The Lisp Linear Algebra Reference Manual

Lisp Linear Algebra, version 0.3.1

Steven Nunez

This manual was generated automatically by Declt 4.0b2.

Table of Contents

Copying

This program is distributed under the terms of the Microsoft Public License.

1 Systems

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

1.1 `lla`

Lisp Linear Algebra

Author Steven Nunez

License ML-PL

Long Description

LLA is a high-level Common Lisp library built on on BLAS and LAPACK, but providing a more abstract interface with the purpose of freeing the user from low-level concerns and reducing the number of bugs in numerical code.

There are four main objectives:

- Provide a high-level, user friendly, interface that hides the details.
- Stay in Lisp and expose the innards of the objects as much as possible.
- Keeping it simple; LLA is only about 3000 lines of code.
- Speed is important, but reliability comes first.

Version 0.3.1

Dependencies

- `anaphora` (system).
- `alexandria` (system).
- `cffi` (system).
- `num-utils` (system).
- `select` (system).
- `let-plus` (system).

Source [`lla.asd`], page 5.

Child Components

- [`package.lisp`], page 5 (file).
- [`configuration-interface.lisp`], page 5 (file).
- [`configuration.lisp`], page 5 (file).
- [`libraries.lisp`], page 5 (file).
- [`conditions.lisp`], page 6 (file).
- [`types.lisp`], page 6 (file).
- [`foreign-memory.lisp`], page 7 (file).
- [`pinned-array.lisp`], page 7 (file).
- [`factorizations.lisp`], page 8 (file).
- [`fortran-call.lisp`], page 9 (file).
- [`linear-algebra.lisp`], page 11 (file).
- [`blas.lisp`], page 12 (file).

2 Files

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

2.1 Lisp

2.1.1 lla/lla.asd

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

ASDF Systems
[lla], page 3.

2.1.2 lla/package.lisp

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Packages [lla], page 13.

2.1.3 lla/configuration-interface.lisp

Dependency
[package.lisp], page 5 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Internals

- [query-configuration], page 44 (function).
- [set-feature], page 44 (function).

2.1.4 lla/configuration.lisp

Dependency
[configuration-interface.lisp], page 5 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Internals [default-libraries], page 41 (function).

2.1.5 lla/libraries.lisp

Dependency
[configuration.lisp], page 5 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

2.1.6 lla/conditions.lisp

Dependency
[libraries.lisp], page 5 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Public Interface

- [*lla-efficiency-warning-array-conversion*], page 19 (special variable).
- [*lla-efficiency-warning-array-type*], page 19 (special variable).
- [lapack-error], page 28 (condition).
- [lapack-failure], page 28 (condition).
- [lapack-invalid-argument], page 28 (condition).
- [lapack-singular-matrix], page 28 (condition).
- [lla-efficiency-warning], page 29 (condition).
- [lla-efficiency-warning-array-conversion], page 29 (condition).
- [lla-efficiency-warning-array-type], page 29 (condition).
- [lla-incompatible-dimensions], page 29 (condition).
- [lla-internal-error], page 30 (condition).
- [lla-unhandled-type], page 30 (condition).
- [print-object], page 27 (method).

2.1.7 lla/types.lisp

Dependency
[conditions.lisp], page 6 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Public Interface

- [lla-complex-double], page 33 (type).
- [lla-complex-single], page 33 (type).
- [lla-double], page 33 (type).
- [lla-integer], page 34 (type).
- [lla-single], page 34 (type).

Internals

- [+complex-double+], page 34 (constant).
- [+complex-single+], page 34 (constant).
- [+double+], page 34 (constant).
- [+float-types+], page 34 (constant).
- [+integer+], page 34 (constant).
- [+internal-types+], page 34 (constant).
- [+single+], page 34 (constant).
- [absolute-square-type], page 38 (function).

- [array-float-type], page 39 (function).
- [common-float-type], page 40 (function).
- [complex?], page 40 (function).
- [epsilon], page 41 (function).
- ⟨undefined⟩ [float-type], page ⟨undefined⟩ (type).
- ⟨undefined⟩ [internal-type], page ⟨undefined⟩ (type).
- [lisp-type], page 42 (function).
- [number-float-type], page 43 (function).

2.1.8 lla/foreign-memory.lisp

Dependency
[types.lisp], page 6 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Internals

- [all-from-specifications%], page 38 (function).
- [all-to-specifications%], page 38 (function).
- [array-clause%], page 36 (macro).
- [copy-array-from-memory], page 40 (function).
- [copy-array-to-memory], page 40 (function).
- [create-array-from-memory], page 41 (function).
- [create-transposed-matrix-from-memory], page 41 (function).
- [define-foreign-aref], page 36 (macro).
- [expand-specifications%], page 41 (function).
- [foreign-size], page 41 (function).
- ⟨undefined⟩ [maximum-array-size], page ⟨undefined⟩ (type).
- [transpose-matrix-from-memory], page 44 (function).
- [transpose-matrix-to-memory], page 44 (function).
- [value-from-memory%], page 45 (function).
- [value-to-memory%], page 45 (function).
- [with-fortran-atom], page 38 (macro).
- [with-fortran-character], page 38 (macro).

2.1.9 lla/pinned-array.lisp

Dependency
[foreign-memory.lisp], page 7 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Internals

- [backing-array], page 39 (function).
- [shareable-array?], page 44 (function).

- [with-array-input], page 37 (macro).
- [with-array-input-output], page 37 (macro).
- [with-array-output], page 37 (macro).
- [with-pinned-array], page 38 (macro).
- [with-work-area], page 38 (macro).

2.1.10 lla/factorizations.lisp

Dependency

[pinned-array.lisp], page 7 (file).

Source [lla.asd], page 5.

Parent Component

[lla], page 3 (system).

Public Interface

- [cholesky], page 30 (structure).
- [e2*], page 27 (method).
- [e2*], page 27 (method).
- [e2+], page 27 (method).
- [e2+], page 27 (method).
- [e2/], page 27 (method).
- [hermitian-factorization], page 32 (class).
- [ipiv], page 20 (function).
- [ipiv-inverse], page 21 (function).
- [left-square-root], page 24 (generic function).
- [lu], page 32 (class).
- [lu-l], page 21 (function).
- [lu-u], page 21 (function).
- [matrix-square-root], page 30 (structure).
- [print-object], page 27 (method).
- [qr], page 26 (reader method).
- [qr], page 33 (class).
- [qr-r], page 21 (function).
- [right-square-root], page 26 (generic function).
- [spectral-factorization], page 31 (structure).
- [spectral-factorization-w], page 22 (reader).
- [(setf spectral-factorization-w)], page 22 (writer).
- [spectral-factorization-z], page 22 (reader).
- [(setf spectral-factorization-z)], page 22 (writer).
- [svd], page 31 (structure).
- [svd-d], page 22 (reader).
- [(setf svd-d)], page 22 (writer).
- [svd-u], page 23 (reader).
- [(setf svd-u)], page 23 (writer).
- [svd-vt], page 23 (reader).

- [(setf svd-vt)], page 23 (writer).
- [xx], page 23 (function).

Internals

- [cholesky-left], page 39 (function).
- [(setf cholesky-left)], page 39 (function).
- [cholesky-p], page 40 (function).
- [copy-cholesky], page 40 (function).
- [copy-matrix-square-root], page 40 (function).
- [copy-spectral-factorization], page 40 (function).
- [copy-svd], page 40 (function).
- [count-permutations%], page 40 (function).
- [define-factorization-eops%], page 36 (macro).
- [define-matrix-square-root-scalar-multiplication], page 36 (macro).
- [factor], page 45 (reader method).
- [ipiv-internal], page 46 (reader method).
- ⟨undefined⟩ [ipiv-mixin], page ⟨undefined⟩ (class).
- [lu-matrix], page 46 (reader method).
- [make-cholesky], page 42 (function).
- [make-cholesky%], page 42 (function).
- [make-matrix-square-root], page 42 (function).
- [make-spectral-factorization], page 43 (function).
- [make-svd], page 43 (function).
- [matrix-square-root-left], page 43 (reader).
- [(setf matrix-square-root-left)], page 43 (writer).
- [matrix-square-root-p], page 43 (function).
- [permutations], page 46 (generic function).
- [spectral-factorization-p], page 44 (function).
- [svd-p], page 44 (function).
- [tau], page 47 (reader method).
- [(setf tau)], page 47 (writer method).

2.1.11 lla/fortran-call.lisp

Dependency

[factorizations.lisp], page 8 (file).

Source [lla.asd], page 5.

Parent Component

[lla], page 3 (system).

Public Interface

[with-fp-traps-masked], page 19 (macro).

Internals

- [&array-in], page 35 (macro).
- [&array-in/out], page 35 (macro).

2.1.12 lla/linear-algebra.lisp

Dependency
[fortran-call.lisp], page 9 (file).

Source [lla.asd], page 5.

Parent Component
[lla], page 3 (system).

Public Interface

- [as-array], page 27 (method).
- [as-array], page 27 (method).
- [as-array], page 27 (method).
- [cholesky], page 23 (generic function).
- [det], page 20 (function).
- [eigenvalues], page 20 (function).
- [hermitian-factorization], page 23 (generic function).
- [invert], page 23 (generic function).
- [invert-xx], page 24 (generic function).
- [least-squares], page 21 (function).
- [left-square-root], page 24 (method).
- [left-square-root], page 24 (method).
- [logdet], page 24 (generic function).
- [lu], page 24 (generic function).
- [mm], page 25 (generic function).
- [mmm], page 21 (function).
- [outer], page 25 (generic function).
- [qr], page 25 (method).
- [solve], page 26 (generic function).
- [spectral-factorization], page 22 (function).
- [svd], page 26 (generic function).
- [tr], page 27 (generic function).

Internals

- [diagonal-log-sum%], page 41 (function).
- [dimensions-as-matrix], page 41 (function).
- [invert-triangular%], page 42 (function).
- [last-rows-ss], page 42 (function).
- [least-squares-qr], page 42 (function).
- [log-with-sign%], page 37 (macro).
- [mm-hermitian%], page 43 (function).
- [sum-diagonal%], page 44 (function).
- [trsm%], page 44 (function).

2.1.13 `lla/blas.lisp`

Dependency
[`linear-algebra.lisp`], page 11 (file).

Source [`lla.asd`], page 5.

Parent Component
[`lla`], page 3 (system).

Public Interface

- [`asum`], page 19 (function).
- [`axpy!`], page 19 (function).
- [`copy!`], page 19 (function).
- [`dot`], page 20 (function).
- [`gemm!`], page 20 (function).
- [`nrm2`], page 21 (function).
- [`scal!`], page 21 (function).

3 Packages

Packages are listed by definition order.

3.1 lla

Source [package.lisp], page 5.

Use List

- alexandria.
- anaphora.
- cffi.
- common-lisp.
- let-plus.
- num-utils.
- select.

Used By List

ls-user.

Public Interface

- [*lla-efficiency-warning-array-conversion*], page 19 (special variable).
- [*lla-efficiency-warning-array-type*], page 19 (special variable).
- [asum], page 19 (function).
- [axpy!], page 19 (function).
- [cholesky], page 23 (generic function).
- [cholesky], page 30 (structure).
- [copy!], page 19 (function).
- [det], page 20 (function).
- [dot], page 20 (function).
- [eigenvalues], page 20 (function).
- [gemm!], page 20 (function).
- [hermitian-factorization], page 23 (generic function).
- [hermitian-factorization], page 32 (class).
- [invert], page 23 (generic function).
- [invert-xx], page 24 (generic function).
- [ipiv], page 20 (function).
- [ipiv-inverse], page 21 (function).
- [lapack-error], page 28 (condition).
- [lapack-failure], page 28 (condition).
- [lapack-invalid-argument], page 28 (condition).
- [lapack-singular-matrix], page 28 (condition).
- [least-squares], page 21 (function).
- [left-square-root], page 24 (generic function).
- [lla-complex-double], page 33 (type).
- [lla-complex-single], page 33 (type).

Internals

4 Definitions

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

4.1 Public Interface

4.1.1 Special variables

`*lla-efficiency-warning-array-conversion*` [Special Variable]
Toggles whether conversion of array elements to another type when used with foreign functions raises an LLA-EFFICIENCY-WARNING-ARRAY-CONVERSION warning.

Effective only when LLA was loaded & compiled with the appropriate settings in CL-USER::*LLA-CONFIGURATION*. See the documentation on configuration in the README.

Package [`lla`], page 13.

Source [`conditions.lisp`], page 6.

`*lla-efficiency-warning-array-type*` [Special Variable]
Toggles whether arrays with types not recognized as LLA types raise an LLA-EFFICIENCY-WARNING-ARRAY-TYPE warning.

Effective only when LLA was loaded & compiled with the appropriate settings in CL-USER::*LLA-CONFIGURATION*. See the documentation on configuration in the README.

Package [`lla`], page 13.

Source [`conditions.lisp`], page 6.

4.1.2 Macros

`with-fp-traps-masked` (**&body** *body*) [Macro]

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

4.1.3 Ordinary functions

`asum` (*x* **&key** *n incx*) [Function]
Return the L1 norm of X.

Package [`lla`], page 13.

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

`axpy!` (*alpha x y* **&key** *n incx incy*) [Function]

Package [`lla`], page 13.

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

`copy!` (*x y* **&key** *n incx incy*) [Function]

Package [`lla`], page 13.

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

`det` (*matrix*) [Function]
Determinant of a matrix. If you need the log of this, use LOGDET directly.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

`dot` (*x y* **&key** *n incx incy*) [Function]

Package [`lla`], page 13.

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

`eigenvalues` (*a* **&key** *abstol*) [Function]
Return the eigenvalues of A. See the documentation of SPECTRAL-FACTORIZATION about ABSTOL.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

`gemm!` (*alpha a b beta c* **&key** *transpose-a? transpose-b? m n k lda ldb ldc*) [Function]
Basically C = ALPHA * A' * B' + BETA * C. A' is A or its transpose depending on TRANSPOSE-A?. B' is B or its transpose depending on TRANSPOSE-B?. Returns C.

A' is an MxK matrix. B' is a KxN matrix. C is an MxN matrix.

LDA is the width of the matrix A (not of A'). If A is not transposed, then K <= LDA, if it's transposed then M <= LDA.

LDB is the width of the matrix B (not of B'). If B is not transposed, then N <= LDB, if it's transposed then K <= LDB.

In the example below M=3, N=2, K=5, LDA=6, LDB=3, LDC=4. The cells marked with + do not feature in the calculation.

```
  N
  –+
  –+
  K -B+
  –+
  –+
  +++
  K
  ––+ –++
  M –A–+ -C++
  ––+ –++
  ++++++ ++++
```

Package [`lla`], page 13.

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

`ipiv` (*object*) [Function]
Pivot indices, counting from 0, in a format understood by SELECT. Example:

```
(let+ (((&accessors-r/o lu-l lu-u ipiv) (lu a)))
(num= (select a ipiv) (mm lu-l lu-u) ipiv-inverse t)) ; => T
```

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`ipiv-inverse` (*object*) [Function]

Inverted permutation for pivot indices, in a format understood by SELECT.

Example:

```
(let+ (((&accessors-r/o lu-l lu-u ipiv-inverse) (lu a)))
(num= a (select (mm lu-l lu-u) ipiv-inverse t))) ; => T
```

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`least-squares` (*y x* **&rest** *rest* **&key** *method* **&allow-other-keys**) [Function]

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`lu-l` (*lu*) [Function]

Return the L part of an LU factorization.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`lu-u` (*lu*) [Function]

Return the U part of an LU factorization.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`mmm` (**&rest** *matrices*) [Function]

Multiply arguments from left to right using MM.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`nrm2` (*x* **&key** *n incx*) [Function]

Return the L2 norm of X.

Package [lla], page 13.

Source [blas.lisp], page 12.

`qr-r` (*qr*) [Function]

Return the R part of a QR factorization.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`scal!` (*alpha x* **&key** *n incx*) [Function]

X = alpha * X.

Package [lla], page 13.

Source [blas.lisp], page 12.

`spectral-factorization` (*a* **&key** *abstol*) [Function]
Return a spectral factorization of A.

The LAPACK manual says the following about ABSTOL:

The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

ABSTOL + EPS * max(|a|,|b|) ,

where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*|T| will be used in its place, where |T| is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

If high relative accuracy is important, set ABSTOL to DLAMCH('Safe minimum'). Doing so will guarantee that eigenvalues are computed to high relative accuracy when possible in future releases. The current code does not make any guarantees about high relative accuracy, but furutre releases will. See J. Barlow and J. Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices", LAPACK Working Note #7, for a discussion of which matrices define their eigenvalues to high relative accuracy.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`spectral-factorization-w` (*instance*) [Reader]
`(setf spectral-factorization-w)` (*instance*) [Writer]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Target Slot
[w], page 31.

`spectral-factorization-z` (*instance*) [Reader]
`(setf spectral-factorization-z)` (*instance*) [Writer]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Target Slot
[z], page 31.

`svd-d` (*instance*) [Reader]
`(setf svd-d)` (*instance*) [Writer]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Target Slot
[d], page 32.

`svd-u` (*instance*) [Reader]
`(setf svd-u)` (*instance*) [Writer]

Package [`lla`], page 13.

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

Target Slot
[`u`], page 32.

`svd-vt` (*instance*) [Reader]
`(setf svd-vt)` (*instance*) [Writer]

Package [`lla`], page 13.

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

Target Slot
[`vt`], page 32.

`xx` (*left-square-root*) [Function]
Convenience function to create a matrix from a left square root.

Package [`lla`], page 13.

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

4.1.4 Generic functions

`cholesky` (*a*) [Generic Function]
Cholesky factorization.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`cholesky` ((*a* `hermitian-matrix`)) [Method]

`hermitian-factorization` (*a*) [Generic Function]
Compute the hermitian factorization.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`hermitian-factorization` ((*a* `hermitian-matrix`)) [Method]

`invert` (*a* **&key** *tolerance* **&allow-other-keys**) [Generic Function]
Invert A. The inverse of matrix factorizations are other factorizations when appropriate, otherwise the result is a matrix. Usage note: inverting dense matrices is unnecessary and unwise in most cases, because it is numerically unstable. If you are solving many Ax=b equations with the same A, use a matrix factorization like LU.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`invert` ((*diagonal* `diagonal-matrix`) **&key** *tolerance*) [Method]
For pseudoinverse, suppressing diagonal elements below TOLERANCE (if given, otherwise / is just used without any checking.

`invert` ((*a* *[*`cholesky`*], page 30*) **&key**) [Method]

`invert` ((*a* `lower-triangular-matrix`) **&key**) [Method]

`invert` ((*a* `upper-triangular-matrix`) **&key**) [Method]

`invert` ((*a* `hermitian-matrix`) **&key**) [Method]

`invert` ((*a* *[*`hermitian-factorization`*], page 32*) **&key**) [Method]

`invert` ((*lu* *[*`lu`*], page 32*) **&key**) [Method]

`invert` ((*a* `array`) **&key**) [Method]

`invert-xx` (*xx*) [Generic Function]

Calculate (X^T X)-1 (which is used for calculating the variance of estimates) and return as a decomposition. Usually XX is a decomposition itself, eg QR returned by least squares. Note: this can be used to generate random draws, etc.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`invert-xx` ((*qr* *[*`qr`*], page 33*)) [Method]

`left-square-root` (*a*) [Generic Function]

Return X such that XX^T=A.

Package [`lla`], page 13.

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

Methods

`left-square-root` ((*a* `diagonal-matrix`)) [Method]

Source [`linear-algebra.lisp`], page 11.

`left-square-root` ((*hermitian-matrix* `hermitian-matrix`)) [Method]

Source [`linear-algebra.lisp`], page 11.

`left-square-root` ((*a* *[*`matrix-square-root`*], page 30*)) [Method]

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

Logarithm of the determinant of a matrix. Return -1, 1 or 0 (or equivalent) to correct for the sign, as a second value.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`logdet` ((*matrix* `upper-triangular-matrix`)) [Method]

`logdet` ((*matrix* `lower-triangular-matrix`)) [Method]

`logdet` ((*matrix* `array`)) [Method]

`lu` (*a*) [Generic Function]

LU decomposition of A

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`lu` ((*a* `array`)) [Method]

`mm` (*a* *b*) [Generic Function]

Matrix multiplication of A and B.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`mm` ((*a* (`eql t`)) (*b* `diagonal-matrix`)) [Method]

`mm` ((*a* `diagonal-matrix`) (*b* (`eql t`))) [Method]

`mm` ((*a* `diagonal-matrix`) (*b* `diagonal-matrix`)) [Method]

`mm` ((*a* `diagonal-matrix`) (*b* `vector`)) [Method]

`mm` ((*a* `vector`) (*b* `diagonal-matrix`)) [Method]

`mm` ((*a* `array`) (*b* `diagonal-matrix`)) [Method]

`mm` ((*a* `diagonal-matrix`) (*b* `array`)) [Method]

`mm` ((*a* (`eql t`)) (*b* `vector`)) [Method]

`mm` ((*a* `vector`) (*b* (`eql t`))) [Method]

`mm` ((*a* `vector`) (*b* `vector`)) [Method]

`mm` (*a* (*b* `wrapped-matrix`)) [Method]

`mm` ((*a* `wrapped-matrix`) *b*) [Method]

`mm` ((*a* `wrapped-matrix`) (*b* `wrapped-matrix`)) [Method]

`mm` ((*a* (`eql t`)) (*b* `array`)) [Method]

`mm` ((*a* `array`) (*b* (`eql t`))) [Method]

`mm` ((*a* `array`) (*b* `array`)) [Method]

`mm` (*a* *b*) [Method]

`outer` (*a* *b*) [Generic Function]

Return the outer product column(a) row(b)^H. If either A or B is T, they are taken to be conjugate transposes of the other argument.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`outer` ((*a* `vector`) (*b* (`eql t`))) [Method]

`outer` ((*a* (`eql t`)) (*b* `vector`)) [Method]

`outer` ((*a* `vector`) (*b* `vector`)) [Method]

`qr` (*object*) [Generic Function]

Package [`lla`], page 13.

Methods

`qr` ((*a* `array`)) [Method]

Source [`linear-algebra.lisp`], page 11.

`qr` ((*qr* [`qr`], *page 33*)) [Reader Method]
matrix storing the QR factorization.

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

Target Slot
[`qr`], page 33.

`right-square-root` (*a*) [Generic Function]
Return Y such that Y^T Y=A.

Efficiency note: decompositions should store the left square root X, and compute Y=X^T on demand, so getting X directly might be more efficient if you don't need X^T.

Package [`lla`], page 13.

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

Methods

`right-square-root` ((*a* [`matrix-square-root`], *page 30*)) [Method]

`right-square-root` (*a*) [Method]

`solve` (*a* *b*) [Generic Function]
Return X that solves AX=B. When B is a vector, so is X.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`solve` ((*a* `diagonal-matrix`) *b*) [Method]

`solve` ((*a* `upper-triangular-matrix`) *b*) [Method]

`solve` ((*a* `lower-triangular-matrix`) *b*) [Method]

`solve` ((*a* `hermitian-matrix`) (*b* `array`)) [Method]

`solve` ((*a* [`hermitian-factorization`], *page 32*) (*b* `array`)) [Method]

`solve` ((*hermitian-matrix* `hermitian-matrix`) *b*) [Method]

`solve` ((*cholesky* [`cholesky`], *page 30*) *b*) [Method]

`solve` ((*a* `array`) (*b* `array`)) [Method]

`solve` ((*lu* [`lu`], *page 32*) (*b* `array`)) [Method]

`solve` (*a* (*b* `wrapped-matrix`)) [Method]

`svd` (*a* **&optional** *vectors*) [Generic Function]
Return singular value decomposition A.

VECTORS determines how singular vectors are calculated:

- NIL sets U and VT to NIL
- :ALL makes U and VT square, with dimensions conforming to A
- :THIN makes the larger of U and VT rectangular. This means that not all of the singular vectors are calculates, and saves computational time when A is far from square.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`svd` ((*a* `array`) **&optional** *vectors*) [Method]

`tr` (*a*) [Generic Function]

Trace of a square matrix.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

Methods

`tr` ((*a* `array`)) [Method]

`tr` ((*a* `wrapped-matrix`)) [Method]

`tr` ((*a* `diagonal-matrix`)) [Method]

4.1.5 Standalone methods

`as-array` ((*matrix-square-root* *[*`matrix-square-root`*], page 30*)) [Method]

Package `array-operations/generic`.

Source [`linear-algebra.lisp`], page 11.

`as-array` ((*svd* *[*`svd`*], page 31*)) [Method]

Package `array-operations/generic`.

Source [`linear-algebra.lisp`], page 11.

`as-array` ((*sf* *[*`spectral-factorization`*], page 31*)) [Method]

Package `array-operations/generic`.

Source [`linear-algebra.lisp`], page 11.

`e2*` ((*a* `number`) (*b* *[*`matrix-square-root`*], page 30*)) [Method]

Package `num-utils.elementwise`.

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

`e2*` ((*a* *[*`matrix-square-root`*], page 30*) (*b* `number`)) [Method]

Package `num-utils.elementwise`.

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

`e2+` (*a* (*b* *[*`matrix-square-root`*], page 30*)) [Method]

Package `num-utils.elementwise`.

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

`e2+` ((*a* *[*`matrix-square-root`*], page 30*) *b*) [Method]

Package `num-utils.elementwise`.

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

`e2/` ((*a* *[*`matrix-square-root`*], page 30*) (*b* `number`)) [Method]

Package `num-utils.elementwise`.

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

`print-object` ((*lu* *[*`lu`*], page 32*) *stream*) [Method]

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

`print-object` ((*object* *[*`lla-efficiency-warning-array-type`*], page 29*) *stream*) [Method]

Source [`conditions.lisp`], page 6.

4.1.6 Conditions

`lapack-error` [Condition]

The LAPACK procedure returned a nonzero info code.

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses

error.

Direct subclasses

- [lapack-failure], page 28.
- [lapack-invalid-argument], page 28.
- [lla-incompatible-dimensions], page 29.

`lapack-failure` [Condition]

Superclass of all LAPACK errors with a positive INFO

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses

[lapack-error], page 28.

Direct subclasses

[lapack-singular-matrix], page 28.

Direct slots

`info` [Slot]

INFO corresponding to error message.

Initargs :info

`lapack-invalid-argument` [Condition]

An argument to a LAPACK procedure had an illegal value. It is very likely that this indicates a bug in LLA and should not happen.

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses

- [lapack-error], page 28.
- [lla-internal-error], page 30.

Direct slots

`position` [Slot]

Position of the illegal argument

Package common-lisp.

Initargs :position

`lapack-singular-matrix` [Condition]

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses

[lapack-failure], page 28.

`lla-efficiency-warning` [Condition]

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses
`warning`.

Direct subclasses
- [lla-efficiency-warning-array-conversion], page 29.
- [lla-efficiency-warning-array-type], page 29.

`lla-efficiency-warning-array-conversion` [Condition]

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses
[lla-efficiency-warning], page 29.

Direct slots

`array` [Slot]
The array that had to be copied.

Package `common-lisp`.

Initargs `:array`

`type` [Slot]
Required element type.

Package `common-lisp`.

Initargs `:type`

`lla-efficiency-warning-array-type` [Condition]
See *LLA-EFFICIENCY-WARNING-ARRAY-TYPE*.

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses
[lla-efficiency-warning], page 29.

Direct methods
[print-object], page 27.

Direct slots

`array` [Slot]

Package `common-lisp`.

Initargs `:array`

`lla-incompatible-dimensions` [Condition]

Package [lla], page 13.

Source [conditions.lisp], page 6.

Direct superclasses
[lapack-error], page 28.

`lla-internal-error` [Condition]
Internal error in LLA.

Package [`lla`], page 13.

Source [`conditions.lisp`], page 6.

Direct superclasses
`error`.

Direct subclasses
[`lapack-invalid-argument`], page 28.

Direct slots

`message` [Slot]
Initform `(quote "")`
Initargs `:message`

`lla-unhandled-type` [Condition]
Could not classify object as a numeric type handled by LLA.

Package [`lla`], page 13.

Source [`conditions.lisp`], page 6.

Direct superclasses
`error`.

Direct slots

`object` [Slot]
Initform `(quote :object)`

4.1.7 Structures

`cholesky` [Structure]
Cholesky factorization a matrix.

Package [`lla`], page 13.

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

Direct superclasses
[`matrix-square-root`], page 30.

Direct methods
- [`invert`], page 24.
- [`solve`], page 26.

`matrix-square-root` [Structure]
General class for representing XX^T decompositions of matrices, regardless of how they were computed. The convention is to store X, the left square root.

Package [`lla`], page 13.

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

Direct superclasses
`structure-object`.

Direct subclasses
[`cholesky`], page 30.

Direct methods

- [as-array], page 27.
- [e2*], page 27.
- [e2*], page 27.
- [e2+], page 27.
- [e2+], page 27.
- [e2/], page 27.
- [left-square-root], page 24.
- [right-square-root], page 26.

Direct slots

`left` [Slot]

Package num-utils.interval.

Readers [matrix-square-root-left], page 43.

Writers [(setf matrix-square-root-left)], page 43.

`spectral-factorization` [Structure]

Z W Z^T factorization of a Hermitian matrix, where the columns of Z contain the eigenvectors and W is a diagonal matrix of the eigenvalues. Z is a unitary matrix.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Direct superclasses

structure-object.

Direct methods

[as-array], page 27.

Direct slots

`z` [Slot]

Readers [spectral-factorization-z], page 22.

Writers [(setf spectral-factorization-z)], page 22.

`w` [Slot]

Readers [spectral-factorization-w], page 22.

Writers [(setf spectral-factorization-w)], page 22.

`svd` [Structure]

Singular value decomposition. Singular values are in S, in descending order. U and VT may be NIL in case they are not computed.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Direct superclasses

structure-object.

Direct methods

[as-array], page 27.

Direct slots

`u` [Slot]

Readers [`svd-u`], page 23.

Writers [`(setf svd-u)`], page 23.

`d` [Slot]

Readers [`svd-d`], page 22.

Writers [`(setf svd-d)`], page 22.

`vt` [Slot]

Readers [`svd-vt`], page 23.

Writers [`(setf svd-vt)`], page 23.

4.1.8 Classes

`hermitian-factorization` [Class]

Factorization for an indefinite hermitian matrix with pivoting.

Package [`lla`], page 13.

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

Direct superclasses

⟨undefined⟩ [`ipiv-mixin`], page ⟨undefined⟩.

Direct methods

- [`factor`], page 45.
- [`invert`], page 24.
- [`solve`], page 26.

Direct slots

`factor` [Slot]

see documentation of *SYTRF and *HETRF, storage is
in the half specified by HERMITIAN-ORIENTATION and otherwise treated as opaque.

Initargs `:factor`

Readers [`factor`], page 45.

Writers *This slot is read-only.*

`lu` [Class]

LU factorization of a matrix with pivoting. (SELECT A IPIV) is (MM L U), when IPIV is used to obtain the permutation.

Package [`lla`], page 13.

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

Direct superclasses

⟨undefined⟩ [`ipiv-mixin`], page ⟨undefined⟩.

Direct methods

- [`invert`], page 24.
- [`lu-matrix`], page 46.

- [permutations], page 46.
- [print-object], page 27.
- [solve], page 26.

Direct slots

`lu` [Slot]
matrix storing the transpose of the LU factorization.

Initargs :lu

Readers [lu-matrix], page 46.

Writers *This slot is read-only.*

`qr` [Class]
QR factorization of a matrix.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Direct methods

- [invert-xx], page 24.
- [qr], page 26.
- [(setf tau)], page 47.
- [tau], page 47.

Direct slots

`qr` [Slot]
matrix storing the QR factorization.

Initargs :qr

Readers [qr], page 26.

Writers *This slot is read-only.*

`tau` [Slot]
complex scalar for elementary reflectors (see documentation of xGEQRF).

Initargs :tau

Readers [tau], page 47.

Writers [(setf tau)], page 47.

4.1.9 Types

`lla-complex-double` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`lla-complex-single` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`lla-double` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`lla-integer` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`lla-single` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

4.2 Internals

4.2.1 Constants

`+complex-double+` [Constant]

Package [lla], page 13.

Source [types.lisp], page 6.

`+complex-single+` [Constant]

Package [lla], page 13.

Source [types.lisp], page 6.

`+double+` [Constant]

Package [lla], page 13.

Source [types.lisp], page 6.

`+float-types+` [Constant]

List of all internal float types.

Package [lla], page 13.

Source [types.lisp], page 6.

`+integer+` [Constant]

Package [lla], page 13.

Source [types.lisp], page 6.

`+internal-types+` [Constant]

List of all internal types.

Package [lla], page 13.

Source [types.lisp], page 6.

`+single+` [Constant]

Package [lla], page 13.

Source [types.lisp], page 6.

4.2.2 Symbol macros

`&info` [Symbol Macro]

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

4.2.3 Macros

`&array-in` (*input* **&key** *type transpose? force-copy?*) [Macro]
Array which serves as an input. See FORTRAN-INPUT-ARRAY.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&array-in/out` ((**&key** *input type transpose? force-copy?*) (**&key** *output dimensions type transpose?*)) [Macro]
Input/output array. See FORTRAN-INPUT-OUTPUT-ARRAY.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&array-out` (*output* **&key** *dimensions type transpose?*) [Macro]
Output array. See FORTRAN-OUTPUT-ARRAY.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&atom` (*value* **&key** *type output*) [Macro]
Atoms passed to FORTRAN. When not given, TYPE is inferred from the call's default. VALUE is coerced to the desired type. When OUTPUT is given, value is read after the call and placed there.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&char` (*character*) [Macro]
Shorthand for character.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&info` (**&key** *condition variable*) [Macro]
Argument for checking whether the call was executed without an error. Automatically takes care of raising the appropriate condition if it wasn't. CONDITION specifies the condition to raise in case of positive error codes, use NIL to ignore these. VARIABLE can be used to specify

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&integer` (*value* **&key** *output*) [Macro]
Shorthand for integer atom.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&integers` (**&rest** *values*) [Macro]
Shorthand for integer atoms (which are not modified). Useful for combining arguments.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`&new` (*variable*) [Macro]
Placeholder macro for newly allocated output variables. Using (&NEW VARIABLE) allocates a new array within the scope of the outer macro, and is usually used for output. See &ARRAY-OUT and &ARRAY-IN/OUT.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`&work` (*size* **&optional** *type*) [Macro]
Allocate a work area of SIZE. When TYPE is not given, the call's default is used.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`&work-query` (**&optional** *type*) [Macro]
Work area query, takes the place of TWO fortran arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`array-clause%` ((*array internal-type clause-element-type clause-internal-type*) **&body** *body*) [Macro]
Macro that generates a lambda form that can bed used in EXPAND-SPECIFICATIONS%.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`blas-call` ((*name type* **&optional** *value return-types)* **&body** *forms*) [Macro]
BLAS call. NAME is either a string or a list of two strings (real/complex). TYPE (internal-type) selects the function to call. VALUE is the form returned after the call.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`define-factorization-eops%` (*type conversion*) [Macro]
Define elementwise operations for TYPE, trying to convert into arrays.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`define-foreign-aref` () [Macro]

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`define-matrix-square-root-scalar-multiplication` (*type* **&key** *make*) [Macro]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`lapack-call` ((*name type value*) **&body** *forms*) [Macro]
LAPACK call, takes an &info argument. NAME is either a string or a list of two strings (real/complex). TYPE (internal-type) selects the function to call. VALUE is the form returned after the call.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`lapack-call-w/query` ((*name type value*) **&body** *forms*) [Macro]
LAPACK call which also takes &work-query arguments (in place of two FORTRAN arguments).

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

`log-with-sign%` (*value sign-changes block-name*) [Macro]
Log of (ABS VALUE), increments SIGN-CHANGES when negative, return-from block-name (values nil 0) when zero.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

`with-array-input` (((*pointer* **&optional** *copied?*) *array internal-type transpose? force-copy?*) **&body** *body*) [Macro]
Ensure that ARRAY is mapped to a corresponding memory area for the duration of BODY (see below for details of semantics). POINTER is bound to the start of the memory address. The representation of values in the memory is determined by INTERNAL-TYPE. When TRANSPOSE?, transpose the array before BODY (only works for matrices, otherwise signal an error).

If FORCE-COPY? is false, POINTER may or may not point to the memory backing ARRAY.

If FORCE-COPY? is true, POINTER always points to a copy of the contents of ARRAY.

COPIED? is bound to indicate whether POINTER points to a copy or the actual array contents.

The value of the expression is always the value of BODY.

Package [`lla`], page 13.

Source [`pinned-array.lisp`], page 7.

`with-array-input-output` (((*pointer* **&optional** *copied?*) *input-array input-internal-type input-transpose? input-force-copy? output-array output-internal-type output-transpose?*) **&body** *body*) [Macro]
Similar to WITH-ARRAY-INPUT, it also ensures that OUTPUT-ARRAY contains the contents the memory area pointed to by POINTER when BODY is finished. If OUTPUT-ARRAY is NIL then it is equivalent to WITH-ARRAY-INPUT.

Package [`lla`], page 13.

Source [`pinned-array.lisp`], page 7.

`with-array-output` (((*pointer* **&optional** *copied?*) *array internal-type transpose?*) **&body** *body*) [Macro]
Ensure that ARRAY is mapped to a corresponding memory area for the duration of BODY (see below for details of semantics). POINTER is bound to the start of the memory address. The representation of values in the memory is determined by INTERNAL-TYPE. When TRANSPOSE?, transpose the array after BODY (only works for matrices, otherwise signal an error).

COPIED? is bound to indicate whether POINTER points to a copy or the actual array contents.

The value of the expression is always the value of BODY.

Package [lla], page 13.

Source [pinned-array.lisp], page 7.

`with-fortran-atom` ((*pointer value type output*) **&body** *body*) [Macro]
Allocate memory for internal TYPE and set it to VALUE for body, which can use POINTER to access it. When OUTPUT is given, the value is assigned to it after BODY. The atom is automatically coerced to the correct type (by FOREIGN-AREF).

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`with-fortran-character` ((*pointer character*) **&body** *body*) [Macro]
Make character available in an allocated memory area at POINTER for BODY.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`with-pinned-array` ((*pointer array*) **&body** *body*) [Macro]

Package [lla], page 13.

Source [pinned-array.lisp], page 7.

`with-work-area` ((*pointer internal-type size*) **&body** *body*) [Macro]
Allocate a work area of SIZE and INTERNAL-TYPE, and bind the POINTER to its start during BODY.

Package [lla], page 13.

Source [pinned-array.lisp], page 7.

4.2.4 Ordinary functions

`absolute-square-type` (*internal-type*) [Function]
Type of (* x (conjugate x)).

Package [lla], page 13.

Source [types.lisp], page 6.

`all-from-specifications%` () [Function]
Return an optimization specification for all functions that copy from foreign memory.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`all-to-specifications%` () [Function]
Return an optimization specification for all functions that copy to foreign memory.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`argument-pointers` (*arguments*) [Function]
Return the list of pointers for all the arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`arguments-for-cffi` (*arguments*) [Function]
Return a list that can be use in a CFFI call.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`array-float-type` (*array*) [Function]
Return an (internal) float type for an array. O(1) when the type can be detected from the specialized array element type, O(n) otherwise.

Package [lla], page 13.

Source [types.lisp], page 6.

`assert-single-lapack-info` (*arguments*) [Function]
Assert that there is at most one LAPACK-INFO in ARGUMENTS.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`backing-array` (*array*) [Function]
Return the array in which the contents of ARRAY are stored. For simple arrays, this is always the array itself. The second value is the displacement.

Package [lla], page 13.

Source [pinned-array.lisp], page 7.

`blas-lapack-call-form` (*type-var name arguments* **&optional** *return-types*) [Function]
Return a form BLAS/LAPACK calls, conditioning on TYPE-VAR. See BLAS-LAPACK-FUNCTION-NAME for the interpretation of FIXME

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`blas-lapack-function-name` (*type name*) [Function]
Return the BLAS/LAPACK foreign function name. TYPE is the internal type, NAME is one of the following: NAME, (NAME), which are used for both complex and real names, or (REAL-NAME COMPLEX-NAME).

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`blas-return-types` (*return-types*) [Function]

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`cholesky-left` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`(setf cholesky-left)` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`cholesky-p` (*object*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`common-float-type` (**&rest** *arrays-or-numbers*) [Function]
Return the common (internal) float type for the arguments.

Package [lla], page 13.

Source [types.lisp], page 6.

`complex?` (*internal-type*) [Function]
True iff the internal type is complex.

Package [lla], page 13.

Source [types.lisp], page 6.

`copy-array-from-memory` (*array pointer internal-type*) [Function]
Copy the memory area of type INTERNAL-TYPE at POINTER to ARRAY.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`copy-array-to-memory` (*array pointer internal-type*) [Function]
Copy the contents of ARRAY to the memory area of type INTERNAL-TYPE at POINTER.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`copy-cholesky` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`copy-matrix-square-root` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`copy-spectral-factorization` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`copy-svd` (*instance*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`count-permutations%` (*ipiv-internal*) [Function]
Count the permutations in a pivoting vector.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`create-array-from-memory` (*pointer internal-type dimensions* **&optional** *element-type*) [Function]
Create an array from contents at POINTER.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`create-transposed-matrix-from-memory` (*pointer internal-type dimensions* **&optional** *element-type*) [Function]
Create a matrix from transposed contents at POINTER.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`default-libraries` () [Function]
Return a list of libraries. The source conditions on the platform, relying TRIVIAL-FEATURES. This function is only called when the libraries were not configured by the user, see the documentation on how to do that.

Package [lla], page 13.

Source [configuration.lisp], page 5.

`diagonal-log-sum%` (*matrix* **&optional** *sign-changes*) [Function]
Sum of the log of the elements in the diagonal. Sign-changes counts the negative values, and may be started at something else than 0 (eg in case of pivoting). Return (values NIL 0) in case of encountering a 0.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`dimensions-as-matrix` (*array orientation*) [Function]
If ARRAY is a vector, return its dimensions as a matrix with given ORIENTATION (:ROW or :COLUMN) as multiple values, and T as the third value. If it is matrix, just return the dimensions and NIL. Used for considering vectors as conforming matrices (eg see MM).

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`epsilon` (*internal-type*) [Function]
Return the float epsilon for the given internal float type.

Package [lla], page 13.

Source [types.lisp], page 6.

`expand-specifications%` (*clause specifications*) [Function]
Expand specifications using (clause internal-type element-type).

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`foreign-size` (*type*) [Function]
Return the size of an internal type in bytes.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`fortran-argument/new-variable` (*form*) [Function]
If FORM is (&NEW VARIABLE), return VARIABLE, otherwise NIL.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`invert-triangular%` (*a upper? unit-diag?*) [Function]
Invert a dense (triangular) matrix using the LAPACK routine *TRTRI. UPPER? indicates if the matrix is in the upper or the lower triangle of a matrix, UNIT-DIAG? indicates whether the diagonal is supposed to consist of 1s. For internal use, not exported.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`lapack-info-wrap-argument` (*argument body*) [Function]
Generate a wrapper for a LAPACK INFO argument, also checking the result and raising a condition if applicable. Useful for WRAP-ARGUMENT.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`last-rows-ss` (*matrix nrhs common-type*) [Function]
Calculate the sum of squares of the last rows of MATRIX columnwise,
omitting the first NRHS rows. If MATRIX is a vector, just do this for the last elements. Used for interfacing with xGELS.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`least-squares-qr` (*y x* **&key &allow-other-keys**) [Function]
Least squares using QR decomposition. Additional values returned: the QR decomposition of X. See LEAST-SQUARES for additional documentation. Usage note: SVD-based methods are recommended over this one, unless X is well-conditioned.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`lisp-type` (*internal-type*) [Function]

Package [lla], page 13.

Source [types.lisp], page 6.

`make-cholesky` (*left*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`make-cholesky%` (*left*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`make-matrix-square-root` (*left*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`make-spectral-factorization` (**&key** *z w*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`make-svd` (**&key** *u d vt*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`matrix-square-root-left` (*instance*) [Reader]
`(setf matrix-square-root-left)` (*instance*) [Writer]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Target Slot
[left], page 31.

`matrix-square-root-p` (*object*) [Function]

Package [lla], page 13.

Source [factorizations.lisp], page 8.

`maybe-default-type` (*type parameters*) [Function]
Return default type from PARAMETERS when TYPE is NIL.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`mm-hermitian%` (*a transpose-left?*) [Function]
Calculate A*op(A) if TRANSPOSE-LEFT? is NIL, and op(A)*A otherwise. op() is always conjugate transpose, but may be implemented as a transpose if A is real, in which case the two are equivalent. This function is meant to be used internally, and is not exported.

Package [lla], page 13.

Source [linear-algebra.lisp], page 11.

`number-float-type` (*number*) [Function]
Return an (internal) float type for a number.

Package [lla], page 13.

Source [types.lisp], page 6.

`output-array-form` (*form*) [Function]
Return a form that can be passed to WITH-OUTPUT-ARRAY (or similar) as an output. The variable is extracted from &NEW forms, otherwise the form is passed as is.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`process-forms` (*forms environment*) [Function]
Process forms and return a list of argument specifications. A form may correspond to multiple arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

`query-configuration` (*indicator* **&optional** *default*) [Function]
Return the property for INDICATOR from the configuration variable, with an optional default, which can be a function (called on demand).

Package [`lla`], page 13.

Source [`configuration-interface.lisp`], page 5.

`set-feature` (*symbol set?*) [Function]
Ensure that symbol is in *FEATURES* iff SET?. Returns no values.

Package [`lla`], page 13.

Source [`configuration-interface.lisp`], page 5.

`shareable-array?` (*array internal-type*) [Function]

Package [`lla`], page 13.

Source [`pinned-array.lisp`], page 7.

`spectral-factorization-p` (*object*) [Function]

Package [`lla`], page 13.

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

`sum-diagonal%` (*array*) [Function]
Sum diagonal of array, checking that it is square.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

`svd-p` (*object*) [Function]

Package [`lla`], page 13.

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

`transpose-matrix-from-memory` (*matrix pointer internal-type*) [Function]
Transpose the contents of ARRAY to the memory area of type INTERNAL-TYPE at POINTER. VECTORs are also handled.

Package [`lla`], page 13.

Source [`foreign-memory.lisp`], page 7.

`transpose-matrix-to-memory` (*matrix pointer internal-type*) [Function]
Transpose the contents of ARRAY to the memory area of type INTERNAL-TYPE at POINTER. VECTORs are also handled.

Package [`lla`], page 13.

Source [`foreign-memory.lisp`], page 7.

`trsm%` (*a a-upper? b*) [Function]
Wrapper for BLAS routine xTRSM. Solve AX=B, where A is triangular.

Package [`lla`], page 13.

Source [`linear-algebra.lisp`], page 11.

`value-from-memory%` (*internal-type*) [Function]
Return a (LAMBDA (POINTER INDEX) ...) form that can be used to read an element from an array in memory.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`value-to-memory%` (*internal-type*) [Function]
Return a (LAMBDA (POINTER INDEX VALUE) ...) form that can be used to write an element to an array in memory.

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

`wrap-arguments` (*arguments pass parameters body*) [Function]
Wrap BODY in arguments. Convenience function used to implement the expansion.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

4.2.5 Generic functions

`argument-pointer` (*object*) [Generic Reader]

Package [lla], page 13.

Methods

`argument-pointer` ((*fortran-argument* [`fortran-argument`], *page 48*)) [Reader Method]
Pointer passed to FORTRAN.

Source [fortran-call.lisp], page 9.

Target Slot
[pointer], page 48.

`factor` (*object*) [Generic Reader]

Package [lla], page 13.

Methods

`factor` ((*hermitian-factorization* [`hermitian-factorization`], *page 32*)) [Reader Method]
see documentation of *SYTRF and *HETRF, storage is
in the half specified by HERMITIAN-ORIENTATION and otherwise treated as opaque.

Source [factorizations.lisp], page 8.

Target Slot
[factor], page 32.

`fortran-argument/output-initializer-form` (*argument parameters*) [Generic Function]
When applicable, return a form that is used to initialize the OUTPUT variable. When NIL is returned, no binding is established.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Methods

`fortran-argument/output-initializer-form` ((*argument* ⟨*undefined*⟩ *[*`fortran-input-output-array`*], page* ⟨*undefined*⟩) *parameters*) [Method]

`fortran-argument/output-initializer-form` ((*argument* ⟨*undefined*⟩ *[*`fortran-output-array`*], page* ⟨*undefined*⟩) *parameters*) [Method]

`fortran-argument/output-initializer-form` ((*argument* *[*`fortran-argument/output`*], page 49*) *parameters*) [Method]

`ipiv-internal` (*object*) [Generic Reader]

Package [`lla`], page 13.

Methods

`ipiv-internal` ((*ipiv-mixin* ⟨*undefined*⟩ *[*`ipiv-mixin`*], page* ⟨*undefined*⟩)) [Reader Method]

pivot indices in LAPACK's representation, counting from 1

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

Target Slot

⟨undefined⟩ [`ipiv-internal`], page ⟨undefined⟩.

`lu-matrix` (*object*) [Generic Reader]

Package [`lla`], page 13.

Methods

`lu-matrix` ((*lu [*`lu`*], page 32*)) [Reader Method]

matrix storing the transpose of the LU factorization.

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

Target Slot

[`lu`], page 33.

`permutations` (*object*) [Generic Function]

Return the number of permutations in object (which is usually a matrix factorization, or a pivot index.

Package [`lla`], page 13.

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

Methods

`permutations` ((*lu [*`lu`*], page 32*)) [Method]

`process-form` (*form environment*) [Generic Function]

Return a list of argument specifications (atoms are converted into lists).

The code in this file defines a DSL for calling BLAS and LAPACK procedures in FORTRAN, which expect pointers to memory locations. The high-level macros BLAS-CALL and LAPACK-CALL take care of pinning arrays or allocating memory and copying array contents (when applicable), and use PROCESS-FORM to interpret their arguments, which correspond to one or more FORTRAN arguments. These are then expanded into code using WRAP-ARGUMENT.

Package [`lla`], page 13.

Source [fortran-call.lisp], page 9.

Methods

`process-form` ((*form* `(eql 1)`) *env*) [Method]

`process-form` ((*form* `(eql 0)`) *env*) [Method]

`process-form` ((*form* `character`) *env*) [Method]

`process-form` ((*form* `null`) *environment*) [Method]

`process-form` (*form environment*) [Method]

`tau` (*object*) [Generic Reader]
`(setf tau)` (*object*) [Generic Writer]

Package [lla], page 13.

Methods

`tau` ((*qr [qr], page 33*)) [Reader Method]
`(setf tau)` ((*qr [qr], page 33*)) [Writer Method]

complex scalar for elementary reflectors (see documentation of xGEQRF).

Source [factorizations.lisp], page 8.

Target Slot

[tau], page 33.

`wrap-argument` (*argument pass parameters body*) [Generic Function]

Return BODY wrapped in an environment generated for ARGUMENT in a given PASS.

The code for calling the function is built using nested calls of WRAP-ARGUMENT. The setup is somewhat complicated because some functions require that they are called twice (eg in order to calculate work area size), and we don't necessarily want to allocate and free memory twice. Because of this, expansions take place in 'passes'.

Currently, the following passes are used:

- BINDINGS: establishes the bindings (also empty variables)

- MAIN: for arguments that are the same regardless of what kind of call is made

- QUERY: for querying work area sizes

- CALL: the actual function call

PARAMETERS is used to specify information that is applicable for all arguments (eg a default array element type).

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Methods

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-work-query-size`*], page* ⟨*undefined*⟩) (*pass* `(eql lla::call)`) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-work-query-area`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::call`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-work-query-size`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::query`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-work-query-area`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::query`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-work-query-area`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::bindings`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-info`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::query`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`lapack-info`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::call`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-work-area`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-input-output-array`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-output-array`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-input-array`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-atom`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument* ⟨*undefined*⟩ *[*`fortran-character`*]*, *page* ⟨*undefined*⟩) (*pass* (`eql lla::main`)) *parameters body*) [Method]

`wrap-argument` ((*argument [*`fortran-argument/output`*]*, *page 49*) (*pass* (`eql lla::bindings`)) *parameters body*) [Method]

`wrap-argument` (*argument pass parameters body*) [Method]

4.2.6 Classes

`fortran-argument` [Class]

Superclass of all arguments with pointers.

Package [`lla`], page 13.

Source [`fortran-call.lisp`], page 9.

Direct subclasses

- [`fortran-argument/output`], page 49.

- [fortran-argument/size], page 49.
- [fortran-argument/type], page ⟨undefined⟩.
- ⟨undefined⟩ [fortran-character], page ⟨undefined⟩.
- ⟨undefined⟩ [fortran-input-array], page ⟨undefined⟩.
- ⟨undefined⟩ [lapack-info], page ⟨undefined⟩.

Direct methods

[argument-pointer], page 45.

Direct slots

`pointer` [Slot]

Pointer passed to FORTRAN.

Initform `(gensym)`

Initargs `:pointer`

Readers [argument-pointer], page 45.

Writers *This slot is read-only.*

`fortran-argument/output` [Class]

Class for arguments that return an output. When FORTRAN-ARGUMENT/OUTPUT-INITIALIZER-FORM returns non-NIL, a local binding of OUTPUT to this form will be wrapped around the relevant BODY. Also see &NEW.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses

[fortran-argument], page 48.

Direct subclasses

- ⟨undefined⟩ [fortran-atom], page ⟨undefined⟩.
- ⟨undefined⟩ [fortran-output-array], page ⟨undefined⟩.

Direct methods

- [fortran-argument/output-initializer-form], page 46.
- [wrap-argument], page 48.

Direct slots

`output` [Slot]

Lisp variable or initialization form mapped to an output.

Initargs `:output`

`fortran-argument/size` [Class]

Number of elements in array mapped to a pointer.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses

[fortran-argument], page 48.

Direct subclasses

- ⟨undefined⟩ [fortran-work-area], page ⟨undefined⟩.
- ⟨undefined⟩ [lapack-work-query-area], page ⟨undefined⟩.

- ⟨undefined⟩ [lapack-work-query-size], page ⟨undefined⟩.

Direct slots

size [Slot]
Number of elements.

Initargs :size

fortran-argument/type [Class]
For arguments which may have multiple types, mostly arrays or atoms (implemented as single-cell arrays).

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
[fortran-argument], page 48.

Direct subclasses
- ⟨undefined⟩ [fortran-atom], page ⟨undefined⟩.
- ⟨undefined⟩ [fortran-work-area], page ⟨undefined⟩.
- ⟨undefined⟩ [lapack-work-query-area], page ⟨undefined⟩.

Direct slots

type [Slot]
Determines (internal) type for array mapped to the pointer.

Package common-lisp.

Initargs :type

fortran-atom [Class]
Atoms passed to FORTRAN. Input/output (when OUTPUT is given).

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
- [fortran-argument/output], page 49.
- [fortran-argument/type], page ⟨undefined⟩.

Direct methods
[wrap-argument], page 48.

Direct slots

value [Slot]

Initargs :value

fortran-character [Class]
Character passed to FORTRAN. Input only, for specifying triangle orientation, etc.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
[fortran-argument], page 48.

Direct methods
[wrap-argument], page 48.

Direct slots

`character` [Slot]
Package common-lisp.
Initargs :character

`fortran-input-array` [Class]
Arrays which serve as inputs. See WITH-ARRAY-INPUT for the semantics of the arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
[fortran-argument], page 48.

Direct subclasses
⟨undefined⟩ [fortran-input-output-array], page ⟨undefined⟩.

Direct methods
[wrap-argument], page 48.

Direct slots

`input` [Slot]
Initargs :input

`input-type` [Slot]
Initargs :input-type

`input-transpose?` [Slot]
Initargs :input-transpose?

`input-force-copy?` [Slot]
Initargs :input-force-copy?

`fortran-input-output-array` [Class]
Input/output array. See WITH-ARRAY-INPUT-OUTPUT for the semantics of the arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
- ⟨undefined⟩ [fortran-input-array], page ⟨undefined⟩.
- ⟨undefined⟩ [fortran-output-array], page ⟨undefined⟩.

Direct methods
- [fortran-argument/output-initializer-form], page 46.
- [wrap-argument], page 48.

`fortran-output-array` [Class]
Output array. See WITH-ARRAY-OUTPUT for the semantics of the arguments.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
[fortran-argument/output], page 49.

Direct subclasses
⟨undefined⟩ [fortran-input-output-array], page ⟨undefined⟩.

Direct methods
- [fortran-argument/output-initializer-form], page 46.
- [wrap-argument], page 48.

Direct slots

`output` [Slot]
Initargs :output

`output-dimensions` [Slot]
Initargs :output-dimensions

`output-type` [Slot]
Initargs :output-type

`output-transpose?` [Slot]
Initargs :output-transpose?

`fortran-work-area` [Class]
Work area.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses
- [fortran-argument/size], page 49.
- [fortran-argument/type], page ⟨undefined⟩.

Direct methods
[wrap-argument], page 48.

`ipiv-mixin` [Class]
Mixin class for objects with pivoting.

Package [lla], page 13.

Source [factorizations.lisp], page 8.

Direct subclasses
- [hermitian-factorization], page 32.
- [lu], page 32.

Direct methods
[ipiv-internal], page 46.

Direct slots

`ipiv-internal` [Slot]
pivot indices in LAPACK's representation, counting from 1

Type vector

Initargs :ipiv-internal

Readers [ipiv-internal], page 46.

Writers *This slot is read-only.*

lapack-info [Class]

Information from a LAPACK call. See the LAPACK manual for error codes.

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses

[fortran-argument], page 48.

Direct methods

- [wrap-argument], page 48.
- [wrap-argument], page 48.

Direct slots

condition [Slot]

Package common-lisp.

Initargs :condition

variable [Slot]

Package common-lisp.

Initargs :variable

lapack-work-query-area [Class]

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses

- [fortran-argument/size], page 49.
- [fortran-argument/type], page ⟨undefined⟩.

Direct methods

- [wrap-argument], page 47.
- [wrap-argument], page 48.
- [wrap-argument], page 48.

lapack-work-query-size [Class]

Package [lla], page 13.

Source [fortran-call.lisp], page 9.

Direct superclasses

[fortran-argument/size], page 49.

Direct methods

- [wrap-argument], page 47.
- [wrap-argument], page 47.

4.2.7 Types

`float-type` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`internal-type` () [Type]

Package [lla], page 13.

Source [types.lisp], page 6.

`maximum-array-size` () [Type]

Package [lla], page 13.

Source [foreign-memory.lisp], page 7.

Appendix A Indexes

A.1 Concepts

(Index is nonexistent)

A.2 Functions

A.3 Variables

A.4 Data types