The LINEAR-ALGEBRA Reference Manual

Linear Algebra for Common Lisp, version 0.1.1

Steve Nunez `<steve@symbolics.tech>`
Thomas M. Hermann `<thomas.m.hermann@odonata-research.com>`

This manual was generated automatically by Declt 4.0b2.

Table of Contents

Copying

1 Systems

The main system appears first, followed by any subsystem dependency.

1.1 `linear-algebra`

Linear Algebra for Common Lisp

Long Name
Linear Algebra for Common Lisp

Maintainer
Steve Nunez `<steve@symbolics.tech>`

Author Thomas M. Hermann `<thomas.m.hermann@odonata-research.com>`

Home Page
`https://lisp-stat.dev/docs/manuals/lla`

Source Control
`(GIT https://github.com/Lisp-Stat/linear-algebra.git)`

Bug Tracker
`https://github.com/Lisp-Stat/linear-algebra/issues`

License MS-PL

Long Description
This system is a high level interface for linear algebra and matrix manipulation. It was forked from Thomas Hermann's linear-algebra library (https://github.com/OdonataResearchLLC/linear-algebra) and currently maintained by Brian Eberman and Steve Nunez.

Current goals are to implement backends that use BLAS/LAPACK and CUDA.

Version 0.1.1

Dependencies
- `closer-mop` (system).
- `floating-point` (system).

Source [`linear-algebra.asd`], page 7.

Child Components
- [`pkgdcl.lisp`], page 7 (file).
- [`kernel`], page 5 (module).
- [`interface`], page 5 (module).
- [`sequence`], page 5 (module).
- [`data-vector.lisp`], page 15 (file).
- [`dense-matrix.lisp`], page 16 (file).
- [`square-matrix.lisp`], page 18 (file).
- [`hermitian-matrix.lisp`], page 18 (file).
- [`symmetric-matrix.lisp`], page 19 (file).

2 Modules

Modules are listed depth-first from the system components tree.

2.1 `linear-algebra/kernel`

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Child Components
- [`pkgdcl.lisp`], page 7 (file).
- [`utility.lisp`], page 7 (file).
- [`permute.lisp`], page 8 (file).
- [`unary-operations.lisp`], page 8 (file).
- [`binary-operations.lisp`], page 8 (file).
- [`rotation.lisp`], page 9 (file).
- [`gauss.lisp`], page 9 (file).
- [`cholesky.lisp`], page 10 (file).
- [`conjugate-gradient.lisp`], page 10 (file).
- [`tridiagonal.lisp`], page 10 (file).

2.2 `linear-algebra/interface`

Dependency
[`kernel`], page 5 (module).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Child Components
- [`fundamental-ops.lisp`], page 11 (file).
- [`vector.lisp`], page 11 (file).
- [`matrix.lisp`], page 12 (file).
- [`identity-matrix.lisp`], page 12 (file).
- [`permutation-matrix.lisp`], page 13 (file).

2.3 `linear-algebra/sequence`

Dependency
[`interface`], page 5 (module).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Child Components
- [`list.lisp`], page 13 (file).
- [`vector.lisp`], page 14 (file).
- [`array.lisp`], page 14 (file).

3 Files

Files are sorted by type and then listed depth-first from the systems components trees.

3.1 Lisp

3.1.1 `linear-algebra/linear-algebra.asd`

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

ASDF Systems
[`linear-algebra`], page 3.

3.1.2 `linear-algebra/pkgdcl.lisp`

Dependency
[`kernel`], page 5 (module).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Packages [`linear-algebra`], page 23.

3.1.3 `linear-algebra/kernel/pkgdcl.lisp`

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Packages [`linear-algebra-kernel`], page 21.

3.1.4 `linear-algebra/kernel/utility.lisp`

Dependency
[`pkgdcl.lisp`], page 7 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
- [`common-array-element-type`], page 27 (function).
- [`common-class-of`], page 27 (function).
- [`complex-equal`], page 27 (function).
- [`copy-array`], page 34 (generic function).
- [`number-equal`], page 30 (function).
- [`specific-array-element-type`], page 31 (function).

Internals
- [`zero-array`], page 69 (function).
- [`zero-vector`], page 69 (function).

3.1.5 `linear-algebra/kernel/permute.lisp`

Dependency
[`pkgdcl.lisp`], page 7 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
- [`left-permute`], page 36 (generic function).
- [`right-permute`], page 50 (generic function).

3.1.6 `linear-algebra/kernel/unary-operations.lisp`

Dependency
[`pkgdcl.lisp`], page 7 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
- [`norm-array`], page 42 (generic function).
- [`norm-vector`], page 43 (generic function).
- [`sump`], page 54 (generic function).
- [`sumsq`], page 55 (generic function).
- [`sumsq-column`], page 31 (function).
- [`sumsq-row`], page 31 (function).
- [`sumsq2`], page 32 (function).
- [`sumsq3`], page 32 (function).

Internals [`%abs-vector`], page 65 (function).

3.1.7 `linear-algebra/kernel/binary-operations.lisp`

Dependency
[`pkgdcl.lisp`], page 7 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
- [`add-array`], page 27 (function).
- [`add-vector`], page 27 (function).
- [`compatible-dimensions-p`], page 34 (generic function).
- [`inner-product-vector`], page 29 (function).
- [`nadd-array`], page 29 (function).
- [`nadd-vector`], page 30 (function).
- [`nsubtract-array`], page 30 (function).
- [`nsubtract-vector`], page 30 (function).
- [`product-array-array`], page 30 (function).

- [product-array-vector], page 30 (function).
- [product-vector-array], page 30 (function).
- [scaled-binary-op], page 50 (generic function).
- [subtract-array], page 31 (function).
- [subtract-vector], page 31 (function).

Internals

- [%array1<-array1-op-array2], page 65 (function).
- [%array<-array1-op-array2], page 65 (function).
- [%product-array-array], page 66 (function).
- [%product-array-vector], page 66 (function).
- [%product-vector-array], page 66 (function).
- [%scaled-product-array-array], page 67 (function).
- [%scaled-product-array-vector], page 67 (function).
- [%scaled-product-vector-array], page 67 (function).
- [%vector1<-vector1-op-vector2], page 68 (function).
- [%vector<-vector1-op-vector2], page 68 (function).

3.1.8 linear-algebra/kernel/rotation.lisp

Dependency
[unary-operations.lisp], page 8 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[kernel], page 5 (module).

Public Interface

- [givens-rotation], page 28 (function).
- [householder-reflection], page 29 (function).
- [jacobi-rotation], page 29 (function).

3.1.9 linear-algebra/kernel/gauss.lisp

Dependency
[pkgdcl.lisp], page 7 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[kernel], page 5 (module).

Public Interface

- [gauss-invert], page 28 (function).
- [gauss-solver], page 28 (function).

Internals

- [column-pivot], page 68 (function).
- [column-pivot-search], page 68 (function).
- [gauss-backsubstitution], page 68 (function).
- [gauss-factorization], page 68 (function).
- [gauss-update], page 68 (function).

- [`initialize-pivot-selection-vector`], page 68 (function).
- [`swap-rows`], page 68 (function).
- [`unit-pivot-value`], page 69 (function).

3.1.10 `linear-algebra/kernel/cholesky.lisp`

Dependency
[`unary-operations.lisp`], page 8 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
- [`hermitian-cholesky-decomposition`], page 28 (function).
- [`hermitian-cholesky-invert`], page 28 (function).
- [`hermitian-cholesky-solver`], page 28 (function).
- [`root-free-hermitian-cholesky-decomposition`], page 30 (function).
- [`root-free-symmetric-cholesky-decomposition`], page 31 (function).
- [`symmetric-cholesky-decomposition`], page 32 (function).
- [`symmetric-cholesky-invert`], page 32 (function).
- [`symmetric-cholesky-solver`], page 32 (function).

3.1.11 `linear-algebra/kernel/conjugate-gradient.lisp`

Dependency
[`binary-operations.lisp`], page 8 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
[`conjugate-gradient-solver`], page 28 (function).

Internals
- [`%default-cg-epsilon`], page 65 (function).
- [`%initialize-cg-residual`], page 65 (function).
- [`%initialize-cg-solution`], page 65 (function).
- [`%negative-residual`], page 66 (function).

3.1.12 `linear-algebra/kernel/tridiagonal.lisp`

Dependency
[`pkgdcl.lisp`], page 7 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`kernel`], page 5 (module).

Public Interface
[`tridiagonal-solver`], page 32 (function).

Internals
- [`tridiagonal-backsubstitution`], page 69 (function).

- [`tridiagonal-factorization`], page 69 (function).
- [`tridiagonal-update`], page 69 (function).

3.1.13 `linear-algebra/interface/fundamental-ops.lisp`

Source [`linear-algebra.asd`], page 7.

Parent Component
[`interface`], page 5 (module).

Public Interface
- [`add`], page 32 (generic function).
- [`invert`], page 35 (generic function).
- [`nadd`], page 40 (generic function).
- [`ninvert`], page 41 (generic function).
- [`norm`], page 42 (generic function).
- [`nscale`], page 43 (generic function).
- [`nsolve`], page 43 (generic function).
- [`nsubtract`], page 44 (generic function).
- [`ntranspose`], page 45 (generic function).
- [`permute`], page 46 (generic function).
- [`product`], page 47 (generic function).
- [`scale`], page 50 (generic function).
- [`solve`], page 51 (generic function).
- [`subtract`], page 53 (generic function).
- [`transpose`], page 55 (generic function).

3.1.14 `linear-algebra/interface/vector.lisp`

Dependency
[`fundamental-ops.lisp`], page 11 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`interface`], page 5 (module).

Public Interface
- [`apply-rotation`], page 33 (generic function).
- [`copy-vector`], page 35 (generic function).
- [`dovector`], page 27 (macro).
- [`make-vector`], page 29 (function).
- [`map-into-vector`], page 36 (generic function).
- [`map-vector`], page 36 (generic function).
- [`napply-rotation`], page 41 (generic function).
- [`replace-vector`], page 49 (generic function).
- [`subvector`], page 54 (generic function).
- [`(setf subvector)`], page 54 (generic function).
- [`vector-element-type`], page 56 (generic function).
- [`vector-in-bounds-p`], page 56 (generic function).

- [vector-length], page 56 (generic function).
- [vref], page 56 (generic function).
- [(setf vref)], page 57 (generic function).

3.1.15 linear-algebra/interface/matrix.lisp

Dependency
[fundamental-ops.lisp], page 11 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[interface], page 5 (module).

Public Interface

- [copy-matrix], page 35 (generic function).
- [make-matrix], page 29 (function).
- [matrix-column-dimension], page 37 (generic function).
- [matrix-dimensions], page 37 (generic function).
- [matrix-element-type], page 38 (generic function).
- [matrix-in-bounds-p], page 38 (generic function).
- [matrix-object], page 62 (class).
- [matrix-row-dimension], page 38 (generic function).
- [matrix-validated-range], page 29 (function).
- [matrixp], page 29 (function).
- [mref], page 39 (generic function).
- [(setf mref)], page 39 (generic function).
- [replace-matrix], page 49 (generic function).
- [submatrix], page 52 (generic function).
- [(setf submatrix)], page 52 (generic function).

Internals [initialize-matrix-contents], page 71 (generic function).

3.1.16 linear-algebra/interface/identity-matrix.lisp

Dependency
[matrix.lisp], page 12 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[interface], page 5 (module).

Public Interface

- [copy-matrix], page 35 (method).
- [identity-matrix], page 61 (class).
- [identity-matrix-p], page 29 (function).
- [initialize-instance], page 57 (method).
- [matrix-column-dimension], page 37 (reader method).
- [matrix-dimensions], page 37 (method).
- [matrix-element-type], page 38 (method).
- [matrix-in-bounds-p], page 38 (method).

- [matrix-row-dimension], page 39 (reader method).
- [mref], page 39 (method).

Internals

- [contents], page 70 (reader method).
- [size], page 73 (reader method).

3.1.17 linear-algebra/interface/permutation-matrix.lisp

Dependency
[matrix.lisp], page 12 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[interface], page 5 (module).

Public Interface

- [copy-matrix], page 35 (method).
- [initialize-instance], page 57 (method).
- [matrix-column-dimension], page 37 (method).
- [matrix-dimensions], page 37 (method).
- [matrix-element-type], page 38 (method).
- [matrix-in-bounds-p], page 38 (method).
- [matrix-row-dimension], page 39 (method).
- [mref], page 39 (method).
- [(setf mref)], page 40 (method).
- [permutation-matrix], page 62 (class).
- [permutation-matrix-p], page 30 (function).
- [transpose], page 56 (method).

Internals

- [%initialize-permutation-matrix-with-seq], page 66 (function).
- [contents], page 70 (reader method).
- [(setf contents)], page 71 (writer method).
- [initialize-matrix-contents], page 72 (method).
- [initialize-matrix-contents], page 72 (method).
- [initialize-matrix-contents], page 72 (method).

3.1.18 linear-algebra/sequence/list.lisp

Source [linear-algebra.asd], page 7.

Parent Component
[sequence], page 5 (module).

Public Interface

- [add], page 33 (method).
- [nadd], page 41 (method).
- [norm], page 42 (method).
- [nscale], page 43 (method).
- [nsubtract], page 45 (method).

- [ntranspose], page 46 (method).
- [permute], page 47 (method).
- [permute], page 47 (method).
- [product], page 48 (method).
- [scale], page 50 (method).
- [subtract], page 54 (method).
- [transpose], page 55 (method).

Internals

- [%norm], page 69 (method).
- [%norm], page 69 (method).
- [%norm], page 69 (method).
- [%norm], page 70 (method).

3.1.19 linear-algebra/sequence/vector.lisp

Source [linear-algebra.asd], page 7.

Parent Component
[sequence], page 5 (module).

Public Interface

- [add], page 33 (method).
- [nadd], page 41 (method).
- [norm], page 42 (method).
- [nscale], page 43 (method).
- [nsubtract], page 45 (method).
- [ntranspose], page 46 (method).
- [permute], page 47 (method).
- [permute], page 47 (method).
- [product], page 48 (method).
- [scale], page 50 (method).
- [subtract], page 54 (method).
- [transpose], page 55 (method).

3.1.20 linear-algebra/sequence/array.lisp

Source [linear-algebra.asd], page 7.

Parent Component
[sequence], page 5 (module).

Public Interface

- [add], page 33 (method).
- [compatible-dimensions-p], page 34 (method).
- [invert], page 36 (method).
- [nadd], page 41 (method).
- [ninvert], page 42 (method).
- [norm], page 42 (method).
- [nscale], page 43 (method).

- [nsolve], page 44 (method).
- [nsubtract], page 45 (method).
- [ntranspose], page 46 (method).
- [permute], page 47 (method).
- [permute], page 47 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [scale], page 50 (method).
- [solve], page 52 (method).
- [subtract], page 54 (method).
- [transpose], page 55 (method).

3.1.21 linear-algebra/data-vector.lisp

Dependency
[interface], page 5 (module).

Source [linear-algebra.asd], page 7.

Parent Component
[linear-algebra], page 3 (system).

Public Interface
- [add], page 33 (method).
- [add], page 33 (method).
- [add], page 33 (method).
- [apply-rotation], page 33 (method).
- [apply-rotation], page 33 (method).
- [column-vector], page 27 (function).
- [column-vector], page 57 (class).
- [column-vector-p], page 27 (function).
- [copy-vector], page 35 (method).
- [data-vector], page 58 (class).
- [initialize-instance], page 57 (method).
- [map-into-vector], page 36 (method).
- [map-into-vector], page 36 (method).
- [map-vector], page 36 (method).
- [map-vector], page 37 (method).
- [nadd], page 40 (method).
- [nadd], page 40 (method).
- [nadd], page 41 (method).
- [napply-rotation], page 41 (method).
- [napply-rotation], page 41 (method).
- [norm], page 42 (method).
- [nscale], page 43 (method).
- [nsubtract], page 45 (method).

- [nsubtract], page 45 (method).
- [nsubtract], page 45 (method).
- [ntranspose], page 45 (method).
- [ntranspose], page 45 (method).
- [permute], page 46 (method).
- [permute], page 46 (method).
- [permute], page 46 (method).
- [permute], page 47 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [replace-vector], page 50 (method).
- [row-vector], page 31 (function).
- [row-vector], page 63 (class).
- [row-vector-p], page 31 (function).
- [scale], page 50 (method).
- [subtract], page 53 (method).
- [subtract], page 53 (method).
- [subtract], page 53 (method).
- [subvector], page 54 (method).
- [(setf subvector)], page 54 (method).
- [transpose], page 55 (method).
- [transpose], page 55 (method).
- [vector-element-type], page 56 (method).
- [vector-in-bounds-p], page 56 (method).
- [vector-length], page 56 (method).
- [vref], page 56 (method).
- [(setf vref)], page 57 (method).

Internals

- [%map-data-vector], page 66 (function).
- [%map-into-data-vector], page 66 (function).
- [contents], page 70 (reader method).
- [(setf contents)], page 70 (writer method).

3.1.22 `linear-algebra/dense-matrix.lisp`

Dependency
[data-vector.lisp], page 15 (file).

Source [linear-algebra.asd], page 7.

Parent Component
[linear-algebra], page 3 (system).

Public Interface

- [add], page 32 (method).
- [add], page 33 (method).
- [compatible-dimensions-p], page 34 (method).

- [copy-matrix], page 35 (method).
- [dense-matrix], page 59 (class).
- [dense-matrix-p], page 28 (function).
- [initialize-instance], page 57 (method).
- [invert], page 36 (method).
- [matrix-column-dimension], page 37 (method).
- [matrix-dimensions], page 37 (method).
- [matrix-element-type], page 38 (method).
- [matrix-in-bounds-p], page 38 (method).
- [matrix-row-dimension], page 39 (method).
- [mref], page 39 (method).
- [(setf mref)], page 40 (method).
- [nadd], page 40 (method).
- [nadd], page 40 (method).
- [ninvert], page 42 (method).
- [norm], page 42 (method).
- [nscale], page 43 (method).
- [nsolve], page 44 (method).
- [nsolve], page 44 (method).
- [nsubtract], page 44 (method).
- [nsubtract], page 44 (method).
- [ntranspose], page 45 (method).
- [permute], page 46 (method).
- [permute], page 46 (method).
- [product], page 47 (method).
- [product], page 47 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [product], page 48 (method).
- [replace-matrix], page 49 (method).
- [scale], page 50 (method).
- [solve], page 51 (method).
- [solve], page 51 (method).
- [submatrix], page 52 (method).
- [(setf submatrix)], page 53 (method).
- [subtract], page 53 (method).
- [subtract], page 53 (method).
- [transpose], page 55 (method).

Internals

- [contents], page 70 (reader method).
- [(setf contents)], page 70 (writer method).

- [`initialize-matrix-contents`], page 72 (method).
- [`initialize-matrix-contents`], page 72 (method).
- [`initialize-matrix-contents`], page 72 (method).
- [`initialize-matrix-contents`], page 72 (method).

3.1.23 `linear-algebra/square-matrix.lisp`

Dependency
[`dense-matrix.lisp`], page 16 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Public Interface

- [`compatible-dimensions-p`], page 34 (method).
- [`invert`], page 35 (method).
- [`ninvert`], page 42 (method).
- [`square-matrix`], page 64 (class).
- [`square-matrix-p`], page 31 (function).
- [`submatrix`], page 52 (method).

Internals [`initialize-matrix-contents`], page 72 (method).

3.1.24 `linear-algebra/hermitian-matrix.lisp`

Dependency
[`square-matrix.lisp`], page 18 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Public Interface

- [`hermitian-matrix`], page 61 (class).
- [`hermitian-matrix-p`], page 28 (function).
- [`invert`], page 35 (method).
- [`(setf mref)`], page 40 (method).
- [`ninvert`], page 41 (method).
- [`nsolve`], page 44 (method).
- [`ntranspose`], page 45 (method).
- [`permute`], page 46 (method).
- [`permute`], page 46 (method).
- [`replace-matrix`], page 49 (method).
- [`replace-matrix`], page 49 (method).
- [`solve`], page 51 (method).
- [`submatrix`], page 52 (method).
- [`(setf submatrix)`], page 53 (method).
- [`(setf submatrix)`], page 53 (method).
- [`transpose`], page 55 (method).

Internals

- [%initialize-hermitian-matrix-with-seq], page 65 (function).
- [%replace-hermitian-matrix-off-diagonal], page 66 (function).
- [%replace-hermitian-matrix-on-diagonal], page 67 (function).
- [%setf-hermitian-submatrix-off-diagonal], page 67 (function).
- [%setf-hermitian-submatrix-on-diagonal], page 67 (function).
- [initialize-matrix-contents], page 71 (method).
- [initialize-matrix-contents], page 71 (method).
- [initialize-matrix-contents], page 71 (method).
- [initialize-matrix-contents], page 72 (method).

3.1.25 `linear-algebra/symmetric-matrix.lisp`

Dependency
[`square-matrix.lisp`], page 18 (file).

Source [`linear-algebra.asd`], page 7.

Parent Component
[`linear-algebra`], page 3 (system).

Public Interface

- [invert], page 35 (method).
- [(setf mref)], page 39 (method).
- [nadd], page 40 (method).
- [nadd], page 40 (method).
- [ninvert], page 41 (method).
- [nsolve], page 44 (method).
- [nsubtract], page 44 (method).
- [nsubtract], page 44 (method).
- [replace-matrix], page 49 (method).
- [replace-matrix], page 49 (method).
- [solve], page 51 (method).
- [submatrix], page 52 (method).
- [(setf submatrix)], page 52 (method).
- [(setf submatrix)], page 52 (method).
- [symmetric-matrix], page 64 (class).
- [symmetric-matrix-p], page 32 (function).

Internals

- [%initialize-symmetric-matrix-with-seq], page 66 (function).
- [%replace-symmetric-matrix-off-diagonal], page 67 (function).
- [%replace-symmetric-matrix-on-diagonal], page 67 (function).
- [%setf-symmetric-submatrix-off-diagonal], page 67 (function).
- [%setf-symmetric-submatrix-on-diagonal], page 68 (function).
- [initialize-matrix-contents], page 71 (method).
- [initialize-matrix-contents], page 71 (method).
- [initialize-matrix-contents], page 71 (method).

4 Packages

Packages are listed by definition order.

4.1 linear-algebra-kernel

Source [pkgdcl.lisp], page 7.

Use List

- common-lisp.
- floating-point.

Used By List
[linear-algebra], page 23.

Public Interface

- [add-array], page 27 (function).
- [add-vector], page 27 (function).
- [common-array-element-type], page 27 (function).
- [common-class-of], page 27 (function).
- [compatible-dimensions-p], page 34 (generic function).
- [complex-equal], page 27 (function).
- [conjugate-gradient-solver], page 28 (function).
- [copy-array], page 34 (generic function).
- [gauss-invert], page 28 (function).
- [gauss-solver], page 28 (function).
- [givens-rotation], page 28 (function).
- [hermitian-cholesky-decomposition], page 28 (function).
- [hermitian-cholesky-invert], page 28 (function).
- [hermitian-cholesky-solver], page 28 (function).
- [householder-reflection], page 29 (function).
- [inner-product-vector], page 29 (function).
- [jacobi-rotation], page 29 (function).
- [left-permute], page 36 (generic function).
- [nadd-array], page 29 (function).
- [nadd-vector], page 30 (function).
- [norm-array], page 42 (generic function).
- [norm-vector], page 43 (generic function).
- [nsubtract-array], page 30 (function).
- [nsubtract-vector], page 30 (function).
- [number-equal], page 30 (function).
- [product-array-array], page 30 (function).
- [product-array-vector], page 30 (function).
- [product-vector-array], page 30 (function).
- [right-permute], page 50 (generic function).
- [root-free-hermitian-cholesky-decomposition], page 30 (function).

- [root-free-symmetric-cholesky-decomposition], page 31 (function).
- [scaled-binary-op], page 50 (generic function).
- [specific-array-element-type], page 31 (function).
- [subtract-array], page 31 (function).
- [subtract-vector], page 31 (function).
- [sump], page 54 (generic function).
- [sumsq], page 55 (generic function).
- [sumsq-column], page 31 (function).
- [sumsq-row], page 31 (function).
- [sumsq2], page 32 (function).
- [sumsq3], page 32 (function).
- [symmetric-cholesky-decomposition], page 32 (function).
- [symmetric-cholesky-invert], page 32 (function).
- [symmetric-cholesky-solver], page 32 (function).
- [tridiagonal-solver], page 32 (function).

Internals

- [%abs-vector], page 65 (function).
- [%array1<-array1-op-array2], page 65 (function).
- [%array<-array1-op-array2], page 65 (function).
- [%default-cg-epsilon], page 65 (function).
- [%initialize-cg-residual], page 65 (function).
- [%initialize-cg-solution], page 65 (function).
- [%negative-residual], page 66 (function).
- [%product-array-array], page 66 (function).
- [%product-array-vector], page 66 (function).
- [%product-vector-array], page 66 (function).
- [%scaled-product-array-array], page 67 (function).
- [%scaled-product-array-vector], page 67 (function).
- [%scaled-product-vector-array], page 67 (function).
- [%vector1<-vector1-op-vector2], page 68 (function).
- [%vector<-vector1-op-vector2], page 68 (function).
- [column-pivot], page 68 (function).
- [column-pivot-search], page 68 (function).
- [gauss-backsubstitution], page 68 (function).
- [gauss-factorization], page 68 (function).
- [gauss-update], page 68 (function).
- [initialize-pivot-selection-vector], page 68 (function).
- [swap-rows], page 68 (function).
- [tridiagonal-backsubstitution], page 69 (function).
- [tridiagonal-factorization], page 69 (function).
- [tridiagonal-update], page 69 (function).
- [unit-pivot-value], page 69 (function).
- [zero-array], page 69 (function).
- [zero-vector], page 69 (function).

4.2 linear-algebra

Source [pkgdcl.lisp], page 7.

Use List

- common-lisp.
- floating-point.
- [linear-algebra-kernel], page 21.

Public Interface

- [add], page 32 (generic function).
- [apply-rotation], page 33 (generic function).
- [column-vector], page 27 (function).
- [column-vector], page 57 (class).
- [column-vector-p], page 27 (function).
- [copy-matrix], page 35 (generic function).
- [copy-vector], page 35 (generic function).
- [data-vector], page 58 (class).
- [dense-matrix], page 59 (class).
- [dense-matrix-p], page 28 (function).
- [dovector], page 27 (macro).
- [hermitian-matrix], page 61 (class).
- [hermitian-matrix-p], page 28 (function).
- [identity-matrix], page 61 (class).
- [identity-matrix-p], page 29 (function).
- [invert], page 35 (generic function).
- [make-matrix], page 29 (function).
- [make-vector], page 29 (function).
- [map-into-vector], page 36 (generic function).
- [map-vector], page 36 (generic function).
- [matrix-column-dimension], page 37 (generic function).
- [matrix-dimensions], page 37 (generic function).
- [matrix-element-type], page 38 (generic function).
- [matrix-in-bounds-p], page 38 (generic function).
- [matrix-object], page 62 (class).
- [matrix-row-dimension], page 38 (generic function).
- [matrix-validated-range], page 29 (function).
- [matrixp], page 29 (function).
- [mref], page 39 (generic function).
- [(setf mref)], page 39 (generic function).
- [nadd], page 40 (generic function).
- [napply-rotation], page 41 (generic function).
- [ninvert], page 41 (generic function).
- [norm], page 42 (generic function).
- [nscale], page 43 (generic function).

- [nsolve], page 43 (generic function).
- [nsubtract], page 44 (generic function).
- [ntranspose], page 45 (generic function).
- [permutation-matrix], page 62 (class).
- [permutation-matrix-p], page 30 (function).
- [permute], page 46 (generic function).
- [product], page 47 (generic function).
- [replace-matrix], page 49 (generic function).
- [replace-vector], page 49 (generic function).
- [row-vector], page 31 (function).
- [row-vector], page 63 (class).
- [row-vector-p], page 31 (function).
- [scale], page 50 (generic function).
- [solve], page 51 (generic function).
- [square-matrix], page 64 (class).
- [square-matrix-p], page 31 (function).
- [submatrix], page 52 (generic function).
- [(setf submatrix)], page 52 (generic function).
- [subtract], page 53 (generic function).
- [subvector], page 54 (generic function).
- [(setf subvector)], page 54 (generic function).
- [symmetric-matrix], page 64 (class).
- [symmetric-matrix-p], page 32 (function).
- [transpose], page 55 (generic function).
- [vector-element-type], page 56 (generic function).
- [vector-in-bounds-p], page 56 (generic function).
- [vector-length], page 56 (generic function).
- [vref], page 56 (generic function).
- [(setf vref)], page 57 (generic function).

Internals

- [%initialize-hermitian-matrix-with-seq], page 65 (function).
- [%initialize-permutation-matrix-with-seq], page 66 (function).
- [%initialize-symmetric-matrix-with-seq], page 66 (function).
- [%map-data-vector], page 66 (function).
- [%map-into-data-vector], page 66 (function).
- [%norm], page 69 (generic function).
- [%replace-hermitian-matrix-off-diagonal], page 66 (function).
- [%replace-hermitian-matrix-on-diagonal], page 67 (function).
- [%replace-symmetric-matrix-off-diagonal], page 67 (function).
- [%replace-symmetric-matrix-on-diagonal], page 67 (function).
- [%setf-hermitian-submatrix-off-diagonal], page 67 (function).
- [%setf-hermitian-submatrix-on-diagonal], page 67 (function).

- [`%setf-symmetric-submatrix-off-diagonal`], page 67 (function).
- [`%setf-symmetric-submatrix-on-diagonal`], page 68 (function).
- [`contents`], page 70 (generic reader).
- [`(setf contents)`], page 70 (generic writer).
- [`initialize-matrix-contents`], page 71 (generic function).
- [`size`], page 72 (generic reader).

5 Definitions

Definitions are sorted by export status, category, package, and then by lexicographic order.

5.1 Public Interface

5.1.1 Macros

`dovector` ((*element vector* **&optional** *result)* **&body** *body*) [Macro]

Iterate over vector returning result.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

5.1.2 Ordinary functions

`add-array` (*array1 array2 scalar1 scalar2*) [Function]

Array binary addition.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`add-vector` (*vector1 vector2 scalar1 scalar2*) [Function]

Vector binary addition.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`column-vector` (**&rest** *numbers*) [Function]

Create a column vector from the numbers.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`column-vector-p` (*object*) [Function]

Return true if object is a column-vector, NIL otherwise.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`common-array-element-type` (*array1 array2*) [Function]

Return the array type common to both arrays.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`common-class-of` (*object1 object2*) [Function]

Return the common class of the 2 objects or default-class.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`complex-equal` (*complex1 complex2* **&optional** *epsilon*) [Function]

Return true if both numbers are complex and equal.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`conjugate-gradient-solver` (*array vector* **&optional** *epsilon limit*) [Function]

Linear system solver using the conjugate gradient method.

Package [`linear-algebra-kernel`], page 21.

Source [`conjugate-gradient.lisp`], page 10.

`dense-matrix-p` (*object*) [Function]

Return true if object is a dense matrix.

Package [`linear-algebra`], page 23.

Source [`dense-matrix.lisp`], page 16.

`gauss-invert` (*array*) [Function]

Find A^-1 via Gauss algorithm with partial column pivot search.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`gauss-solver` (*array vector*) [Function]

Gauss algorithm with column pivot search.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`givens-rotation` (*f g*) [Function]

Return c,s,r defined from the Givens rotation.

Package [`linear-algebra-kernel`], page 21.

Source [`rotation.lisp`], page 9.

`hermitian-cholesky-decomposition` (*array*) [Function]

Factor A = LL^T.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`hermitian-cholesky-invert` (*array*) [Function]

Invert a positive definite matrices using the root-free Cholesky decomposition.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`hermitian-cholesky-solver` (*array vector*) [Function]

Linear system solver for positive definite matrices using the root-free Cholesky decomposition.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`hermitian-matrix-p` (*object*) [Function]

Return true if object is a hermitian-matrix, NIL otherwise.

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`householder-reflection` (*alpha vector*) [Function]

Return Beta, Tau and the Householder vector.

Package [`linear-algebra-kernel`], page 21.

Source [`rotation.lisp`], page 9.

`identity-matrix-p` (*object*) [Function]

Return true if object is an identity-matrix.

Package [`linear-algebra`], page 23.

Source [`identity-matrix.lisp`], page 12.

`inner-product-vector` (*vector1 vector2 scalar*) [Function]

Return the vector inner product.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`jacobi-rotation` (*x y z*) [Function]

Return a, b, cos(theta) and sin(theta) terms from the Jacobi rotation.

Package [`linear-algebra-kernel`], page 21.

Source [`rotation.lisp`], page 9.

`make-matrix` (*rows columns* **&key** *matrix-type element-type initial-element initial-contents*) [Function]

Return a new matrix instance.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

`make-vector` (*size* **&key** *vector-type element-type initial-element initial-contents*) [Function]

Create the data structure to represent a vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

`matrix-validated-range` (*matrix start-row start-column* **&optional** *end-row end-column*) [Function]

Returns a validated range of rows and columns for the matrix.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

`matrixp` (*object*) [Function]

Return true if object is a matrix, NIL otherwise.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

`nadd-array` (*array1 array2 scalar1 scalar2*) [Function]

Destructive array binary addition.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`nadd-vector` (*vector1 vector2 scalar1 scalar2*) [Function]
Destructive vector binary addition.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`nsubtract-array` (*array1 array2 scalar1 scalar2*) [Function]
Destructive array binary subtraction.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`nsubtract-vector` (*vector1 vector2 scalar1 scalar2*) [Function]
Destructive vector binary subtraction.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`number-equal` (*number1 number2* **&optional** *epsilon*) [Function]
Return true if the numbers are equal using the appropriate comparison.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`permutation-matrix-p` (*object*) [Function]
Return true if object is a permutation-matrix.

Package [`linear-algebra`], page 23.

Source [`permutation-matrix.lisp`], page 13.

`product-array-array` (*array1 array2* **&optional** *scalar result*) [Function]
Return the scaled result of the product of 2 arrays.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`product-array-vector` (*array vector* **&optional** *scalar result*) [Function]
Return the result of the array postmultiplied by the vector and scaled.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`product-vector-array` (*vector array* **&optional** *scalar result*) [Function]
Return the result of the array premultiplied by the vector and scaled.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`root-free-hermitian-cholesky-decomposition` (*array*) [Function]
Factor A = LDL^t.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`root-free-symmetric-cholesky-decomposition` (*array*) [Function]
Factor A = LDL^t.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`row-vector` (**&rest** *numbers*) [Function]
Create a row vector from the numbers.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`row-vector-p` (*object*) [Function]
Return true if object is a row-vector, NIL otherwise.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`specific-array-element-type` (*array* **&rest** *subscripts*) [Function]
Return the specific type of the element specified by subscripts.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`square-matrix-p` (*object*) [Function]
Return true if OBJECT is a square matrix.

Package [`linear-algebra`], page 23.

Source [`square-matrix.lisp`], page 18.

`subtract-array` (*array1 array2 scalar1 scalar2*) [Function]
Array binary subtraction.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`subtract-vector` (*vector1 vector2 scalar1 scalar2*) [Function]
Vector binary subtraction.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`sumsq-column` (*array column* **&key** *scale sumsq start end*) [Function]
Return the scaling parameter and the sum of the squares of the array column.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

`sumsq-row` (*array row* **&key** *scale sumsq start end*) [Function]
Return the scaling parameter and the sum of the squares of the array row.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

`sumsq2` (*x y*) [Function]
Return the square root of `|x|^2 + |y|^2`.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

`sumsq3` (*x y z*) [Function]
Return the square root of `|x|^2 + |y|^2 + |z|^2`.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

`symmetric-cholesky-decomposition` (*array*) [Function]
Factor A = LL^T.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`symmetric-cholesky-invert` (*array*) [Function]
Invert a positive definite matrices using the root-free Cholesky decomposition.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`symmetric-cholesky-solver` (*array vector*) [Function]
Linear system solver for positive definite matrices using the root-free Cholesky decomposition.

Package [`linear-algebra-kernel`], page 21.

Source [`cholesky.lisp`], page 10.

`symmetric-matrix-p` (*object*) [Function]
Return true if object is a symmetric-matrix, NIL otherwise.

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`tridiagonal-solver` (*array vector*) [Function]
Linear equation solver for a tridiagonal matrix.

Package [`linear-algebra-kernel`], page 21.

Source [`tridiagonal.lisp`], page 10.

5.1.3 Generic functions

`add` (*vector-or-matrix-1 vector-or-matrix-2* **&key** *scalar1 scalar2*) [Generic Function]
Vector or matrix binary addition.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`add` ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]
Return the addition of the 2 matrices.

Source [`dense-matrix.lisp`], page 16.

`add` *:before ((matrix1 [*`dense-matrix`*], page 59) (matrix2 [*`dense-matrix`*], page 59)* **&key** *scalar1 scalar2)* [Method]

Audit the input data.

Source [`dense-matrix.lisp`], page 16.

`add` *((vector1 [*`row-vector`*], page 63) (vector2 [*`row-vector`*], page 63)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar1*vector1 with scalar2*vector2.

Source [`data-vector.lisp`], page 15.

`add` *((vector1 [*`column-vector`*], page 57) (vector2 [*`column-vector`*], page 57)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar1*vector1 with scalar2*vector2.

Source [`data-vector.lisp`], page 15.

`add` *:before ((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58)* **&key** *scalar1 scalar2)* [Method]

Verify that the dimensions are equal.

Source [`data-vector.lisp`], page 15.

`add` *((array1* `array`*) (array2* `array`*)* **&key** *scalar1 scalar2)* [Method]

Return the addition of the 2 arrays.

Source [`array.lisp`], page 14.

`add` *((vector1* `vector`*) (vector2* `vector`*)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar1*vector1 with scalar2*vector2

Source [`vector.lisp`], page 14.

`add` *((list1* `list`*) (list2* `list`*)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar1*list1 with scalar2*list2

Source [`list.lisp`], page 13.

`apply-rotation` *(vector1 vector2 cc ss)* [Generic Function]

Return the plane rotations of vector1 and vector2 by cc and ss.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`apply-rotation` *((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58) cc ss)* [Method]

Return the plane rotations of vector1 and vector2 by cc and ss.

Source [`data-vector.lisp`], page 15.

`apply-rotation` *:before ((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58) cc ss)* [Method]

Verify the input to apply-rotation.

Source [`data-vector.lisp`], page 15.

`compatible-dimensions-p` (*operation vector-or-matrix-1 vector-or-matrix-2*) [Generic Function]

Return true if the vector and matrix dimensions are compatible for the operation.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

Methods

`compatible-dimensions-p` ((*operation* `(eql :solve)`) (*matrix [*`square-matrix`*], page 64*) (*vector [*`column-vector`*], page 57*)) [Method]

Return true if the array dimensions are compatible for product.

Source [`square-matrix.lisp`], page 18.

`compatible-dimensions-p` ((*operation* `(eql :solve)`) (*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*)) [Method]

Return true if the array dimensions are compatible for product.

Source [`dense-matrix.lisp`], page 16.

`compatible-dimensions-p` ((*operation* `(eql :solve)`) (*array* `array`) (*vector* `vector`)) [Method]

Return true if the array dimensions are compatible for product.

Source [`array.lisp`], page 14.

`compatible-dimensions-p` ((*operation* `(eql :product)`) (*array1* `array`) (*array2* `array`)) [Method]

Return true if the array dimensions are compatible for product.

`compatible-dimensions-p` ((*operation* `(eql :add)`) (*array1* `array`) (*array2* `array`)) [Method]

Return true if the array dimensions are compatible for an addition.

`compatible-dimensions-p` ((*operation* `(eql :product)`) (*array* `array`) (*vector* `vector`)) [Method]

Return true if the array dimensions are compatible for product.

`compatible-dimensions-p` ((*operation* `(eql :product)`) (*vector* `vector`) (*array* `array`)) [Method]

Return true if the array dimensions are compatible for product.

`compatible-dimensions-p` ((*operation* `(eql :add)`) (*vector1* `vector`) (*vector2* `vector`)) [Method]

Return true if the vector dimensions are compatible for an addition.

`copy-array` (*array*) [Generic Function]

Return an element-wise copy of the original array.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

Methods

`copy-array` ((*original* `array`)) [Method]

Return an element-wise copy of the original array.

`copy-array` ((*original* `vector`)) [Method]

Return an element-wise copy of the original vector.

`copy-matrix` (*matrix*) [Generic Function]

Return a copy of the matrix.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`copy-matrix` ((*matrix [*`dense-matrix`*], page 59*)) [Method]

Return a copy of the dense matrix.

Source [`dense-matrix.lisp`], page 16.

`copy-matrix` ((*matrix [*`permutation-matrix`*], page 62*)) [Method]

Return a copy of the permutation matrix.

Source [`permutation-matrix.lisp`], page 13.

`copy-matrix` ((*matrix [*`identity-matrix`*], page 61*)) [Method]

Return a copy of the matrix.

Source [`identity-matrix.lisp`], page 12.

`copy-vector` (*vector*) [Generic Function]

Return a copy of the vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`copy-vector` ((*vector [*`data-vector`*], page 58*)) [Method]

Return a copy of the vector.

Source [`data-vector.lisp`], page 15.

`invert` (*matrix*) [Generic Function]

Return the invert of the matrix.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`invert` ((*matrix [*`symmetric-matrix`*], page 64*)) [Method]

Return the invert of the symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`invert` ((*matrix [*`hermitian-matrix`*], page 61*)) [Method]

Return the invert of the hermitian matrix.

Source [`hermitian-matrix.lisp`], page 18.

`invert` ((*matrix [*`square-matrix`*], page 64*)) [Method]

Return the invert of the square matrix.

Source [`square-matrix.lisp`], page 18.

`invert` ((*matrix* [`dense-matrix`], *page 59*)) [Method]
Return the invert of the dense matrix.

Source [`dense-matrix.lisp`], page 16.

`invert` ((*array* `array`)) [Method]
Return the invert of the array.

Source [`array.lisp`], page 14.

`left-permute` (*permutation vector-or-array*) [Generic Function]
Permute the column vector or rows of the array.

Package [`linear-algebra-kernel`], page 21.

Source [`permute.lisp`], page 8.

Methods

`left-permute` ((*permutation* `vector`) (*data* `array`)) [Method]
Permute the rows of the array.

`left-permute` ((*permutation* `vector`) (*data* `vector`)) [Method]
Permute the column vector to create a row vector.

`map-into-vector` (*result-vector function* **&rest** *vectors*) [Generic Function]
Destructively modifies the result vector with the result of applying the function to each element of the vectors.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`map-into-vector` ((*result-vector* [`data-vector`], *page 58*) (*function* `function`) **&rest** *vectors*) [Method]
Destructively modifies the result vector with the result of applying the function to each element of the vectors.

Source [`data-vector.lisp`], page 15.

`map-into-vector` *:before* ((*result-vector* [`data-vector`], *page 58*) (*function* `function`) **&rest** *vectors*) [Method]
Verify the arguments to map-into-vector.

Source [`data-vector.lisp`], page 15.

`map-vector` (*result-type function first-vector* **&rest** *more-vectors*) [Generic Function]
Calls function on successive sets of vector objects.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`map-vector` (*result-type* (*function* `function`) (*first-vector* [`data-vector`], *page 58*) **&rest** *more-vectors*) [Method]
Calls function on successive sets of data vectors.

Source [`data-vector.lisp`], page 15.

`map-vector` *:before* (*result-type* (*function* `function`) (*first-vector* [`data-vector`], *page 58*) **&rest** *more-vectors*) [Method]

Verify the arguments to map-vector.

Source [`data-vector.lisp`], page 15.

`matrix-column-dimension` (*matrix*) [Generic Function]

Return the number of columns in MATRIX.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`matrix-column-dimension` ((*matrix* [`dense-matrix`], *page 59*)) [Method]

Return the number of columns in matrix.

Source [`dense-matrix.lisp`], page 16.

`matrix-column-dimension` ((*matrix* [`permutation-matrix`], *page 62*)) [Method]

Return the number of columns in matrix.

Source [`permutation-matrix.lisp`], page 13.

`matrix-column-dimension` ((*identity-matrix* [`identity-matrix`], *page 61*)) [Reader Method]

automatically generated reader method

Source [`identity-matrix.lisp`], page 12.

Target Slot

[`size`], page 62.

`matrix-dimensions` (*matrix*) [Generic Function]

Return the number of rows and columns in MATRIX.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`matrix-dimensions` ((*matrix* [`dense-matrix`], *page 59*)) [Method]

Return the number of rows and columns in matrix.

Source [`dense-matrix.lisp`], page 16.

`matrix-dimensions` ((*matrix* [`permutation-matrix`], *page 62*)) [Method]

Return the number of rows and columns in matrix.

Source [`permutation-matrix.lisp`], page 13.

`matrix-dimensions` ((*matrix* [`identity-matrix`], *page 61*)) [Method]

Return the number of rows and columns in matrix.

Source [`identity-matrix.lisp`], page 12.

`matrix-element-type` (*matrix*) [Generic Function]

Return the element type of MATRIX.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

> `matrix-element-type` ((*matrix [*`dense-matrix`*], page 59*)) [Method]
>
> Return the element type of the matrix.
>
> **Source** [`dense-matrix.lisp`], page 16.
>
> `matrix-element-type` ((*matrix [*`permutation-matrix`*], page 62*)) [Method]
>
> Element type of the permutation matrix.
>
> **Source** [`permutation-matrix.lisp`], page 13.
>
> `matrix-element-type` ((*matrix [*`identity-matrix`*], page 61*)) [Method]
>
> Return the element type of the identity matrix.
>
> **Source** [`identity-matrix.lisp`], page 12.

`matrix-in-bounds-p` (*matrix row column*) [Generic Function]

Return true if ROW and COLUMN do not exceed the dimensions of MATRIX.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

> `matrix-in-bounds-p` ((*matrix [*`dense-matrix`*], page 59*) (*row* `integer`) (*column* `integer`)) [Method]
>
> Return true if row and column do not exceed the dimensions of matrix.
>
> **Source** [`dense-matrix.lisp`], page 16.
>
> `matrix-in-bounds-p` ((*matrix [*`permutation-matrix`*], page 62*) (*row* `integer`) (*column* `integer`)) [Method]
>
> Return true if row and column do not exceed the dimensions of matrix.
>
> **Source** [`permutation-matrix.lisp`], page 13.
>
> `matrix-in-bounds-p` ((*matrix [*`identity-matrix`*], page 61*) (*row* `integer`) (*column* `integer`)) [Method]
>
> Return true if row and column do not exceed the dimensions of matrix.
>
> **Source** [`identity-matrix.lisp`], page 12.

`matrix-row-dimension` (*matrix*) [Generic Function]

Return the number of rows in MATRIX.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`matrix-row-dimension` ((*matrix [*`dense-matrix`*], page 59*)) [Method]
Return the number of rows in matrix.

Source [`dense-matrix.lisp`], page 16.

`matrix-row-dimension` ((*matrix [*`permutation-matrix`*], page 62*)) [Method]
Return the number of rows in matrix.

Source [`permutation-matrix.lisp`], page 13.

`matrix-row-dimension` ((*identity-matrix [*`identity-matrix`*], page 61*)) [Reader Method]
automatically generated reader method

Source [`identity-matrix.lisp`], page 12.

Target Slot
[`size`], page 62.

`mref` (*matrix row column*) [Generic Function]
Return the matrix element at ROW,COLUMN.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`mref` ((*matrix [*`dense-matrix`*], page 59*) (*row* `integer`) (*column* `integer`)) [Method]
Return the element of matrix at row,column.

Source [`dense-matrix.lisp`], page 16.

`mref` ((*matrix [*`permutation-matrix`*], page 62*) (*row* `integer`) (*column* `integer`)) [Method]
Return 1 if a permutation and 0 otherwise.

Source [`permutation-matrix.lisp`], page 13.

`mref` ((*matrix [*`identity-matrix`*], page 61*) (*row* `integer`) (*column* `integer`)) [Method]
Return the element of the matrix at row,column.

Source [`identity-matrix.lisp`], page 12.

`(setf mref)` (*matrix row column*) [Generic Function]
Set the element at row,column of matrix to data.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`(setf mref)` ((*matrix [*`symmetric-matrix`*], page 64*) (*row* `integer`) (*column* `integer`)) [Method]
Set the element of matrix at row,column.

Source [`symmetric-matrix.lisp`], page 19.

`(setf mref)` ((*matrix [*`hermitian-matrix`*], page 61*) (*row* `integer`) (*column* `integer`)) [Method]

Set the element at row,column of matrix to data.

Source [`hermitian-matrix.lisp`], page 18.

`(setf mref)` ((*matrix [*`dense-matrix`*], page 59*) (*row* `integer`) (*column* `integer`)) [Method]

Set the element of matrix at row,column.

Source [`dense-matrix.lisp`], page 16.

`(setf mref)` ((*matrix [*`permutation-matrix`*], page 62*) (*row* `integer`) (*column* `integer`)) [Method]

Swap rows of the permutation matrix.

Source [`permutation-matrix.lisp`], page 13.

`nadd` (*vector-or-matrix-1 vector-or-matrix-2* **&key** *scalar1 scalar2*) [Generic Function]

Destructive vector or matrix addition.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`nadd` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`symmetric-matrix`*], page 64*) **&key** *scalar1 scalar2*) [Method]

Source [`symmetric-matrix.lisp`], page 19.

`nadd` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Generate an error if a non-symmetric matrix is destructively added to a symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`nadd` ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Return the addition of the 2 matrices.

Source [`dense-matrix.lisp`], page 16.

`nadd` *:before* ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Audit the input data.

Source [`dense-matrix.lisp`], page 16.

`nadd` ((*vector1 [*`row-vector`*], page 63*) (*vector2 [*`row-vector`*], page 63*) **&key** *scalar1 scalar2*) [Method]

Return the addition of scalar2*vector2 to scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`nadd` ((*vector1 [*`column-vector`*], page 57*) (*vector2 [*`column-vector`*], page 57*) **&key** *scalar1 scalar2*) [Method]

Return the addition of scalar2*vector2 to scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`nadd` *:before ((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58)* **&key** *scalar1 scalar2)* [Method]

Verify that the dimensions are equal.

Source [`data-vector.lisp`], page 15.

`nadd` *((array1* `array`*) (array2* `array`*)* **&key** *scalar1 scalar2)* [Method]

Destructively add array2 to array1.

Source [`array.lisp`], page 14.

`nadd` *((vector1* `vector`*) (vector2* `vector`*)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar2*vector2 to scalar1*vector1.

Source [`vector.lisp`], page 14.

`nadd` *((list1* `list`*) (list2* `list`*)* **&key** *scalar1 scalar2)* [Method]

Return the addition of scalar2*list2 to scalar1*list1.

Source [`list.lisp`], page 13.

`napply-rotation` *(vector1 vector2 cc ss)* [Generic Function]

Return the plane rotations of vector1 and vector2 by cc and ss.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`napply-rotation` *((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58) cc ss)* [Method]

Return the plane rotations of vector1 and vector2 by cc and ss.

Source [`data-vector.lisp`], page 15.

`napply-rotation` *:before ((vector1 [*`data-vector`*], page 58) (vector2 [*`data-vector`*], page 58) cc ss)* [Method]

Verify the input to napply-rotation.

Source [`data-vector.lisp`], page 15.

`ninvert` *(matrix)* [Generic Function]

Return the invert of the matrix with in-place decomposition.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`ninvert` *((matrix [*`symmetric-matrix`*], page 64))* [Method]

Return the invert of the symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`ninvert` *((matrix [*`hermitian-matrix`*], page 61))* [Method]

Return the invert of the hermitian matrix.

Source [`hermitian-matrix.lisp`], page 18.

`ninvert` ((*matrix [*`square-matrix`*], page 64*)) [Method]
Return the invert of the square matrix.

Source [`square-matrix.lisp`], page 18.

`ninvert` ((*matrix [*`dense-matrix`*], page 59*)) [Method]
Return the invert of the dense matrix.

Source [`dense-matrix.lisp`], page 16.

`ninvert` ((*array* `array`)) [Method]
Return the invert of the array.

Source [`array.lisp`], page 14.

`norm` (*vector-or-matrix* **&optional** *measure*) [Generic Function]
Return the norm according to measure.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`norm` ((*matrix [*`dense-matrix`*], page 59*) **&optional** *measure*) [Method]
Return the norm of the matrix.

Source [`dense-matrix.lisp`], page 16.

`norm` ((*vector [*`data-vector`*], page 58*) **&optional** *measure*) [Method]
Return the p-norm of the vector.

Source [`data-vector.lisp`], page 15.

`norm` ((*data* `array`) **&optional** *measure*) [Method]
Return the norm of the array.

Source [`array.lisp`], page 14.

`norm` ((*data* `vector`) **&optional** *measure*) [Method]
Source [`vector.lisp`], page 14.

`norm` ((*data* `list`) **&optional** *measure*) [Method]
Source [`list.lisp`], page 13.

`norm-array` (*data measure*) [Generic Function]
Return the norm of the array according to the measure.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

Methods

`norm-array` ((*data* `array`) (*measure* (`eql :infinity`))) [Method]
Return the infinity norm of the array.

`norm-array` ((*data* `array`) (*measure* (`eql :frobenius`))) [Method]
Return the Frobenius norm of the array.

`norm-array` ((*data* `array`) (*measure* (`eql :max`))) [Method]
Return the max norm of the array.

`norm-array` ((*data* `array`) (*measure* (`eql 1`))) [Method]
Return the 1 norm of the array.

`norm-vector` (*data measure*) [Generic Function]
Return the norm of the vector according to the measure.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

Methods

`norm-vector` ((*data* `vector`) (*measure* (`eql :infinity`))) [Method]
Return the infinity, or maximum, norm of vector.

`norm-vector` ((*data* `vector`) (*measure* `integer`)) [Method]
Return the p-norm of the vector.

`norm-vector` ((*data* `vector`) (*measure* (`eql 2`))) [Method]
Return the Euclidean norm of the vector.

`norm-vector` ((*data* `vector`) (*measure* (`eql 1`))) [Method]
Return the Taxicab norm of the list.

`nscale` (*scalar vector-or-matrix*) [Generic Function]
Destructively scale each element by the scalar.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`nscale` ((*scalar* `number`) (*matrix [*`dense-matrix`*], page 59*)) [Method]
Scale each element of the dense matrix.

Source [`dense-matrix.lisp`], page 16.

`nscale` ((*scalar* `number`) (*vector [*`data-vector`*], page 58*)) [Method]
Return the vector destructively scaled by scalar.

Source [`data-vector.lisp`], page 15.

`nscale` ((*scalar* `number`) (*data* `array`)) [Method]
Scale each element of the array.

Source [`array.lisp`], page 14.

`nscale` ((*scalar* `number`) (*data* `vector`)) [Method]
Return the vector destructively scaled by scalar.

Source [`vector.lisp`], page 14.

`nscale` ((*scalar* `number`) (*data* `list`)) [Method]
Return the list destructively scaled by scalar.

Source [`list.lisp`], page 13.

`nsolve` (*matrix vector*) [Generic Function]
Return the solution to the system of equations in-place.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`nsolve` ((*matrix [*`symmetric-matrix`*], page 64*) (*vector [*`column-vector`*], page 57*)) [Method]

Return the solution to the system of equations.

Source [`symmetric-matrix.lisp`], page 19.

`nsolve` ((*matrix [*`hermitian-matrix`*], page 61*) (*vector [*`column-vector`*], page 57*)) [Method]

Return the solution to the system of equations.

Source [`hermitian-matrix.lisp`], page 18.

`nsolve` ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*)) [Method]

Return the solution to the system of equations.

Source [`dense-matrix.lisp`], page 16.

`nsolve` *:before* ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*)) [Method]

Return the solution to the system of equations.

Source [`dense-matrix.lisp`], page 16.

`nsolve` ((*array* `array`) (*vector* `vector`)) [Method]

Return the solution to the system of equations.

Source [`array.lisp`], page 14.

`nsubtract` (*vector-or-matrix-1 vector-or-matrix-2* **&key** *scalar1 scalar2*) [Generic Function]

Destructive vector or matrix subtraction.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`nsubtract` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`symmetric-matrix`*], page 64*) **&key** *scalar1 scalar2*) [Method]

Source [`symmetric-matrix.lisp`], page 19.

`nsubtract` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Generate an error if a non-symmetric matrix is destructively subtracted to a symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`nsubtract` ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Return the addition of the 2 matrices.

Source [`dense-matrix.lisp`], page 16.

`nsubtract` *:before* ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Audit the input data.

Source [`dense-matrix.lisp`], page 16.

`nsubtract` ((*vector1* *[*`row-vector`*], page 63*) (*vector2* *[*`row-vector`*], page 63*) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`nsubtract` ((*vector1* *[*`column-vector`*], page 57*) (*vector2* *[*`column-vector`*], page 57*) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`nsubtract` *:before* ((*vector1* *[*`data-vector`*], page 58*) (*vector2* *[*`data-vector`*], page 58*) **&key** *scalar1 scalar2*) [Method]

Verify that the dimensions are equal.

Source [`data-vector.lisp`], page 15.

`nsubtract` ((*array1* `array`) (*array2* `array`) **&key** *scalar1 scalar2*) [Method]

Destructively subtract array2 from array1.

Source [`array.lisp`], page 14.

`nsubtract` ((*vector1* `vector`) (*vector2* `vector`) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`vector.lisp`], page 14.

`nsubtract` ((*list1* `list`) (*list2* `list`) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*list2 from scalar1*list1.

Source [`list.lisp`], page 13.

`ntranspose` (*vector-or-matrix*) [Generic Function]

Destructively transpose the vector or matrix.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`ntranspose` ((*matrix* *[*`hermitian-matrix`*], page 61*)) [Method]

The destructive transpose of a Hermitian matrix is itself.

Source [`hermitian-matrix.lisp`], page 18.

`ntranspose` ((*matrix* *[*`dense-matrix`*], page 59*)) [Method]

Replace the contents of the dense matrix with the transpose.

Source [`dense-matrix.lisp`], page 16.

`ntranspose` ((*vector* *[*`row-vector`*], page 63*)) [Method]

Return a column vector destructively.

Source [`data-vector.lisp`], page 15.

`ntranspose` ((*vector* *[*`column-vector`*], page 57*)) [Method]

Return a row vector destructively.

Source [`data-vector.lisp`], page 15.

`ntranspose` ((*data* `array`)) [Method]

Replace the contents of the array with the transpose.

Source [`array.lisp`], page 14.

`ntranspose` ((*data* `vector`)) [Method]

Return a row vector destructively.

Source [`vector.lisp`], page 14.

`ntranspose` ((*data* `list`)) [Method]

Return a row vector destructively.

Source [`list.lisp`], page 13.

`permute` (*vector-or-matrix-1 vector-or-matrix-2*) [Generic Function]

Permute the vector or matrix.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`permute` ((*permutation [*`permutation-matrix`*], page 62*) (*matrix [*`hermitian-matrix`*], page 61*)) [Method]

Source [`hermitian-matrix.lisp`], page 18.

`permute` ((*matrix [*`hermitian-matrix`*], page 61*) (*permutation [*`permutation-matrix`*], page 62*)) [Method]

Source [`hermitian-matrix.lisp`], page 18.

`permute` ((*permutation [*`permutation-matrix`*], page 62*) (*matrix [*`dense-matrix`*], page 59*)) [Method]

Source [`dense-matrix.lisp`], page 16.

`permute` ((*matrix [*`dense-matrix`*], page 59*) (*permutation [*`permutation-matrix`*], page 62*)) [Method]

Source [`dense-matrix.lisp`], page 16.

`permute` ((*matrix [*`permutation-matrix`*], page 62*) (*vector [*`column-vector`*], page 57*)) [Method]

Return the permutation of the column vector.

Source [`data-vector.lisp`], page 15.

`permute` *:before* ((*matrix [*`permutation-matrix`*], page 62*) (*vector [*`column-vector`*], page 57*)) [Method]

Verify that the dimensions are compatible.

Source [`data-vector.lisp`], page 15.

`permute` ((*vector [*`row-vector`*], page 63*) (*matrix [*`permutation-matrix`*], page 62*)) [Method]

Return the permutation of the row vector.

Source [`data-vector.lisp`], page 15.

`permute` *:before* ((*vector* [`row-vector`], *page 63*) (*matrix* [`permutation-matrix`], *page 62*)) [Method]

Verify that the dimensions are compatible.

Source [`data-vector.lisp`], page 15.

`permute` ((*matrix* [`permutation-matrix`], *page 62*) (*data* `array`)) [Method]

Source [`array.lisp`], page 14.

`permute` ((*data* `array`) (*matrix* [`permutation-matrix`], *page 62*)) [Method]

Source [`array.lisp`], page 14.

`permute` ((*matrix* [`permutation-matrix`], *page 62*) (*data* `vector`)) [Method]

Return the permutation of the list.

Source [`vector.lisp`], page 14.

`permute` ((*data* `vector`) (*matrix* [`permutation-matrix`], *page 62*)) [Method]

Return the permutation of the list.

Source [`vector.lisp`], page 14.

`permute` ((*matrix* [`permutation-matrix`], *page 62*) (*data* `list`)) [Method]

Return the permutation of the list.

Source [`list.lisp`], page 13.

`permute` ((*data* `list`) (*matrix* [`permutation-matrix`], *page 62*)) [Method]

Return the permutation of the list.

Source [`list.lisp`], page 13.

`product` (*vector-or-matrix-1 vector-or-matrix-2* **&optional** *scalar*) [Generic Function]

Return the vector-vector, matrix-vector or matrix-matrix product.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`product` ((*matrix1* [`dense-matrix`], *page 59*) (*matrix2* [`dense-matrix`], *page 59*) **&optional** *scalar*) [Method]

Return the product of the dense matrices.

Source [`dense-matrix.lisp`], page 16.

`product` *:before* ((*matrix1* [`dense-matrix`], *page 59*) (*matrix2* [`dense-matrix`], *page 59*) **&optional** *scalar*) [Method]

Verify the input.

Source [`dense-matrix.lisp`], page 16.

`product` ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*) **&optional** *scalar*) [Method]

Return a column vector generated by the multiplication of the dense matrix with a column vector.

Source [`dense-matrix.lisp`], page 16.

`product` *:before* ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*) **&optional** *scalar*) [Method]

Verify the input.

Source [`dense-matrix.lisp`], page 16.

`product` ((*vector [*`row-vector`*], page 63*) (*matrix [*`dense-matrix`*], page 59*) **&optional** *scalar*) [Method]

Return a row vector generated by the pre-multiplication of a dense matrix by a row vector.

Source [`dense-matrix.lisp`], page 16.

`product` *:before* ((*vector [*`row-vector`*], page 63*) (*matrix [*`dense-matrix`*], page 59*) **&optional** *scalar*) [Method]

Verify the inputs.

Source [`dense-matrix.lisp`], page 16.

`product` ((*vector1 [*`row-vector`*], page 63*) (*vector2 [*`column-vector`*], page 57*) **&optional** *scalar*) [Method]

Return the dot product of vector1 and vector2.

Source [`data-vector.lisp`], page 15.

`product` *:before* ((*vector1 [*`row-vector`*], page 63*) (*vector2 [*`column-vector`*], page 57*) **&optional** *scalar*) [Method]

Verify that the dimensions are equal.

Source [`data-vector.lisp`], page 15.

`product` ((*array1* `array`) (*array2* `array`) **&optional** *scalar*) [Method]

Return the product of the arrays.

Source [`array.lisp`], page 14.

`product` ((*array* `array`) (*vector* `vector`) **&optional** *scalar*) [Method]

Return a vector generated by the multiplication of the array with a vector.

Source [`array.lisp`], page 14.

`product` ((*vector* `vector`) (*array* `array`) **&optional** *scalar*) [Method]

Return a vector generated by the pre-multiplication of a array by a vector.

Source [`array.lisp`], page 14.

`product` ((*vector1* `vector`) (*vector2* `vector`) **&optional** *scalar*) [Method]

Return the dot product of vector1 and vector2.

Source [`vector.lisp`], page 14.

`product` ((*list1* `list`) (*list2* `list`) **&optional** *scalar*) [Method]

Return the dot product of list1 and list2.

Source [`list.lisp`], page 13.

`replace-matrix` (*matrix1 matrix2* **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Generic Function]

Destructively replace elements of matrix1 with matrix2.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`replace-matrix` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Method]

Replace the elements of MATRIX1 with MATRIX2.

Source [`symmetric-matrix.lisp`], page 19.

`replace-matrix` ((*matrix1 [*`symmetric-matrix`*], page 64*) (*matrix2 [*`symmetric-matrix`*], page 64*) **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Method]

Replace the elements of MATRIX1 with MATRIX2.

Source [`symmetric-matrix.lisp`], page 19.

`replace-matrix` ((*matrix1 [*`hermitian-matrix`*], page 61*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Method]

Replace the elements of matrix1 with matrix2.

Source [`hermitian-matrix.lisp`], page 18.

`replace-matrix` ((*matrix1 [*`hermitian-matrix`*], page 61*) (*matrix2 [*`hermitian-matrix`*], page 61*) **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Method]

Replace the elements of matrix1 with matrix2.

Source [`hermitian-matrix.lisp`], page 18.

`replace-matrix` ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *start-row1 end-row1 start-column1 end-column1 start-row2 end-row2 start-column2 end-column2*) [Method]

Replace the elements of matrix1 with matrix2.

Source [`dense-matrix.lisp`], page 16.

`replace-vector` (*vector1 vector2* **&key** *start1 end1 start2 end2*) [Generic Function]

Destructively replace the elements of vector1 with vector2.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`replace-vector` ((*vector1* [`data-vector`], *page 58*) (*vector2* [`data-vector`], *page 58*) **&key** *start1 end1 start2 end2*) [Method]

Destructively replace the elements of vector1 with vector2.

Source [`data-vector.lisp`], page 15.

`right-permute` (*vector-or-array permutation*) [Generic Function]

Permute the row vector or columns of the array.

Package [`linear-algebra-kernel`], page 21.

Source [`permute.lisp`], page 8.

Methods

`right-permute` ((*data* `array`) (*permutation* `vector`)) [Method]

Permute the columns of the array.

`right-permute` ((*data* `vector`) (*permutation* `vector`)) [Method]

Permute the row vector to create a column vector.

`scale` (*scalar vector-or-matrix*) [Generic Function]

Scale each element by the scalar.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`scale` ((*scalar* `number`) (*matrix* [`dense-matrix`], *page 59*)) [Method]

Scale each element of the dense matrix.

Source [`dense-matrix.lisp`], page 16.

`scale` ((*scalar* `number`) (*vector* [`data-vector`], *page 58*)) [Method]

Return the vector scaled by scalar.

Source [`data-vector.lisp`], page 15.

`scale` ((*scalar* `number`) (*data* `array`)) [Method]

Scale each element of the array.

Source [`array.lisp`], page 14.

`scale` ((*scalar* `number`) (*data* `vector`)) [Method]

Return the vector scaled by scalar.

Source [`vector.lisp`], page 14.

`scale` ((*scalar* `number`) (*data* `list`)) [Method]

Return the list scaled by scalar.

Source [`list.lisp`], page 13.

`scaled-binary-op` (*op scalar1 scalar2*) [Generic Function]

Compile and return a scaled binary operation.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

Methods

`scaled-binary-op` ((*op* (`eql #<function ->`)) (*scalar1* `number`) (*scalar2* `number`)) [Method]
Return the scaled operation.

`scaled-binary-op` ((*op* (`eql #<function +>`)) (*scalar1* `number`) (*scalar2* `number`)) [Method]
Return the scaled operation.

`scaled-binary-op` ((*op* (`eql #<function ->`)) (*scalar1* (`eql nil`)) (*scalar2* `number`)) [Method]
Return the scaled operation.

`scaled-binary-op` ((*op* (`eql #<function +>`)) (*scalar1* (`eql nil`)) (*scalar2* `number`)) [Method]
Return the scaled operation.

`scaled-binary-op` ((*op* (`eql #<function ->`)) (*scalar1* `number`) (*scalar2* (`eql nil`))) [Method]
Return the scaled operation.

`scaled-binary-op` ((*op* (`eql #<function +>`)) (*scalar1* `number`) (*scalar2* (`eql nil`))) [Method]
Return the scaled operation.

`scaled-binary-op` (*op* (*scalar1* (`eql nil`)) (*scalar2* (`eql nil`))) [Method]
Return the operation.

`solve` (*matrix vector*) [Generic Function]
Return the solution to the system of equations.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`solve` ((*matrix [*`symmetric-matrix`*], page 64*) (*vector [*`column-vector`*], page 57*)) [Method]
Return the solution to the system of equations.

Source [`symmetric-matrix.lisp`], page 19.

`solve` ((*matrix [*`hermitian-matrix`*], page 61*) (*vector [*`column-vector`*], page 57*)) [Method]
Return the solution to the system of equations.

Source [`hermitian-matrix.lisp`], page 18.

`solve` ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*)) [Method]
Return the solution to the system of equations.

Source [`dense-matrix.lisp`], page 16.

`solve` *:before* ((*matrix [*`dense-matrix`*], page 59*) (*vector [*`column-vector`*], page 57*)) [Method]
Return the solution to the system of equations.

Source [`dense-matrix.lisp`], page 16.

`solve` ((*array* `array`) (*vector* `vector`)) [Method]

Return the solution to the system of equations.

Source [`array.lisp`], page 14.

`submatrix` (*matrix start-row start-column* **&key** *end-row end-column*) [Generic Function]

Return a submatrix of the matrix.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`submatrix` ((*matrix [*`symmetric-matrix`*], page 64*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Return a matrix created from the submatrix of matrix.

Source [`symmetric-matrix.lisp`], page 19.

`submatrix` ((*matrix [*`hermitian-matrix`*], page 61*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Return a matrix created from the submatrix of matrix.

Source [`hermitian-matrix.lisp`], page 18.

`submatrix` ((*matrix [*`square-matrix`*], page 64*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Return a matrix created from the submatrix of matrix.

Source [`square-matrix.lisp`], page 18.

`submatrix` ((*matrix [*`dense-matrix`*], page 59*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Return a dense matrix created from the submatrix of a matrix.

Source [`dense-matrix.lisp`], page 16.

`(setf submatrix)` (*matrix start-row start-column* **&key** *end-row end-column*) [Generic Function]

Set the submatrix of the matrix.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`(setf submatrix)` ((*matrix [*`symmetric-matrix`*], page 64*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Set a submatrix of MATRIX.

Source [`symmetric-matrix.lisp`], page 19.

`(setf submatrix)` ((*matrix [*`symmetric-matrix`*], page 64*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Set a submatrix of the matrix.

Source [`symmetric-matrix.lisp`], page 19.

`(setf submatrix)` ((*matrix [*`hermitian-matrix`*], page 61*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Set a submatrix of the matrix.

Source [`hermitian-matrix.lisp`], page 18.

`(setf submatrix)` ((*matrix [*`hermitian-matrix`*], page 61*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Set a submatrix of the matrix.

Source [`hermitian-matrix.lisp`], page 18.

`(setf submatrix)` ((*matrix [*`dense-matrix`*], page 59*) (*start-row* `integer`) (*start-column* `integer`) **&key** *end-row end-column*) [Method]

Set the submatrix of matrix.

Source [`dense-matrix.lisp`], page 16.

`subtract` (*vector-or-matrix-1 vector-or-matrix-2* **&key** *scalar1 scalar2*) [Generic Function]

Vector or matrix binary subtraction.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`subtract` ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Return the addition of the 2 matrices.

Source [`dense-matrix.lisp`], page 16.

`subtract` *:before* ((*matrix1 [*`dense-matrix`*], page 59*) (*matrix2 [*`dense-matrix`*], page 59*) **&key** *scalar1 scalar2*) [Method]

Audit the input data.

Source [`dense-matrix.lisp`], page 16.

`subtract` ((*vector1 [*`row-vector`*], page 63*) (*vector2 [*`row-vector`*], page 63*) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`subtract` ((*vector1 [*`column-vector`*], page 57*) (*vector2 [*`column-vector`*], page 57*) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`data-vector.lisp`], page 15.

`subtract` *:before* ((*vector1 [*`data-vector`*], page 58*) (*vector2 [*`data-vector`*], page 58*) **&key** *scalar1 scalar2*) [Method]

Verify that the dimensions are equal.

Source [`data-vector.lisp`], page 15.

`subtract` ((*array1* `array`) (*array2* `array`) **&key** *scalar1 scalar2*) [Method]

Return the subtraction of the 2 arrays.

Source [`array.lisp`], page 14.

`subtract` ((*vector1* `vector`) (*vector2* `vector`) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*vector2 from scalar1*vector1.

Source [`vector.lisp`], page 14.

`subtract` ((*list1* `list`) (*list2* `list`) **&key** *scalar1 scalar2*) [Method]

Return the subraction of scalar2*list2 from scalar1*list1.

Source [`list.lisp`], page 13.

`subvector` (*vector start* **&optional** *end*) [Generic Function]

Return a new vector that is a subvector of the vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`subvector` ((*vector [*`data-vector`*], page 58*) *start* **&optional** *end*) [Method]

Return a new data vector that is a subset of vector.

Source [`data-vector.lisp`], page 15.

`(setf subvector)` (*vector start* **&optional** *end*) [Generic Function]

Set the subvector of the vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`(setf subvector)` ((*vector [*`data-vector`*], page 58*) *start* **&optional** *end*) [Method]

Set the subvector of the data vector.

Source [`data-vector.lisp`], page 15.

`sump` (*vector-or-array p* **&optional** *scale sump*) [Generic Function]

Return the scaling parameter and the sum of the P powers.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

Methods

`sump` ((*data* `array`) *p* **&optional** *scale sump*) [Method]

Return the scaling parameter and the sum of the P powers of the matrix.

`sump` ((*data* `vector`) *p* **&optional** *scale sump*) [Method]

Return the scaling parameter and the sum of the powers of p of the vector.

`sump` ((*data* `list`) (*p* `real`) **&optional** *scale sump*) [Method]
Return the scaling parameter and the sum of the powers of p of the data.

`sumsq` (*vector-or-array* **&optional** *scale sumsq*) [Generic Function]
Return the scaling parameter and the sum of the squares.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

Methods

`sumsq` ((*data* `array`) **&optional** *scale sumsq*) [Method]
Return the scaling parameter and the sum of the squares of the array.

`sumsq` ((*data* `vector`) **&optional** *scale sumsq*) [Method]
Return the scaling parameter and the sum of the squares of the vector.

`sumsq` ((*data* `list`) **&optional** *scale sumsq*) [Method]
Return the scaling parameter and the sum of the squares of the list.

`transpose` (*vector-or-matrix*) [Generic Function]
Transpose the vector or matrix.

Package [`linear-algebra`], page 23.

Source [`fundamental-ops.lisp`], page 11.

Methods

`transpose` ((*matrix [*`hermitian-matrix`*], page 61*)) [Method]
The transpose of a Hermitian matrix is itself.

Source [`hermitian-matrix.lisp`], page 18.

`transpose` ((*matrix [*`dense-matrix`*], page 59*)) [Method]
Return the transpose of the matrix.

Source [`dense-matrix.lisp`], page 16.

`transpose` ((*vector [*`row-vector`*], page 63*)) [Method]
Return a column vector.

Source [`data-vector.lisp`], page 15.

`transpose` ((*vector [*`column-vector`*], page 57*)) [Method]
Return a row vector.

Source [`data-vector.lisp`], page 15.

`transpose` ((*data* `array`)) [Method]
Return the transpose of the array.

Source [`array.lisp`], page 14.

`transpose` ((*data* `vector`)) [Method]
Return a row vector.

Source [`vector.lisp`], page 14.

`transpose` ((*data* `list`)) [Method]
Return a row vector.

Source [`list.lisp`], page 13.

`transpose` ((*matrix [*`permutation-matrix`*], page 62*)) [Method]
Transpose the permutation matrix.

Source [`permutation-matrix.lisp`], page 13.

`vector-element-type` (*vector*) [Generic Function]
Return the element type of vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`vector-element-type` ((*vector [*`data-vector`*], page 58*)) [Method]
Return the element type of vector.

Source [`data-vector.lisp`], page 15.

`vector-in-bounds-p` (*vector index*) [Generic Function]
Return true if index does not exceed the dimensions of vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`vector-in-bounds-p` ((*vector [*`data-vector`*], page 58*) (*index* `integer`)) [Method]
Return true if index does not exceed the dimensions of vector.

Source [`data-vector.lisp`], page 15.

`vector-length` (*vector*) [Generic Function]
Return the length of the vector.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`vector-length` ((*vector [*`data-vector`*], page 58*)) [Method]
Return the length of the vector.

Source [`data-vector.lisp`], page 15.

`vref` (*vector index*) [Generic Function]
Return the element of vector at index.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`vref` ((*vector [*`data-vector`*], page 58*) (*index* `integer`)) [Method]
Return the element of vector at index.

Source [`data-vector.lisp`], page 15.

`(setf vref)` (*vector index*) [Generic Function]

Set the element of vector at index to data.

Package [`linear-algebra`], page 23.

Source [`vector.lisp`], page 11.

Methods

`(setf vref)` ((*vector [*`data-vector`*], page 58*) (*index* `integer`)) [Method]

Set the element of vector at index to data.

Source [`data-vector.lisp`], page 15.

5.1.4 Standalone methods

`initialize-instance` *:after* ((*self [*`data-vector`*], page 58*) **&rest** *initargs* **&key** *size element-type initial-element initial-contents*) [Method]

Source [`data-vector.lisp`], page 15.

`initialize-instance` *:after* ((*self [*`identity-matrix`*], page 61*) **&rest** *initargs* **&key** *dimensions element-type initial-element initial-contents*) [Method]

Initialize the identity matrix.

Source [`identity-matrix.lisp`], page 12.

`initialize-instance` *:after* ((*self [*`dense-matrix`*], page 59*) **&rest** *initargs* **&key** *dimensions element-type initial-element initial-contents*) [Method]

Source [`dense-matrix.lisp`], page 16.

`initialize-instance` *:after* ((*self [*`permutation-matrix`*], page 62*) **&rest** *initargs* **&key** *dimensions element-type initial-element initial-contents*) [Method]

Verify that the element-type was not set and that rows equals columns.

Source [`permutation-matrix.lisp`], page 13.

5.1.5 Classes

`column-vector` [Class]

A column vector.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

Direct superclasses

[`data-vector`], page 58.

Direct methods

- [`add`], page 33.
- [`compatible-dimensions-p`], page 34.
- [`compatible-dimensions-p`], page 34.
- [`nadd`], page 40.
- [`nsolve`], page 44.
- [`nsolve`], page 44.
- [`nsolve`], page 44.
- [`nsolve`], page 44.

- [nsubtract], page 45.
- [ntranspose], page 45.
- [permute], page 46.
- [permute], page 46.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- [solve], page 51.
- [solve], page 51.
- [solve], page 51.
- [solve], page 51.
- [subtract], page 53.
- [transpose], page 55.

`data-vector` [Class]

A data vector.

Package [linear-algebra], page 23.

Source [data-vector.lisp], page 15.

Direct subclasses

- [column-vector], page 57.
- [row-vector], page 63.

Direct methods

- [add], page 33.
- [apply-rotation], page 33.
- [apply-rotation], page 33.
- [(setf contents)], page 70.
- [contents], page 70.
- [copy-vector], page 35.
- `float-equal`.
- `float-equal`.
- [initialize-instance], page 57.
- [map-into-vector], page 36.
- [map-into-vector], page 36.
- [map-vector], page 36.
- [map-vector], page 37.
- [nadd], page 41.
- [napply-rotation], page 41.
- [napply-rotation], page 41.
- [norm], page 42.
- [nscale], page 43.
- [nsubtract], page 45.
- `rational-equal`.

- `rational-equal`.
- [`replace-vector`], page 50.
- [`scale`], page 50.
- [`subtract`], page 53.
- [`(setf subvector)`], page 54.
- [`subvector`], page 54.
- [`vector-element-type`], page 56.
- [`vector-in-bounds-p`], page 56.
- [`vector-length`], page 56.
- [`(setf vref)`], page 57.
- [`vref`], page 56.

Direct slots

`contents` [Slot]

Type `(array * (*))`

Initargs `:contents`

Readers [`contents`], page 70.

Writers [`(setf contents)`], page 70.

`dense-matrix` [Class]

Dense matrix object.

Package [`linear-algebra`], page 23.

Source [`dense-matrix.lisp`], page 16.

Direct superclasses
[`matrix-object`], page 62.

Direct subclasses
[`square-matrix`], page 64.

Direct methods

- [`add`], page 32.
- [`add`], page 33.
- [`compatible-dimensions-p`], page 34.
- [`(setf contents)`], page 70.
- [`contents`], page 70.
- [`copy-matrix`], page 35.
- `float-equal`.
- `float-equal`.
- `float-equal`.
- `float-equal`.
- [`initialize-instance`], page 57.
- [`initialize-matrix-contents`], page 72.
- [`initialize-matrix-contents`], page 72.
- [`initialize-matrix-contents`], page 72.
- [`initialize-matrix-contents`], page 72.

- [invert], page 36.
- [matrix-column-dimension], page 37.
- [matrix-dimensions], page 37.
- [matrix-element-type], page 38.
- [matrix-in-bounds-p], page 38.
- [matrix-row-dimension], page 39.
- [(setf mref)], page 40.
- [mref], page 39.
- [nadd], page 40.
- [nadd], page 40.
- [nadd], page 40.
- [ninvert], page 42.
- [norm], page 42.
- [nscale], page 43.
- [nsolve], page 44.
- [nsolve], page 44.
- [nsubtract], page 44.
- [nsubtract], page 44.
- [nsubtract], page 44.
- [ntranspose], page 45.
- [permute], page 46.
- [permute], page 46.
- [product], page 47.
- [product], page 47.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- rational-equal.
- rational-equal.
- rational-equal.
- rational-equal.
- [replace-matrix], page 49.
- [replace-matrix], page 49.
- [replace-matrix], page 49.
- [scale], page 50.
- [solve], page 51.
- [solve], page 51.
- [(setf submatrix)], page 52.
- [(setf submatrix)], page 53.
- [(setf submatrix)], page 53.
- [submatrix], page 52.
- [subtract], page 53.

- [subtract], page 53.
- [transpose], page 55.

Direct slots

`contents` [Slot]

Type	`(array * (* *))`
Initargs	`:contents`
Readers	[`contents`], page 70.
Writers	[`(setf contents)`], page 70.

`hermitian-matrix` [Class]

Hermitian matrix object.

Package	[`linear-algebra`], page 23.
Source	[`hermitian-matrix.lisp`], page 18.

Direct superclasses

[`square-matrix`], page 64.

Direct subclasses

[`symmetric-matrix`], page 64.

Direct methods

- [`initialize-matrix-contents`], page 71.
- [`initialize-matrix-contents`], page 71.
- [`initialize-matrix-contents`], page 71.
- [`initialize-matrix-contents`], page 72.
- [`invert`], page 35.
- [`(setf mref)`], page 40.
- [`ninvert`], page 41.
- [`nsolve`], page 44.
- [`ntranspose`], page 45.
- [`permute`], page 46.
- [`permute`], page 46.
- [`replace-matrix`], page 49.
- [`replace-matrix`], page 49.
- [`solve`], page 51.
- [`(setf submatrix)`], page 53.
- [`(setf submatrix)`], page 53.
- [`submatrix`], page 52.
- [`transpose`], page 55.

`identity-matrix` [Class]

Identity matrix object.

Package	[`linear-algebra`], page 23.
Source	[`identity-matrix.lisp`], page 12.

Direct superclasses

[`matrix-object`], page 62.

Direct methods

- [`contents`], page 70.
- [`copy-matrix`], page 35.
- [`initialize-instance`], page 57.
- [`matrix-column-dimension`], page 37.
- [`matrix-dimensions`], page 37.
- [`matrix-element-type`], page 38.
- [`matrix-in-bounds-p`], page 38.
- [`matrix-row-dimension`], page 39.
- [`mref`], page 39.
- [`size`], page 73.

Direct slots

`size` [Slot]

Type `fixnum`

Initargs `:size`

Readers

- [`matrix-column-dimension`], page 37.
- [`matrix-row-dimension`], page 39.
- [`size`], page 73.

Writers *This slot is read-only.*

`contents` [Slot]

Type `(array * (2))`

Initargs `:contents`

Readers [`contents`], page 70.

Writers *This slot is read-only.*

`matrix-object` [Class]

A superclass for all matrices.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Direct subclasses

- [`dense-matrix`], page 59.
- [`identity-matrix`], page 61.
- [`permutation-matrix`], page 62.

`permutation-matrix` [Class]

Permutation matrix object.

Package [`linear-algebra`], page 23.

Source [`permutation-matrix.lisp`], page 13.

Direct superclasses

[`matrix-object`], page 62.

Direct methods

- [(setf contents)], page 71.
- [contents], page 70.
- [copy-matrix], page 35.
- [initialize-instance], page 57.
- [initialize-matrix-contents], page 72.
- [initialize-matrix-contents], page 72.
- [initialize-matrix-contents], page 72.
- [matrix-column-dimension], page 37.
- [matrix-dimensions], page 37.
- [matrix-element-type], page 38.
- [matrix-in-bounds-p], page 38.
- [matrix-row-dimension], page 39.
- [(setf mref)], page 40.
- [mref], page 39.
- [permute], page 46.
- [permute], page 46.
- [permute], page 46.
- [permute], page 46.
- [permute], page 46.
- [permute], page 46.
- [permute], page 46.
- [permute], page 47.
- [permute], page 47.
- [permute], page 47.
- [permute], page 47.
- [permute], page 47.
- [permute], page 47.
- [permute], page 47.
- [transpose], page 56.

Direct slots

`contents` [Slot]

Type	`(array fixnum (*))`
Initargs	`:contents`
Readers	[contents], page 70.
Writers	[(setf contents)], page 71.

`row-vector` [Class]

A row vector.

Package	[linear-algebra], page 23.
Source	[data-vector.lisp], page 15.

Direct superclasses

[data-vector], page 58.

Direct methods

- [add], page 33.
- [nadd], page 40.
- [nsubtract], page 45.
- [ntranspose], page 45.
- [permute], page 46.
- [permute], page 47.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- [product], page 48.
- [subtract], page 53.
- [transpose], page 55.

`square-matrix` [Class]

Square matrix object.

Package [linear-algebra], page 23.

Source [square-matrix.lisp], page 18.

Direct superclasses

[dense-matrix], page 59.

Direct subclasses

[hermitian-matrix], page 61.

Direct methods

- [compatible-dimensions-p], page 34.
- [initialize-matrix-contents], page 72.
- [invert], page 35.
- [ninvert], page 42.
- [submatrix], page 52.

`symmetric-matrix` [Class]

Symmetric matrix object.

Package [linear-algebra], page 23.

Source [symmetric-matrix.lisp], page 19.

Direct superclasses

[hermitian-matrix], page 61.

Direct methods

- [initialize-matrix-contents], page 71.
- [initialize-matrix-contents], page 71.
- [initialize-matrix-contents], page 71.
- [invert], page 35.
- [(setf mref)], page 39.
- [nadd], page 40.
- [nadd], page 40.
- [ninvert], page 41.

- [`nsolve`], page 44.
- [`nsubtract`], page 44.
- [`nsubtract`], page 44.
- [`replace-matrix`], page 49.
- [`replace-matrix`], page 49.
- [`solve`], page 51.
- [`(setf submatrix)`], page 52.
- [`(setf submatrix)`], page 52.
- [`submatrix`], page 52.

5.2 Internals

5.2.1 Ordinary functions

`%abs-vector` (*vector*) [Function]

Return a vector containing absolute value of each element.

Package [`linear-algebra-kernel`], page 21.

Source [`unary-operations.lisp`], page 8.

`%array1<-array1-op-array2` (*operation array1 array2*) [Function]

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%array<-array1-op-array2` (*operation array1 array2*) [Function]

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%default-cg-epsilon` (*array vector*) [Function]

Return a default epsilon for the conjugate gradient method.

Package [`linear-algebra-kernel`], page 21.

Source [`conjugate-gradient.lisp`], page 10.

`%initialize-cg-residual` (*array vector solution*) [Function]

Return the initial residual vector for the conjugate gradient.

Package [`linear-algebra-kernel`], page 21.

Source [`conjugate-gradient.lisp`], page 10.

`%initialize-cg-solution` (*array*) [Function]

Return an initial solution vector for the conjugate gradient.

Package [`linear-algebra-kernel`], page 21.

Source [`conjugate-gradient.lisp`], page 10.

`%initialize-hermitian-matrix-with-seq` (*matrix data dimensions element-type*) [Function]

Initialize and validate a Hermitian matrix with a sequence.

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`%initialize-permutation-matrix-with-seq` (*matrix data size*) [Function]

Package [`linear-algebra`], page 23.

Source [`permutation-matrix.lisp`], page 13.

`%initialize-symmetric-matrix-with-seq` (*matrix data dimensions element-type*) [Function]

Initialize and validate a symmetric matrix with a sequence.

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`%map-data-vector` (*result-type function first-vector* **&rest** *more-vectors*) [Function]

Non-validating version of map-vector.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`%map-into-data-vector` (*result-vector function* **&rest** *vectors*) [Function]

Non-validating version of map-into-vector.

Package [`linear-algebra`], page 23.

Source [`data-vector.lisp`], page 15.

`%negative-residual` (*residual*) [Function]

Return the negative of the residual.

Package [`linear-algebra-kernel`], page 21.

Source [`conjugate-gradient.lisp`], page 10.

`%product-array-array` (*array1 array2* **&optional** *result*) [Function]

Return the result of the product of 2 arrays.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%product-array-vector` (*array vector* **&optional** *result*) [Function]

Return the result of the array postmultiplied by the vector.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%product-vector-array` (*vector array* **&optional** *result*) [Function]

Return the result of the array premultiplied by the vector.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%replace-hermitian-matrix-off-diagonal` (*matrix1 matrix2 row1 column1 row2 column2 numrows numcols*) [Function]

Destructively replace a subset off the diagonal of matrix1 with matrix2.

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`%replace-hermitian-matrix-on-diagonal` (*matrix1 matrix2 row1 column1 row2 column2 numrows numcols*) [Function]

Destructively replace a subset on the diagonal of matrix1 with matrix2.

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`%replace-symmetric-matrix-off-diagonal` (*matrix1 matrix2 row1 column1 row2 column2 numrows numcols*) [Function]

Destructively replace a subset off the diagonal of matrix1 with matrix2.

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`%replace-symmetric-matrix-on-diagonal` (*matrix1 matrix2 row1 column1 row2 column2 numrows numcols*) [Function]

Destructively replace a subset on the diagonal of matrix1 with matrix2.

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`%scaled-product-array-array` (*scalar array1 array2* **&optional** *result*) [Function]

Return the scaled result of the product of 2 arrays.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%scaled-product-array-vector` (*scalar array vector* **&optional** *result*) [Function]

Return the result of the array postmultiplied by the vector and scaled.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%scaled-product-vector-array` (*scalar vector array* **&optional** *result*) [Function]

Return the result of the array premultiplied by the vector and scaled.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%setf-hermitian-submatrix-off-diagonal` (*matrix data row column numrows numcols*) [Function]

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`%setf-hermitian-submatrix-on-diagonal` (*matrix data row numrows*) [Function]

Package [`linear-algebra`], page 23.

Source [`hermitian-matrix.lisp`], page 18.

`%setf-symmetric-submatrix-off-diagonal` (*matrix data row column numrows numcols*) [Function]

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`%setf-symmetric-submatrix-on-diagonal` (*matrix data row numrows*) [Function]

Package [`linear-algebra`], page 23.

Source [`symmetric-matrix.lisp`], page 19.

`%vector1<-vector1-op-vector2` (*operation vector1 vector2*) [Function]

Store the result of the binary operation in vector1.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`%vector<-vector1-op-vector2` (*operation vector1 vector2*) [Function]

Store the result of the binary operation in a new vector.

Package [`linear-algebra-kernel`], page 21.

Source [`binary-operations.lisp`], page 8.

`column-pivot` (*array pivot-selection-vector column*) [Function]

Return the LR pivot of the array.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`column-pivot-search` (*array column*) [Function]

Return the row index of the maximum value in the column.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`gauss-backsubstitution` (*factored solution*) [Function]

Calculate the solution by backsubstitution.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`gauss-factorization` (*array*) [Function]

Return the Gauss factorization of the array.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`gauss-update` (*factored pivot-selection-vector vector*) [Function]

Update the solution vector.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`initialize-pivot-selection-vector` (*size*) [Function]

Return a new, initialized, pivot vector.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`swap-rows` (*array i0 jth*) [Function]

Interchange the

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`tridiagonal-backsubstitution` (*array vector*) [Function]

Perform backsubstitution to obtain the solution.

Package [`linear-algebra-kernel`], page 21.

Source [`tridiagonal.lisp`], page 10.

`tridiagonal-factorization` (*array*) [Function]

Return the factorization of the tridiagonal array.

Package [`linear-algebra-kernel`], page 21.

Source [`tridiagonal.lisp`], page 10.

`tridiagonal-update` (*array vector*) [Function]

Update the solution vector using the factored array.

Package [`linear-algebra-kernel`], page 21.

Source [`tridiagonal.lisp`], page 10.

`unit-pivot-value` (*pivot-selection-vector row column* **&optional** *array-type*) [Function]

Return 1.0 if column equals the value at row of the pivot selection vector, otherwise 0.0.

Package [`linear-algebra-kernel`], page 21.

Source [`gauss.lisp`], page 9.

`zero-array` (*rows columns* **&optional** *element-type*) [Function]

Return an array of zeros.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

`zero-vector` (*size* **&optional** *element-type*) [Function]

Return a vector of zeros.

Package [`linear-algebra-kernel`], page 21.

Source [`utility.lisp`], page 7.

5.2.2 Generic functions

`%norm` (*data measure*) [Generic Function]

Package [`linear-algebra`], page 23.

Methods

`%norm` ((*data* `list`) (*measure* `(eql :infinity)`)) [Method]

Return the infinity, or maximum, norm of vector.

Source [`list.lisp`], page 13.

`%norm` ((*data* `list`) (*measure* `integer`)) [Method]

Return the p-norm of the vector.

Source [`list.lisp`], page 13.

`%norm` ((*data* `list`) (*measure* `(eql 2)`)) [Method]

Return the Euclidean norm of the vector.

Source [`list.lisp`], page 13.

`%norm` ((*data* `list`) (*measure* (`eql 1`))) [Method]

Return the Taxicab norm of the list.

Source [`list.lisp`], page 13.

`contents` (*object*) [Generic Reader]

Package [`linear-algebra`], page 23.

Methods

`contents` ((*dense-matrix [*`dense-matrix`*], page 59*)) [Reader Method]

automatically generated reader method

Source [`dense-matrix.lisp`], page 16.

Target Slot

[`contents`], page 61.

`contents` ((*data-vector [*`data-vector`*], page 58*)) [Reader Method]

automatically generated reader method

Source [`data-vector.lisp`], page 15.

Target Slot

[`contents`], page 59.

`contents` ((*permutation-matrix [*`permutation-matrix`*], page 62*)) [Reader Method]

automatically generated reader method

Source [`permutation-matrix.lisp`], page 13.

Target Slot

[`contents`], page 63.

`contents` ((*identity-matrix [*`identity-matrix`*], page 61*)) [Reader Method]

automatically generated reader method

Source [`identity-matrix.lisp`], page 12.

Target Slot

[`contents`], page 62.

`(setf contents)` (*object*) [Generic Writer]

Package [`linear-algebra`], page 23.

Methods

`(setf contents)` ((*dense-matrix [*`dense-matrix`*], page 59*)) [Writer Method]

automatically generated writer method

Source [`dense-matrix.lisp`], page 16.

Target Slot

[`contents`], page 61.

`(setf contents)` ((*data-vector [*`data-vector`*], page 58*)) [Writer Method]

automatically generated writer method

Source [`data-vector.lisp`], page 15.

Target Slot
[`contents`], page 59.

`(setf contents)` ((*permutation-matrix* *[*`permutation-matrix`*], page 62*)) [Writer Method]
automatically generated writer method

Source [`permutation-matrix.lisp`], page 13.

Target Slot
[`contents`], page 63.

`initialize-matrix-contents` (*matrix initial-contents initargs*) [Generic Function]
Initialize the matrix with data.

Package [`linear-algebra`], page 23.

Source [`matrix.lisp`], page 12.

Methods

`initialize-matrix-contents` ((*matrix* *[*`symmetric-matrix`*], page 64*) (*initial-contents* `array`) *initargs*) [Method]
Initialize a symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`initialize-matrix-contents` ((*matrix* *[*`symmetric-matrix`*], page 64*) (*initial-contents* `vector`) *initargs*) [Method]
Initialize a symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`initialize-matrix-contents` ((*matrix* *[*`symmetric-matrix`*], page 64*) (*initial-contents* `list`) *initargs*) [Method]
Initialize a symmetric matrix.

Source [`symmetric-matrix.lisp`], page 19.

`initialize-matrix-contents` ((*matrix* *[*`hermitian-matrix`*], page 61*) (*initial-contents* `array`) *initargs*) [Method]
Initialize the Hermitian matrix with a 2D array.

Source [`hermitian-matrix.lisp`], page 18.

`initialize-matrix-contents` ((*matrix* *[*`hermitian-matrix`*], page 61*) (*initial-contents* `vector`) *initargs*) [Method]
Initialize the Hermitian matrix with a nested sequence.

Source [`hermitian-matrix.lisp`], page 18.

`initialize-matrix-contents` ((*matrix* *[*`hermitian-matrix`*], page 61*) (*initial-contents* `list`) *initargs*) [Method]
Initialize the Hermitian matrix with a nested sequence.

Source [`hermitian-matrix.lisp`], page 18.

`initialize-matrix-contents` ((*matrix* [`hermitian-matrix`], *page 61*) (*initial-element* `complex`) *initargs*) [Method]

It is an error to initialize a Hermitian matrix with a complex element.

Source [`hermitian-matrix.lisp`], page 18.

`initialize-matrix-contents` *:before* ((*matrix* [`square-matrix`], *page 64*) *initial-contents initargs*) [Method]

Verify that the number of rows and colums are equal.

Source [`square-matrix.lisp`], page 18.

`initialize-matrix-contents` ((*matrix* [`dense-matrix`], *page 59*) (*initial-contents* `array`) *initargs*) [Method]

Verify that the size of the data is valid.

Source [`dense-matrix.lisp`], page 16.

`initialize-matrix-contents` ((*matrix* [`dense-matrix`], *page 59*) (*initial-contents* `vector`) *initargs*) [Method]

Initialize the dense matrix with a nested sequence.

Source [`dense-matrix.lisp`], page 16.

`initialize-matrix-contents` ((*matrix* [`dense-matrix`], *page 59*) (*initial-contents* `list`) *initargs*) [Method]

Initialize the dense matrix with a nested sequence.

Source [`dense-matrix.lisp`], page 16.

`initialize-matrix-contents` ((*matrix* [`dense-matrix`], *page 59*) (*initial-element* `number`) *initargs*) [Method]

Initialize the dense matrix with an initial element.

Source [`dense-matrix.lisp`], page 16.

`initialize-matrix-contents` ((*matrix* [`permutation-matrix`], *page 62*) (*initial-contents* `array`) *initargs*) [Method]

Initialize the permutation matrix with a 2D array.

Source [`permutation-matrix.lisp`], page 13.

`initialize-matrix-contents` ((*matrix* [`permutation-matrix`], *page 62*) (*initial-contents* `vector`) *initargs*) [Method]

Initialize the permutation matrix with a list.

Source [`permutation-matrix.lisp`], page 13.

`initialize-matrix-contents` ((*matrix* [`permutation-matrix`], *page 62*) (*initial-contents* `list`) *initargs*) [Method]

Initialize the permutation matrix with a list.

Source [`permutation-matrix.lisp`], page 13.

`size` (*object*) [Generic Reader]

Package [`linear-algebra`], page 23.

Methods

`size` ((*identity-matrix [`identity-matrix`], page 61*)) [Reader Method]

automatically generated reader method

Source [`identity-matrix.lisp`], page 12.

Target Slot
[`size`], page 62.

Appendix A Indexes

A.1 Concepts

(Index is nonexistent)

A.2 Functions

A.3 Variables

A.4 Data types