

Octave C++ Classes

Edition 1.0 for Octave version [No value for "VERSION"]
September 1993

John W. Eaton

Copyright © 1996, 1997 John W. Eaton.

This is the first edition of the documentation for Octave's C++ classes, and is consistent with version [No value for "VERSION"] of Octave.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

Table of Contents

1	Acknowledgements	1
	Contributors to Octave	1
	GNU GENERAL PUBLIC LICENSE	2
	Preamble	2
	TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	3
	Appendix: How to Apply These Terms to Your New Programs ..	7
2	A Brief Introduction to Octave	9
3	Arrays	10
	3.1 Constructors and Assignment	10
4	Matrix and Vector Operations	15
5	Matrix Factorizations	32
6	Ranges	37
7	Nonlinear Functions	38
8	Nonlinear Equations	39
9	Optimization	40
	9.1 Objective Functions	40
	9.2 Bounds	40
	9.3 Linear Constraints	41
	9.4 Nonlinear Constraints	41
	9.5 Quadratic Programming	42
	9.6 Nonlinear Programming	42
10	Quadrature	44
	10.1 Collocation Weights	45
11	Ordinary Differential Equations	46
12	Differential Algebraic Equations	47

13	Error Handling	48
14	Installation	49
15	Bugs	50
	Concept Index	51
	Function Index	52

1 Acknowledgements

Contributors to Octave

In addition to John W. Eaton, several people have written parts of liboctave. (This has been removed because it is the same as what is in the Octave manual.)

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions

for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does.

Copyright (C) yyyy *name of author*

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA. ■

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy *name of author*

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. ■

This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

2 A Brief Introduction to Octave

This manual documents how to run, install and port Octave's C++ classes, and how to report bugs.

3 Arrays

3.1 Constructors and Assignment

Array<T> (void)	Constructor
Create an array with no elements.	
Array<T> (int <i>n</i> [, const T &val])	Constructor
Create an array with <i>n</i> elements. If the optional argument <i>val</i> is supplied, the elements are initialized to <i>val</i> ; otherwise, they are left uninitialized. If <i>n</i> is less than zero, the current error handler is invoked (see Chapter 13 [Error Handling] , page 48).	
Array<T> (const Array<T> &a)	Constructor
Create a copy of the Array<T> object <i>a</i> . Memory for the Array<T> class is managed using a reference counting scheme, so the cost of this operation is independent of the size of the array.	
Array<T>& operator = (const Array<T> &a)	Assignment on Array<T>
Assignment operator. Memory for the Array<T> class is managed using a reference counting scheme, so the cost of this operation is independent of the size of the array.	
int capacity (void) const	Method on Array<T>
int length (void) const	Method on Array<T>
Return the length of the array.	
T& elem (int <i>n</i>)	Method on Array<T>
T& checkelem (int <i>n</i>)	Method on Array<T>
If <i>n</i> is within the bounds of the array, return a reference to the element indexed by <i>n</i> ; otherwise, the current error handler is invoked (see Chapter 13 [Error Handling] , page 48).	
T& operator () (int <i>n</i>)	Indexing on Array<T>
T elem (int <i>n</i>) const	Method on Array<T>
T checkelem (int <i>n</i>) const	Method on Array<T>
If <i>n</i> is within the bounds of the array, return the value indexed by <i>n</i> ; otherwise, call the current error handler. See Chapter 13 [Error Handling] , page 48.	
T operator () (int <i>n</i>) const	Indexing on Array<T>
T& xelem (int <i>n</i>)	Method on Array<T>
T xelem (int <i>n</i>) const	Method on Array<T>
Return a reference to, or the value of, the element indexed by <i>n</i> . These methods never perform bounds checking.	

<code>void resize (int <i>n</i> [, const T &<i>val</i>])</code>	Method on <code>Array<T></code>
Change the size of the array to be <i>n</i> elements. All elements are unchanged, except that if <i>n</i> is greater than the current size and the optional argument <i>val</i> is provided, the additional elements are initialized to <i>val</i> ; otherwise, any additional elements are left uninitialized. In the current implementation, if <i>n</i> is less than the current size, the length is updated but no memory is released.	
<code>const T* data (void) const</code>	Method on <code>Array<T></code>
<code>Array2<T> Array2<T> Array2 (void)</code>	Constructor
<code>Array2<T> (int <i>n</i>, int <i>m</i>)</code>	Constructor
<code>Array2<T> (int <i>n</i>, int <i>m</i>, const T &<i>val</i>)</code>	Constructor
<code>Array2<T> (const Array2<T> &<i>a</i>)</code>	Constructor
<code>Array2<T> (const DiagArray<T> &<i>a</i>)</code>	Constructor
<code>Array2<T>& operator = (const Array2<T> &<i>a</i>)</code>	Assignment on <code>Array2<T></code>
<code>int dim1 (void) const</code>	Method on <code>Array2<T></code>
<code>int rows (void) const</code>	Method on <code>Array2<T></code>
<code>int dim2 (void) const</code>	Method on <code>Array2<T></code>
<code>int cols (void) const</code>	Method on <code>Array2<T></code>
<code>int columns (void) const</code>	Method on <code>Array2<T></code>
<code>T& elem (int <i>i</i>, int <i>j</i>)</code>	Method on <code>Array2<T></code>
<code>T& checkelem (int <i>i</i>, int <i>j</i>)</code>	Method on <code>Array2<T></code>
<code>T& operator () (int <i>i</i>, int <i>j</i>)</code>	Indexing on <code>Array2<T></code>
<code>void resize (int <i>n</i>, int <i>m</i>)</code>	Method on <code>Array2<T></code>
<code>void resize (int <i>n</i>, int <i>m</i>, const T &<i>val</i>)</code>	Method on <code>Array2<T></code>
<code>Array3<T> (void)</code>	Constructor
<code>Array3<T> (int <i>n</i>, int <i>m</i>, int <i>k</i>)</code>	Constructor
<code>Array3<T> (int <i>n</i>, int <i>m</i>, int <i>k</i>, const T &<i>val</i>)</code>	Constructor
<code>Array3<T> (const Array3<T> &<i>a</i>)</code>	Constructor
<code>Array3<T>& operator = (const Array3<T> &<i>a</i>)</code>	Assignment on <code>Array3<T></code>
<code>int dim1 (void) const</code>	Method on <code>Array3<T></code>
<code>int dim2 (void) const</code>	Method on <code>Array3<T></code>
<code>int dim3 (void) const</code>	Method on <code>Array3<T></code>
<code>T& elem (int <i>i</i>, int <i>j</i>, int <i>k</i>)</code>	Method on <code>Array3<T></code>
<code>T& checkelem (int <i>i</i>, int <i>j</i>, int <i>k</i>)</code>	Method on <code>Array3<T></code>
<code>T& operator () (int <i>i</i>, int <i>j</i>, int <i>k</i>)</code>	Indexing on <code>Array3<T></code>

<code>void resize (int n, int m, int k)</code>	Method on <code>Array3<T></code>
<code>void resize (int n, int m, int k, const T &val)</code>	Method on <code>Array3<T></code>
<code>DiagArray<T> (void)</code>	Constructor
<code>DiagArray<T> (int n)</code>	Constructor
<code>DiagArray<T> (int n, const T &val)</code>	Constructor
<code>DiagArray<T> (int r, int c)</code>	Constructor
<code>DiagArray<T> (int r, int c, const T &val)</code>	Constructor
<code>DiagArray<T> (const Array<T> &a)</code>	Constructor
<code>DiagArray<T> (const DiagArray<T> &a)</code>	Constructor
<code>operator = (const DiagArray<T> &a)</code>	Assignment on <code>DiagArray<T>&</code>
<code>int dim1 (void) const</code>	Method on <code>DiagArray<T></code>
<code>int rows (void) const</code>	Method on <code>DiagArray<T></code>
<code>int dim2 (void) const</code>	Method on <code>DiagArray<T></code>
<code>int cols (void) const</code>	Method on <code>DiagArray<T></code>
<code>int columns (void) const</code>	Method on <code>DiagArray<T></code>
<code>T& elem (int r, int c)</code>	Method on <code>DiagArray<T></code>
<code>T& checkelem (int r, int c)</code>	Method on <code>DiagArray<T></code>
<code>T& operator () (int r, int c)</code>	Indexing on <code>DiagArray<T></code>
<code>void resize (int n, int m)</code>	Method on <code>DiagArray<T></code>
<code>void resize (int n, int m, const T &val)</code>	Method on <code>DiagArray<T></code>

The real and complex `ColumnVector` and `RowVector` classes all have the following functions. These will eventually be part of an `MArray<T>` class, derived from the `Array<T>` class. Then the `ColumnVector` and `RowVector` classes will be derived from the `MArray<T>` class.

Element by element vector by scalar ops.

```
RowVector operator + (const RowVector &a, const double &s)
RowVector operator - (const RowVector &a, const double &s)
RowVector operator * (const RowVector &a, const double &s)
RowVector operator / (const RowVector &a, const double &s)
```

Element by element scalar by vector ops.

```
RowVector operator + (const double &s, const RowVector &a)
RowVector operator - (const double &s, const RowVector &a)
RowVector operator * (const double &s, const RowVector &a)
RowVector operator / (const double &s, const RowVector &a)
```

Element by element vector by vector ops.

```
RowVector operator + (const RowVector &a, const RowVector &b)
RowVector operator - (const RowVector &a, const RowVector &b)
```

RowVector product (const RowVector &a, const RowVector &b)

RowVector quotient (const RowVector &a, const RowVector &b)

Unary MArray ops.

RowVector operator - (const RowVector &a)

The `Matrix` classes share the following functions. These will eventually be part of an `MArray2<T>` class, derived from the `Array2<T>` class. Then the `Matrix` class will be derived from the `MArray<T>` class.

Element by element matrix by scalar ops.

Matrix operator + (const Matrix &a, const double &s)

Matrix operator - (const Matrix &a, const double &s)

Matrix operator * (const Matrix &a, const double &s)

Matrix operator / (const Matrix &a, const double &s)

Element by element scalar by matrix ops.

Matrix operator + (const double &s, const Matrix &a)

Matrix operator - (const double &s, const Matrix &a)

Matrix operator * (const double &s, const Matrix &a)

Matrix operator / (const double &s, const Matrix &a)

Element by element matrix by matrix ops.

Matrix operator + (const Matrix &a, const Matrix &b)

Matrix operator - (const Matrix &a, const Matrix &b)

Matrix product (const Matrix &a, const Matrix &b)

Matrix quotient (const Matrix &a, const Matrix &b)

Unary matrix ops.

Matrix operator - (const Matrix &a)

The `DiagMatrix` classes share the following functions. These will eventually be part of an `MDiagArray<T>` class, derived from the `DiagArray<T>` class. Then the `DiagMatrix` class will be derived from the `MDiagArray<T>` class.

Element by element MDiagArray by scalar ops.

DiagMatrix operator * (const DiagMatrix &a, const double &s)

DiagMatrix operator / (const DiagMatrix &a, const double &s)

Element by element scalar by MDiagArray ops.

DiagMatrix operator * (const double &s, const DiagMatrix &a)

Element by element MDiagArray by MDiagArray ops.

DiagMatrix **operator** + (const DiagMatrix &a, const DiagMatrix &b)
DiagMatrix **operator** - (const DiagMatrix &a, const DiagMatrix &b)

DiagMatrix **product** (const DiagMatrix &a, const DiagMatrix &b)

Unary MDiagArray ops.

DiagMatrix **operator** - (const DiagMatrix &a)

4 Matrix and Vector Operations

```

(void)
(int r, int c)
(int r, int c, double val)
(const Array2<double> &a)
(const Matrix &a)
(const DiagArray<double> &a)
(const DiagMatrix &a)

Matrix& operator = (const Matrix &a)

int operator == (const Matrix &a) const
int operator != (const Matrix &a) const

Matrix& insert (const Matrix &a, int r, int c)
Matrix& insert (const RowVector &a, int r, int c)
Matrix& insert (const ColumnVector &a, int r, int c)
Matrix& insert (const DiagMatrix &a, int r, int c)

Matrix& fill (double val)
Matrix& fill (double val, int r1, int c1, int r2, int c2)

Matrix append (const Matrix &a) const
Matrix append (const RowVector &a) const
Matrix append (const ColumnVector &a) const
Matrix append (const DiagMatrix &a) const

Matrix stack (const Matrix &a) const
Matrix stack (const RowVector &a) const
Matrix stack (const ColumnVector &a) const
Matrix stack (const DiagMatrix &a) const

Matrix transpose (void) const

Matrix extract (int r1, int c1, int r2, int c2) const

RowVector row (int i) const
RowVector row (char *s) const

ColumnVector column (int i) const
ColumnVector column (char *s) const

Matrix inverse (void) const
Matrix inverse (int &info) const
Matrix inverse (int &info, double &rcond) const

```

```

ComplexMatrix fourier (void) const
ComplexMatrix ifourier (void) const

DET determinant (void) const
DET determinant (int &info) const
DET determinant (int &info, double &rcond) const

Matrix solve (const Matrix &b) const
Matrix solve (const Matrix &b, int &info) const
Matrix solve (const Matrix &b, int &info, double &rcond) const

ComplexMatrix solve (const ComplexMatrix &b) const
ComplexMatrix solve (const ComplexMatrix &b, int &info) const
ComplexMatrix solve (const ComplexMatrix &b, int &info, double &rcond)
    const

ColumnVector solve (const ColumnVector &b) const
ColumnVector solve (const ColumnVector &b, int &info) const
ColumnVector solve (const ColumnVector &b, int &info, double &rcond)
    const

ComplexColumnVector solve (const ComplexColumnVector &b) const
ComplexColumnVector solve (const ComplexColumnVector &b, int &info)
    const
ComplexColumnVector solve (const ComplexColumnVector &b, int &info,
    double &rcond) const

Matrix lssolve (const Matrix &b) const
Matrix lssolve (const Matrix &b, int &info) const
Matrix lssolve (const Matrix &b, int &info, int &rank) const

ComplexMatrix lssolve (const ComplexMatrix &b) const
ComplexMatrix lssolve (const ComplexMatrix &b, int &info) const
ComplexMatrix lssolve (const ComplexMatrix &b, int &info, int &rank)
    const

ColumnVector lssolve (const ColumnVector &b) const
ColumnVector lssolve (const ColumnVector &b, int &info) const
ColumnVector lssolve (const ColumnVector &b, int &info, int &rank) const

ComplexColumnVector lssolve (const ComplexColumnVector &b) const
ComplexColumnVector lssolve (const ComplexColumnVector &b, int &info)
    const
ComplexColumnVector lssolve (const ComplexColumnVector &b, int &info,
    int &rank) const

Matrix& operator += (const Matrix &a)
Matrix& operator -= (const Matrix &a)

```

Matrix& **operator** += (const DiagMatrix &a)
 Matrix& **operator** -= (const DiagMatrix &a)

Matrix **operator** ! (void) const

ComplexMatrix **operator** + (const Matrix &a, const Complex &s)
 ComplexMatrix **operator** - (const Matrix &a, const Complex &s)
 ComplexMatrix **operator** * (const Matrix &a, const Complex &s)
 ComplexMatrix **operator** / (const Matrix &a, const Complex &s)

ComplexMatrix **operator** + (const Complex &s, const Matrix &a)
 ComplexMatrix **operator** - (const Complex &s, const Matrix &a)
 ComplexMatrix **operator** * (const Complex &s, const Matrix &a)
 ComplexMatrix **operator** / (const Complex &s, const Matrix &a)

ColumnVector **operator** * (const Matrix &a, const ColumnVector &b)
 ComplexColumnVector **operator** * (const Matrix &a, const
 ComplexColumnVector &b)

Matrix **operator** + (const Matrix &a, const DiagMatrix &b)
 Matrix **operator** - (const Matrix &a, const DiagMatrix &b)
 Matrix **operator** * (const Matrix &a, const DiagMatrix &b)

ComplexMatrix **operator** + (const Matrix &a, const ComplexDiagMatrix &b)
 ComplexMatrix **operator** - (const Matrix &a, const ComplexDiagMatrix &b)
 ComplexMatrix **operator** * (const Matrix &a, const ComplexDiagMatrix &b)

Matrix **operator** * (const Matrix &a, const Matrix &b)
 ComplexMatrix **operator** * (const Matrix &a, const ComplexMatrix &b)

ComplexMatrix **operator** + (const Matrix &a, const ComplexMatrix &b)
 ComplexMatrix **operator** - (const Matrix &a, const ComplexMatrix &b)

ComplexMatrix **product** (const Matrix &a, const ComplexMatrix &b)
 ComplexMatrix **quotient** (const Matrix &a, const ComplexMatrix &b)

Matrix **map** (d_d_Mapper f, const Matrix &a)
 void **map** (d_d_Mapper f)

Matrix **all** (void) const
 Matrix **any** (void) const

Matrix **cumprod** (void) const
 Matrix **cumsum** (void) const
 Matrix **prod** (void) const
 Matrix **sum** (void) const
 Matrix **sumsq** (void) const

```

ColumnVector diag (void) const
ColumnVector diag (int k) const

ColumnVector row_min (void) const
ColumnVector row_min_loc (void) const

ColumnVector row_max (void) const
ColumnVector row_max_loc (void) const

RowVector column_min (void) const
RowVector column_min_loc (void) const

RowVector column_max (void) const
RowVector column_max_loc (void) const

ostream& operator << (ostream &os, const Matrix &a)
istream& operator >> (istream &is, Matrix &a)

(void)
(int n)
(int n, double val)
(const Array<double> &a)
(const ColumnVector &a)

ColumnVector& operator = (const ColumnVector &a)

int operator == (const ColumnVector &a) const
int operator != (const ColumnVector &a) const

ColumnVector& insert (const ColumnVector &a, int r)

ColumnVector& fill (double val)
ColumnVector& fill (double val, int r1, int r2)

ColumnVector stack (const ColumnVector &a) const

RowVector transpose (void) const

ColumnVector extract (int r1, int r2) const

ColumnVector& operator += (const ColumnVector &a)
ColumnVector& operator -= (const ColumnVector &a)

ComplexColumnVector operator + (const ColumnVector &a, const Complex
    &s)
ComplexColumnVector operator - (const ColumnVector &a, const Complex
    &s)
ComplexColumnVector operator * (const ColumnVector &a, const Complex
    &s)
ComplexColumnVector operator / (const ColumnVector &a, const Complex
    &s)

```

```

ComplexColumnVector operator + (const Complex &s, const ColumnVector
    &a)
ComplexColumnVector operator - (const Complex &s, const ColumnVector
    &a)
ComplexColumnVector operator * (const Complex &s, const ColumnVector
    &a)
ComplexColumnVector operator / (const Complex &s, const ColumnVector
    &a)

Matrix operator * (const ColumnVector &a, const RowVector &a)

ComplexMatrix operator * (const ColumnVector &a, const
    ComplexRowVector &b)

ComplexColumnVector operator + (const ComplexColumnVector &a, const
    ComplexColumnVector &b)

ComplexColumnVector operator - (const ComplexColumnVector &a, const
    ComplexColumnVector &b)

ComplexColumnVector product (const ComplexColumnVector &a, const
    ComplexColumnVector &b)

ComplexColumnVector quotient (const ComplexColumnVector &a, const
    ComplexColumnVector &b)

ColumnVector map (d_d_Mapper f, const ColumnVector &a)
void map (d_d_Mapper f)

double min (void) const
double max (void) const

ostream& operator << (ostream &os, const ColumnVector &a)

(void)
(int n)
(int n, double val)
(const Array<double> &a)
(const RowVector &a)

RowVector& operator = (const RowVector &a)

int operator == (const RowVector &a) const
int operator != (const RowVector &a) const

RowVector& insert (const RowVector &a, int c)

RowVector& fill (double val)
RowVector& fill (double val, int c1, int c2)

```

RowVector **append** (const RowVector &a) const

ColumnVector **transpose** (void) const

RowVector **extract** (int c1, int c2) const

RowVector& **operator +=** (const RowVector &a)

RowVector& **operator -=** (const RowVector &a)

ComplexRowVector **operator +** (const RowVector &a, const Complex &s)

ComplexRowVector **operator -** (const RowVector &a, const Complex &s)

ComplexRowVector **operator *** (const RowVector &a, const Complex &s)

ComplexRowVector **operator /** (const RowVector &a, const Complex &s)

ComplexRowVector **operator +** (const Complex &s, const RowVector &a)

ComplexRowVector **operator -** (const Complex &s, const RowVector &a)

ComplexRowVector **operator *** (const Complex &s, const RowVector &a)

ComplexRowVector **operator /** (const Complex &s, const RowVector &a)

double **operator *** (const RowVector &a, ColumnVector &b)

Complex **operator *** (const RowVector &a, const ComplexColumnVector &b)

RowVector **operator *** (const RowVector &a, const Matrix &b)

ComplexRowVector **operator *** (const RowVector &a, const ComplexMatrix
&b)

ComplexRowVector **operator +** (const RowVector &a, const
ComplexRowVector &b)

ComplexRowVector **operator -** (const RowVector &a, const
ComplexRowVector &b)

ComplexRowVector **product** (const RowVector &a, const ComplexRowVector
&b)

ComplexRowVector **quotient** (const RowVector &a, const ComplexRowVector
&b)

RowVector **map** (d_d_Mapper f, const RowVector &a)

void **map** (d_d_Mapper f)

double **min** (void) const

double **max** (void) const

ostream& **operator <<** (ostream &os, const RowVector &a)

```

(void)
(int n)
(int n, double val)
(int r, int c)
(int r, int c, double val)
(const RowVector &a)
(const ColumnVector &a)
(const DiagArray<double> &a)
(const DiagMatrix &a)

DiagMatrix& operator = (const DiagMatrix &a)

int operator == (const DiagMatrix &a) const
int operator != (const DiagMatrix &a) const

DiagMatrix& fill (double val)
DiagMatrix& fill (double val, int beg, int end)
DiagMatrix& fill (const ColumnVector &a)
DiagMatrix& fill (const RowVector &a)
DiagMatrix& fill (const ColumnVector &a, int beg)
DiagMatrix& fill (const RowVector &a, int beg)

DiagMatrix transpose (void) const

Matrix extract (int r1, int c1, int r2, int c2) const

RowVector row (int i) const
RowVector row (char *s) const

ColumnVector column (int i) const
ColumnVector column (char *s) const

DiagMatrix inverse (void) const
DiagMatrix inverse (int &info) const

DiagMatrix& operator += (const DiagMatrix &a)
DiagMatrix& operator -= (const DiagMatrix &a)

Matrix operator + (const DiagMatrix &a, double s)
Matrix operator - (const DiagMatrix &a, double s)

ComplexMatrix operator + (const DiagMatrix &a, const Complex &s)
ComplexMatrix operator - (const DiagMatrix &a, const Complex &s)

ComplexDiagMatrix operator * (const DiagMatrix &a, const Complex &s)
ComplexDiagMatrix operator / (const DiagMatrix &a, const Complex &s)

Matrix operator + (double s, const DiagMatrix &a)
Matrix operator - (double s, const DiagMatrix &a)

```

```

ComplexMatrix operator + (const Complex &s, const DiagMatrix &a)
ComplexMatrix operator - (const Complex &s, const DiagMatrix &a)

ComplexDiagMatrix operator * (const Complex &s, const DiagMatrix &a)

ColumnVector operator * (const DiagMatrix &a, const ColumnVector &b)

ComplexColumnVector operator * (const DiagMatrix &a, const
    ComplexColumnVector &b)

ComplexDiagMatrix operator + (const DiagMatrix &a, const
    ComplexDiagMatrix &b)
ComplexDiagMatrix operator - (const DiagMatrix &a, const
    ComplexDiagMatrix &b)

ComplexDiagMatrix product (const DiagMatrix &a, const
    ComplexDiagMatrix &b)

Matrix operator + (const DiagMatrix &a, const Matrix &b)
Matrix operator - (const DiagMatrix &a, const Matrix &b)
Matrix operator * (const DiagMatrix &a, const Matrix &b)

ComplexMatrix operator + (const DiagMatrix &a, const ComplexMatrix &b)
ComplexMatrix operator - (const DiagMatrix &a, const ComplexMatrix &b)
ComplexMatrix operator * (const DiagMatrix &a, const ComplexMatrix &b)

ColumnVector diag (void) const
ColumnVector diag (int k) const

ostream& operator << (ostream &os, const DiagMatrix &a)

(void)
(int r, int c)
(int r, int c, const Complex &val)
(const Matrix &a)
(const Array2<Complex> &a)
(const ComplexMatrix &a)
(const DiagMatrix &a)
(const DiagArray<Complex> &a)
(const ComplexDiagMatrix &a)

ComplexMatrix& operator = (const ComplexMatrix &a)

int operator == (const ComplexMatrix &a) const
int operator != (const ComplexMatrix &a) const

ComplexMatrix& insert (const Matrix &a, int r, int c)
ComplexMatrix& insert (const RowVector &a, int r, int c)
ComplexMatrix& insert (const ColumnVector &a, int r, int c)
ComplexMatrix& insert (const DiagMatrix &a, int r, int c)

```

```
ComplexMatrix& insert (const ComplexMatrix &a, int r, int c)
ComplexMatrix& insert (const ComplexRowVector &a, int r, int c)
ComplexMatrix& insert (const ComplexColumnVector &a, int r, int c)
ComplexMatrix& insert (const ComplexDiagMatrix &a, int r, int c)
```

```
ComplexMatrix& fill (double val)
ComplexMatrix& fill (const Complex &val)
ComplexMatrix& fill (double val, int r1, int c1, int r2, int c2)
ComplexMatrix& fill (const Complex &val, int r1, int c1, int r2, int c2)
```

```
ComplexMatrix append (const Matrix &a) const
ComplexMatrix append (const RowVector &a) const
ComplexMatrix append (const ColumnVector &a) const
ComplexMatrix append (const DiagMatrix &a) const
```

```
ComplexMatrix append (const ComplexMatrix &a) const
ComplexMatrix append (const ComplexRowVector &a) const
ComplexMatrix append (const ComplexColumnVector &a) const
ComplexMatrix append (const ComplexDiagMatrix &a) const
```

```
ComplexMatrix stack (const Matrix &a) const
ComplexMatrix stack (const RowVector &a) const
ComplexMatrix stack (const ColumnVector &a) const
ComplexMatrix stack (const DiagMatrix &a) const
```

```
ComplexMatrix stack (const ComplexMatrix &a) const
ComplexMatrix stack (const ComplexRowVector &a) const
ComplexMatrix stack (const ComplexColumnVector &a) const
ComplexMatrix stack (const ComplexDiagMatrix &a) const
```

```
ComplexMatrix transpose (void) const
```

```
Matrix real (const ComplexMatrix &a)
Matrix imag (const ComplexMatrix &a)
ComplexMatrix conj (const ComplexMatrix &a)
```

```
ComplexMatrix extract (int r1, int c1, int r2, int c2) const
```

```
ComplexRowVector row (int i) const
ComplexRowVector row (char *s) const
```

```
ComplexColumnVector column (int i) const
ComplexColumnVector column (char *s) const
```

```
ComplexMatrix inverse (void) const
ComplexMatrix inverse (int &info) const
ComplexMatrix inverse (int &info, double &rcond) const
```

ComplexMatrix **fourier** (void) const
 ComplexMatrix **ifourier** (void) const

ComplexDET **determinant** (void) const
 ComplexDET **determinant** (int &info) const
 ComplexDET **determinant** (int &info, double &rcond) const

ComplexMatrix **solve** (const Matrix &b) const
 ComplexMatrix **solve** (const Matrix &b, int &info) const
 ComplexMatrix **solve** (const Matrix &b, int &info, double &rcond) const

ComplexMatrix **solve** (const ComplexMatrix &b) const
 ComplexMatrix **solve** (const ComplexMatrix &b, int &info) const
 ComplexMatrix **solve** (const ComplexMatrix &b, int &info, double &rcond)
 const

ComplexColumnVector **solve** (const ComplexColumnVector &b) const
 ComplexColumnVector **solve** (const ComplexColumnVector &b, int &info)
 const
 ComplexColumnVector **solve** (const ComplexColumnVector &b, int &info,
 double &rcond) const

ComplexMatrix **lssolve** (const ComplexMatrix &b) const
 ComplexMatrix **lssolve** (const ComplexMatrix &b, int &info) const
 ComplexMatrix **lssolve** (const ComplexMatrix &b, int &info, int &rank)
 const

ComplexColumnVector **lssolve** (const ComplexColumnVector &b) const
 ComplexColumnVector **lssolve** (const ComplexColumnVector &b, int &info)
 const
 ComplexColumnVector **lssolve** (const ComplexColumnVector &b, int &info,
 int &rank) const

ComplexMatrix& **operator +=** (const DiagMatrix &a)
 ComplexMatrix& **operator -=** (const DiagMatrix &a)

ComplexMatrix& **operator +=** (const ComplexDiagMatrix &a)
 ComplexMatrix& **operator -=** (const ComplexDiagMatrix &a)

ComplexMatrix& **operator +=** (const Matrix &a)
 ComplexMatrix& **operator -=** (const Matrix &a)

ComplexMatrix& **operator +=** (const ComplexMatrix &a)
 ComplexMatrix& **operator -=** (const ComplexMatrix &a)

Matrix **operator !** (void) const

```

ComplexMatrix operator + (const ComplexMatrix &a, double s)
ComplexMatrix operator - (const ComplexMatrix &a, double s)
ComplexMatrix operator * (const ComplexMatrix &a, double s)
ComplexMatrix operator / (const ComplexMatrix &a, double s)

ComplexMatrix operator + (double s, const ComplexMatrix &a)
ComplexMatrix operator - (double s, const ComplexMatrix &a)
ComplexMatrix operator * (double s, const ComplexMatrix &a)
ComplexMatrix operator / (double s, const ComplexMatrix &a)

ComplexColumnVector operator * (const ComplexMatrix &a, const
    ColumnVector &b)

ComplexColumnVector operator * (const ComplexMatrix &a, const
    ComplexColumnVector &b)

ComplexMatrix operator + (const ComplexMatrix &a, const DiagMatrix &b)
ComplexMatrix operator - (const ComplexMatrix &a, const DiagMatrix &b)
ComplexMatrix operator * (const ComplexMatrix &a, const DiagMatrix &b)

ComplexMatrix operator + (const ComplexMatrix &a, const
    ComplexDiagMatrix &b)
ComplexMatrix operator - (const ComplexMatrix &a, const
    ComplexDiagMatrix &b)
ComplexMatrix operator * (const ComplexMatrix &a, const
    ComplexDiagMatrix &b)

ComplexMatrix operator + (const ComplexMatrix &a, const Matrix &b)
ComplexMatrix operator - (const ComplexMatrix &a, const Matrix &b)

ComplexMatrix operator * (const ComplexMatrix &a, const Matrix &b)
ComplexMatrix operator * (const ComplexMatrix &a, const ComplexMatrix
    &b)

ComplexMatrix product (const ComplexMatrix &a, const Matrix &b)
ComplexMatrix quotient (const ComplexMatrix &a, const Matrix &b)

ComplexMatrix map (c_c_Mapper f, const ComplexMatrix &a)
Matrix map (d_c_Mapper f, const ComplexMatrix &a)
void map (c_c_Mapper f)

Matrix all (void) const
Matrix any (void) const

ComplexMatrix cumprod (void) const
ComplexMatrix cumsum (void) const
ComplexMatrix prod (void) const
ComplexMatrix sum (void) const
ComplexMatrix sumsq (void) const

```

```

ComplexColumnVector diag (void) const
ComplexColumnVector diag (int k) const

ComplexColumnVector row_min (void) const
ComplexColumnVector row_min_loc (void) const

ComplexColumnVector row_max (void) const
ComplexColumnVector row_max_loc (void) const

ComplexRowVector column_min (void) const
ComplexRowVector column_min_loc (void) const

ComplexRowVector column_max (void) const
ComplexRowVector column_max_loc (void) const

ostream& operator << (ostream &os, const ComplexMatrix &a)
istream& operator >> (istream &is, ComplexMatrix &a)

(void)
(int n)
(int n, const Complex &val)
(const ColumnVector &a)
(const Array<Complex> &a)
(const ComplexColumnVector &a)

ComplexColumnVector& operator = (const ComplexColumnVector &a)

int operator == (const ComplexColumnVector &a) const
int operator != (const ComplexColumnVector &a) const

ComplexColumnVector& insert (const ColumnVector &a, int r)
ComplexColumnVector& insert (const ComplexColumnVector &a, int r)

ComplexColumnVector& fill (double val)
ComplexColumnVector& fill (const Complex &val)
ComplexColumnVector& fill (double val, int r1, int r2)
ComplexColumnVector& fill (const Complex &val, int r1, int r2)

ComplexColumnVector stack (const ColumnVector &a) const
ComplexColumnVector stack (const ComplexColumnVector &a) const

ComplexRowVector transpose (void) const

ColumnVector real (const ComplexColumnVector &a)
ColumnVector imag (const ComplexColumnVector &a)
ComplexColumnVector conj (const ComplexColumnVector &a)

ComplexColumnVector extract (int r1, int r2) const

```

```

ComplexColumnVector& operator += (const ColumnVector &a)
ComplexColumnVector& operator -= (const ColumnVector &a)

ComplexColumnVector& operator += (const ComplexColumnVector &a)
ComplexColumnVector& operator -= (const ComplexColumnVector &a)

ComplexColumnVector operator + (const ComplexColumnVector &a, double
    s)
ComplexColumnVector operator - (const ComplexColumnVector &a, double
    s)
ComplexColumnVector operator * (const ComplexColumnVector &a, double
    s)
ComplexColumnVector operator / (const ComplexColumnVector &a, double
    s)

ComplexColumnVector operator + (double s, const ComplexColumnVector
    &a)
ComplexColumnVector operator - (double s, const ComplexColumnVector
    &a)
ComplexColumnVector operator * (double s, const ComplexColumnVector
    &a)
ComplexColumnVector operator / (double s, const ComplexColumnVector
    &a)

ComplexMatrix operator * (const ComplexColumnVector &a, const
    ComplexRowVector &b)

ComplexColumnVector operator + (const ComplexColumnVector &a, const
    ColumnVector &b)
ComplexColumnVector operator - (const ComplexColumnVector &a, const
    ColumnVector &b)

ComplexColumnVector product (const ComplexColumnVector &a, const
    ColumnVector &b)
ComplexColumnVector quotient (const ComplexColumnVector &a, const
    ColumnVector &b)

ComplexColumnVector map (c_c_Mapper f, const ComplexColumnVector &a)
ColumnVector map (d_c_Mapper f, const ComplexColumnVector &a)
void map (c_c_Mapper f)

Complex min (void) const
Complex max (void) const

ostream& operator << (ostream &os, const ComplexColumnVector &a)

```

```

(void)
(int n)
(int n, const Complex &val)
(const RowVector &a)
(const Array<Complex> &a)
(const ComplexRowVector &a)

ComplexRowVector& operator = (const ComplexRowVector &a)

int operator == (const ComplexRowVector &a) const
int operator != (const ComplexRowVector &a) const

ComplexRowVector& insert (const RowVector &a, int c)
ComplexRowVector& insert (const ComplexRowVector &a, int c)

ComplexRowVector& fill (double val)
ComplexRowVector& fill (const Complex &val)
ComplexRowVector& fill (double val, int c1, int c2)
ComplexRowVector& fill (const Complex &val, int c1, int c2)

ComplexRowVector append (const RowVector &a) const
ComplexRowVector append (const ComplexRowVector &a) const

ComplexColumnVector transpose (void) const

RowVector real (const ComplexRowVector &a)
RowVector imag (const ComplexRowVector &a)
ComplexRowVector conj (const ComplexRowVector &a)

ComplexRowVector extract (int c1, int c2) const

ComplexRowVector& operator += (const RowVector &a)
ComplexRowVector& operator -= (const RowVector &a)

ComplexRowVector& operator += (const ComplexRowVector &a)
ComplexRowVector& operator -= (const ComplexRowVector &a)

ComplexRowVector operator + (const ComplexRowVector &a, double s)
ComplexRowVector operator - (const ComplexRowVector &a, double s)
ComplexRowVector operator * (const ComplexRowVector &a, double s)
ComplexRowVector operator / (const ComplexRowVector &a, double s)

ComplexRowVector operator + (double s, const ComplexRowVector &a)
ComplexRowVector operator - (double s, const ComplexRowVector &a)
ComplexRowVector operator * (double s, const ComplexRowVector &a)
ComplexRowVector operator / (double s, const ComplexRowVector &a)

Complex operator * (const ComplexRowVector &a, const ColumnVector &b)

```

Complex operator * (const ComplexRowVector &a, const
ComplexColumnVector &b)

ComplexRowVector operator * (const ComplexRowVector &a, const
ComplexMatrix &b)

ComplexRowVector operator + (const ComplexRowVector &a, const
RowVector &b)

ComplexRowVector operator - (const ComplexRowVector &a, const
RowVector &b)

ComplexRowVector product (const ComplexRowVector &a, const RowVector
&b)

ComplexRowVector quotient (const ComplexRowVector &a, const RowVector
&b)

ComplexRowVector map (c_c_Mapper f, const ComplexRowVector &a)

RowVector map (d_c_Mapper f, const ComplexRowVector &a)

void map (c_c_Mapper f)

Complex min (void) const

Complex max (void) const

ostream& operator << (ostream &os, const ComplexRowVector &a)

(void)

(int n)

(int n, const Complex &val)

(int r, int c)

(int r, int c, const Complex &val)

(const RowVector &a)

(const ComplexRowVector &a)

(const ColumnVector &a)

(const ComplexColumnVector &a)

(const DiagMatrix &a)

(const DiagArray<Complex> &a)

(const ComplexDiagMatrix &a)

ComplexDiagMatrix& operator = (const ComplexDiagMatrix &a)

int operator == (const ComplexDiagMatrix &a) const

int operator != (const ComplexDiagMatrix &a) const

```

ComplexDiagMatrix& fill (double val)
ComplexDiagMatrix& fill (const Complex &val)
ComplexDiagMatrix& fill (double val, int beg, int end)
ComplexDiagMatrix& fill (const Complex &val, int beg, int end)
ComplexDiagMatrix& fill (const ColumnVector &a)
ComplexDiagMatrix& fill (const ComplexColumnVector &a)
ComplexDiagMatrix& fill (const RowVector &a)
ComplexDiagMatrix& fill (const ComplexRowVector &a)
ComplexDiagMatrix& fill (const ColumnVector &a, int beg)
ComplexDiagMatrix& fill (const ComplexColumnVector &a, int beg)
ComplexDiagMatrix& fill (const RowVector &a, int beg)
ComplexDiagMatrix& fill (const ComplexRowVector &a, int beg)

ComplexDiagMatrix transpose (void) const

DiagMatrix real (const ComplexDiagMatrix &a)
DiagMatrix imag (const ComplexDiagMatrix &a)
ComplexDiagMatrix conj (const ComplexDiagMatrix &a)

ComplexMatrix extract (int r1, int c1, int r2, int c2) const

ComplexRowVector row (int i) const
ComplexRowVector row (char *s) const

ComplexColumnVector column (int i) const
ComplexColumnVector column (char *s) const

ComplexDiagMatrix inverse (int &info) const
ComplexDiagMatrix inverse (void) const

ComplexDiagMatrix& operator += (const DiagMatrix &a)
ComplexDiagMatrix& operator -= (const DiagMatrix &a)

ComplexDiagMatrix& operator += (const ComplexDiagMatrix &a)
ComplexDiagMatrix& operator -= (const ComplexDiagMatrix &a)

ComplexMatrix operator + (const ComplexDiagMatrix &a, double s)
ComplexMatrix operator - (const ComplexDiagMatrix &a, double s)

ComplexMatrix operator + (const ComplexDiagMatrix &a, const Complex &s)
ComplexMatrix operator - (const ComplexDiagMatrix &a, const Complex &s)

ComplexDiagMatrix operator * (const ComplexDiagMatrix &a, double s)
ComplexDiagMatrix operator / (const ComplexDiagMatrix &a, double s)

ComplexMatrix operator + (double s, const ComplexDiagMatrix &a)
ComplexMatrix operator - (double s, const ComplexDiagMatrix &a)

```

`ComplexMatrix operator + (const Complex &s, const ComplexDiagMatrix &a)`

`ComplexMatrix operator - (const Complex &s, const ComplexDiagMatrix &a)`

`ComplexDiagMatrix operator * (double s, const ComplexDiagMatrix &a)`

`ComplexColumnVector operator * (const ComplexDiagMatrix &a, const
ColumnVector &b)`

`ComplexColumnVector operator * (const ComplexDiagMatrix &a, const
ComplexColumnVector &b)`

`ComplexDiagMatrix operator + (const ComplexDiagMatrix &a, const
DiagMatrix &b)`

`ComplexDiagMatrix operator - (const ComplexDiagMatrix &a, const
DiagMatrix &b)`

`ComplexDiagMatrix product (const ComplexDiagMatrix &a, const
DiagMatrix &b)`

`ComplexMatrix operator + (const ComplexDiagMatrix &a, const Matrix &b)`

`ComplexMatrix operator - (const ComplexDiagMatrix &a, const Matrix &b)`

`ComplexMatrix operator * (const ComplexDiagMatrix &a, const Matrix &b)`

`ComplexMatrix operator + (const ComplexDiagMatrix &a, const
ComplexMatrix &b)`

`ComplexMatrix operator - (const ComplexDiagMatrix &a, const
ComplexMatrix &b)`

`ComplexMatrix operator * (const ComplexDiagMatrix &a, const
ComplexMatrix &b)`

`ComplexColumnVector diag (void) const`

`ComplexColumnVector diag (int k) const`

`ostream& operator << (ostream &os, const ComplexDiagMatrix &a)`

5 Matrix Factorizations

```

(void)
(const Matrix &a, const char *balance_job)
(const AEPBALANCE &a)

AEPBALANCE& operator = (const AEPBALANCE &a)

Matrix balanced_matrix (void) const
Matrix balancing_matrix (void) const

ostream& operator << (ostream &os, const AEPBALANCE &a)

ComplexAEPBALANCE (void)
ComplexAEPBALANCE (const ComplexMatrix &a, const char *balance_job)
ComplexAEPBALANCE (const ComplexAEPBALANCE &a)

ComplexAEPBALANCE& operator = (const ComplexAEPBALANCE &a)

ComplexMatrix balanced_matrix (void) const
ComplexMatrix balancing_matrix (void) const

ostream& operator << (ostream &os, const ComplexAEPBALANCE &a)

(void)
(const DET &a)

DET& operator = (const DET &a)

int value_will_overflow (void) const
int value_will_underflow (void) const

double coefficient (void) const
int exponent (void) const
double value (void) const

ostream& operator << (ostream &os, const DET &a)

(void)
(const ComplexDET &a)

ComplexDET& operator = (const ComplexDET &a)

int value_will_overflow (void) const
int value_will_underflow (void) const

Complex coefficient (void) const
int exponent (void) const
Complex value (void) const

```

```
ostream& operator << (ostream &os, const ComplexDET &a)
```

```
(void)
```

```
(const Matrix &a, const Matrix &, const char *balance_job)
```

```
(const GEPBALANCE &a)
```

```
GEPBALANCE& operator = (const GEPBALANCE &a)
```

```
Matrix balanced_a_matrix (void) const
```

```
Matrix balanced_b_matrix (void) const
```

```
Matrix left_balancing_matrix (void) const
```

```
Matrix right_balancing_matrix (void) const
```

```
ostream& operator << (ostream &os, const GEPBALANCE &a)
```

```
(void)
```

```
(const Matrix &a)
```

```
(const Matrix &a, int &info)
```

```
(const CHOL &a)
```

```
CHOL& operator = (const CHOL &a)
```

```
Matrix chol_matrix (void) const
```

```
ostream& operator << (ostream &os, const CHOL &a)
```

```
(void)
```

```
(const ComplexMatrix &a)
```

```
(const ComplexMatrix &a, int &info)
```

```
(const ComplexCHOL &a)
```

```
ComplexCHOL& operator = (const ComplexCHOL &a)
```

```
ComplexMatrix chol_matrix (void) const
```

```
ostream& operator << (ostream &os, const ComplexCHOL &a)
```

```
(void)
```

```
(const Matrix &a)
```

```
(const Matrix&a, int &info)
```

```
(const HESS &a)
```

```
HESS& operator = (const HESS &a)
```

```
Matrix hess_matrix (void) const
```

```
Matrix unitary_hess_matrix (void) const
```

```
ostream& operator << (ostream &os, const HESS &a)
```

```

(void)
(const ComplexMatrix &a)
(const ComplexMatrix &a, int &info)
(const ComplexHESS &a)

ComplexHESS& operator = (const ComplexHESS &a)

ComplexMatrix hess_matrix (void) const
ComplexMatrix unitary_hess_matrix (void) const

ostream& operator << (ostream &os, const ComplexHESS &a)

(void)
(const Matrix &a, const char *ord)
(const Matrix &a, const char *ord, int &info)
(const SCHUR &a, const char *ord)

SCHUR& operator = (const SCHUR &a)

Matrix schur_matrix (void) const
Matrix unitary_matrix (void) const

ostream& operator << (ostream &os, const SCHUR &a)

(void)
(const ComplexMatrix &a, const char *ord)
(const ComplexMatrix &a, const char *ord, int &info)
(const ComplexSCHUR &a, const char *ord)

ComplexSCHUR& operator = (const ComplexSCHUR &a)

ComplexMatrix schur_matrix (void) const
ComplexMatrix unitary_matrix (void) const

ostream& operator << (ostream &os, const ComplexSCHUR &a)

(void)
(const Matrix &a)
(const Matrix &a, int &info)
(const SVD &a)

SVD& operator = (const SVD &a)

DiagMatrix singular_values (void) const
Matrix left_singular_matrix (void) const
Matrix right_singular_matrix (void) const

ostream& operator << (ostream &os, const SVD &a)

```

```

(void)
(const ComplexMatrix &a)
(const ComplexMatrix &a, int &info)
(const ComplexSVD &a)

ComplexSVD& operator = (const ComplexSVD &a)

DiagMatrix singular_values (void) const
ComplexMatrix left_singular_matrix (void) const
ComplexMatrix right_singular_matrix (void) const

ostream& operator << (ostream &os, const ComplexSVD &a)

(void)
(const Matrix &a)
(const Matrix &a, int &info)
(const ComplexMatrix &a)
(const ComplexMatrix &a, int &info)
(const EIG &a)

EIG& operator = (const EIG &a)

ComplexColumnVector eigenvalues (void) const

ComplexMatrix eigenvectors (void) const

ostream& operator << (ostream &os, const EIG &a)

(void)
(const Matrix &a)
(const LU &a)

LU& operator = (const LU &a)

Matrix L (void) const
Matrix U (void) const
Matrix P (void) const

ostream& operator << (ostream &os, const LU &a)

(void)
(const ComplexMatrix &a)
(const ComplexLU &a)

ComplexLU& operator = (const ComplexLU &a)

ComplexMatrix L (void) const
ComplexMatrix U (void) const
Matrix P (void) const

```

```
ostream& operator << (ostream &os, const ComplexLU &a)
```

```
(void)  
(const Matrix &A)  
(const QR &a)
```

```
QR& operator = (const QR &a)
```

```
Matrix Q (void) const  
Matrix R (void) const
```

```
ostream& operator << (ostream &os, const QR &a)
```

```
(void)  
(const ComplexMatrix &A)  
(const ComplexQR &a)
```

```
ComplexQR& operator = (const ComplexQR &a)
```

```
ComplexMatrix Q (void) const  
ComplexMatrix R (void) const
```

```
ostream& operator << (ostream &os, const ComplexQR &a)
```

6 Ranges

```
(void)
(const Range &r)
(double b, double l)
(double b, double l, double i)

double base (void) const
double limit (void) const
double inc (void) const

void set_base (double b)
void set_limit (double l)
void set_inc (double i)

int nelem (void) const

double min (void) const
double max (void) const

void sort (void)

ostream& operator << (ostream &os, const Range &r)
istream& operator >> (istream &is, Range &r)

void print_range (void)
```

7 Nonlinear Functions

```
(void)
(const nonlinear_fcn)
(const nonlinear_fcn, const jacobian_fcn)
(const NLFunc &a)

NLFunc& operator = (const NLFunc &a)

nonlinear_fcn function (void) const;

NLFunc& set_function (const nonlinear_fcn f)

jacobian_fcn jacobian_function (void) const;

NLFunc& set_jacobian_function (const jacobian_fcn j)
```

8 Nonlinear Equations

```
(void)
(const NLEqn_options &opt)

NLEqn_options& operator = (const NLEqn_options &opt)

void init (void)

void copy (const NLEqn_options &opt)

void set_default_options (void)

void set_tolerance (double val)

double tolerance (void)

(void)
(const ColumnVector&, const NLFunc)
(const NLEqn &a)

NLEqn& operator = (const NLEqn &a)

void resize (int n)

void set_states (const ColumnVector &x)

ColumnVector states (void) const

int size (void) const

ColumnVector solve (void)
ColumnVector solve (const ColumnVector &x)

ColumnVector solve (int &info)
ColumnVector solve (const ColumnVector &x, int &info)
```

9 Optimization

9.1 Objective Functions

```

(void)
(const objective_fcn)
(const objective_fcn, const gradient_fcn)
(const Objective &a)

Objective& operator = (const Objective &a)

objective_fcn objective_function (void) const;

Objective& set_objective_function (const objective_fcn)

gradient_fcn gradient_function (void) const;

Objective& set_gradient_function (const gradient_fcn)

```

9.2 Bounds

```

(void)
(int n)
(const ColumnVector lb, const ColumnVector ub)
(const Bounds &a)

Bounds& operator = (const Bounds &a)

Bounds& resize (int n)

double lower_bound (int index) const;
double upper_bound (int index) const;

ColumnVector lower_bounds (void) const;
ColumnVector upper_bounds (void) const;

int size (void) const;

Bounds& set_bound (int index, double low, double high)

Bounds& set_bounds (double low, double high)
Bounds& set_bounds (const ColumnVector lb, const ColumnVector ub)

Bounds& set_lower_bound (int index, double low)
Bounds& set_upper_bound (int index, double high)

```

```
Bounds& set_lower_bounds (double low)
Bounds& set_upper_bounds (double high)
```

```
Bounds& set_lower_bounds (const ColumnVector lb)
Bounds& set_upper_bounds (const ColumnVector ub)
```

```
ostream& operator << (ostream &os, const Bounds &b)
```

9.3 Linear Constraints

```
(void)
(int nclin, int nx)
(int nclin_eq, int nclin_ineq, int nx)
(const ColumnVector &lb, const Matrix &A, const ColumnVector &ub)
(const Matrix &A_eq, const ColumnVector &b_eq, const Matrix &A_ineq,
 const ColumnVector &b_ineq)
(const LinConst &a)
```

```
LinConst& operator = (const LinConst &a)
```

```
LinConst& resize (int nclin, int n)
```

```
Matrix constraint_matrix (void) const;
```

```
LinConst& set_constraint_matrix (const Matrix &A)
```

```
Matrix eq_constraint_matrix (void) const;
Matrix ineq_constraint_matrix (void) const;
```

```
ColumnVector eq_constraint_vector (void) const;
ColumnVector ineq_constraint_vector (void) const;
```

```
ostream& operator << (ostream &os, const LinConst &b)
```

9.4 Nonlinear Constraints

```
(void)
(int n)
(const ColumnVector lb, const NLFunc f, const ColumnVector ub)
(const NLConst &a)
```

```
NLConst& operator = (const NLConst &a)
```

9.5 Quadratic Programming

```

(void)
(const ColumnVector &x, const Matrix &H)
(const ColumnVector &x, const Matrix &H, const ColumnVector &c)
(const ColumnVector &x, const Matrix &H, const Bounds &b)
(const ColumnVector &x, const Matrix &H, const LinConst &lc)
(const ColumnVector &x, const Matrix &H, const ColumnVector &c, const
    Bounds &b)
(const ColumnVector &x, const Matrix &H, const ColumnVector &c, const
    LinConst &lc)
(const ColumnVector &x, const Matrix &H, const Bounds &b, const LinConst
    &lc)
(const ColumnVector &x, const Matrix &H, const ColumnVector &c, const
    Bounds &b, const LinConst &lc)

virtual ColumnVector minimize (void)
virtual ColumnVector minimize (double &objf)
virtual ColumnVector minimize (double &objf, int &inform)
virtual ColumnVector minimize (double &objf, int &inform, ColumnVector
    &lambda) = 0;

virtual ColumnVector minimize (const ColumnVector &x)
virtual ColumnVector minimize (const ColumnVector &x, double &objf)
virtual ColumnVector minimize (const ColumnVector &x, double &objf,
    int &inform)
virtual ColumnVector minimize (const ColumnVector &x, double &objf,
    int &inform, ColumnVector &lambda)

ColumnVector minimize (double &objf, int &inform, ColumnVector &lambda)

```

9.6 Nonlinear Programming

```

(void)
(const ColumnVector &x, const Objective &phi)
(const ColumnVector &x, const Objective &phi, const Bounds &b)
(const ColumnVector &x, const Objective &phi, const Bounds &b, const
    LinConst &lc)
(const ColumnVector &x, const Objective &phi, const Bounds &b, const
    LinConst &lc, const NLConst &nlc)
(const ColumnVector &x, const Objective &phi, const LinConst &lc)
(const ColumnVector &x, const Objective &phi, const LinConst &lc, const
    NLConst &nlc)
(const ColumnVector &x, const Objective &phi, const NLConst &nlc)
(const ColumnVector &x, const Objective &phi, const Bounds &b, const
    NLConst &nlc)

```

```
NLP& operator = (const NLP &a)

int size (void) const

ColumnVector minimize (void)
ColumnVector minimize (double &objf)
ColumnVector minimize (double &objf, int &inform)
ColumnVector minimize (double &objf, int &inform, ColumnVector &lambda)

ColumnVector minimize (const ColumnVector &x)
ColumnVector minimize (const ColumnVector &x, double &objf)
ColumnVector minimize (const ColumnVector &x, double &objf, int
    &inform)
ColumnVector minimize (const ColumnVector &x, double &objf, int
    &inform, ColumnVector &lambda)
```

10 Quadrature

```

(integrand_fcn fcn)
(integrand_fcn fcn, double abs, double rel)

virtual double integrate (void)
virtual double integrate (int &ier)
virtual double integrate (int &ier, int &neval)
virtual double integrate (int &ier, int &neval, double &abserr) = 0

Quad_options (void)
Quad_options (const Quad_options &opt)

Quad_options& operator = (const Quad_options &opt)

void init (void)

void copy (const Quad_options &opt)

void set_default_options (void)

void set_absolute_tolerance (double val)

void set_relative_tolerance (double val)

double absolute_tolerance (void)
double relative_tolerance (void)

(integrand_fcn fcn)
(integrand_fcn fcn, double ll, double ul)
(integrand_fcn fcn, double ll, double ul, double abs, double rel)
(integrand_fcn fcn, double ll, double ul, const ColumnVector &sing)
(integrand_fcn fcn, const ColumnVector &sing, double abs, double rel)
(integrand_fcn fcn, const ColumnVector &sing)
(integrand_fcn fcn, double ll, double ul, const ColumnVector &sing, double
    abs, double rel)

(integrand_fcn fcn)
(integrand_fcn fcn, double b, IntegralType t)
(integrand_fcn fcn, double b, IntegralType t, double abs, double rel)
(integrand_fcn fcn, double abs, double rel)

```

10.1 Collocation Weights

```

(void)
(int n, int inc_l, int inc_r)
(int n, int inc_l, int inc_r, double l, double r)
(int n, double a, double b, int inc_l, int inc_r)
(int n, int inc_l, int inc_r, double l, double r)
(const CollocWt&)

CollocWt& operator = (const CollocWt&)

CollocWt& resize (int ncol)

CollocWt& add_left (void)
CollocWt& add_right (void)

CollocWt& delete_left (void)
CollocWt& delete_right (void)

CollocWt& set_left (double val)
CollocWt& set_right (double val)

CollocWt& set_alpha (double val)
CollocWt& set_beta (double val)

int ncol (void) const

int left_included (void) const
int right_included (void) const

double left (void) const
double right (void) const
double width (void) const

double alpha (void) const
double beta (void) const

ColumnVector roots (void)
ColumnVector quad (void)
ColumnVector quad_weights (void)

Matrix first (void)
Matrix second (void)

ostream& operator << (ostream &os, const CollocWt &c)

```

11 Ordinary Differential Equations

```

(void)
(const ODE_options &opt)
ODE_options& operator = (const ODE_options &opt)
void init (void)
void copy (const ODE_options &opt)
void set_default_options (void)
void set_absolute_tolerance (double val)
void set_initial_step_size (double val)
void set_maximum_step_size (double val)
void set_minimum_step_size (double val)
void set_relative_tolerance (double val)

double absolute_tolerance (void)
double initial_step_size (void)
double maximum_step_size (void)
double minimum_step_size (void)
double relative_tolerance (void)

(void)
(int n)
(const ColumnVector &state, double time, const ODEFunc &f)

virtual int size (void) const
virtual ColumnVector state (void) const
virtual double time (void) const
virtual void force_restart (void)
virtual void initialize (const ColumnVector &x, double t)
virtual void set_stop_time (double t)
virtual void clear_stop_time (void)
virtual ColumnVector integrate (double t)
void integrate (int nsteps, double tstep, ostream &s)
Matrix integrate (const ColumnVector &tout)
Matrix integrate (const ColumnVector &tout, const ColumnVector &tcrit)

```

12 Differential Algebraic Equations

```
(void)
(int n)
(const ColumnVector &x, double time, DAEFunc &f)
(const ColumnVector &x, ColumnVector &xdot, double time, DAEFunc &f)

ColumnVector deriv (void)

virtual void initialize (const ColumnVector &x, double t)
virtual void initialize (const ColumnVector &x, ColumnVector &xdot,
    double t)

ColumnVector integrate (double t)

Matrix integrate (const ColumnVector &tout, Matrix &xdot_out)
Matrix integrate (const ColumnVector &tout, Matrix &xdot_out, const
    ColumnVector &tcrit)
```

13 Error Handling

14 Installation

15 Bugs

Concept Index

A

acknowledgements	1
arrays	10

B

bounds	40
bugs, known	50

C

collocation weights	45
contributors	1
copyright	1

D

DAE	47
-----------	----

F

factorizations	32
----------------------	----

I

installation	49
installation trouble	50
integration	44
introduction	9

K

known causes of trouble	50
-------------------------------	----

L

linear Constraints	41
--------------------------	----

M

matrix factorizations	32
matrix manipulations	15

N

NLP	42
nonlinear Constraints	41
nonlinear equations	39
nonlinear functions	38
nonlinear programming	42
numerical integration	44

O

objective functions	40
ODE	46
optimization	40
orthogonal collocation	45

Q

QP	42
quadratic programming	42
quadrature	44

R

ranges	37
--------------	----

T

troubleshooting	50
-----------------------	----

V

vector manipulations	15
----------------------------	----

W

warranty	1
----------------	---

Function Index

- (
 (const ... 15, 18, 19, 21, 22, 26, 28, 29, 32, 33, 34,
 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47
 (const on DiagArray<T>& 12
 (double 37
 (int .. 15, 18, 19, 21, 22, 26, 28, 29, 40, 41, 45, 46,
 47
 (integrand_fcn 44
 (void) ... 15, 18, 19, 21, 22, 26, 27, 29, 32, 33, 34,
 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47
- ## A
- absolute_tolerance 44, 46
 add_left 45
 add_right 45
 all 17, 25
 alpha 45
 any 17, 25
 append 15, 20, 23, 28
 Array<T> 10
 Array2<T> 11
 Array3<T> 11
- ## B
- balanced_a_matrix 33
 balanced_b_matrix 33
 balanced_matrix 32
 balancing_matrix 32
 base 37
 beta 45
- ## C
- capacity on Array<T> 10
 checkelem on Array<T> 10
 checkelem on Array2<T> 11
 checkelem on Array3<T> 11
 checkelem on DiagArray<T> 12
 chol_matrix 33
 clear_stop_time 46
 coefficient 32
 cols on Array2<T> 11
 cols on DiagArray<T> 12
 column 15, 21, 23, 30
 column_max 18, 26
 column_max_loc 18, 26
 column_min 18, 26
 column_min_loc 18, 26
 columns on Array2<T> 11
 columns on DiagArray<T> 12
- conj 23, 26, 28, 30
 constraint_matrix 41
 copy 39, 44, 46
 cumprod 17, 25
 cumsum 17, 25
- ## D
- data on Array<T> 11
 delete_left 45
 delete_right 45
 deriv 47
 determinant 16, 24
 diag 18, 22, 26, 31
 DiagArray<T> 12
 dim1 on Array2<T> 11
 dim1 on Array3<T> 11
 dim1 on DiagArray<T> 12
 dim2 on Array2<T> 11
 dim2 on Array3<T> 11
 dim2 on DiagArray<T> 12
 dim3 on Array3<T> 11
- ## E
- eigenvalues 35
 eigenvectors 35
 elem on Array<T> 10
 elem on Array2<T> 11
 elem on Array3<T> 11
 elem on DiagArray<T> 12
 eq_constraint_matrix 41
 eq_constraint_vector 41
 exponent 32
 extract 15, 18, 20, 21, 23, 26, 28, 30
- ## F
- fill 15, 18, 19, 21, 23, 26, 28, 29, 30
 first 45
 force_restart 46
 fourier 16, 24
 function 38
- ## G
- gradient_function 40
- ## H
- hess_matrix 33, 34

I

ifourier 16, 24
 imag 23, 26, 28, 30
 inc 37
 ineq_constraint_matrix 41
 ineq_constraint_vector 41
 init 39, 44, 46
 initial_step_size 46
 initialize 46, 47
 insert 15, 18, 19, 22, 23, 26, 28
 integrate 44, 46, 47
 inverse 15, 21, 23, 30

J

jacobian_function 38

L

L 35
 left 45
 left_balancing_matrix 33
 left_included 45
 left_singular_matrix 34, 35
 length on Array<T> 10
 limit 37
 lower_bound 40
 lower_bounds 40
 lssolve 16, 24

M

map 17, 19, 20, 25, 27, 29
 max 19, 20, 27, 29, 37
 maximum_step_size 46
 min 19, 20, 27, 29, 37
 minimize 42, 43
 minimum_step_size 46

N

ncol 45
 nelem 37

O

objective_function 40
 operator 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
 36, 37, 38, 39, 40, 41, 43, 44, 45, 46
 operator () on Array<T> 10

operator () on Array2<T> 11
 operator () on Array3<T> 11
 operator () on DiagArray<T> 12
 operator = on Array<T> 10
 operator = on Array2<T> 11
 operator = on Array3<T> 11

P

P 35
 print_range 37
 prod 17, 25
 product 13, 14, 17, 19, 20, 22, 25, 27, 29, 31

Q

Q 36
 quad 45
 quad_weights 45
 quotient 13, 17, 19, 20, 25, 27, 29

R

R 36
 real 23, 26, 28, 30
 relative_tolerance 44, 46
 resize 39, 40, 41, 45
 resize on Array<T> 11
 resize on Array2<T> 11
 resize on Array3<T> 12
 resize on DiagArray<T> 12
 right 45
 right_balancing_matrix 33
 right_included 45
 right_singular_matrix 34, 35
 roots 45
 row 15, 21, 23, 30
 row_max 18, 26
 row_max_loc 18, 26
 row_min 18, 26
 row_min_loc 18, 26
 rows on Array2<T> 11
 rows on DiagArray<T> 12

S

schur_matrix 34
 second 45
 set_absolute_tolerance 44, 46
 set_alpha 45
 set_base 37
 set_beta 45
 set_bound 40
 set_bounds 40
 set_constraint_matrix 41
 set_default_options 39, 44, 46
 set_function 38
 set_gradient_function 40
 set_inc 37
 set_initial_step_size 46
 set_jacobian_function 38
 set_left 45
 set_limit 37
 set_lower_bound 40
 set_lower_bounds 41
 set_maximum_step_size 46
 set_minimum_step_size 46
 set_objective_function 40
 set_relative_tolerance 44, 46
 set_right 45
 set_states 39
 set_stop_time 46
 set_tolerance 39
 set_upper_bound 40
 set_upper_bounds 41
 singular_values 34, 35
 size 39, 40, 43, 46
 solve 16, 24, 39
 sort 37

stack 15, 18, 23, 26
 state 46
 states 39
 sum 17, 25
 sumsq 17, 25

T

time 46
 tolerance 39
 transpose 15, 18, 20, 21, 23, 26, 28, 30

U

U 35
 unitary_hess_matrix 33, 34
 unitary_matrix 34
 upper_bound 40
 upper_bounds 40

V

value 32
 value_will_overflow 32
 value_will_underflow 32

W

width 45

X

xelem on Array<T> 10