

**Description**

\( \texttt{X=fc\_amat.random.randher(N,d)} \)

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

\( \texttt{X=fc\_amat.random.randher(...,'class',classname)} \)

returns an amat object with values of class \( \text{classname} \). \( \text{classname} \) could be 'single' or 'double' (default).

In Listing 16, some examples are provided.

**Listing 16: examples of fc\_amat.random.randher function usage**

```matlab
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:')
disp('Print_Y amat object:')
```

**Output**

List current variables:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 8 elements using 25 bytes

Print Y amat object:

Y =

is a 50x2x2 amat[complex single] object

matrix(1)=
0.37703 + 0.89750i 0.04810 + 0.19992i
0.04810 - 0.19992i 0.22310 - 0.96485i

matrix(2)=
0.97920 - 0.40479i 0.35806 + 0.42714i
0.35806 - 0.42714i 0.64391 - 0.87017i

...

matrix(49)=
0.939823 - 0.061725i 0.046003 - 0.349185i
0.046003 + 0.349185i 0.000790 - 0.77446i

matrix(50)=
0.73612 + 0.76445i 0.84192 + 0.40467i
0.84192 - 0.40467i 0.05102 - 0.54484i

### 4.3.8 fc\_amat.random.randnher function

The \( \texttt{fc\_amat.random.randnher} \) function returns an amat object whose matrices are hermitian with normally distributed random real and imaginary part elements having zero mean and variance one.
Syntaxe

\[
X = \text{fc amat.random.randnher}(N,d) \\
X = \text{fc amat.random.randnher}(...,'class',value)
\]

Description

\[
X = \text{fc amat.random.randnher}(N,d)
\]

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

\[
X = \text{fc amat.random.randnher}(...,'class',classname)
\]

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 \text{fc amat.random.randnher} function usage

```matlab
X = \text{fc amat.random.randnher}(100, 3); 
\% X : 100-by-3-by-3 amat
Y = \text{fc amat.random.randnher}(50, 2, 'class', 'single'); 
\% Y : 50-by-2-by-2 single amat
disp('List current variables:')
disp('Print Y amat object:')
who
\text{disp(' amat object:')}
Y
```

**Output**

List current variables :

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 8 elements using 25 bytes

Print Y amat object :

\(Y = \)

is a 50x2x2 amat[complex single] object

matrix(1) =

\[-0.59442 - 0.42816i 0.73292 - 2.53702i \\
0.73292 + 2.53702i -0.46465 - 1.07502i\]

matrix(2) =

\[1.28017 - 0.11936i 0.01663 - 0.30064i \\
0.01663 + 0.30064i -1.12893 + 1.16056i\]

...

matrix(49) =

\[0.58809 - 2.40979i 0.02161 - 0.35857i \\
0.02161 + 0.35857i 0.38315 + 0.44575i\]

matrix(50) =

\[1.2362 + 1.1539i 0.0988 + 1.9253i \\
0.0988 - 1.9253i -1.1539 - 2.5880i\]

---

### 4.3.9 fc amat.random.randiher function

The \text{fc amat.random.randiher} function return an amat object whose matrices are Hermitian with random integers values.
Syntaxe

\[
X = \text{fc amat.random.randiher}(\text{Imax}, N, d) \\
X = \text{fc amat.random.randiher}([\text{Imin}, \text{Imax}], ...) \\
X = \text{fc amat.random.randiher}(..., 'class', \text{classname})
\]

Description

\[
X = \text{fc amat.random.randiher}(\text{Imax}, N, d)
\]

returns a \(N\)-by-\(d\)-by-\(d\) \text{amat} object whose matrices are Hermitian where real and imaginary part values are respectively drawn from the discrete uniform distribution on \(1:\text{Imax}\) and the discrete uniform distribution on \(1:\text{Imax}\) times a random sign.

\[
X = \text{fc amat.random.randiher}([\text{Imin, Imax}], ...)
\]
pseudorandom integer values are drawn from the discrete uniform distribution on \(\text{Imin:Imax}\)

\[
X = \text{fc amat.random.randiher}(..., 'class', \text{classname})
\]
returns an \text{amat} object with values of class \text{classname}. Accepted \text{classname} strings are those of the \text{randi Matlab function}. Default is ‘double’.

In Listing 18 some examples are provided.

**Listing 18**: examples of \text{fc amat.random.randiher} function usage

```matlab
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 amat object:'); Y
```

Output

X is a 100x3x3 amat[complex double] object

Y amat object:

```
% Y amat object:
```

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

4.3.10 \text{fc amat.random.randdiag} function

The \text{fc amat.random.randdiag} function return an \text{amat} object whose matrices are diagonal with non zeros elements drawn from the uniform distribution on the interval \([a, b]=[-0.1, 1]\).
Syntaxe

\[ X = \text{fc amat.random.randdiag}(N, d) \]
\[ X = \text{fc amat.random.randdiag}(..., \text{key}, \text{value}) \]

Description

\[ X = \text{fc amat.random.randdiag}(N, d) \]
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 \([a, b] = [0, 1]\).

\[ X = \text{fc amat.random.randdiag}(..., \text{key}, \text{value}) \]
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 \([a, b] = [0, 1]\). (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 \texttt{fc\_amat.random.randdiag} function usage

\begin{verbatim}
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(60,3,'class','single','k',1,'b',5);
\end{verbatim}

Output

\begin{verbatim}
X is a 100x3x3 amat\{double\} object
Y is a 200x3x3 amat\{complex double\} object
Print Z amat object:
Z is a 60x3x3 amat\{single\} object
Z(1)=
0.00000 4.41160 0.00000
0.00000 0.00000 0.87940
0.00000 0.00000 0.00000
Z(2)=
0.00000 0.30972 0.00000
0.00000 0.00000 2.46224
0.00000 0.00000 0.00000
...
Z(49)=
0.00000 2.22852 0.00000
0.00000 0.00000 4.26874
0.00000 0.00000 0.00000
Z(50)=
0.00000 3.97789 0.00000
0.00000 0.00000 2.56508
0.00000 0.00000 0.00000
\end{verbatim}

4.3.11 \texttt{fc\_amat.random.randndiag} function

The \texttt{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.

\textbf{Syntaxe}

\begin{verbatim}
X=fc_amat.random.randndiag(N,d)
X=fc_amat.random.randndiag(...,key,value)
\end{verbatim}

\textbf{Description}

\begin{verbatim}
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.
\end{verbatim}

\begin{verbatim}
X=fc_amat.random.randndiag(...,key,value)
Some optional key/value pairs arguments are available with keys:
\begin{itemize}
  \item \texttt{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)
\end{itemize}
\end{verbatim}
• '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.

```
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
5.89470 0.00000 0.00000
0.00000 4.16606 0.00000
Z(2)=
0.00000 0.00000 0.00000
2.79067 0.00000 0.00000
0.00000 4.73109 0.00000
...
Z(49)=
0.00000 0.00000 0.00000
3.61998 0.00000 0.00000
0.00000 3.57442 0.00000
Z(50)=
0.00000 0.00000 0.00000
2.68792 0.00000 0.00000
0.00000 2.22889 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

\[ X = \text{fc amat.random.randidiag(Imax,N,d)} \]

returns a \( N \)-by-\( d \)-by-\( d \) amat object whose matrices are diagonal and nonzeros elements are pseudorandom integer drawn from the discrete uniform distribution on \( 1: \text{Imax} \).

\[ X = \text{fc amat.random.randidiag([Imin,Imax],N,d)} \]

returns a \( N \)-by-\( d \)-by-\( d \) amat object whose matrices are diagonal and nonzeros elements are pseudorandom integer drawn from the discrete uniform distribution on \( \text{Imin:Imax} \).

\[ X = \text{fc amat.random.randidiag(...,key,value)} \]

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 \text{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 \texttt{fcamat.random.randdiag} function usage

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

disp(Z,'n',2)
\end{verbatim}

Output

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Attr Name & Size & Bytes & Class \\
\hline
SaveOptions & 1x6 & 25 & cell \\
X & 1x1 & 0 & \texttt{amat} \\
Y & 1x1 & 0 & \texttt{amat} \\
Z & 1x1 & 0 & \texttt{amat} \\
\hline
\end{tabular}
\end{table}

Total is 9 elements using 25 bytes

Print Z amat object:
\texttt{Z} is a 50x3x3 \texttt{amat}[\texttt{single}] object
\begin{verbatim}
Z(1)=
0 -4 0
0 0 0
0 0 0
\end{verbatim}
\begin{verbatim}
Z(2)=
0 0 0
0 0 5
0 0 0
\end{verbatim}
\begin{verbatim}
Z(3)=
0 0 5
0 0 0
\end{verbatim}
\begin{verbatim}
Z(4)=
0 1 0
0 0 1
0 0 0
\end{verbatim}
\begin{verbatim}
Z(5)=
0 -5 0
0 0 -5
0 0 0
\end{verbatim}

4.3.13 \texttt{fcamat.random.randtril} function

The \texttt{fcamat.random.randtril} function return an \texttt{amat} object whose matrices are lower triangular with non zeros elements drawn from the uniform distribution on the interval \([a, b[=0, 1].

Syntaxe

\begin{verbatim}
X=fc_amat.random.randtril(N,d)
X=fc_amat.random.randtril(...,key,value)
\end{verbatim}

Description

\texttt{X=fc_amat.random.randtril(N,d)}

returns a \(N\)-by-\(d\)-by-\(d\) \texttt{amat} object whose matrices are lower triangular with non zeros elements drawn from the uniform distribution on the interval \([a, b[=0, 1].
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 \times 3 \times 3 amat
Y=fc amat random randtril(200,3,'nc',4,'complex',true,'a',-1);
info(Y)
% Y : 200 \times 3 \times 4 amat
Z=fc amat random randtril(60,3,'class','single','k',1,'b',5);
% Z : 50 \times 3 \times 3 single amat
disp('Print 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)= 2.3534 4.6018 0.0000
3.6035 0.6812 3.8119
4.1786 4.7306 2.8629
Z(2)= 3.5722 0.1197 0.0000
4.9158 4.2029 4.6242
3.9697 2.6703 0.5788
...
Z(49)= 2.4959 0.3437 0.0000
2.9926 1.8096 2.1136
2.9556 2.9697 0.6512
Z(50)= 4.5002 1.4738 0.2804
```

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

\[
X = \text{fc amat.random.randntril}(N, d) \\
X = \text{fc amat.random.randntril}(..., key, value)
\]

Description

\[
X = \text{fc amat.random.randntril}(N, d)
\]

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.

\[
X = \text{fc amat.random.randntril}(..., key, value)
\]

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

```matlab
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 100-by-3-by-3 amat[double] object
Y is a 200-by-3-by-4 amat[complex double] object
Print Z amat object:
Z is a 50-by-3-by-3 amat[single] object
Z(1)=
 0.00000 0.00000 0.00000
 2.90558 0.00000 0.00000
 3.50448 5.76827 0.00000
Z(2)=
 0.00000 0.00000 0.00000
 3.42248 0.00000 0.00000
 3.41635 3.47000 0.00000
...
Z(49)=
 0.00000 0.00000 0.00000
 0.83085 0.00000 0.00000
 4.02537 5.14715 0.00000
Z(50)=
 0.00000 0.00000 0.00000
 6.20763 0.00000 0.00000
 5.63155 3.21133 0.00000
```

4.3.15 `fc_amat.random.randntril` function

The `fc_amat.random.randntril` function returns an amat object whose matrices are lower triangular and non-zeros elements are random integers.

**Syntax**

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

**Description**

- `X = fc_amat.random.randntril(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.randntril([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.

```matlab
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:
Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:
Z is a 50x3x3 amat[single] object
Z(1):
3 -5 0
-4 3 -5
0 -4 -5
Z(2):
5 5 0
3 3 -5
-2 3 -4
...
Z(49):
3 2 0
1 -2 4
-5 4 0
Z(50):
5 3 0
-1 4 0
-5 0 0
```

Output

List current variables:
Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:
Z is a 50x3x3 amat[single] object
Z(1):
3 -5 0
-4 3 -5
0 -4 -5
Z(2):
5 5 0
3 3 -5
-2 3 -4
...
Z(49):
3 2 0
1 -2 4
-5 4 0
Z(50):
5 3 0
-1 4 0
-5 0 0

35
4.3.16  \texttt{fc amat.random.randtriu} function

The \texttt{fc amat.random.randtriu} function return an \texttt{amat} object whose matrices are upper triangular with non zeros elements drawn from the uniform distribution on the interval \([a, b]=]0, 1].

\textbf{Syntaxe}

\begin{verbatim}
X=fc amat.random.randtriu(N,d)
X=fc amat.random.randtriu(...,key,value)
\end{verbatim}

\textbf{Description}

\begin{verbatim}
X=fc amat.random.randtriu(N,d)
\end{verbatim}
returns a \(N\)-by-\(d\)-by-\(d\) \texttt{amat} object whose matrices are diagonal with non zeros elements drawn from the uniform distribution on the interval \([a, b]=]0, 1].

\begin{verbatim}
X=fc amat.random.randtriu(...,key,value)
\end{verbatim}
Some optional key/value pairs arguments are available with keys:

- \texttt{'complex}, if value is \texttt{true} the \texttt{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 \texttt{false}, i.e. real \texttt{amat} object)
- \texttt{'class}, to set \texttt{amat} object data type; value could be \texttt{'single'} or \texttt{'double'} (default).
- \texttt{'nc}, number of columns of the matrices (default: \(d\))
- \texttt{'k}, offset of \(k\) diagonals above or below the main diagonal; above for positive \(k\) and below for negative \(k\).
- \texttt{'a}, to set \(a\) (lower bound of the interval) value (0 by default).
- \texttt{'b'}, to set \(b\) (upper bound of the interval) value (1 by default).

In Listing\textsuperscript{25} some examples are provided.
4.3.17  \texttt{fc\_amat.random.randntriu} function

The \texttt{fc\_amat.random.randntriu} function returns an \texttt{amat} object whose matrices are upper triangular with non-zeros elements drawn from the normal distribution having zero mean and unit standard deviation.

**Syntax**

\[
X = \text{fc\_amat.random.randntriu}(N,d) \\
X = \text{fc\_amat.random.randntriu}(\ldots, \text{key}, \text{value})
\]

**Description**

\[
X = \text{fc\_amat.random.randntriu}(N,d)
\]

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

\[
X = \text{fc\_amat.random.randntriu}(\ldots, \text{key}, \text{value})
\]

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

- 'complex', if value is \texttt{true} the \texttt{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 \texttt{false} i.e real \texttt{amat} object)
• 'class', to set amat object data type; value could be 'single' or 'double' (default).
• 'nc', number of columns of the matrices (default: d)
• 'k', offset of k diagonals above or below the main diagonal; above for positive k and below for negative k.
• 'mean', to set mean of the normal distribution (0 by default).
• 'sigma', to set standard deviation of the normal distribution (1 by default).

In Listing 26, some examples are provided.

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

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

Output

```
X is a 100x3x3 amat[double] object
Y is a 200x3x4 amat[complex double] object
Print Z amat object:
Z is a 50x3x3 amat[single] object
Z(1)=
  3.68445  3.69596  5.01343
  4.67820  4.15220  4.28478
  0.00000  3.19696  3.22938
Z(2)=
  4.91812  3.75382  4.26172
  4.95688  5.33169  3.08506
  0.00000  3.14698  4.73564
...
Z(49)=
  3.30760  3.90885  3.47341
  2.40127  5.59333  3.20737
  0.00000  3.96006  4.06464
Z(50)=
  5.08935  3.48016  4.58166
  4.27741  5.40893  4.76488
  0.00000  2.88230  4.86348
```

4.3.18 fc_amat.random.randntriu function

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

Syntaxe

```
X=fc_amat.random.randntriu(Imax,N,d)
X=fc_amat.random.randntriu([Imin,Imax],...)
X=fc_amat.random.randntriu(...,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 \( Imin:Imax \).

\[
X = fc\_amat\_random\_randitriu(\ldots, key, value)
\]
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

```matlab
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/uni2423current/uni2423variables/uni2423:')
whos
disp('Print/uni2423Z/uni2423amat/uni2423object/uni2423:')
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 -1
 0 0 0
 0 0 0
Z(2)=
 0 -3 -1
 0 0 -2
 0 0 0
   ...
Z(49)=
 0 3 -5
 0 0 1
 0 0 0
Z(50)=
 0 4 2
 0 0 0
 0 0 0
```

4.3.19 fc_amat.random.randsdd function

The `fc_amat.random.randsdd` function returns 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 \([a, b]=[0, 1]\).
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 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,uni2423current,uni2423variables,uni2423:')
disp(Z,'n',2)
```

Output

```
List current variables:

Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:

Z is a 50x3x3 amat[complex single] object

Z(1)=
-1.85628 - 1.80281i -0.94583 + 0.20185i 0.92199 - 0.55566i
-0.81580 - 0.20185i -2.60677 + 0.21298i 0.73502 + 0.83627i
0.53696 - 0.55948i -0.80205 + 0.27069i 0.74772 - 1.66913i

Z(2)=
1.32723 - 2.04260i 0.33750 + 0.29106i 0.59967 + 0.97000i
-0.72419 + 0.51464i 2.55980 + 0.59173i 0.15225 + 0.71241i
0.12219 + 0.25053i 0.14538 - 0.43840i 1.38329 - 0.35646i

... 

Z(49)=
-1.60668 + 1.91827i -0.43788 - 0.95819i 0.45215 + 0.89148i
-0.91632 - 0.45536i -0.01719 + 2.62895i -0.36519 + 0.52504i
-0.07924 + 0.45581i -0.28387 - 0.76538i 2.21512 - 0.44204i

Z(50)=
-0.59302 + 2.30869i 0.26381 + 0.62386i 0.80799 + 0.36992i
-0.97939 + 0.66681i 2.38443 - 1.10421i 0.06622 + 0.60132i
0.16618 + 0.38183i -0.86612 - 0.38654i -0.55452 + 2.04148i
```

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 stan-
dard 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.randisdd function usage

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

Output

```
List current variables:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables in the current scope:

- `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 50x3 amat[single] object

Z(1)=
-14.0707  3.0876  5.1594
 3.7519 -15.1119  4.4601
 5.2102  5.5365 -15.8501
Z(2)=
-17.9278  7.4604  4.9124
 5.4429 -14.1214  5.0393
 4.7808  4.4440 -12.6257
...
Z(49)=
-14.4591  4.2202  4.6114
 6.1097  14.8370  4.3985
 4.0389  6.6770  -16.3307
Z(50)=
-13.2340  6.6334  3.9592
 4.5351  -10.6589  4.7686
 4.2893  5.8596  15.8153
```

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

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

Description

`X=fc_amat.random.randisdd(Imax,N,d)` returns a `N`-by-`d`-by-`d` amat object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on `1:Imax`. 
\[
X = \text{fc amat.random.randisdd}([\text{Imin}, \text{Imax}], N, d)
\]
returns a \(N\)-by-\(d\) amat object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on \(\text{Imin:Imax}\).

\[
X = \text{fc amat.random.randisdd}(\ldots, \text{key, value})
\]
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.

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

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:
Z is a 50x3x3 amat[complex single] object

```

\[Z(1)\]  
-14 - 4i 1 - 4i 0 + 5i  
-4 - 3i 11 - 11i 5 - 3i  
-4 + 5i 3 + 5i 8 - 19i

\[Z(2)\]  
2 - 12i 2 + 5i -2 - 3i  
0 - 1i 9 + 7i -3 + 3i  
2 - 3i 5 + 5i -17 - 5i

\[Z(49)\]  
-6 - 13i 0 + 1i 2 + 5i  
0 + 4i -13 - 4i -4 - 4i  
-5 - 1i 1 - 0i 10 - 12i

\[Z(50)\]  
-8 - 13i -1 + 5i 0 + 1i  
-2 + 3i -2 - 2i 1 + 1i  
2 - 2i -1 + 0i -4 - 9i
```

44
4.3.22 \texttt{fc\_amat\_random\_randsympd} function

The \texttt{fc\_amat\_random\_randsympd} function returns an \texttt{amat} object whose matrices are symmetric positive definite. This object is generated by using \texttt{randsdd} function from \texttt{fc\_amat\_random} namespace.

Syntaxe

\begin{verbatim}
X=fc_amat.random.randsympd(N,d)
X=fc_amat.random.randsympd(...,key,value)
\end{verbatim}

Description

\begin{verbatim}
X=fc_amat.random.randsympd(N,d)
\end{verbatim}

returns a N-by-d-by-d \texttt{amat} object whose matrices are symmetric positive definite.

\begin{verbatim}
X=fc_amat.random.randsympd(...,key,value)
\end{verbatim}

Optional key/value pairs arguments are those of the \texttt{fc\_amat\_random\_randnsdd} function except for \texttt{'complex'} key which is forced to \texttt{false}. Keys can be:

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

```matlab
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:**

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
</tbody>
</table>

Total is 9 elements using 26 bytes

**Print Z amat object:**

Z is a 50x3x3 amat[single] object

```matlab
Z(1)=
  3.12824 -0.56898 -0.98676
 -0.56898  0.57577 -0.36037
-0.98676 -0.36037  2.74345
Z(2)=
  1.96795  2.13488 -1.02405
  2.13488  2.70007 -1.13869
-1.02405 -1.13869  0.93555
Z(49)=
  2.97042  0.28426  0.58873
  0.28426  1.78646  0.96416
  0.58873  0.96416  1.08743
Z(50)=
  2.6050 -1.4400  2.0404
  -1.4400  2.3544 -2.0398
  2.0404 -2.0398  6.2490
```

4.3.23 `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 `fc_amat.random.randnsdd` function.

**Syntaxe**

```matlab
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.
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.randnsypd` function usage

```matlab
X=fc_amat.random.randnsypd(100,3);
Y=fc_amat.random.randnsypd(200,3,'sigma',5);
Z=fc_amat.random.randnsypd(50,3,'class','single','mean',5);

disp('List current variables:

Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr</th>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td></td>
<td>Z</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:
Z is a 50x3x3 amat[single] object
Z(1)=
277.1917 6.3279 41.1924
6.3279 394.2693 -191.0545
41.1924 -191.0545 321.6099
Z(2)=
236.388 14.523 10.591
14.523 296.594 171.795
10.591 171.795 216.996
...
Z(49)=
260.159 165.958 15.695
165.958 304.895 21.491
15.695 21.491 307.055
Z(50)=
253.58 162.06 162.79
162.06 290.53 121.14
162.79 121.14 207.50
```

### 4.3.24 `fc_amat.random.randnisypd` function

The `fc_amat.random.randnisypd` function returns an `amat` object whose matrices are symmetric positive definite with random integers. This object is generated by using `randisypd` function from `fc_amat.random` namespace.
Syntaxe

\[
X = \text{fc amat.random.randisymdp}(\text{Imax}, N, d) \\
X = \text{fc amat.random.randisymdp}([\text{Imin}, \text{Imax}], ...) \\
X = \text{fc amat.random.randisymdp}(..., \text{key}, \text{value})
\]

Description

\[
X = \text{fc amat.random.randisymdp}(\text{Imax}, N, d)
\]
returns a \(N\)-by-\(d\)-by-\(d\) amat object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on \(1: \text{Imax}\).

\[
X = \text{fc amat.random.randisymdp}([\text{Imin}, \text{Imax}], N, d)
\]
returns a \(N\)-by-\(d\)-by-\(d\) amat object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudo random integer drawn from the discrete uniform distribution on \(\text{Imin}: \text{Imax}\).

\[
X = \text{fc amat.random.randisymdp}(..., \text{key}, \text{value})
\]
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

whos

disp('list current variables:')
whos

disp('print Z amat object:')
disp(Z,'n',2)

Output

List current variables:

Variables in the current scope:

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saveoptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes

Print Z amat object:

Z is a 50x3x3 amat [single] object

Z(1) =
65 27 9
27 90 18
9 18 162

Z(2) =
53 8 57
8 52 38
57 38 147

...

Z(49) =
116 -24 92
-24 68 12
92 12 201

Z(50) =
78 -22 76
-22 144 41
76 41 153

4.3.25 fc_amat.random.randherpd function

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

Syntaxe

\[
X=fc\_amat\_random\_randherpd(N,d)\\
X=fc\_amat\_random\_randherpd(\ldots,\text{key, value})
\]

Description

\[
X=fc\_amat\_random\_randherpd(N,d)
\]
returns a $N$-by-$d$-by-$d$ amat object whose matrices are symmetric positive definite.

49
Optional key/value pairs arguments are those of the `fc_amat.random.randherpd` 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.
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 randnadd 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 randnadd 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 [55], some examples are provided.
Listing 35: Examples of fc_amat.random.randnherpd function usage

```matlab
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-3 amat
Z = fc_amat.random.randnherpd(50, 3, 'class', 'single', 'mean', 5);
% Z: 50-by-3-by-3 single amat
disp('List current variables:');
disp('Print Z amat object:');
disp(Z, 'n', 2)
```

Output

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td>amat</td>
</tr>
<tr>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
</tbody>
</table>

Total is 9 elements using 26 bytes

Print Z amat object:

Z is a 50x3x3 amat[complex single] object

```
Z(1)=
 480.705 + 0.000i 200.505 - 28.135i -94.025 - 64.111i
 200.505 + 28.135i 626.745 + 0.000i 58.934 - 41.632i
-94.025 + 64.111i 58.934 + 41.632i 464.941 + 0.000i
Z(2)=
 579.026 + 0.000i 13.406 + 108.032i -144.922 + 64.164i
 13.406 + 108.032i 509.861 + 0.000i 116.951 + 20.770i
-144.922 + 64.164i 116.951 - 20.770i 509.386 + 0.000i
... 
Z(49)=
 587.53 + 0.001i -228.65 - 116.891i -227.52 - 54.24i
-228.65 + 116.891i 613.41 + 0.001i -251.63 + 183.40i
-227.52 + 54.24i -251.63 - 183.40i 643.36 + 0.000i
Z(50)=
 613.569 + 0.000i 209.766 - 65.099i 73.116 - 39.240i
 209.766 + 65.099i 513.685 + 0.000i -58.454 + 111.764i
 73.116 + 39.240i -58.454 - 111.764i 505.958 + 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.

\[ X = \text{fc amat.random.randiherpd}([\text{Imin}, \text{Imax}], N, d) \]

returns a \( N \times d \times d \) amat object whose matrices are strictly diagonally dominant and non-diagonal elements are pseudorandom integer drawn from the discrete uniform distribution on \( \text{Imin:Imax} \).

\[ X = \text{fc amat.random.randiherpd}(..., \text{key}, \text{value}) \]

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:');
disp('Print Z amat object:');
disp(Z, 'n', 2)
```

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

<table>
<thead>
<tr>
<th>Attr Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SaveOptions</td>
<td>1x6</td>
<td>25</td>
<td>cell</td>
</tr>
<tr>
<td>I</td>
<td>1x1</td>
<td>0 amat</td>
<td>amat</td>
</tr>
<tr>
<td>Y</td>
<td>1x1</td>
<td>0 amat</td>
<td>amat</td>
</tr>
<tr>
<td>Z</td>
<td>1x1</td>
<td>0 amat</td>
<td>amat</td>
</tr>
</tbody>
</table>

Total is 9 elements using 25 bytes
Print Z amat object:
2 is a 50x3x3 amat[complex single] object
Z(1)=
183 + 0i  141 - 52i -10 + 12i
141 + 52i  357 + 0i -54 + 74i
-10 - 12i -54 - 74i 262 + 0i
Z(2)=
395 + 0i -74 + 27i -63 + 16i
-74 + 27i  283 + 0i -8 + 40i
-63 - 16i -8 - 40i 202 + 0i
...
Z(49)=
242 + 0i  5 + 59i -31 - 187i
5 - 59i  142 + 0i -23 + 28i
-31 + 187i -23 - 28i 303 + 0i
Z(50)=
300 + 0i -105 - 43i -2 + 41i
-105 + 43i  216 + 0i 92 + 31i
-2 + 41i  92 - 31i 260 + 0i
```

53
Let $A$ be an $n$-by-$m$-by-$n$ amat object.

### 5.1.1 $A(K,I,J)$

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

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

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

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

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

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

The commands $A(K,I,:)$ and $A(K,I,1:end)$ are equivalent to $A(K,I,1:n)$.

The commands $A(K,:,:)$ and $A(K,,:)$ 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 $A(:,,:,:)$ and $A(1:end,1:end)$ 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 \texttt{subsref} method usage

\begin{verbatim}
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([K,p,q],*[1:R]);
I=fc_amat.random.randi(m,[K,p,q]);
J=fc_amat.random.randi(n,[K,p,q]);
sK=1:2:R;
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
\end{verbatim}

\textbf{Output}

\begin{tabular}{|l|}
\hline
\textbf{B} is a 2x2x2 amat\texttt{double} object \\
\textbf{D} is a 100x2x3 amat\texttt{double} object \\
\textbf{E} is a 100x1x1 amat\texttt{double} object \\
\textbf{F} is a 100x1x2 amat\texttt{double} object \\
\textbf{G} is a 50x2x2 amat\texttt{double} object \\
\textbf{H} is a 100x2x2 amat\texttt{double} object \\
List of some variables: \\
Variables in the current scope: \\
\hline
\textbf{Attr Name} & \textbf{Size} & \textbf{Bytes} & \textbf{Class} \\
\hline
A & 1x1 & 8 & double \\
C & 2x3 & 48 & double \\
sK & 1x50 & 24 & double \\
\hline
\textbf{Total is 57 elements using 80 bytes}
\end{tabular}

## 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
    \[
    A(k,i,j)=B, \quad \forall i \in I, \forall j \in J, \forall k \in K.
    \]

  - \( B \) is a \texttt{length(I)} \times \texttt{length(J)} matrix, then \( \forall k \in 1:\texttt{length(K)} \) the \( K(k) \)-th matrix of \( A \) is set to \( B \), i.e. \( \forall i \in 1:\texttt{length(I)}, \forall j \in 1:\texttt{length(J)}, \)
    \[
    A(K(k),I(i),J(j))=B(i,j).
    \]
- B is a \textit{length}(K)-by-\textit{length}(I)-by-\textit{length}(J) amat object then \\
  \( \forall k \in 1:\textit{length}(K) \) the \( K(k) \)-th matrix of A is set to \( k \)-th ma-
  trix of B, i.e. \( \forall i \in 1:\textit{length}(I) , \forall j \in 1:\textit{length}(J) , \)
  \( A(K(k),I(i),J(j))=B(k,i,j) \).

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

Then three cases are possible

- B is a scalar, then \( \forall i \in 1:p, \forall j \in 1:q, \forall k \in 1:M \)
  
  \( A(K(k,i,j),I(k,i,j),J(k,i,j))=B \)

- (\textit{experimental}) B is a M-by-p-by-q amat object then \( \forall i \in 1:p, \forall j \in 
  1:q, \forall k \in 1:M \)
  
  \( A(K(k,i,j),I(k,i,j),J(k,i,j))=B(k,i,j) \)

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

5.2.2 \( A(K)=B \)

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

5.2.3 \( A(I,J)=B \)

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

In Listing \( \text{38} \) some examples are provided.
6 Elementary operations

6.1 Arithmetic operations

The implemented element by element arithmetic operators/methods for \texttt{amat} objects are:

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

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

### Listing 38: examples of \texttt{subsasgn} method usage

```matlab
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 \( X \) \texttt{amat} object:

\[
X =
\begin{bmatrix}
\text{is a 100x3x3 amat [int32] object} \\
\text{matrix(1)=} \\
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & -1 & 1 \\
0 & 0 & 2 \\
\end{bmatrix} \\
\text{matrix(2)=} \\
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 2 \\
\end{bmatrix} \\
\text{...} \\
\text{matrix(99)=} \\
\begin{bmatrix}
1 & 1 & 0 \\
1 & 1 & 0 \\
1 & 1 & 99 \\
\end{bmatrix} \\
\text{matrix(100)=} \\
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 100 \\
\end{bmatrix}
\end{bmatrix}
\]
1. Let $\mathbb{B} \in (\mathcal{M}_{m,n}(K))^N$, we have

$$\mathbf{A} \otimes \mathbb{B} \overset{\text{def}}{=} \mathbb{C} \in (\mathcal{M}_{m,n}(K))^N$$

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

$$C_k(i, j) = A_k(i, j) \otimes B_k(i, j), \ \forall i \in [1, m], \forall j \in [1, n].$$

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

$$\mathbf{A} \otimes \mathbb{B} \overset{\text{def}}{=} \mathbb{C} \in (\mathcal{M}_{m,n}(K))^N$$

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

$$C_k(i, j) = A_k(i, j) \otimes B(i, j), \ \forall i \in [1, m], \forall j \in [1, n].$$

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

$$\mathbf{A} \otimes \mathbf{B} \overset{\text{def}}{=} \mathbb{C} \in (\mathcal{M}_{m,n}(K))^N$$

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

$$C_k(i, j) = A_k(i, j) \otimes B(k), \ \forall i \in [1, m], \forall j \in [1, n].$$

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

$$\mathbf{A} \otimes \mathbf{B} \overset{\text{def}}{=} \mathbb{C} \in (\mathcal{M}_{m,n}(K))^N$$

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

$$C_k(i, j) = A_k(i, j) \otimes B, \ \forall i \in [1, m], \forall j \in [1, n].$$

When $\mathbf{A}$ 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

```plaintext
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:

<table>
<thead>
<tr>
<th>Attr</th>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>2x3</td>
<td>48 double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>1x1</td>
<td>8 double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SaveOptions</td>
<td>1x6</td>
<td>25 cell</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>1x1</td>
<td>0 amat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>1x1</td>
<td>8 double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>1x1</td>
<td>8 double</td>
<td></td>
</tr>
</tbody>
</table>

Total is 20 elements using 97 bytes

Print D amat object:
D is a 100x2x3 amat[double] object
D(1)=
 -0.240228 -0.341488 -0.190240
 -0.323099 -0.469056 -0.073520
...
D(100)=
 -0.240228 -0.341488 -0.190240
 -0.323099 -0.469056 -0.073520

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:

- `&`, logical and
- `|`, logical or
- `~`, 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

\[
\begin{align*}
N=100; & \quad \text{n=2;=} \quad \text{m=3;} \\
X=( \text{fc amat random randi([-2,2],N,m,n)} & \geq 0 ) ; \\
y=( \text{randi([-2,2],n,n)} & < 0 ) ; \\
w=( \text{randi([-2,2],N,1)} & \leq 1 ) ; \\
A= X & \& y ; \\
info(A) & \\
B= X | v ; \\
info(B) & \\
C= \neg B ; \\
info(C) & \\
D= \text{xor}(X,C) ; \\
disp(D) & \\
\end{align*}
\]

Output

\[
\begin{align*}
A & \quad \text{is a 100x2x3 amat[logical] object} \\
B & \quad \text{is a 100x2x3 amat[logical] object} \\
C & \quad \text{is a 100x2x3 amat[logical] object} \\
D & \quad \text{is a 100x2x3 amat[logical] object} \\
\text{D(1)} & = \\
1 & 0 \\
1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 \\
\text{D(99)} & = \\
1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{align*}
\]

6.3.1 all method

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

Syntaxe

\[
\begin{align*}
\text{B}=\text{all}(X) \\
\text{B}=\text{all}(X, \text{dim})
\end{align*}
\]

Description

\[
\begin{align*}
\text{B}=\text{all}(X) \\
\text{B}=\text{all}(X, \text{dim})
\end{align*}
\]

\[
\begin{align*}
\text{B}=\text{all}(X) \\
\text{B}=\text{all}(X, \text{dim})
\end{align*}
\]

- \( \text{dim}=1 \), along rows of matrices of \( X \). Returns a \( N \)-by-\( 1 \)-by-\( n \) logical amat object such that \( B(k,1,j) \) is one (logical true) if \( \forall i \in \text{1:}\text{m}, \forall j \in \text{1:n}, A(k,i,j) \) is nonzero. Otherwise \( B(k,1,j) \) is zero (logical false).
$A(k,i,j)$ 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 $\forall j \in [1:n], A(k,i,j)$ is nonzero. Otherwise, $B(k,i,1)$ is zero (logical false).

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

- **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 $\forall k \in [1:N], A(k,i,j)$ is nonzero. Otherwise, $B(i,j)$ is zero (logical false).

In Listing 42 some examples are provided.

```
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('%i
%i
%i
')
disp(D)
E=all(all(X>0),0);
fprintf('%i
%i
%i
')
disp(E)
```

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>X is a 100x2x3 amat[double] object</td>
</tr>
<tr>
<td>A is a 100x1x1 amat[logical] object</td>
</tr>
<tr>
<td>B is a 100x1x3 amat[logical] object</td>
</tr>
<tr>
<td>C is a 100x2x1 amat[logical] object</td>
</tr>
<tr>
<td>D is</td>
</tr>
<tr>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
</tr>
<tr>
<td>E is</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

### 6.3.2 any method

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

**Syntaxe**

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

\[ B = \text{any}(X) \]

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

\[ B = \text{any}(X, \text{dim}) \]

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

In Listing 43, some examples are provided.

\[
\text{Listing 43: examples of } \text{fc amat random randbar} \text{ function usage}
\]

\[
X = \text{fc amat random randbar}(100,2,3);
\]

\[
\text{info}(X)
\]

\[
A = \text{any}(X>0); \text{info}(A)
\]

\[
B = \text{any}(X>0,1); \text{info}(B)
\]

\[
C = \text{any}(X>0,2); \text{info}(C)
\]

\[
D = \text{any}(X>0,0);
\]

\[
\text{fprintf}(\text{'}\text{D is/\n}\text{'}); \text{disp}(D)
\]

\[
E = \text{any(any(X>0),0)};
\]

\[
\text{fprintf}(\text{'}\text{E is/\n}\text{'}); \text{disp}(E)
\]

\[
\text{Output}
\]

\[
\begin{array}{l}
X \text{ is a 100x2x3 amat[double] object} \\
A \text{ is a 100x1x3 amat[logical] object} \\
B \text{ is a 100x1x3 amat[logical] object} \\
C \text{ is a 100x2x1 amat[logical] object} \\
D \text{ is 1 1 1} \\
E \text{ is 1 1}
\end{array}
\]

63
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
\begin{verbatim}
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
\end{verbatim}
Output
\begin{verbatim}
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
\end{verbatim}

7.1 trigonometric functions

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

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

7.3 Complex Arithmetic

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

7.4 Utility methods

- `ceil`, round toward positive infinity
- `fix`, round toward zero
- `floor`, round toward negative infinity
- `round`, Round to the nearest integer
7.4.1 \texttt{max} method

Let $X$ be a $n$-by-$m$-by-$n$ amat object. The \texttt{max} method of $X$ return its maximum values.

\textbf{Syntaxe}

\begin{verbatim}
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)
\end{verbatim}

\textbf{Description}

\begin{itemize}
\item \texttt{W=max(X)}
\end{itemize}

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

\begin{itemize}
\item \texttt{W = max (X, [], dim)}
\end{itemize}

- \texttt{dim=0}, along the number of matrices of $X$. Same as \texttt{W = max (X)}.
- \texttt{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 $X(k,:,j)$.
- \texttt{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 $X(k,i,:)$.
- \texttt{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 $X(k,:,:)$.

\begin{itemize}
\item \texttt{W = max (X, Y)}
\end{itemize}

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

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

\begin{itemize}
\item \texttt{[W, K] = max (X)}
\end{itemize}

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

- \texttt{W(i,j)=max(X(:,i,j))} and \texttt{W(i,j)=X(K(i,j),i,j)}
\[ [W, \text{Idx}] = \max (X, [], \text{DIM}) \]

- if DIM=0, command is equivalent to \([W, \text{Idx}] = \max (X)\).
- if DIM=1, returns two \(N\)-by-1-by-\(n\) amat objects such that
  \[ W(k,1,j) = \max(X(k,:,j)) \text{ and } W(k,1,j) = X(K,\text{Idx}(k,1,j),j) \],
- if DIM=2, returns two \(N\)-by-\(m\)-by-1 amat objects such that
  \[ W(k,i,1) = \max(X(k,i,:)) \text{ and } W(k,i,1) = X(K,i,\text{Idx}(k,i,1)) \].

\[ [W, I, J] = \max (X, [], 3) \]
returns three \(N\)-by-1-by-1 amat objects such that
\[ W(k,1,1) = \max(X(k,:,:)) \text{ and } W(k,1,1) = X(K,I(k,1,1),J(k,1,1)) \].

In Listing 45, some examples are provided.

```
Listing 45: examples of \textit{fc amat.random.randher} function usage
\begin{verbatim}
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)/uni2423->
')
disp(W)
W1=max(X,[],1);
fprintf('W1=max(X,[],1)/uni2423->/uni2423
')
disp(W1)
\end{verbatim}
```

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

7.4.2 \textit{min} method

Let \(X\) be a \(N\)-by-\(m\)-by-\(n\) amat object. The \texttt{min} method of \(X\) return its minimum values.

Syntaxe

\[ W = \min (X) \]
\[ W = \min (X, [], \text{DIM}) \]
\[ W = \min (X, Y) \]
\[ [W, I] = \min (X) \]
\[ [W, I] = \min (X, [], \text{DIM}) \]
\[ W, I, J = \text{min} (X, [], 3) \]

**Description**

\[ W = \text{min} (X) \]

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

\[ W = \text{min} (X, [], \text{dim}) \]

- \( \text{dim} = 0 \), along the number of matrices of \( X \). Same as \( W = \text{min} (X) \).
- \( \text{dim} = 1 \), 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 \( X(k,:,j) \).
- \( \text{dim} = 2 \), 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 \( X(k,i,:) \).
- \( \text{dim} = 3 \), 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 \( X(k,:,:) \).

\[ W = \text{min} (X, Y) \]

Returns a \( N \)-by-m-by-n amat object such that

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

\[ [W, K] = \text{min} (X) \]

Returns two m-by-n matrices such that

\( W(i,j) = \text{min}(X(:,i,j)) \) and \( W(i,j) = X(K(i,j),i,j) \)

\[ [W, \text{Idx}] = \text{min} (X, [], \text{DIM}) \]

- if \( \text{DIM} = 0 \), command is equivalent to \( [W, \text{Idx}] = \text{min} (X) \),
- if \( \text{DIM} = 1 \), returns two \( N \)-by-1-by-n amat objects such that
  \( W(k,1,j) = \text{min}(X(k,:,j)) \) and \( W(k,1,j) = X(K,\text{Idx}(k,1,j),j) \),
- if \( \text{DIM} = 2 \), returns two \( N \)-by-m-by-1 amat objects such that
  \( W(k,i,1) = \text{min}(X(k,i,:)) \) and \( W(k,i,1) = X(K,i,\text{Idx}(k,i,1)) \).
\[ [W, I, J] = \min(X, [], 3) \]

returns three \(N\)-by-1-by-1 amat objects such that

\[ W(k, 1, 1) = \min(X(k, :, :)) \quad \text{and} \quad W(k, 1, 1) = X(K, I(k, 1, 1), J(k, 1, 1)). \]

In Listing 46, some examples are provided.

**Listing 46:** Examples of fc amat random randbar function usage

```matlab
N=10;n=2;m=3;
X=fc_amat.random.randi([N,m,n]);
disp(X)
W=min(X);
fprintf('W=min(X)/uni2423->
')
disp(W)
W1=min(X,[],1);
fprintf('W1=min(X,[],1)/uni2423
')
disp(W1)
```

Output

\[
\begin{align*}
X(1) &= 1 \ 8 \ 6 \\
 & \quad 2 \ 5 \ 1 \\
X(2) &= 2 \ 1 \ 5 \\
 & \quad 6 \ 6 \ 6 \\
\cdots
\end{align*}
\]

\[
\begin{align*}
X(9) &= 3 \ 7 \ 1 \\
X(10) &= 7 \ 7 \ 9 \\
W &= \min(X) \rightarrow \\
 & 1 \ 1 \ 1 \\
W1 &= \min(X,[],1) \rightarrow \\
W1 &= \text{a 10x1x3 amat[double] object}
\end{align*}
\]

8 Linear algebra

8.1 Linear combination

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

\[
Z = \alpha \times X + \beta \times Y
\]  
(5)
where \( Z \) be also a \( N \)-by-\( m \)-by-\( n \) amat object, and we have \( \forall k \in 1:N, \forall i \in 1:m, \forall j \in 1:n \),

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

In Listing 47 some examples are provided.

<table>
<thead>
<tr>
<th>Listing 47: examples of linear combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N=100; \ m=2; \ n=3 ); \</td>
</tr>
<tr>
<td>( X=fc_amat_random_randi(9, [N, m, n]); \</td>
</tr>
<tr>
<td>( info(X) );</td>
</tr>
<tr>
<td>( info(Y) ); \</td>
</tr>
<tr>
<td>( A=3<em>X-2</em>Y; \</td>
</tr>
<tr>
<td>( info(A) );</td>
</tr>
<tr>
<td>( B=2<em>Y2-4</em>X; \</td>
</tr>
<tr>
<td>( info(B) ); \</td>
</tr>
<tr>
<td>( C=3*X-1; \</td>
</tr>
<tr>
<td>( info(C) );</td>
</tr>
<tr>
<td>( D=3*Y3-X; \</td>
</tr>
<tr>
<td>( info(D) );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X ) is a 100x2x3 amat[double] object \</td>
</tr>
<tr>
<td>( Y ) is a 100x2x3 amat[double] object \</td>
</tr>
<tr>
<td>( A ) is a 100x2x3 amat[double] object \</td>
</tr>
<tr>
<td>( B ) is a 100x2x3 amat[double] object \</td>
</tr>
<tr>
<td>( C ) is a 100x2x3 amat[double] object \</td>
</tr>
<tr>
<td>( D ) is a 100x2x3 amat[double] object \</td>
</tr>
</tbody>
</table>

### 8.2 Matrix product

We define (and extend) matricial products for amat objects by using operator \( \ast \) (i.e. {mtimes} method)

\[
Z = X \ast Y
\]

(6)

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

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

- With \( Y \) a \( N \)-by-\( m \)-by-\( p \) amat object (compatible dimensions), instruction \( \text{[6]} \) defines \( Z \) as a \( N \)-by-\( m \)-by-\( p \) amat object and is equivalent to the \( N \) matricial products

\[
Z(k) = X(k) \ast Y(k), \ \forall k \in 1:N
\]

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

\[
Z(k,i,j) = \sum_{r=1}^{n} X(k,i,r) \ast Y(k,r,j), \ \forall k \in 1:N.
\]
• With \( Y \) a \( n \)-by-\( p \) matrix (compatible dimensions), instruction \( Z \) defines 
\( Z \) as an \( n \)-by-\( m \)-by-\( p \) \texttt{amat} object and is equivalent to the \( n \) matricial 
products 
\[
Z(k) = X(k) \ast Y, \quad \forall k \in 1:N
\]
i.e. \( \forall i \in 1:m, \forall j \in 1:p, 
\]
\[
Z(k,i,j) = \sum_{r=1}^{n} X(k,i,r) \ast Y(r,j), \quad \forall k \in 1:N.
\]

• With \( Y \) a \( N \)-by-1 1D-array, instruction \( Z \) defines \( Z \) as a \( N \)-by-\( m \)-by-\( n \) 
amat object and we have 
\[
Z(k) = X(k) \ast Y(k), \quad \forall k \in 1:N
\]
i.e. \( \forall i \in 1:m, \forall j \in 1:n, 
\]
\[
Z(k,i,j) = X(k,i,j) \ast Y(k), \quad \forall k \in 1:N.
\]

• With \( Y \) a scalar, instruction \( Z \) defines \( Z \) as a \( N \)-by-\( m \)-by-\( n \) amat object and we have 
\[
Z(k) = X(k) \ast Y, \quad \forall k \in 1:N
\]
i.e. \( \forall i \in 1:m, \forall j \in 1:n, 
\]
\[
Z(k,i,j) = X(k,i,j) \ast Y, \quad \forall k \in 1:N.
\]

In Listing 48, some examples are provided.

Listing 48: examples of matricial products

```matlab
N=100;m=2;n=4;p=3;
X=fcamat.random.randi(9, [N,m,n]);
info(X)
Y=fcamat.random.randi(9, [N,n,p]);
info(Y)
A=X\ast Y; % < matricial products
info(A)
X2=randi(9, [m,n]);
B=X2\ast Y; % < matricial products
info(B)
Y2=randi(9, [n,p]);
C=X\ast Y2; % < matricial products
info(C)
T=C(1)-X(1)*Y2;
fprintf('T/uni2423is
')
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
```

71
8.2.1 Efficiency

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

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

$$\begin{array}{cc}
N & \text{mtimes} \\
200 000 & 0.550(s) \\
400 000 & 2.550(s) \\
600 000 & 3.673(s) \\
800 000 & 5.033(s) \\
1 000 000 & 6.224(s) \\
5 000 000 & 33.229(s) \\
10 000 000 & 67.099(s)
\end{array}$$

Table 2: Computational times in seconds of \texttt{mtimes}(X,Y) ($X \times Y$ matrix product) where $X$ and $Y$ are $N$-by-$d$-by-$d$ \texttt{amat} objects.

![Function mtimes, N-by-d-by-d amat with N=1000000](image)

Figure 1: Computational times in seconds of \texttt{mtimes}(X,Y) or $X \times Y$ (matrix product) where $X$ and $Y$ are $N$-by-$d$-by-$d$ \texttt{amat} objects.

8.2.2 Benchmark function

The function \texttt{fc amat.benchs.mtimes} measures performance of matricial products of \texttt{amat} objects done by \texttt{mtimes}(X,Y) or $X \times Y$ 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**

```
f amat.benchs.mtimes(LN)
f amat.benchs.mtimes(LN,key,value,...)
```

**Description**

`fc amat.benchs.mtimes(LN)`

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

`fc amat.benchs.mtimes(LN,key,value,...)`

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

- `'d'`, left and right matrices dimension (default value is `[2,2]`)
- `'type'`, to set type of left and right operands. `value` is either `'amat'` (`amat` object), `'mat'` (matrix), `'array1d'` ($N$-by-1 1D-array) or `'scalar'` (default value is `'amat'`).
- `'class'`, to set classname of left and right operands. Value can be `'double'` (default), `'single'`, `'int32'` , ...
- `'complex'`, if true left and right operands are complex (default value is `false`).
- `'ld'`, same as `'d'` but only for left operand.
- `'rd'`, same as `'d'` but only for right operand.
- `'ltype'` same as `'type'` but only for left operand.
- `'rtype'` same as `'type'` but only for right operand.
- `'lclass'` same as `'class'` but only for left operand.
- `'rclass'` same as `'class'` but only for right operand.
- `'lcomplex'` same as `'complex'` but only for left operand.
- `'rcomplex'` same as `'complex'` but only for right operand.

In Listings 49 and 50 two examples with outputs are provided.
Listing 49: Benchmarking mtimes(X,Y) with X a 3-by-4 matrix and Y a N-by-4-by-5 amat object

\[ \text{LN}=10^5 \times [2:2:10]; \]
\[ \text{fc amat.benchs.mtimes(LN,'ltype','mat','ld', [3,4], 'rd', [4,5]);} \]

Output

# computer: rhum-ubuntu18-04
# system: Ubuntu 18.04.3 LTS (x86_64)
# processor: Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.90GHz
# RAM: 62.9 Go
# software: Octave
# release: 5.1.0
#--------------------------
# list parameter is:
# -> matrix[double] with (m,n)=(3,4), size=[3 4]
# 2nd parameter is:
# -> amat[double] with (N,mr,nc)=(200000,4,5), size=[200000 4 5]
#--------------------------
#date:2020/01/02 01:04:57
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: N mtimes(s)

200000 0.734
400000 2.085
600000 3.084
800000 4.046
1000000 5.130

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

\[ \text{LN}=10^5 \times [2:2:10]; \]
\[ \text{fc amat.benchs.mtimes(LN,'d',[4,4], 'complex',true,'class','single', ... 'info',false);} \]

Output

#--------------------------
# list parameter is:
# -> amat[complex single] with (N,mr,nc)=(200000,4,4), size=[200000 4 4]
# 2nd parameter is:
# -> amat[complex single] with (N,mr,nc)=(200000,4,4), size=[200000 4 4]
#--------------------------
#date:2020/01/02 01:06:45
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: N mtimes(s)

200000 0.785
400000 2.779
600000 4.179
800000 5.456
1000000 6.875

8.3 LU Factorization

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

\[ P \cdot A = L \cdot U \quad \text{or} \quad A = P' \cdot L \cdot U. \quad (7) \]

Here, operator \(*\) is the amat matricial product, i.e.

\[ \forall k \in 1:N, \quad P(k) * A(k) = L(k) * U(k). \]

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

Let $A$ be a $N$-by-$m$ $amat$ object.

\[
[L,U,P] = lu(A) \\
[L,U,P] = lu(A, \text{type})
\]

Description

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

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

\[
P*\text{A}=L*U \text{ or } A=P'*L*U.
\]

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

\[
\forall k \in 1:N, \ P(k)A(k) = L(k)U(k).
\]

\[
[L,U,P] = lu(A, \text{type})
\]

- If $\text{type}$ is 'amat', then the command is equivalent to \( [L,U,P] = lu(A) \).
- If $\text{type}$ 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: \( \text{examples of } \text{lu} \text{ method usage} \)

\[
A = \text{complex}(\text{fc amat random.randn}(100,3,3), \text{fc amat random.randn}(100,3,3));
\]

\[
\text{info}(A);
\]

\[
[L, U, P] = \text{lu}(A);
\]

\[
\text{info}(L); \text{info}(U); \text{info}(P);
\]

\[
E = PA - L*U;
\]

\[
\text{disp}(E);
\]

**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

\[
\begin{align*}
E(1) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i & 0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 5.5511e-17i & 0.0000e+00 + 0.0000e+00i \\
-2.7756e-17 - 3.2960e-17i & 0.0000e+00 + 0.0000e+00i
\end{pmatrix} \\
\text{Column 3:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(2) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 5.5511e-17i \\
0.0000e+00 + 0.0000e+00i
\end{pmatrix} \\
\text{Column 3:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(99) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i
\end{pmatrix} \\
\text{Column 3:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(100) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 5.5511e-17i \\
2.2204e-16 + 0.0000e+00i \\
2.7756e-17 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(100) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
2.2204e-16 + 0.0000e+00i \\
2.7756e-17 + 0.0000e+00i \\
-5.5511e-17 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(100) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 5.5511e-17i \\
2.2204e-16 + 0.0000e+00i \\
1.1102e-16 - 1.1102e-16i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(100) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
-1.1102e-16 + 1.1102e-16i \\
2.7756e-17 + 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
E(100) &= \\
\text{Columns 1 and 2:} &
\begin{pmatrix}
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 0.0000e+00i \\
0.0000e+00 + 5.5511e-17i \\
2.7756e-17 - 0.0000e+00i
\end{pmatrix}
\end{align*}
\]

### 8.3.1 Efficiency

For benchmarking purpose the function \texttt{fc amat.benchs.lu} can be used and is described in Section \ref{section:bc.8.3.2}. This function uses the FC-BENCH Octave package described in \cite{fc-bench} and performs all computational times of this section.

Let \( A \) be a \( N \)-by-\( d \)-by-\( d \) amat object, in Table \ref{table:ett} computational times in seconds of \( [L, U, P] = \text{lu}(A) \) are given. In Figure \ref{fig:ett} computational times in seconds...
for a given $N$ are represented in function of very small values of $d$.

<table>
<thead>
<tr>
<th>N</th>
<th>$d=2$</th>
<th>$d=4$</th>
<th>$d=6$</th>
<th>$d=8$</th>
<th>$d=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 000</td>
<td>0.064(s)</td>
<td>0.414(s)</td>
<td>3.155(s)</td>
<td>9.916(s)</td>
<td>23.861(s)</td>
</tr>
<tr>
<td>400 000</td>
<td>0.133(s)</td>
<td>1.249(s)</td>
<td>6.765(s)</td>
<td>19.745(s)</td>
<td>47.441(s)</td>
</tr>
<tr>
<td>600 000</td>
<td>0.231(s)</td>
<td>2.289(s)</td>
<td>10.081(s)</td>
<td>29.568(s)</td>
<td>72.387(s)</td>
</tr>
<tr>
<td>800 000</td>
<td>0.313(s)</td>
<td>3.016(s)</td>
<td>13.682(s)</td>
<td>41.049(s)</td>
<td>98.218(s)</td>
</tr>
<tr>
<td>1 000 000</td>
<td>0.354(s)</td>
<td>4.036(s)</td>
<td>17.833(s)</td>
<td>52.070(s)</td>
<td>120.526(s)</td>
</tr>
</tbody>
</table>

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 $[L,U,P]=lu(A)$ where $A$ is a $N$-by-$d$-by-$d$ amat object.

### 8.3.2 Benchmark function

The function `fc amat benches 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**

```matlab
fc amat benches lu(LN)
fc amat benchmarks lu(LN,key,value,...)
```
Description

\texttt{fc amat.benchs.lu(LN)}

runs a benchmark of the \texttt{lu} method on a N-by-2-by-2 \texttt{amat} object for all N in LN.

\texttt{fc amat.benchs.lu(LN,key,value,...)}

Optional key/value pairs arguments are available and can modify the input N-by-d-by-d \texttt{amat} object of the \texttt{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 \texttt{amat} object. Value can be 'double' (default) or 'single'.
- 'complex', if true the input \texttt{amat} object is complex (default value is false).

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

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

\begin{verbatim}
LN=10^5*[2:2:10];
fc amat.benchs.lu(LN,'d',4);
\end{verbatim}
Listing 53: Benchmarking \( [L,U,P]=\text{lu}(A) \) where \( A \) is \( N \times 3 \times 3 \) amat object with complex single values.

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

Output

<table>
<thead>
<tr>
<th>LN</th>
<th>lu(s)</th>
<th>Error[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000</td>
<td>0.273</td>
<td>1.041e-06</td>
</tr>
<tr>
<td>400000</td>
<td>0.607</td>
<td>9.441e-07</td>
</tr>
<tr>
<td>600000</td>
<td>1.082</td>
<td>1.103e-06</td>
</tr>
<tr>
<td>800000</td>
<td>1.557</td>
<td>1.027e-06</td>
</tr>
<tr>
<td>1000000</td>
<td>1.956</td>
<td>1.073e-06</td>
</tr>
</tbody>
</table>

8.4 Cholesky Factorization

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

**Syntaxe**

Let \( A \) be a \( N \times N \times N \) symmetric (or hermitian) positive definite amat object.

- \( B = \text{chol}(A) \)
- \( B = \text{chol}(A, \text{type}) \)

**Description**

- \( B = \text{chol}(A) \)

  returns the positive Cholesky factorization of \( A \) as an \( N \times N \times N \) upper triangular amat object \( B \) with strictly positive diagonal entries such that

  \[
  A = B' \cdot B \tag{9}
  \]

  Here, operator \( * \) is the amat matricial product, i.e.

  \[
  \forall k \in 1:N, \quad A(k) = B(k)' \cdot B(k).
  \]

- \( B = \text{chol}(A, \text{type}) \)

  - If \( \text{type} \) is ‘upper’, then the command is equivalent to \( B = \text{chol}(A) \).
If type is 'lower', then \( B \) is a \( N \)-by-\( d \)-by-\( d \) lower triangular amat object with strictly positive diagonal entries such that

\[ A = B \cdot B' \]  

(10)

Here, operator \( \cdot \) is the amat matricial product, i.e.

\( \forall k \in 1:N, \quad A(k) = B(k) \cdot B(k)' \).

In Listing 54, some examples are provided.

Listing 54: examples of chol method usage

```matlab
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)=
 0 + 0i 0 + 0i 0 + 0i
 0 + 0i 0 + 0i 0 + 0i
 0 + 0i 0 + 0i 0 + 0i
E(2)=
 0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
 0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
 0.00000 + 0.00000i 0.00000 + 0.00000i 0.00000 + 0.00000i
...
E(99)=
Columns 1 and 2:
-1.7764e-15 + 0.0000e+00i 0.0000e+00 - 2.2204e-16i
 0.0000e+00 + 2.2204e-16i 0.0000e+00 + 0.0000e+00i
 0.0000e+00 + 0.0000e+00i 4.4409e-16 + 0.0000e+00i
Column 3:
 0.0000e+00 + 0.0000e+00i
 4.4409e-16 + 0.0000e+00i
 0.0000e+00 + 0.0000e+00i
E(100)=
Columns 1 and 2:
 1.7764e-15 + 0.0000e+00i -8.8818e-16 + 0.0000e+00i
-8.8818e-16 + 0.0000e+00i -3.5527e-15 + 0.0000e+00i
 0.0000e+00 + 0.0000e+00i 0.0000e+00 + 8.8818e-16i
Column 3:
 0.0000e+00 + 0.0000e+00i
 0.0000e+00 - 8.8818e-16i
 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 \( B = \text{chol}(A) \) are given. In Figure 3 computational times in seconds for a given \( N \) are represented in fonction of very small values of \( d \).
<table>
<thead>
<tr>
<th>N</th>
<th>d=2</th>
<th>d=4</th>
<th>d=6</th>
<th>d=8</th>
<th>d=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 000</td>
<td>0.008(s)</td>
<td>0.026(s)</td>
<td>0.097(s)</td>
<td>0.182(s)</td>
<td>0.300(s)</td>
</tr>
<tr>
<td>400 000</td>
<td>0.014(s)</td>
<td>0.071(s)</td>
<td>0.196(s)</td>
<td>0.371(s)</td>
<td>0.616(s)</td>
</tr>
<tr>
<td>600 000</td>
<td>0.021(s)</td>
<td>0.111(s)</td>
<td>0.277(s)</td>
<td>0.574(s)</td>
<td>0.973(s)</td>
</tr>
<tr>
<td>800 000</td>
<td>0.029(s)</td>
<td>0.151(s)</td>
<td>0.407(s)</td>
<td>0.751(s)</td>
<td>1.315(s)</td>
</tr>
<tr>
<td>1 000 000</td>
<td>0.036(s)</td>
<td>0.211(s)</td>
<td>0.465(s)</td>
<td>0.894(s)</td>
<td>1.674(s)</td>
</tr>
</tbody>
</table>

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

Figure 3: Computational times in seconds of $B=\text{chol}(A)$ 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 \( B = \text{chol}(A) \) 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)
fcc_amat.benchs.chol(LN,key,value,...)
```

**Description**

- `fc_amat.benchs.chol(LN)`
  runs a benchmark of the \texttt{chol} method on a \( N \)-by-\( d \)-by-\( d \) 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 \texttt{chol} function. \texttt{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 \texttt{value} is `'lower'`, then the output is a lower triangular amat object with strictly positive diagonal entries. Default \texttt{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 \texttt{true} the input amat object is Hermitian positive definite (default value is \texttt{false}).

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

The \texttt{det(A)} command returns determinants of the matrices of the square \texttt{amat} object. Explanations on programming techniques can be found in \cite{1}.

\textbf{Syntaxe} \ Let \( \mathbf{A} \) be a \( N \)-by-\( d \)-by-\( d \) \texttt{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

∀k ∈ 1:N, D(k)=det(A(k)).

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

when value is

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

In Listing 57, some examples are provided.

```matlab
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)= 1.8310e-15
E(2)= 0
E(3)=
E(99)= 4.4409e-16
E(100)= 9.9301e-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 an $N$-by-$d$-by-$d$ amat object, in Table 5 computational times in seconds of $B=\det(A)$ are given. In Figure 4 computational times in seconds for a given $N$ are represented in function of very small values of $d$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>d=2</th>
<th>d=4</th>
<th>d=6</th>
<th>d=8</th>
<th>d=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 000</td>
<td>0.074(s)</td>
<td>0.453(s)</td>
<td>3.538(s)</td>
<td>10.015(s)</td>
<td>25.434(s)</td>
</tr>
<tr>
<td>400 000</td>
<td>0.178(s)</td>
<td>1.286(s)</td>
<td>7.115(s)</td>
<td>20.968(s)</td>
<td>50.041(s)</td>
</tr>
<tr>
<td>600 000</td>
<td>0.262(s)</td>
<td>2.350(s)</td>
<td>10.161(s)</td>
<td>30.395(s)</td>
<td>74.450(s)</td>
</tr>
<tr>
<td>800 000</td>
<td>0.373(s)</td>
<td>3.048(s)</td>
<td>13.220(s)</td>
<td>38.759(s)</td>
<td>100.856(s)</td>
</tr>
<tr>
<td>1 000 000</td>
<td>0.391(s)</td>
<td>4.204(s)</td>
<td>16.959(s)</td>
<td>50.394(s)</td>
<td>118.775(s)</td>
</tr>
</tbody>
</table>

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 $B=\det(A)$ where $A$ is a $N$-by-$d$-by-$d$ amat object.

8.5.2 Benchmark function

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

Syntaxe

```
fc\_amat\_benchs\_det(LN)
fc\_amat\_benchs\_det(LN,key,value,...)
```
Description

```
fc_amat.benchs.det(LN)
```

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

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

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

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

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

---

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

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

```
#-----------------------------
# computer: rhum-ubuntu18-04
# system: Ubuntu 18.04.3 LTS (x86_64)
# processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
# (2 proc/6 cores by proc/2 threads by core)
# RAM: 62.9 Go
# software: Octave
# release: 5.1.0
#-----------------------------
# input parameter for N=200000 is:
# -> amat[double] with (N,nr,nc)=(200000,4,4), size=[200000 4 4]
#-----------------------------
# Benchmarking command: @(A) det(A,'select','lu');
#-----------------------------
# date:2020/01/02 09:50:44
#nbruns:5
#numpy: i4 f4
#format: %d %.3f
#labels: N det(s)
<table>
<thead>
<tr>
<th>N</th>
<th>det(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000</td>
<td>0.474</td>
</tr>
<tr>
<td>400000</td>
<td>0.476</td>
</tr>
<tr>
<td>600000</td>
<td>1.524</td>
</tr>
<tr>
<td>800000</td>
<td>2.332</td>
</tr>
<tr>
<td>1000000</td>
<td>4.067</td>
</tr>
</tbody>
</table>
```
### 8.6 Solving particular linear systems

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

- $X = \text{solvetriu}(A,B)$, if $A$ is an upper triangular amat object.
- $X = \text{solvetril}(A,B)$, if $A$ is a lower triangular amat object.
- $X = \text{solvediag}(A,B)$, if $A$ is a diagonal amat object.

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

The $X = \text{solvetriu}(A,B)$ command returns solutions of the linear systems $A\times X = B$ where $A$ is a regular upper triangular amat object. If $A$ is not upper triangular, then $X$ is solution of $\text{triu}(A)\times X = B$.

**Description**

$X = \text{solvetriu}(A,B)$

The input $A$ supposes to be a $N$-by-$d$-by-$n$ 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 output $X$ is the $N$-by-$d$-by-$n$ amat object such that

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

In Listing 60, some examples are provided.
Listing 60: examples of solve triu method usage

\begin{verbatim}
N=100; d=3;
A=fc amat.random.randtriu(N,d);
info(A)
B1=fc amat.random.randn(N,d,4);
info(B1)
X1=solve triu(A,B1);
info(X1)
fprintf('Max.\_of\_errors\_in\_Inf.\_norm:\_%.4e\n',max(norm(A*X1-B1)))
B2=randn(d,1);
X2=solve triu(A,B2);
info(X2)
E2=A*X2-B2;
disp(E2)
\end{verbatim}

Output

| A is a 100x3x3 amat\{double\} object |
| B1 is a 100x3x4 amat\{double\} object |
| E2(1)= 2.2204e-16 |
| E2(2)= 0 |
| E2(3)= 0 |
| E2(99)= 0 |
| E2(100)= 0 |

8.6.1 Efficiency

For benchmarking purpose the function \texttt{fc amat.benchs.solve triu} can be used and is described in Section 8.6.2. This function uses the \texttt{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 \( X=solve triu(A,B) \) are given. In Figure 5 computational times in seconds for a given \( N \) are represented in function of very small values of \( d \).

\begin{table}[h]
\centering
\begin{tabular}{ccc}
\hline
\textbf{N} & \textbf{solve triu} & \textbf{Error} \\
\hline
200 000 & 0.022(a) & 7.1050e-15 \\
400 000 & 0.042(a) & 9.3260e-15 \\
600 000 & 0.064(a) & 6.6680e-15 \\
800 000 & 0.093(a) & 1.2920e-14 \\
1 000 000 & 0.131(a) & 1.2490e-14 \\
5 000 000 & 1.150(a) & 1.5540e-14 \\
10 000 000 & 2.223(a) & 1.4650e-14 \\
\hline
\end{tabular}
\caption{Computational times in seconds of \( X=solve triu(A,B) \) where \( A \) is a \( N \)-by-\( d \)-by-\( d \) amat object and \( B \) is a \( N \)-by-\( d \)-by-1 amat object with \( d=4 \).}
\end{table}

88
Figure 5: Computational times in seconds of of \( X = \text{solvetriu}(A, B) \) where \( A \) is a \( N \)-by-\( d \)-by-\( d \) amat object and \( B \) is a \( N \)-by-\( d \)-by-1 amat object.

### 8.6.2 Benchmark function

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

**Syntaxe**

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

**Description**

- `fc_amat.benchs.solvetriu(LN)`
  - runs a benchmark of the \( X = \text{solvetriu}(A, B) \) 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 \text{solvetriu} function. \textit{key} can be one of the following strings:
    - `'d'`, to set \( d \) (default value is 2)
• `n`, to set n (default value is 1)
• `rhstype`, to set the kind of B: `amat` (default) for amat object and `mat` for matrix
• `class`, to set classname of the two inputs. Value can be `double` (default) or `single`.
• `complex`, if true the inputs are complex (default value is false).
• `a`, to set the lower bound of the interval of the uniform distribution used to generate input data (default value is 0.5).
• `b`, to set b the upper bound of the interval of the uniform distribution used to generate input data (default value is 2).

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

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

```
LN=10^5*[2:2:10];
f6.amat.benchs.solvetriu(LN,'d',4,'n',5, 'rhstype','mat);
```

```
#------------------------------------------------------------------------------------------
# computer: rhum-ubuntu18-04
# system: Ubuntu 18.04.3 LTS (x86_64)
# processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
# (2 procs/6 cores by proc/2 threads by core)
# RAM: 62.9 Go
# software: Octave
# release: 5.1.0
#------------------------------------------------------------------------------------------
# 1st parameter is :
# -> amat[double] with (N,m,n)=(N,4,4)
# containing upper triangular matrices
# 2nd parameter is :
# -> matrix[double] with (m,n)=(4,5), size=[4 5]
# Error function: @(X)max(norm(A*X-B))
#------------------------------------------------------------------------------------------
# Benchmarking command: @(A,B) solvetriu(A,B);
#------------------------------------------------------------------------------------------
#date:2020/01/02 09:57:16
#nbruns:5
#numpy: i4 f4 f4
#format: %d %.3f %.3e
#labels: N solvetriu(s) Error[0]
#   200000 0.248 1.109e-14
#   400000 0.898 9.881e-15
#   600000 1.611 7.515e-15
#   800000 2.086 1.442e-14
# 1000000 2.647 1.028e-14
#------------------------------------------------------------------------------------------
```
Listing 62: Benchmarking \( X = \text{solvetriu}(A,B) \) 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.

\[
\text{LS} = 10^5 \times [2:2:10];
\]

\[ \text{fc amat.benchs.solvetriu(\text{LS},’d’,3,’complex’,true,’class’,’single’,...’info’,false}); \]

---

### Output

---

# Benchmarking command: \( \text{solvetriu}(A,B) \);

---

<table>
<thead>
<tr>
<th>Parameter 1</th>
<th>Parameter 2</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) amat [complex single] with ( (N,m,n) = (N,3,3) ) containing upper triangular matrices</td>
<td>( B ) amat [complex single] with ( (N,mr,nc) = (200000,3,1) ), size = ( [200000 \ 3 \ 1] )</td>
<td>[200000 \ 0.040 \ 3.459 \times 10^{-6}]</td>
</tr>
<tr>
<td>[400000 \ 0.085 \ 2.149 \times 10^{-6}]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[600000 \ 0.133 \ 2.431 \times 10^{-6}]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[800000 \ 0.210 \ 3.446 \times 10^{-6}]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1000000 \ 0.246 \ 2.779 \times 10^{-6}]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 8.7 Solving linear systems

The \( X = \text{mldivide}(A,B) \) or \( X = A \backslash B \) commands return solutions of the linear systems \( A \times X = B \) where \( A \) is a regular \ amat \ object. Explanations on programming techniques can be found in [II].

### Description

\[
X = \text{mldivide}(A,B) \text{ or } X = A \backslash B
\]

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 output \( X \) is the \( N \)-by-\( d \)-by-\( n \) \ amat \ object such that

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

In Listing 63, some examples are provided.
Listing 63: examples of mldivide method operator usage

```matlab
N=100; d=3;
A=fc amat.random.randn(N,d);
info(A)
B1=fc amat.random.randn(N,d,4);
info(B1)
X1=mldivide(A,B1); % using function
info(X1)
fprintf('Max./uni2423of/uni2423errors/uni2423in/uni2423Inf./uni2423norm:/uni2423%.4e
',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: 6.1062e-15 |
| E2 is a 100x3x1 amat[double] object |
| E2(1)= |
| -2.7756e-17 |
| 5.5511e-17 |
| 0.0000e+00 |
| E2(2)= |
| -2.7756e-17 |
| -2.2204e-16 |
| 0.0000e+00 |
| ...
| E2(99)= |
| -2.7756e-17 |
| 0.0000e+00 |
| 1.1102e-16 |
| E2(100)= |
| 0.0000e+00 |
| 1.1102e-16 |
| 0.0040e+00 |

8.7.1 Efficiency

For benchmarking purpose the function fc amat.benchs.mldivide can be used and is described in Section 8.7.2. This function uses the FC-BENCH Octave package described in [2] and performs all computational times of this section.

Let A be a N-by-d-by-d regular triangular upper amat object and B be a N-by-d-by-1 amat object. In Table 7 computational times in seconds of X=mldivide(A,B) are given. In Figure 6 computational times in seconds for a given N are represented in function of very small values of d.

<table>
<thead>
<tr>
<th>N</th>
<th>d=2</th>
<th>d=4</th>
<th>d=6</th>
<th>d=8</th>
<th>d=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000</td>
<td>0.115(s)</td>
<td>0.599(s)</td>
<td>3.298(s)</td>
<td>11.065(s)</td>
<td>25.845(s)</td>
</tr>
<tr>
<td>400000</td>
<td>0.214(s)</td>
<td>1.366(s)</td>
<td>7.518(s)</td>
<td>21.789(s)</td>
<td>51.753(s)</td>
</tr>
<tr>
<td>600000</td>
<td>0.337(s)</td>
<td>2.788(s)</td>
<td>11.338(s)</td>
<td>32.855(s)</td>
<td>70.990(s)</td>
</tr>
<tr>
<td>800000</td>
<td>0.442(s)</td>
<td>3.661(s)</td>
<td>15.426(s)</td>
<td>43.183(s)</td>
<td>104.000(s)</td>
</tr>
<tr>
<td>1000000</td>
<td>0.545(s)</td>
<td>4.744(s)</td>
<td>18.041(s)</td>
<td>52.548(s)</td>
<td>133.478(s)</td>
</tr>
</tbody>
</table>

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 \( X = \text{mldivide}(A,B) \) 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 \( X = \text{mldivide}(A,B) \) 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 \( X = \text{mldivide}(A,B) \) 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.

```
f6 amat.benchs.mldivide(LN,'d',4,'n',5, 'rhstype','mat');
```

---

```
#-----------------------------
# computer: rhum-ubuntu18-04
# system: Ubuntu 18.04.3 LTS (x86_64)
# processor: Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz
# (2 procs/6 cores by proc/2 threads by core)
# RAM: 62.9 Go
# software: Octave
# release: 5.1.0
#-----------------------------
# 1st parameter is :
# -> amat[d] with (N,m,n)=(N,4,4)
# containing strictly diagonally dominant matrices
# 2nd parameter is :
# -> matrix[d] with (m,n)=(4,5), size=[4 5]
# Error function: @(X)max(norm(A*X-B))
#-----------------------------
# date:2020/01/02 12:12:01
#nruns:5
#nmpy: i4 f4 f4
#format: %d %.3f %.3e
#labels: X mldivide(s) Error[0]
#   20000  2.638  2.871e-14
#   40000  6.516  5.437e-14
#   80000 11.852  3.257e-14
#  100000 20.377  3.708e-14
```

---

94
Listing 65: Benchmarking \(x = \text{mldivide}(A, B)\) where \(A\) is \(N\)-by-\(3\)-by-\(3\) \text{amat} object and \(B\) is \(N\)-by-\(3\)-by-1 \text{amat} object with both complex single values.

\[
\text{LN} = 10^5 \times [2:2:10];
\]
\[
\text{fc amat.benchs.mldivide}([\text{LN}, 'd', 3, 'complex', true, 'class', 'single', ...
\text{'info',false});
\]

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,mr,nc)=(200000,3,1), size=[200000 3 1]
# Error function: @(X)max(norm(A*X-B))
#---------------------------------------------------------------
# date:2020/01/02 12:19:19
# nbruns:5
# comp: i4 f4 f4
# format: %d %.3f %.3e
# labels: N mldivide(s) Error[0]
#-------------------------------------------------------------
200000 0.461 3.831e-06
400000 1.015 2.186e-06
600000 1.670 2.651e-06
800000 2.332 3.002e-06
1000000 2.950 2.159e-06
```

8 References
