

linalg

1.4.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Introduction . . . . .	1
<b>2</b>	<b>Modules Index</b>	<b>3</b>
2.1	Modules List . . . . .	3
<b>3</b>	<b>Data Type Index</b>	<b>5</b>
3.1	Data Types List . . . . .	5
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
4.1	lapack Module Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.2	linalg_c_binding Module Reference . . . . .	7
4.2.1	Detailed Description . . . . .	9
4.2.2	Function/Subroutine Documentation . . . . .	9
4.2.2.1	cholesky_factor_c(n, a, upper, err) . . . . .	9
4.2.2.2	cholesky_rank1_downdate_c(n, r, u, err) . . . . .	10
4.2.2.3	cholesky_rank1_update_c(n, r, u, err) . . . . .	10
4.2.2.4	cmtx_mult_c(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c) . . . . .	11
4.2.2.5	det_c(n, a, err) . . . . .	11
4.2.2.6	diag_cmtx_mult_left_c(m, n, k, alpha, a, b, beta, c) . . . . .	12
4.2.2.7	diag_cmtx_mult_right_c(m, n, k, alpha, a, b, beta, c) . . . . .	12
4.2.2.8	diag_mtx_mult_cmplx_left_c(m, n, k, alpha, a, b, beta, c) . . . . .	13
4.2.2.9	diag_mtx_mult_cmplx_right_c(m, n, k, alpha, a, b, beta, c) . . . . .	13
4.2.2.10	diag_mtx_mult_left_c(m, n, k, alpha, a, b, beta, c) . . . . .	13

4.2.2.11	<code>disg_mtx_mult_right_c(m, n, k, alpha, a, b, beta, c)</code>	14
4.2.2.12	<code>eigen_asymm_c(n, a, vals, vecs, err)</code>	14
4.2.2.13	<code>eigen_gen_c(n, a, b, alpha, beta, vecs, err)</code>	15
4.2.2.14	<code>eigen_symm_c(n, vecs, a, vals, err)</code>	15
4.2.2.15	<code>form_lu_c(n, lu, ipvt, u, p)</code>	16
4.2.2.16	<code>form_qr_c(m, n, r, tau, q, err)</code>	17
4.2.2.17	<code>form_qr_pivot_c(m, n, r, tau, pvt, q, p, err)</code>	17
4.2.2.18	<code>lu_factor_c(m, n, a, ipvt, err)</code>	18
4.2.2.19	<code>mtx_inverse_c(n, a, err)</code>	18
4.2.2.20	<code>mtx_mult_c(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c)</code>	19
4.2.2.21	<code>mtx_pinverse_c(m, n, a, ainv, err)</code>	19
4.2.2.22	<code>mtx_rank_c(m, n, a, err)</code>	20
4.2.2.23	<code>mult_qr_c(trans, m, n, q, tau, c, err)</code>	20
4.2.2.24	<code>mult_rz_c(trans, m, n, l, a, tau, c, err)</code>	21
4.2.2.25	<code>qr_factor_c(m, n, a, tau, err)</code>	22
4.2.2.26	<code>qr_factor_pivot_c(m, n, a, tau, jpvt, err)</code>	22
4.2.2.27	<code>qr_rank1_update_c(m, n, q, r, u, v, err)</code>	23
4.2.2.28	<code>rank1_update_c(m, n, alpha, x, y, a)</code>	23
4.2.2.29	<code>rz_factor_c(m, n, a, tau, err)</code>	24
4.2.2.30	<code>solve_cholesky_c(upper, n, nrhs, a, b)</code>	24
4.2.2.31	<code>solve_least_squares_c(m, n, nrhs, a, b, err)</code>	24
4.2.2.32	<code>solve_lu_c(n, nrhs, a, ipvt, b)</code>	25
4.2.2.33	<code>solve_qr_c(m, n, nrhs, a, tau, b, err)</code>	25
4.2.2.34	<code>solve_qr_pivot_c(m, n, nrhs, a, tau, jpvt, b, err)</code>	26
4.2.2.35	<code>solve_tri_mtx_c(upper, trans, nunit, n, nrhs, alpha, a, b)</code>	26
4.2.2.36	<code>sort_cmplx_ind_c(ascend, n, x, ind)</code>	27
4.2.2.37	<code>sort_dbl_ind_c(ascend, n, x, ind)</code>	27
4.2.2.38	<code>sort_eigen_cmplx_c(ascend, n, vals, vecs)</code>	28
4.2.2.39	<code>sort_eigen_dbl_c(ascend, n, vals, vecs)</code>	28
4.2.2.40	<code>svd_c(m, n, a, s, u, vt, err)</code>	28

4.2.2.41	<code>swap_c(n, x, y)</code>	29
4.2.2.42	<code>trace_c(m, n, x)</code>	29
4.3	<code>linalg_constants</code> Module Reference	30
4.3.1	Detailed Description	30
4.4	<code>linalg_core</code> Module Reference	30
4.4.1	Detailed Description	31
4.4.2	Function/Subroutine Documentation	31
4.4.2.1	<code>det(a, iwork, err)</code>	31
4.4.2.2	<code>diag_mtx_mult_mtx(lside, trans, alpha, a, b, beta, c, err)</code>	32
4.4.2.3	<code>diag_mtx_mult_mtx2(lside, alpha, a, b, err)</code>	33
4.4.2.4	<code>diag_mtx_mult_mtx3(lside, trans, alpha, a, b, beta, c, err)</code>	33
4.4.2.5	<code>diag_mtx_mult_mtx4(lside, trans, alpha, a, b, beta, c, err)</code>	34
4.4.2.6	<code>mtx_mult_mtx(transa, transb, alpha, a, b, beta, c, err)</code>	34
4.4.2.7	<code>mtx_mult_vec(trans, alpha, a, b, beta, c, err)</code>	35
4.4.2.8	<code>mtx_rank(a, tol, work, olwork, err)</code>	36
4.4.2.9	<code>rank1_update(alpha, x, y, a, err)</code>	36
4.4.2.10	<code>recip_mult_array(a, x)</code>	37
4.4.2.11	<code>swap(x, y, err)</code>	37
4.4.2.12	<code>trace(x)</code>	38
4.4.2.13	<code>tri_mtx_mult(upper, alpha, a, beta, b, err)</code>	38
4.5	<code>linalg_eigen</code> Module Reference	38
4.5.1	Detailed Description	39
4.5.2	Function/Subroutine Documentation	39
4.5.2.1	<code>eigen_asymm(a, vals, vecs, work, olwork, err)</code>	39
4.5.2.2	<code>eigen_gen(a, b, alpha, beta, vecs, work, olwork, err)</code>	40
4.5.2.3	<code>eigen_symm(vecs, a, vals, work, olwork, err)</code>	41
4.6	<code>linalg_factor</code> Module Reference	42
4.6.1	Detailed Description	43
4.6.2	Function/Subroutine Documentation	44
4.6.2.1	<code>cholesky_factor(a, upper, err)</code>	44

4.6.2.2	cholesky_rank1_downdate(r, u, work, err)	44
4.6.2.3	cholesky_rank1_update(r, u, work, err)	45
4.6.2.4	form_lu_all(lu, ipvt, u, p, err)	46
4.6.2.5	form_lu_only(lu, u, err)	46
4.6.2.6	form_qr_no_pivot(r, tau, q, work, olwork, err)	47
4.6.2.7	form_qr_pivot(r, tau, pvt, q, p, work, olwork, err)	48
4.6.2.8	lu_factor(a, ipvt, err)	48
4.6.2.9	mult_qr_mtx(lside, trans, a, tau, c, work, olwork, err)	49
4.6.2.10	mult_qr_vec(trans, a, tau, c, work, olwork, err)	50
4.6.2.11	mult_rz_mtx(lside, trans, l, a, tau, c, work, olwork, err)	51
4.6.2.12	mult_rz_vec(trans, l, a, tau, c, work, olwork, err)	52
4.6.2.13	qr_factor_no_pivot(a, tau, work, olwork, err)	52
4.6.2.14	qr_factor_pivot(a, tau, jpvt, work, olwork, err)	54
4.6.2.15	qr_rank1_update(q, r, u, v, work, err)	55
4.6.2.16	rz_factor(a, tau, work, olwork, err)	56
4.6.2.17	svd(a, s, u, vt, work, olwork, err)	57
4.7	linalg_solve Module Reference	59
4.7.1	Detailed Description	60
4.7.2	Function/Subroutine Documentation	60
4.7.2.1	mtx_inverse(a, iwork, work, olwork, err)	60
4.7.2.2	mtx_pinverse(a, ainvtol, work, olwork, err)	61
4.7.2.3	solve_cholesky_mtx(upper, a, b, err)	62
4.7.2.4	solve_cholesky_vec(upper, a, b, err)	63
4.7.2.5	solve_least_squares_mtx(a, b, work, olwork, err)	63
4.7.2.6	solve_least_squares_mtx_pvt(a, b, ipvt, arnk, work, olwork, err)	64
4.7.2.7	solve_least_squares_mtx_svd(a, b, arnk, s, work, olwork, err)	65
4.7.2.8	solve_least_squares_vec(a, b, work, olwork, err)	66
4.7.2.9	solve_least_squares_vec_pvt(a, b, ipvt, arnk, work, olwork, err)	66
4.7.2.10	solve_least_squares_vec_svd(a, b, arnk, s, work, olwork, err)	67
4.7.2.11	solve_lu_mtx(a, ipvt, b, err)	68

4.7.2.12	<code>solve_lu_vec(a, ipvt, b, err)</code>	69
4.7.2.13	<code>solve_qr_no_pivot_mtx(a, tau, b, work, olwork, err)</code>	69
4.7.2.14	<code>solve_qr_no_pivot_vec(a, tau, b, work, olwork, err)</code>	70
4.7.2.15	<code>solve_qr_pivot_mtx(a, tau, jpvt, b, work, olwork, err)</code>	70
4.7.2.16	<code>solve_qr_pivot_vec(a, tau, jpvt, b, work, olwork, err)</code>	71
4.7.2.17	<code>solve_tri_mtx(lside, upper, trans, nounit, alpha, a, b, err)</code>	72
4.7.2.18	<code>solve_tri_vec(upper, trans, nounit, a, x, err)</code>	73
4.8	<code>linalg_sorting</code> Module Reference	74
4.8.1	Detailed Description	74
4.8.2	Function/Subroutine Documentation	75
4.8.2.1	<code>cmplx_partition(ascend, x, marker)</code>	75
4.8.2.2	<code>cmplx_partition_ind(ascend, x, ind, marker)</code>	75
4.8.2.3	<code>dbl_partition_ind(ascend, x, ind, marker)</code>	76
4.8.2.4	<code>qsort_cmplx(ascend, x)</code>	76
4.8.2.5	<code>qsort_cmplx_ind(ascend, x, ind)</code>	76
4.8.2.6	<code>qsort_dbl_ind(ascend, x, ind)</code>	77
4.8.2.7	<code>sort_cmplx_array(x, ascend)</code>	77
4.8.2.8	<code>sort_cmplx_array_ind(x, ind, ascend, err)</code>	78
4.8.2.9	<code>sort_dbl_array(x, ascend)</code>	78
4.8.2.10	<code>sort_dbl_array_ind(x, ind, ascend, err)</code>	79
4.8.2.11	<code>sort_eigen_cmplx(vals, vecs, ascend, err)</code>	79
4.8.2.12	<code>sort_eigen_dbl(vals, vecs, ascend, err)</code>	80

<b>5</b>	<b>Data Type Documentation</b>	<b>81</b>
5.1	linalg_core::diag_mtx_mult Interface Reference	81
5.1.1	Detailed Description	81
5.1.2	Member Function/Subroutine Documentation	81
5.1.2.1	diag_mtx_mult_mtx(lside, trans, alpha, a, b, beta, c, err)	81
5.1.2.2	diag_mtx_mult_mtx2(lside, alpha, a, b, err)	82
5.1.2.3	diag_mtx_mult_mtx3(lside, trans, alpha, a, b, beta, c, err)	83
5.1.2.4	diag_mtx_mult_mtx4(lside, trans, alpha, a, b, beta, c, err)	83
5.2	lapack::DLAMCH Interface Reference	84
5.2.1	Detailed Description	84
5.3	linalg_eigen::eigen Interface Reference	84
5.3.1	Detailed Description	85
5.3.2	Member Function/Subroutine Documentation	85
5.3.2.1	eigen_asymm(a, vals, vecs, work, olwork, err)	85
5.3.2.2	eigen_gen(a, b, alpha, beta, vecs, work, olwork, err)	86
5.3.2.3	eigen_symm(vecs, a, vals, work, olwork, err)	87
5.4	linalg_factor::form_lu Interface Reference	88
5.4.1	Detailed Description	88
5.4.2	Member Function/Subroutine Documentation	89
5.4.2.1	form_lu_all(lu, ipvt, u, p, err)	89
5.4.2.2	form_lu_only(lu, u, err)	89
5.5	linalg_factor::form_qr Interface Reference	90
5.5.1	Detailed Description	90
5.5.2	Member Function/Subroutine Documentation	90
5.5.2.1	form_qr_no_pivot(r, tau, q, work, olwork, err)	90
5.5.2.2	form_qr_pivot(r, tau, pvt, q, p, work, olwork, err)	91
5.6	linalg_core::mtx_mult Interface Reference	92
5.6.1	Detailed Description	92
5.6.2	Member Function/Subroutine Documentation	92
5.6.2.1	mtx_mult_mtx(transa, transb, alpha, a, b, beta, c, err)	92



5.6.2.2	<a href="#">mtx_mult_vec(trans, alpha, a, b, beta, c, err)</a>	93
5.7	<a href="#">linalg_factor::mult_qr Interface Reference</a>	94
5.7.1	<a href="#">Detailed Description</a>	94
5.7.2	<a href="#">Member Function/Subroutine Documentation</a>	94
5.7.2.1	<a href="#">mult_qr_mtx(lside, trans, a, tau, c, work, olwork, err)</a>	94
5.7.2.2	<a href="#">mult_qr_vec(trans, a, tau, c, work, olwork, err)</a>	95
5.8	<a href="#">linalg_factor::mult_rz Interface Reference</a>	96
5.8.1	<a href="#">Detailed Description</a>	96
5.8.2	<a href="#">Member Function/Subroutine Documentation</a>	96
5.8.2.1	<a href="#">mult_rz_mtx(lside, trans, l, a, tau, c, work, olwork, err)</a>	96
5.8.2.2	<a href="#">mult_rz_vec(trans, l, a, tau, c, work, olwork, err)</a>	97
5.9	<a href="#">linalg_factor::qr_factor Interface Reference</a>	98
5.9.1	<a href="#">Detailed Description</a>	98
5.9.2	<a href="#">Member Function/Subroutine Documentation</a>	98
5.9.2.1	<a href="#">qr_factor_no_pivot(a, tau, work, olwork, err)</a>	98
5.9.2.2	<a href="#">qr_factor_pivot(a, tau, jpvt, work, olwork, err)</a>	100
5.10	<a href="#">linalg_solve::solve_cholesky Interface Reference</a>	101
5.10.1	<a href="#">Detailed Description</a>	101
5.10.2	<a href="#">Member Function/Subroutine Documentation</a>	102
5.10.2.1	<a href="#">solve_cholesky_mtx(upper, a, b, err)</a>	102
5.10.2.2	<a href="#">solve_cholesky_vec(upper, a, b, err)</a>	102
5.11	<a href="#">linalg_solve::solve_least_squares Interface Reference</a>	103
5.11.1	<a href="#">Detailed Description</a>	103
5.11.2	<a href="#">Member Function/Subroutine Documentation</a>	103
5.11.2.1	<a href="#">solve_least_squares_mtx(a, b, work, olwork, err)</a>	103
5.11.2.2	<a href="#">solve_least_squares_vec(a, b, work, olwork, err)</a>	104
5.12	<a href="#">linalg_solve::solve_least_squares_full Interface Reference</a>	104
5.12.1	<a href="#">Detailed Description</a>	105
5.12.2	<a href="#">Member Function/Subroutine Documentation</a>	105
5.12.2.1	<a href="#">solve_least_squares_mtx_pvt(a, b, ipvt, arnk, work, olwork, err)</a>	105

5.12.2.2	<code>solve_least_squares_vec_pvt(a, b, ipvt, arnk, work, olwork, err)</code>	106
5.13	<code>linalg_solve::solve_least_squares_svd</code> Interface Reference	107
5.13.1	Detailed Description	107
5.13.2	Member Function/Subroutine Documentation	107
5.13.2.1	<code>solve_least_squares_mtx_svd(a, b, arnk, s, work, olwork, err)</code>	107
5.13.2.2	<code>solve_least_squares_vec_svd(a, b, arnk, s, work, olwork, err)</code>	108
5.14	<code>linalg_solve::solve_lu</code> Interface Reference	109
5.14.1	Detailed Description	109
5.14.2	Member Function/Subroutine Documentation	109
5.14.2.1	<code>solve_lu_mtx(a, ipvt, b, err)</code>	109
5.14.2.2	<code>solve_lu_vec(a, ipvt, b, err)</code>	110
5.15	<code>linalg_solve::solve_qr</code> Interface Reference	110
5.15.1	Detailed Description	111
5.15.2	Member Function/Subroutine Documentation	111
5.15.2.1	<code>solve_qr_no_pivot_mtx(a, tau, b, work, olwork, err)</code>	111
5.15.2.2	<code>solve_qr_no_pivot_vec(a, tau, b, work, olwork, err)</code>	112
5.15.2.3	<code>solve_qr_pivot_mtx(a, tau, jpvt, b, work, olwork, err)</code>	112
5.15.2.4	<code>solve_qr_pivot_vec(a, tau, jpvt, b, work, olwork, err)</code>	113
5.16	<code>linalg_solve::solve_triangular_system</code> Interface Reference	114
5.16.1	Detailed Description	114
5.16.2	Member Function/Subroutine Documentation	114
5.16.2.1	<code>solve_tri_mtx(lside, upper, trans, nounit, alpha, a, b, err)</code>	114
5.16.2.2	<code>solve_tri_vec(upper, trans, nounit, a, x, err)</code>	115
5.17	<code>linalg_sorting::sort</code> Interface Reference	116
5.17.1	Detailed Description	117
5.17.2	Member Function/Subroutine Documentation	117
5.17.2.1	<code>sort_cmplx_array(x, ascend)</code>	117
5.17.2.2	<code>sort_cmplx_array_ind(x, ind, ascend, err)</code>	118
5.17.2.3	<code>sort_dbl_array(x, ascend)</code>	118
5.17.2.4	<code>sort_dbl_array_ind(x, ind, ascend, err)</code>	119
5.17.2.5	<code>sort_eigen_cmplx(vals, vecs, ascend, err)</code>	119
5.17.2.6	<code>sort_eigen_dbl(vals, vecs, ascend, err)</code>	120

## Chapter 1

# Main Page

### 1.1 Introduction

LINALG is a linear algebra library that provides a user-friendly interface to several BLAS and LAPACK routines.

#### Author

Jason Christopherson

#### Version

1.4.0



## Chapter 2

# Modules Index

### 2.1 Modules List

Here is a list of all documented modules with brief descriptions:

<a href="#">lapack</a>		
<a href="#">lapack</a>	. . . . .	7
<a href="#">linalg_c_binding</a>		
<a href="#">linalg_c_binding</a>	. . . . .	7
<a href="#">linalg_constants</a>		
<a href="#">linalg_constants</a>	. . . . .	30
<a href="#">linalg_core</a>		
<a href="#">linalg_core</a>	. . . . .	30
<a href="#">linalg_eigen</a>		
<a href="#">linalg_eigen</a>	. . . . .	38
<a href="#">linalg_factor</a>		
<a href="#">linalg_factor</a>	. . . . .	42
<a href="#">linalg_solve</a>		
<a href="#">linalg_solve</a>	. . . . .	59
<a href="#">linalg_sorting</a>		
<a href="#">linalg_sorting</a>	. . . . .	74



## Chapter 3

# Data Type Index

### 3.1 Data Types List

Here are the data types with brief descriptions:

<a href="#">linalg_core::diag_mtx_mult</a>	81
Multiplies a diagonal matrix with another matrix or array . . . . .	
<a href="#">lapack::DLAMCH</a> . . . . .	84
<a href="#">linalg_eigen::eigen</a>	
Computes the eigenvalues, and optionally the eigenvectors, of a matrix . . . . .	84
<a href="#">linalg_factor::form_lu</a>	
Extracts the L and U matrices from the condensed [L\U] storage format used by the <a href="#">lu_factor</a> . . . . .	88
<a href="#">linalg_factor::form_qr</a>	
Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm . . . . .	90
<a href="#">linalg_core::mtx_mult</a>	
Performs the matrix operation: $C = \alpha * op(A) * op(B) + \beta * C$ . . . . .	92
<a href="#">linalg_factor::mult_qr</a>	
Multiplies a general matrix by the orthogonal matrix Q from a QR factorization . . . . .	94
<a href="#">linalg_factor::mult_rz</a>	
Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization . . . . .	96
<a href="#">linalg_factor::qr_factor</a>	
Computes the QR factorization of an M-by-N matrix . . . . .	98
<a href="#">linalg_solve::solve_cholesky</a>	
Solves a system of Cholesky factored equations . . . . .	101
<a href="#">linalg_solve::solve_least_squares</a>	
Solves the overdetermined or underdetermined system ( $A*X = B$ ) of M equations of N unknowns . . . . .	103
<a href="#">linalg_solve::solve_least_squares_full</a>	
Solves the overdetermined or underdetermined system ( $A*X = B$ ) of M equations of N unknowns, but uses a full orthogonal factorization of the system . . . . .	104
<a href="#">linalg_solve::solve_least_squares_svd</a>	
Solves the overdetermined or underdetermined system ( $A*X = B$ ) of M equations of N unknowns using a singular value decomposition of matrix A . . . . .	107
<a href="#">linalg_solve::solve_lu</a>	
Solves a system of LU-factored equations . . . . .	109
<a href="#">linalg_solve::solve_qr</a>	
Solves a system of M QR-factored equations of N unknowns . . . . .	110
<a href="#">linalg_solve::solve_triangular_system</a>	
Solves a triangular system of equations . . . . .	114
<a href="#">linalg_sorting::sort</a>	
Sorts an array . . . . .	116





## Chapter 4

# Module Documentation

### 4.1 lapack Module Reference

#### lapack

##### Data Types

- interface [DLAMCH](#)

#### 4.1.1 Detailed Description

#### lapack

##### Purpose

Provides interfaces to various LAPACK routines.

### 4.2 linalg\_c\_binding Module Reference

#### [linalg\\_c\\_binding](#)

##### Functions/Subroutines

- subroutine [mtx\\_mult\\_c](#) (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c)  
*Performs the matrix operation:  $C = \alpha * op(A) * op(B) + \beta * C$ .*
- subroutine [cmtx\\_mult\\_c](#) (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c)  
*Performs the matrix operation:  $C = \alpha * op(A) * op(B) + \beta * C$ .*
- subroutine [diag\\_mtx\\_mult\\_left\\_c](#) (m, n, k, alpha, a, b, beta, c)  
*Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $A$  is a diagonal matrix.*
- subroutine [diag\\_mtx\\_mult\\_right\\_c](#) (m, n, k, alpha, a, b, beta, c)  
*Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $B$  is a diagonal matrix.*
- subroutine [diag\\_mtx\\_mult\\_cmplx\\_left\\_c](#) (m, n, k, alpha, a, b, beta, c)

- Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $A$  is a diagonal matrix.*

  - subroutine [diag\\_mtx\\_mult\\_cmplx\\_right\\_c](#) (m, n, k, alpha, a, b, beta, c)
- Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $B$  is a diagonal matrix.*

  - subroutine [diag\\_cmtx\\_mult\\_left\\_c](#) (m, n, k, alpha, a, b, beta, c)
- Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $A$  is a diagonal matrix.*

  - subroutine [diag\\_cmtx\\_mult\\_right\\_c](#) (m, n, k, alpha, a, b, beta, c)
- Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where  $B$  is a diagonal matrix.*

  - subroutine [rank1\\_update\\_c](#) (m, n, alpha, x, y, a)
- Performs the rank-1 update to matrix  $A$  such that:  $A = \alpha * X * Y^{**}T + A$ , where  $A$  is an  $M$ -by- $N$  matrix,  $\alpha$  is a scalar,  $X$  is an  $M$ -element array, and  $N$  is an  $N$ -element array.*

  - pure real(dp) function [trace\\_c](#) (m, n, x)
- Computes the trace of a matrix (the sum of the main diagonal elements).*

  - integer(i32) function [mtx\\_rank\\_c](#) (m, n, a, err)
- Computes the rank of a matrix.*

  - real(dp) function [det\\_c](#) (n, a, err)
- Computes the determinant of a square matrix.*

  - subroutine [swap\\_c](#) (n, x, y)
- Swaps the contents of two arrays.*

  - subroutine [tri\\_mtx\\_mult\\_c](#) (upper, n, alpha, a, beta, b, err)
  - subroutine [lu\\_factor\\_c](#) (m, n, a, ipvt, err)
- Computes the LU factorization of an  $M$ -by- $N$  matrix.*

  - subroutine [form\\_lu\\_c](#) (n, lu, ipvt, u, p)
- Extracts the  $L$ ,  $U$ , and  $P$  matrices from the output of the [lu\\_factor](#) routine.*

  - subroutine [qr\\_factor\\_c](#) (m, n, a, tau, err)
- Computes the QR factorization of an  $M$ -by- $N$  matrix without pivoting.*

  - subroutine [qr\\_factor\\_pivot\\_c](#) (m, n, a, tau, jpvt, err)
- Computes the QR factorization of an  $M$ -by- $N$  matrix with column pivoting such that  $A * P = Q * R$ .*

  - subroutine [form\\_qr\\_c](#) (m, n, r, tau, q, err)
- Forms the full  $M$ -by- $M$  orthogonal matrix  $Q$  from the elementary reflectors returned by the base QR factorization algorithm.*

  - subroutine [form\\_qr\\_pivot\\_c](#) (m, n, r, tau, pvt, q, p, err)
- Forms the full  $M$ -by- $M$  orthogonal matrix  $Q$  from the elementary reflectors returned by the base QR factorization algorithm.*

  - subroutine [mult\\_qr\\_c](#) (trans, m, n, q, tau, c, err)
- Multiplies a general matrix by the orthogonal matrix  $Q$  from a QR factorization such that:  $C = op(Q) * C$ .*

  - subroutine [qr\\_rank1\\_update\\_c](#) (m, n, q, r, u, v, err)
- Computes the rank 1 update to an  $M$ -by- $N$  QR factored matrix  $A$  ( $M \geq N$ ) where  $A = Q * R$ , and  $A1 = A + U * V^{**}T$  such that  $A1 = Q1 * R1$ .*

  - subroutine [cholesky\\_factor\\_c](#) (n, a, upper, err)
- Computes the Cholesky factorization of a symmetric, positive definite matrix.*

  - subroutine [cholesky\\_rank1\\_update\\_c](#) (n, r, u, err)
- Computes the rank 1 update to a Cholesky factored matrix (upper triangular).*

  - subroutine [cholesky\\_rank1\\_downdate\\_c](#) (n, r, u, err)
- Computes the rank 1 downdate to a Cholesky factored matrix (upper triangular).*

  - subroutine [rz\\_factor\\_c](#) (m, n, a, tau, err)
- Factors an upper trapezoidal matrix by means of orthogonal transformations such that  $A = R * Z = (R \ 0) * Z$ .  $Z$  is an orthogonal matrix of dimension  $N$ -by- $N$ , and  $R$  is an  $M$ -by- $M$  upper triangular matrix.*

  - subroutine [mult\\_rz\\_c](#) (trans, m, n, l, a, tau, c, err)
- Multiplies a general matrix by the orthogonal matrix  $Z$  from an RZ factorization such that:  $C = op(Z) * C$ .*

  - subroutine [svd\\_c](#) (m, n, a, s, u, vt, err)
- Computes the singular value decomposition of a matrix  $A$ . The SVD is defined as:  $A = U * S * V^{**}T$ , where  $U$  is an  $M$ -by- $M$  orthogonal matrix,  $S$  is an  $M$ -by- $N$  diagonal matrix, and  $V$  is an  $N$ -by- $N$  orthogonal matrix.*

- subroutine [solve\\_tri\\_mtx\\_c](#) (upper, trans, nounit, n, nrhs, alpha, a, b)  
*Solves one of the matrix equations:  $op(A) * X = alpha * B$ , where  $A$  is a triangular matrix.*
- subroutine [solve\\_lu\\_c](#) (n, nrhs, a, ipvt, b)  
*Solves a system of LU-factored equations.*
- subroutine [solve\\_qr\\_c](#) (m, n, nrhs, a, tau, b, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where  $M \geq N$ .*
- subroutine [solve\\_qr\\_pivot\\_c](#) (m, n, nrhs, a, tau, jpvt, b, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where the QR factorization made use of column pivoting.*
- subroutine [solve\\_cholesky\\_c](#) (upper, n, nrhs, a, b)  
*Solves a system of Cholesky factored equations.*
- subroutine [mtx\\_inverse\\_c](#) (n, a, err)  
*Computes the inverse of a square matrix.*
- subroutine [mtx\\_pinverse\\_c](#) (m, n, a, ainvt, err)  
*Computes the Moore-Penrose pseudo-inverse of a  $M$ -by- $N$  matrix using the singular value decomposition of the matrix.*
- subroutine [solve\\_least\\_squares\\_c](#) (m, n, nrhs, a, b, err)  
*Solves the overdetermined or underdetermined system ( $A * X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.*
- subroutine [eigen\\_symm\\_c](#) (n, vecs, a, vals, err)  
*Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.*
- subroutine [eigen\\_asymm\\_c](#) (n, a, vals, vecs, err)  
*Computes the eigenvalues, and the right eigenvectors of a square matrix.*
- subroutine [eigen\\_gen\\_c](#) (n, a, b, alpha, beta, vecs, err)  
*Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A * X = lambda * B * X$ .*
- subroutine [sort\\_dbl\\_ind\\_c](#) (ascend, n, x, ind)  
*Sorts an array of double-precision values.*
- subroutine [sort\\_cmplx\\_ind\\_c](#) (ascend, n, x, ind)  
*Sorts an array of complex values according to their real components.*
- subroutine [sort\\_eigen\\_cmplx\\_c](#) (ascend, n, vals, vecs)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*
- subroutine [sort\\_eigen\\_dbl\\_c](#) (ascend, n, vals, vecs)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*

### 4.2.1 Detailed Description

#### [linalg\\_c\\_binding](#)

##### Purpose

Provides a C friendly interface to the LINALG library.

### 4.2.2 Function/Subroutine Documentation

- 4.2.2.1 subroutine [linalg\\_c\\_binding::cholesky\\_factor\\_c](#) ( integer(i32), intent(in), value *n*, real(dp), dimension(n,n), intent(inout) *a*, logical(c\_bool), intent(in), value *upper*, type(errorhandler), intent(inout) *err* )

Computes the Cholesky factorization of a symmetric, positive definite matrix.

## Parameters

<i>in</i>	<i>n</i>	The dimension of the matrix.
<i>in, out</i>	<i>a</i>	On input, the N-by-N matrix to factor. On output, the factored matrix is returned in either the upper or lower triangular portion of the matrix, dependent upon the value of <i>upper</i> .
<i>in</i>	<i>upper</i>	An optional input that, if specified, provides control over whether the factorization is computed as $A = U^{**T} * U$ (set to true), or as $A = L * L^{**T}$ (set to false). The default value is true such that $A = U^{**T} * U$ .
<i>in, out</i>	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_MATRIX_FORMAT_ERROR: Occurs if <i>a</i> is not positive definite.</li> </ul>

Definition at line 814 of file linalg\_c\_binding.f90.

**4.2.2.2** subroutine linalg\_c\_binding::cholesky\_rank1\_downdate\_c ( integer(i32), intent(in), value *n*, real(dp), dimension(n,n), intent(inout) *r*, real(dp), dimension(n), intent(inout) *u*, type(errorhandler), intent(inout) *err* )

Computes the rank 1 downdate to a Cholesky factored matrix (upper triangular).

## Parameters

<i>in</i>	<i>n</i>	The dimension of the matrix.
<i>in, out</i>	<i>r</i>	On input, the N-by-N upper triangular matrix R. On output, the updated matrix R1.
<i>in, out</i>	<i>u</i>	On input, the N-element update vector U. On output, the rotation sines used to transform R to R1.
<i>in, out</i>	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_MATRIX_FORMAT_ERROR: Occurs if the downdated matrix is not positive definite.</li> <li>LA_SINGULAR_MATRIX_ERROR: Occurs if <i>r</i> is singular.</li> </ul>

Definition at line 888 of file linalg\_c\_binding.f90.

**4.2.2.3** subroutine linalg\_c\_binding::cholesky\_rank1\_update\_c ( integer(i32), intent(in), value *n*, real(dp), dimension(n,n), intent(inout) *r*, real(dp), dimension(n), intent(inout) *u*, type(errorhandler), intent(inout) *err* )

Computes the rank 1 update to a Cholesky factored matrix (upper triangular).

## Parameters

<i>in</i>	<i>n</i>	The dimension of the matrix.
<i>in, out</i>	<i>r</i>	On input, the N-by-N upper triangular matrix R. On output, the updated matrix R1.

## Parameters

in, out	<i>u</i>	On input, the N-element update vector U. On output, the rotation sines used to transform R to R1.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 849 of file linalg\_c\_binding.f90.

**4.2.2.4** subroutine linalg\_c\_binding::cmtx\_mult\_c ( logical(c\_bool), intent(in), value *transa*, logical(c\_bool), intent(in), value *transb*, integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, complex(dp), dimension(lda,\*), intent(in) *a*, integer(i32), intent(in), value *lda*, complex(dp), dimension(ldb,\*), intent(in) *b*, integer(i32), intent(in), value *ldb*, real(dp), intent(in), value *beta*, complex(dp), dimension(m,n), intent(inout) *c* )

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

## Parameters

in	<i>transa</i>	Set to true if $\text{op}(A) == A^*T$ ; else, set to false if $\text{op}(A) == A$ .
in	<i>transb</i>	Set to true if $\text{op}(B) == B^*T$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>m</i>	The number of rows in matrix C, and the number of rows in matrix $\text{op}(A)$ .
in	<i>n</i>	The number of columns in matrix C, and the number of columns in matrix $\text{op}(B)$ .
in	<i>k</i>	The number of columns in matrix $\text{op}(A)$ , and the number of rows in the matrix $\text{op}(B)$ .
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	The M-by-K matrix A.
in	<i>lda</i>	The leading dimension of matrix A. If <i>transa</i> is true, this value must be at least MAX(1, K); else, if <i>transa</i> is false, this value must be at least MAX(1, M).
in	<i>b</i>	The K-by-N matrix B.
in	<i>ldb</i>	The leading dimension of matrix B. If <i>transb</i> is true, this value must be at least MAX(1, N); else, if <i>transb</i> is false, this value must be at least MAX(1, K).
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 96 of file linalg\_c\_binding.f90.

**4.2.2.5** real(dp) function linalg\_c\_binding::det\_c ( integer(i32), intent(in), value *n*, real(dp), dimension(n,n), intent(inout) *a*, type(errorhandler), intent(inout) *err* )

Computes the determinant of a square matrix.

## Parameters

in	<i>n</i>	The dimension of the matrix.
in, out	<i>a</i>	On input, the N-by-N matrix on which to operate. On output the contents are overwritten by the LU factorization of the original matrix.

## Parameters

<i>in, out</i>	<i>err</i>	<p>The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if the input matrix is not square.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
----------------	------------	---

Definition at line 374 of file `linalg_c_binding.f90`.

**4.2.2.6** subroutine `linalg_c_binding::diag_cmtx_mult_left_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, complex(dp), dimension(min(*m*, *k*)), intent(in) *a*, complex(dp), dimension(*k*, *n*), intent(in) *b*, real(dp), intent(in), value *beta*, complex(dp), dimension(*m*, *n*), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where *A* is a diagonal matrix.

## Parameters

<i>in</i>	<i>m</i>	The number of rows in matrix <i>C</i> .
<i>in</i>	<i>n</i>	The number of columns in matrix <i>C</i> .
<i>in</i>	<i>k</i>	The number of rows in matrix <i>B</i> .
<i>in</i>	<i>alpha</i>	The scalar multiplier to matrix <i>A</i> .
<i>in</i>	<i>a</i>	A MIN( <i>M</i> , <i>K</i> )-element array containing the diagonal elements of matrix <i>A</i> .
<i>in</i>	<i>b</i>	The <i>K</i> -by- <i>N</i> matrix <i>B</i> .
<i>in</i>	<i>beta</i>	The scalar multiplier to matrix <i>C</i> .
<i>in, out</i>	<i>c</i>	The <i>M</i> -by- <i>N</i> matrix <i>C</i> .

Definition at line 240 of file `linalg_c_binding.f90`.

**4.2.2.7** subroutine `linalg_c_binding::diag_cmtx_mult_right_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, complex(dp), dimension(*m*, *k*), intent(in) *a*, complex(dp), dimension(min(*k*, *n*)), intent(in) *b*, real(dp), intent(in), value *beta*, complex(dp), dimension(*m*, *n*), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where *B* is a diagonal matrix.

## Parameters

<i>in</i>	<i>m</i>	The number of rows in matrix <i>C</i> .
<i>in</i>	<i>n</i>	The number of columns in matrix <i>C</i> .
<i>in</i>	<i>k</i>	The number of columns in matrix <i>A</i> .
<i>in</i>	<i>alpha</i>	The scalar multiplier to matrix <i>A</i> .
<i>in</i>	<i>a</i>	The <i>M</i> -by- <i>K</i> matrix <i>A</i> .
<i>in</i>	<i>b</i>	A MIN( <i>K</i> , <i>N</i> )-element array containing the diagonal elements of matrix <i>B</i> .
<i>in</i>	<i>beta</i>	The scalar multiplier to matrix <i>C</i> .
<i>in, out</i>	<i>c</i>	The <i>M</i> -by- <i>N</i> matrix <i>C</i> .

Definition at line 265 of file linalg\_c\_binding.f90.

**4.2.2.8** subroutine linalg\_c\_binding::diag\_mtx\_mult\_cmplx\_left\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, complex(dp), dimension(min(*m*, *k*)), intent(in) *a*, real(dp), dimension(*k*, *n*), intent(in) *b*, real(dp), intent(in), value *beta*, complex(dp), dimension(*m*, *n*), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where A is a diagonal matrix.

#### Parameters

in	<i>m</i>	The number of rows in matrix C.
in	<i>n</i>	The number of columns in matrix C.
in	<i>k</i>	The number of rows in matrix B.
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	A MIN(M,K)-element array containing the diagonal elements of matrix A.
in	<i>b</i>	The K-by-N matrix B.
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 188 of file linalg\_c\_binding.f90.

**4.2.2.9** subroutine linalg\_c\_binding::diag\_mtx\_mult\_cmplx\_right\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, real(dp), dimension(*m*, *k*), intent(in) *a*, complex(dp), dimension(min(*k*, *n*)), intent(in) *b*, real(dp), intent(in), value *beta*, complex(dp), dimension(*m*, *n*), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where B is a diagonal matrix.

#### Parameters

in	<i>m</i>	The number of rows in matrix C.
in	<i>n</i>	The number of columns in matrix C.
in	<i>k</i>	The number of columns in matrix A.
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	The M-by-K matrix A.
in	<i>b</i>	A MIN(K,N)-element array containing the diagonal elements of matrix B.
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 214 of file linalg\_c\_binding.f90.

**4.2.2.10** subroutine linalg\_c\_binding::diag\_mtx\_mult\_left\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, real(dp), dimension(min(*m*, *k*)), intent(in) *a*, real(dp), dimension(*k*, *n*), intent(in) *b*, real(dp), intent(in), value *beta*, real(dp), dimension(*m*, *n*), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where A is a diagonal matrix.

## Parameters

in	<i>m</i>	The number of rows in matrix C.
in	<i>n</i>	The number of columns in matrix C.
in	<i>k</i>	The number of rows in matrix B.
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	A MIN(M,K)-element array containing the diagonal elements of matrix A.
in	<i>b</i>	The K-by-N matrix B.
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 138 of file linalg\_c\_binding.f90.

**4.2.2.11** subroutine linalg\_c\_binding::disg\_mtx\_mult\_right\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, real(dp), dimension(m, k), intent(in) *a*, real(dp), dimension(min(k, n)), intent(in) *b*, real(dp), intent(in), value *beta*, real(dp), dimension(m, n), intent(inout) *c* )

Computes the matrix operation:  $C = \alpha * A * B + \beta * C$ , where B is a diagonal matrix.

## Parameters

in	<i>m</i>	The number of rows in matrix C.
in	<i>n</i>	The number of columns in matrix C.
in	<i>k</i>	The number of columns in matrix A.
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	The M-by-K matrix A.
in	<i>b</i>	A MIN(K,N)-element array containing the diagonal elements of matrix B.
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 163 of file linalg\_c\_binding.f90.

**4.2.2.12** subroutine linalg\_c\_binding::eigen\_asymm\_c ( integer(i32), intent(in), value *n*, real(dp), dimension(n,n), intent(inout) *a*, complex(dp), dimension(n), intent(out) *vals*, complex(dp), dimension(n,n), intent(out) *vecs*, type(errorhandler), intent(inout) *err* )

Computes the eigenvalues, and the right eigenvectors of a square matrix.

## Parameters

in	<i>n</i>	The dimension of the matrix.
in, out	<i>a</i>	On input, the N-by-N matrix on which to operate. On output, the contents of this matrix are overwritten.
out	<i>vals</i>	An N-element array containing the eigenvalues of the matrix on output. The eigenvalues are not sorted.
out	<i>vecs</i>	An N-by-N matrix containing the right eigenvectors (one per column) on output.



## Parameters

<i>in, out</i>	<i>err</i>	<p>The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>
----------------	------------	--

Definition at line 1372 of file `linalg_c_binding.f90`.

**4.2.2.13** subroutine `linalg_c_binding::eigen_gen_c` ( integer(*i32*), intent(in), value *n*, real(*dp*), dimension(*n*, *n*), intent(inout) *a*, real(*dp*), dimension(*n*, *n*), intent(inout) *b*, complex(*dp*), dimension(*n*), intent(out) *alpha*, real(*dp*), dimension(*n*), intent(out) *beta*, complex(*dp*), dimension(*n*,*n*), intent(out) *vecs*, type(errorhandler), intent(inout) *err* )

Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A \cdot X = \lambda B \cdot X$ .

## Parameters

<i>in</i>	<i>n</i>	The dimension of the matrix.
<i>in, out</i>	<i>a</i>	On input, the N-by-N matrix A. On output, the contents of this matrix are overwritten.
<i>in, out</i>	<i>b</i>	On input, the N-by-N matrix B. On output, the contents of this matrix are overwritten.
<i>out</i>	<i>alpha</i>	An N-element array that, on output, contains the numerator of the eigenvalue ration ALPHA / BETA. Computation of this ratio isn't necessarily as trivial as it seems as it is entirely possible, and likely, that ALPHA / BETA can overflow or underflow. With that said, the values in ALPHA will always be less than and usually comparable with the NORM(A).
<i>out</i>	<i>beta</i>	An N-element array that, on output, contains the denominator used to determine the eigenvalues as ALPHA / BETA. The values in this array will always be less than and usually comparable with the NORM(B).
<i>out</i>	<i>vecs</i>	An N-by-N matrix containing the right eigenvectors (one per column) on output.
<i>in, out</i>	<i>err</i>	<p>The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>

Definition at line 1421 of file `linalg_c_binding.f90`.

**4.2.2.14** subroutine `linalg_c_binding::eigen_symm_c` ( integer(*i32*), intent(in), value *n*, logical(*c\_bool*), intent(in), value *vecs*, real(*dp*), dimension(*n*,*n*), intent(inout) *a*, real(*dp*), dimension(*n*), intent(out) *vals*, type(errorhandler), intent(inout) *err* )

Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.

## Parameters

in	<i>n</i>	The dimension of the matrix.
in	<i>vecs</i>	Set to true to compute the eigenvectors as well as the eigenvalues; else, set to false to just compute the eigenvalues.
in, out	<i>a</i>	On input, the N-by-N symmetric matrix on which to operate. On output, and if <i>vecs</i> is set to true, the matrix will contain the eigenvectors (one per column) corresponding to each eigenvalue in <i>vals</i> . If <i>vecs</i> is set to false, the lower triangular portion of the matrix is overwritten.
out	<i>vals</i>	An N-element array that will contain the eigenvalues sorted into ascending order.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>

Definition at line 1333 of file `linalg_c_binding.f90`.

```
4.2.2.15 subroutine linalg_c_binding::form_lu_c ( integer(i32), intent(in), value n, real(dp), dimension(n,n), intent(inout) lu,
integer(i32), dimension(n), intent(in) ipvt, real(dp), dimension(n,n), intent(out) u, real(dp), dimension(n,n), intent(out) p
)
```

Extracts the L, U, and P matrices from the output of the `lu_factor` routine.

## Parameters

in	<i>n</i>	The dimension of the original matrix.
in, out	<i>lu</i>	On input, the N-by-N matrix as output by <code>lu_factor</code> . On output, the N-by-N lower triangular matrix L.
in	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
out	<i>u</i>	An N-by-N matrix where the U matrix will be written.
out	<i>p</i>	An N-by-N matrix where the row permutation matrix will be written.

## Remarks

This routine allows extraction of the actual "L", "U", and "P" matrices of the decomposition. To use these matrices to solve the system  $A \cdot X = B$ , the following approach is used.

1. First, solve the linear system:  $L \cdot Y = P \cdot B$  for  $Y$ .
2. Second, solve the linear system:  $U \cdot X = Y$  for  $X$ .

Notice, as both L and U are triangular in structure, the above equations can be solved by forward and backward substitution.

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 521 of file `linalg_c_binding.f90`.

**4.2.2.16** subroutine `linalg_c_binding::form_qr_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *r*, real(dp), dimension(min(m,n)), intent(in) *tau*, real(dp), dimension(m,m), intent(out) *q*, type(errorhandler), intent(inout) *err* )

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

#### Parameters

in	<i>m</i>	The number of rows in the original matrix.
in	<i>n</i>	The number of columns in the original matrix.
in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that $M > N$ , Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 ( $Q = [Q1, Q2]$ ) as the factorization can be written as $Q * R = [Q1, Q2] * [R1; 0]$ .
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the scalar factor array is not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 643 of file `linalg_c_binding.f90`.

**4.2.2.17** subroutine `linalg_c_binding::form_qr_pivot_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *r*, real(dp), dimension(min(m,n)), intent(in) *tau*, integer(i32), dimension(n), intent(in) *pvt*, real(dp), dimension(m,m), intent(out) *q*, real(dp), dimension(n,n), intent(out) *p*, type(errorhandler), intent(inout) *err* )

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

#### Parameters

in	<i>m</i>	The number of rows in the original matrix.
in	<i>n</i>	The number of columns in the original matrix.
in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
in	<i>pvt</i>	An N-element column pivot array as returned by the QR factorization.
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that $M > N$ , Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 ( $Q = [Q1, Q2]$ ) as the factorization can be written as $Q * R = [Q1, Q2] * [R1; 0]$ .

## Parameters

out	<i>p</i>	An N-by-N matrix where the pivot matrix will be written.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the scalar factor array is not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 692 of file linalg\_c\_binding.f90.

```
4.2.2.18 subroutine linalg_c_binding::lu_factor_c ( integer(i32), intent(in), value m, integer(i32), intent(in), value n, real(dp),
dimension(m,n), intent(inout) a, integer(i32), dimension(min(m,n)), intent(out) ipvt, type(errorhandler), intent(inout) err
)
```

Computes the LU factorization of an M-by-N matrix.

## Parameters

in	<i>m</i>	The number of rows in the matrix.
in	<i>n</i>	The number of columns in the matrix.
in, out	<i>a</i>	On input, the M-by-N matrix on which to operate. On output, the LU factored matrix in the form [L\U] where the unit diagonal elements of L are not stored.
out	<i>ipvt</i>	An MIN(M, N)-element array used to track row-pivot operations. The array stored pivot information such that row <i>l</i> is interchanged with row IPVT( <i>l</i> ).
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the pivot array is not sized appropriately.</li> <li>LA_SINGULAR_MATRIX_ERROR: Occurs as a warning if <i>a</i> is found to be singular.</li> </ul>

Definition at line 475 of file linalg\_c\_binding.f90.

```
4.2.2.19 subroutine linalg_c_binding::mtx_inverse_c ( integer(i32), intent(in), value n, real(dp), dimension(n,n), intent(inout) a,
type(errorhandler), intent(inout) err )
```

Computes the inverse of a square matrix.

## Parameters

in	<i>n</i>	The dimension of the matrix.
in, out	<i>a</i>	On input, the N-by-N matrix to invert. On output, the inverted matrix.

## Parameters

in, out	err	<p>The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_SINGULAR_MATRIX_ERROR: Occurs if the input matrix is singular.</li> </ul>
---------	-----	--

Definition at line 1214 of file linalg\_c\_binding.f90.

**4.2.2.20** subroutine linalg\_c\_binding::mtx\_mult\_c ( logical(c\_bool), intent(in), value *transa*, logical(c\_bool), intent(in), value *transb*, integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *k*, real(dp), intent(in), value *alpha*, real(dp), dimension(lda,\*), intent(in) *a*, integer(i32), intent(in), value *lda*, real(dp), dimension(ldb,\*), intent(in) *b*, integer(i32), intent(in), value *ldb*, real(dp), intent(in), value *beta*, real(dp), dimension(m,n), intent(inout) *c* )

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

## Parameters

in	<i>transa</i>	Set to true if $\text{op}(A) == A^{**T}$ ; else, set to false if $\text{op}(A) == A$ .
in	<i>transb</i>	Set to true if $\text{op}(B) == B^{**T}$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>m</i>	The number of rows in matrix C, and the number of rows in matrix $\text{op}(A)$ .
in	<i>n</i>	The number of columns in matrix C, and the number of columns in matrix $\text{op}(B)$ .
in	<i>k</i>	The number of columns in matrix $\text{op}(A)$ , and the number of rows in the matrix $\text{op}(B)$ .
in	<i>alpha</i>	The scalar multiplier to matrix A.
in	<i>a</i>	The M-by-K matrix A.
in	<i>lda</i>	The leading dimension of matrix A. If <i>transa</i> is true, this value must be at least MAX(1, K); else, if <i>transa</i> is false, this value must be at least MAX(1, M).
in	<i>b</i>	The K-by-N matrix B.
in	<i>ldb</i>	The leading dimension of matrix B. If <i>transb</i> is true, this value must be at least MAX(1, N); else, if <i>transb</i> is false, this value must be at least MAX(1, K).
in	<i>beta</i>	The scalar multiplier to matrix C.
in, out	<i>c</i>	The M-by-N matrix C.

Definition at line 47 of file linalg\_c\_binding.f90.

**4.2.2.21** subroutine linalg\_c\_binding::mtx\_pinverse\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(n,m), intent(out) *ainv*, type(errorhandler), intent(inout) *err* )

Computes the Moore-Penrose pseudo-inverse of a M-by-N matrix using the singular value decomposition of the matrix.

## Parameters

in	<i>m</i>	The number of rows in the matrix to invert.
in	<i>n</i>	The number of columns in the matrix to invert.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to invert. The matrix is overwritten on output.
out	<i>ainv</i>	The N-by-M matrix where the pseudo-inverse of <i>a</i> will be written.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_CONVERGENCE_ERROR: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

Definition at line 1251 of file `linalg_c_binding.f90`.

**4.2.2.22** `integer(i32) function linalg_c_binding::mtx_rank_c ( integer(i32), intent(in), value m, integer(i32), intent(in), value n, real(dp), dimension(m,n), intent(inout) a, type(errorhandler), intent(inout) err )`

Computes the rank of a matrix.

## Parameters

in	<i>m</i>	The number of rows in the matrix.
in	<i>n</i>	The number of columns in the matrix.
in, out	<i>a</i>	On input, the M-by-N matrix of interest. On output, the contents of the matrix are overwritten.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_CONVERGENCE_ERROR: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

Definition at line 341 of file `linalg_c_binding.f90`.

**4.2.2.23** `subroutine linalg_c_binding::mult_qr_c ( logical(c_bool), intent(in), value trans, integer(i32), intent(in), value m, integer(i32), intent(in), value n, real(dp), dimension(m,m), intent(inout) q, real(dp), dimension(min(m,n)), intent(in) tau, real(dp), dimension(m,n), intent(inout) c, type(errorhandler), intent(inout) err )`

Multiplies a general matrix by the orthogonal matrix *Q* from a QR factorization such that:  $C = op(Q) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Q^*T$ ; else, set to false.
in	<i>m</i>	The number of rows in the matrix <i>c</i> .

## Parameters

in	<i>n</i>	The number of columns in the matrix <i>c</i> .
in	<i>q</i>	On input, an M-by-M matrix containing the elementary reflectors output from the QR factorization. Notice, the contents of this matrix are restored on exit. that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M,N)-element array containing the scalar factors of each elementary reflector defined in <i>a</i> .
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the product of the orthogonal matrix Q and the original matrix C.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the scalar factor array is not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 737 of file linalg\_c\_binding.f90.

4.2.2.24 subroutine linalg\_c\_binding::mult\_rz\_c ( logical(c\_bool), intent(in), value *trans*, integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *l*, real(dp), dimension(m,m), intent(inout) *a*, real(dp), dimension(m), intent(in) *tau*, real(dp), dimension(m,n), intent(inout) *c*, type(errorhandler), intent(inout) *err* )

Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Z^{**T}$ ; else, set to false.
in	<i>m</i>	The number of rows in the matrix <i>c</i> .
in	<i>n</i>	The number of columns in the matrix <i>c</i> .
in	<i>l</i>	The number of columns in matrix <i>a</i> containing the meaningful part of the Householder vectors ( $M \geq L \geq 0$ ).
in, out	<i>a</i>	On input, the M-by-M matrix Z as output by <i>rz_factor</i> . The matrix is used as in-place storage during execution; however, the contents of the matrix are restored on exit.
in	<i>tau</i>	An M-element array containing the scalar factors of the elementary reflectors found in <i>a</i> .
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the product of the orthogonal matrix Z and the original matrix C.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 969 of file linalg\_c\_binding.f90.

**4.2.2.25** subroutine `linalg_c_binding::qr_factor_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(min(m,n)), intent(out) *tau*, type(errorhandler), intent(inout) *err* )

Computes the QR factorization of an M-by-N matrix without pivoting.

#### Parameters

in	<i>m</i>	The number of rows in the matrix.
in	<i>n</i>	The number of columns in the matrix.
in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <code>tau</code> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the scalar factor array is not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 553 of file `linalg_c_binding.f90`.

**4.2.2.26** subroutine `linalg_c_binding::qr_factor_pivot_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(min(m,n)), intent(out) *tau*, integer(i32), dimension(n), intent(inout) *jpvt*, type(errorhandler), intent(inout) *err* )

Computes the QR factorization of an M-by-N matrix with column pivoting such that  $A * P = Q * R$ .

#### Parameters

in	<i>m</i>	The number of rows in the matrix.
in	<i>n</i>	The number of columns in the matrix.
in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <code>tau</code> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
in, out	<i>jpvt</i>	On input, an N-element array that if JPVT(I) .ne. 0, the I-th column of A is permuted to the front of $A * P$ ; if JPVT(I) = 0, the I-th column of A is a free column. On output, if JPVT(I) = K, then the I-th column of $A * P$ was the K-th column of A.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if the scalar factor array is not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>



Definition at line 598 of file linalg\_c\_binding.f90.

**4.2.2.27** subroutine linalg\_c\_binding::qr\_rank1\_update\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,m), intent(inout) *q*, real(dp), dimension(m,n), intent(inout) *r*, real(dp), dimension(m), intent(inout) *u*, real(dp), dimension(n), intent(inout) *v*, type(errorhandler), intent(inout) *err* )

Computes the rank 1 update to an M-by-N QR factored matrix A ( $M \geq N$ ) where  $A = Q * R$ , and  $A1 = A + U * V^{**}T$  such that  $A1 = Q1 * R1$ .

#### Parameters

in	<i>m</i>	The number of rows in the original matrix.
in	<i>n</i>	The number of columns in the original matrix.
in, out	<i>q</i>	On input, the original M-by-M orthogonal matrix Q. On output, the updated matrix Q1.
in, out	<i>r</i>	On input, the M-by-N matrix R. On output, the updated matrix R1.
in, out	<i>u</i>	On input, the M-element U update vector. On output, the original content of the array is overwritten.
in, out	<i>v</i>	On input, the N-element V update vector. On output, the original content of the array is overwritten.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 778 of file linalg\_c\_binding.f90.

**4.2.2.28** subroutine linalg\_c\_binding::rank1\_update\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), intent(in), value *alpha*, real(dp), dimension(m), intent(in) *x*, real(dp), dimension(n), intent(in) *y*, real(dp), dimension(m,n), intent(inout) *a* )

Performs the rank-1 update to matrix A such that:  $A = \alpha * X * Y^{**}T + A$ , where A is an M-by-N matrix, alpha is a scalar, X is an M-element array, and Y is an N-element array.

#### Parameters

in	<i>m</i>	The number of elements in <i>x</i> , and the number of rows in matrix <i>a</i> .
in	<i>n</i>	The number of elements in <i>y</i> , and the number of columns in matrix <i>a</i> .
in	<i>alpha</i>	The scalar multiplier.
in	<i>x</i>	An M-element array.
in	<i>y</i>	An N-element array.
in, out	<i>a</i>	On input, the M-by-N matrix to update. On output, the updated M-by-N matrix.

#### Notes

This routine is based upon the BLAS routine DGER.

Definition at line 294 of file linalg\_c\_binding.f90.

**4.2.2.29** subroutine `linalg_c_binding::rz_factor_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(m), intent(out) *tau*, type(errorhandler), intent(inout) *err* )

Factors an upper trapezoidal matrix by means of orthogonal transformations such that  $A = R * Z = (R \ 0) * Z$ . *Z* is an orthogonal matrix of dimension N-by-N, and *R* is an M-by-M upper triangular matrix.

#### Parameters

in	<i>m</i>	The number of rows in the original matrix.
in	<i>n</i>	The number of columns in the original matrix.
in, out	<i>a</i>	On input, the M-by-N upper trapezoidal matrix to factor. On output, the leading M-by-M upper triangular part of the matrix contains the upper triangular matrix <i>R</i> , and elements N-L+1 to N of the first M rows of <i>A</i> , with the array <i>tau</i> , represent the orthogonal matrix <i>Z</i> as a product of M elementary reflectors.
out	<i>tau</i>	An M-element array used to store the scalar factors of the elementary reflectors.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 927 of file `linalg_c_binding.f90`.

**4.2.2.30** subroutine `linalg_c_binding::solve_cholesky_c` ( logical(c\_bool), intent(in), value *upper*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), dimension(n,n), intent(in) *a*, real(dp), dimension(n,nrhs), intent(inout) *b* )

Solves a system of Cholesky factored equations.

#### Parameters

in	<i>upper</i>	Set to true if the original matrix <i>A</i> was factored such that $A = U^{**T} * U$ ; else, set to false if the factorization of <i>A</i> was $A = L^{**T} * L$ .
in	<i>n</i>	The dimension of the original matrix <i>a</i> .
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix <i>b</i> ).
in	<i>a</i>	The N-by-N Cholesky factored matrix.
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side matrix <i>B</i> . On output, the solution matrix <i>X</i> .

Definition at line 1190 of file `linalg_c_binding.f90`.

**4.2.2.31** subroutine `linalg_c_binding::solve_least_squares_c` ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), dimension(m, n), intent(inout) *a*, real(dp), dimension(max(m,n), nrhs), intent(inout) *b*, type(errorhandler), intent(inout) *err* )

Solves the overdetermined or underdetermined system ( $A * X = B$ ) of M equations of N unknowns using a QR or LQ factorization of the matrix *A*. Notice, it is assumed that matrix *A* has full rank.

#### Parameters

in	<i>m</i>	The number of rows in the original coefficient matrix <i>A</i> .
----	----------	--

## Parameters

in	<i>n</i>	The number of columns in the original coefficient matrix A.
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix b).
in, out	<i>a</i>	On input, the M-by-N matrix A. On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the M-by-NRHS matrix B. On output, the first N rows contain the N-by-NRHS solution matrix X. If $M < N$ , an N-by-NRHS matrix with the first M rows containing the matrix B. On output, the N-by-NRHS solution matrix X.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 1292 of file linalg\_c\_binding.f90.

**4.2.2.32** subroutine linalg\_c\_binding::solve\_lu\_c ( integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), dimension(n,n), intent(in) *a*, integer(i32), dimension(n), intent(in) *ipvt*, real(dp), dimension(n,nrhs), intent(inout) *b* )

Solves a system of LU-factored equations.

## Parameters

in	<i>n</i>	The dimension of the original matrix a.
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix b).
in	<i>a</i>	The N-by-N LU factored matrix as output by lu_factor.
in	<i>ipvt</i>	The N-element pivot array as output by lu_factor.
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side matrix. On output, the N-by-NRHS solution matrix.

Definition at line 1079 of file linalg\_c\_binding.f90.

**4.2.2.33** subroutine linalg\_c\_binding::solve\_qr\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(min(m,n)), intent(in) *tau*, real(dp), dimension(m,nrhs), intent(inout) *b*, type(errorhandler), intent(inout) *err* )

Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .

## Parameters

in	<i>m</i>	The number of rows in the original coefficient matrix A.
in	<i>n</i>	The number of columns in the original coefficient matrix A.
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix b).
in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by qr_factor. On output, the contents of this matrix are restored. Notice, M must be greater than or equal to N.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by qr_factor.

## Parameters

in	<i>b</i>	On input, the M-by-NRHS right-hand-side matrix. On output, the first N columns are overwritten by the solution matrix X.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 1112 of file linalg\_c\_binding.f90.

4.2.2.34 subroutine linalg\_c\_binding::solve\_qr\_pivot\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), dimension(m,n), intent(inout) *a*, real(dp), dimension(min(m,n)), intent(in) *tau*, integer(i32), dimension(n), intent(in) *jpvt*, real(dp), dimension(max(m,n),nrhs), intent(inout) *b*, type(errorhandler), intent(inout) *err* )

Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.

## Parameters

in	<i>m</i>	The number of rows in the original coefficient matrix A.
in	<i>n</i>	The number of columns in the original coefficient matrix A.
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix b).
in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by qr_factor. On output, the contents of this matrix are altered.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by qr_factor.
in	<i>jpvt</i>	An N-element array, as output by qr_factor, used to track the column pivots.
in	<i>b</i>	On input, the MAX(M, N)-by-NRHS matrix where the first M rows contain the right-hand-side matrix B. On output, the first N rows are overwritten by the solution matrix X.
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

Definition at line 1157 of file linalg\_c\_binding.f90.

4.2.2.35 subroutine linalg\_c\_binding::solve\_tri\_mtx\_c ( logical(c\_bool), intent(in), value *upper*, logical(c\_bool), intent(in), value *trans*, logical(c\_bool), intent(in), value *nounit*, integer(i32), intent(in), value *n*, integer(i32), intent(in), value *nrhs*, real(dp), intent(in), value *alpha*, real(dp), dimension(n,n), intent(in) *a*, real(dp), dimension(n,nrhs), intent(inout) *b* )

Solves one of the matrix equations:  $op(A) * X = \alpha * B$ , where A is a triangular matrix.

## Parameters

in	<i>upper</i>	Set to true if A is an upper triangular matrix; else, set to false if A is a lower triangular matrix.
in	<i>trans</i>	Set to true if $op(A) = A^{**T}$ ; else, set to false if $op(A) = A$ .
in	<i>nounit</i>	Set to true if A is not a unit-diagonal matrix (ones on every diagonal element); else, set to false if A is a unit-diagonal matrix.
in	<i>n</i>	The dimension of the triangular matrix a.
in	<i>nrhs</i>	The number of right-hand-side vectors (number of columns in matrix b).
in	<i>alpha</i>	The scalar multiplier to B.
in	<i>a</i>	N-by-N triangular matrix on which to operate.
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side. On output, the N-by-NRHS solution.

Definition at line 1056 of file linalg\_c\_binding.f90.

**4.2.2.36** subroutine linalg\_c\_binding::sort\_cmplx\_ind\_c ( logical(c\_bool), intent(in), value *ascend*, integer(i32), intent(in), value *n*, complex(dp), dimension(n), intent(inout) *x*, type(c\_ptr), intent(in), value *ind* )

Sorts an array of complex values according to their real components.

## Parameters

in	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
in	<i>n</i>	The number of elements in the array.
in, out	<i>x</i>	On input, the N-element array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, a pointer to an integer array. If NULL, this argument is ignored, and <i>x</i> is sorted as expected. However, if used, on output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .

Definition at line 1495 of file linalg\_c\_binding.f90.

**4.2.2.37** subroutine linalg\_c\_binding::sort\_dbl\_ind\_c ( logical(c\_bool), intent(in), value *ascend*, integer(i32), intent(in), value *n*, real(dp), dimension(n), intent(inout) *x*, type(c\_ptr), intent(in), value *ind* )

Sorts an array of double-precision values.

## Parameters

in	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
in	<i>n</i>	The number of elements in the array.
in, out	<i>x</i>	On input, the N-element array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, a pointer to an integer array. If NULL, this argument is ignored, and <i>x</i> is sorted as expected. However, if used, on output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .

Definition at line 1459 of file linalg\_c\_binding.f90.

**4.2.2.38** `subroutine linalg_c_binding::sort_eigen_cmplx_c ( logical(c_bool), intent(in), value ascend, integer(i32), intent(in), value n, complex(dp), dimension(n), intent(inout) vals, complex(dp), dimension(n,n), intent(inout) vecs )`

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

#### Parameters

<i>in</i>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<i>in</i>	<i>n</i>	The number of eigenvalues.
<i>in, out</i>	<i>vals</i>	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
<i>in, out</i>	<i>vecs</i>	On input, an N-by-N matrix containing the eigenvectors associated with <i>vals</i> (one vector per column). On output, the sorted eigenvector matrix.

Definition at line 1527 of file linalg\_c\_binding.f90.

**4.2.2.39** `subroutine linalg_c_binding::sort_eigen_dbl_c ( logical(c_bool), intent(in), value ascend, integer(i32), intent(in), value n, real(dp), dimension(n), intent(inout) vals, real(dp), dimension(n,n), intent(inout) vecs )`

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

#### Parameters

<i>in</i>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<i>in</i>	<i>n</i>	The number of eigenvalues.
<i>in, out</i>	<i>vals</i>	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
<i>in, out</i>	<i>vecs</i>	On input, an N-by-N matrix containing the eigenvectors associated with <i>vals</i> (one vector per column). On output, the sorted eigenvector matrix.

Definition at line 1550 of file linalg\_c\_binding.f90.

**4.2.2.40** `subroutine linalg_c_binding::svd_c ( integer(i32), intent(in), value m, integer(i32), intent(in), value n, real(dp), dimension(m,n), intent(inout) a, real(dp), dimension(min(m,n)), intent(out) s, real(dp), dimension(m,m), intent(out) u, real(dp), dimension(n,n), intent(out) vt, type(errorhandler), intent(inout) err )`

Computes the singular value decomposition of a matrix A. The SVD is defined as:  $A = U * S * V^{**T}$ , where U is an M-by-M orthogonal matrix, S is an M-by-N diagonal matrix, and V is an N-by-N orthogonal matrix.

#### Parameters

<i>in</i>	<i>m</i>	The number of rows in the original matrix.
<i>in</i>	<i>n</i>	The number of columns in the original matrix.
<i>in, out</i>	<i>a</i>	On input, the M-by-N matrix to factor. The matrix is overwritten on output. that the remaining matrix is simply the M-by-N matrix R.

## Parameters

out	<i>s</i>	A MIN(M, N)-element array containing the singular values of <i>a</i> sorted in descending order.
out	<i>u</i>	An M-by-M matrix that on output contains the left singular vectors (matrix U in the decomposition: $A = U * S * V^{**T}$ )
out	<i>vt</i>	An N-by-N matrix that on output contains the right singular vectors (matrix $V^{**T}$ in the decomposition: $A = U * S * V^{**T}$ ).
in, out	<i>err</i>	The errorhandler object. If no error handling is desired, simply pass NULL, and errors will be dealt with by the default internal error handler. Possible errors that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if the singular value array is not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

Definition at line 1016 of file linalg\_c\_binding.f90.

**4.2.2.41** subroutine linalg\_c\_binding::swap\_c ( integer(i32), intent(in), value *n*, real(dp), dimension(n), intent(inout) *x*, real(dp), dimension(n), intent(inout) *y* )

Swaps the contents of two arrays.

## Parameters

in	<i>n</i>	The number of elements either array.
in, out	<i>x</i>	One of the N-element arrays.
in, out	<i>y</i>	The other N-element array.

Definition at line 399 of file linalg\_c\_binding.f90.

**4.2.2.42** pure real(dp) function linalg\_c\_binding::trace\_c ( integer(i32), intent(in), value *m*, integer(i32), intent(in), value *n*, real(dp), dimension(m,n), intent(in) *x* )

Computes the trace of a matrix (the sum of the main diagonal elements).

## Parameters

in	<i>m</i>	The number of rows in the matrix.
in	<i>n</i>	The number of columns in the matrix.
in	<i>x</i>	The matrix on which to operate.

## Returns

The trace of *x*.

Definition at line 314 of file linalg\_c\_binding.f90.

## 4.3 linalg\_constants Module Reference

### [linalg\\_constants](#)

#### Variables

- integer, parameter [dp](#) = real64  
*Defines a double-precision (64-bit) floating-point type.*
- integer, parameter [i32](#) = int32  
*Defines a 32-bit signed integer type.*
- integer, parameter [la\\_invalid\\_input\\_error](#) = 101  
*An error flag denoting an invalid input.*
- integer, parameter [la\\_array\\_size\\_error](#) = 102  
*An error flag denoting an improperly sized array.*
- integer, parameter [la\\_singular\\_matrix\\_error](#) = 103  
*An error flag denoting a singular matrix.*
- integer, parameter [la\\_matrix\\_format\\_error](#) = 104  
*An error flag denoting an issue with the matrix format.*
- integer, parameter [la\\_out\\_of\\_memory\\_error](#) = 105  
*An error flag denoting that there is insufficient memory available.*
- integer, parameter [la\\_convergence\\_error](#) = 106  
*An error flag denoting a convergence failure.*
- integer, parameter [la\\_invalid\\_operation\\_error](#) = 107  
*An error resulting from an invalid operation.*

#### 4.3.1 Detailed Description

### [linalg\\_constants](#)

#### Purpose

Provides a set of constants and error flags for the library.

## 4.4 linalg\_core Module Reference

### [linalg\\_core](#)

#### Data Types

- interface [diag\\_mtx\\_mult](#)  
*Multiplies a diagonal matrix with another matrix or array.*
- interface [mtx\\_mult](#)  
*Performs the matrix operation:  $C = \alpha * op(A) * op(B) + \beta * C$ .*



## Functions/Subroutines

- subroutine [mtx\\_mult\\_mtx](#) (transa, transb, alpha, a, b, beta, c, err)  
*Performs the matrix operation:  $C = \alpha * op(A) * op(B) + \beta * C$ .*
- subroutine [mtx\\_mult\\_vec](#) (trans, alpha, a, b, beta, c, err)  
*Performs the matrix-vector operation:  $c = \alpha * op(A) * b + \beta * c$ .*
- subroutine, public [rank1\\_update](#) (alpha, x, y, a, err)  
*Performs the rank-1 update to matrix A such that:  $A = \alpha * X * Y^{**T} + A$ , where A is an M-by-N matrix, alpha is a scalar, X is an M-element array, and Y is an N-element array.*
- subroutine [diag\\_mtx\\_mult\\_mtx](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ .*
- subroutine [diag\\_mtx\\_mult\\_mtx2](#) (lside, alpha, a, b, err)  
*Computes the matrix operation:  $B = \alpha * A * op(B)$ , or  $B = \alpha * op(B) * A$ .*
- subroutine [diag\\_mtx\\_mult\\_mtx3](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ , where A and C are complex-valued.*
- subroutine [diag\\_mtx\\_mult\\_mtx4](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ , where A, B, and C are complex-valued.*
- pure real(dp) function, public [trace](#) (x)  
*Computes the trace of a matrix (the sum of the main diagonal elements).*
- integer(i32) function, public [mtx\\_rank](#) (a, tol, work, olwork, err)  
*Computes the rank of a matrix.*
- real(dp) function, public [det](#) (a, iwork, err)  
*Computes the determinant of a square matrix.*
- subroutine, public [swap](#) (x, y, err)  
*Swaps the contents of two arrays.*
- subroutine, public [recip\\_mult\\_array](#) (a, x)  
*Multiplies a vector by the reciprocal of a real scalar.*
- subroutine, public [tri\\_mtx\\_mult](#) (upper, alpha, a, beta, b, err)  
*Computes the triangular matrix operation:  $B = \alpha * A^{**T} * A + \beta * B$ , or  $B = \alpha * A * A^{**T} + \beta * B$ , where A is a triangular matrix.*

### 4.4.1 Detailed Description

#### [linalg\\_core](#)

##### Purpose

Provides common "core" linear algebra routines.

### 4.4.2 Function/Subroutine Documentation

- 4.4.2.1 real(dp) function, public [linalg\\_core::det](#) ( real(dp), dimension(:,:), intent(inout) a, integer(i32), dimension(:), intent(out), optional, pointer iwork, class(errors), intent(inout), optional, target err )

Computes the determinant of a square matrix.

## Parameters

in, out	<i>a</i>	On input, the N-by-N matrix on which to operate. On output the contents are overwritten by the LU factorization of the original matrix.
out	<i>iwork</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least N-elements.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Returns

The determinant of *a*.

Definition at line 1088 of file `linalg_core.f90`.

4.4.2.2 subroutine `linalg_core::diag_mtx_mult_mtx` ( `logical, intent(in) lside`, `logical, intent(in) trans`, `real(dp) alpha`, `real(dp), dimension(:), intent(in) a`, `real(dp), dimension(:, :), intent(in) b`, `real(dp) beta`, `real(dp), dimension(:, :), intent(inout) c`, `class(errors), intent(inout), optional, target err` ) [private]

Computes the matrix operation:  $C = \alpha * A * \text{op}(B) + \beta * C$ , or  $C = \alpha * \text{op}(B) * A + \beta * C$ .

## Parameters

in	<i>lside</i>	Set to true to apply matrix A from the left; else, set to false to apply matrix A from the left.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^*T$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of A where $K = \text{MIN}(M,P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N,P)$ .
in	<i>b</i>	The LDB-by-TDB matrix B where (LDB = leading dimension of B, and TDB = trailing dimension of B): <ul style="list-style-type: none"> <li>• <i>lside</i> == true &amp; <i>trans</i> == true: LDB = N, TDB = P</li> <li>• <i>lside</i> == true &amp; <i>trans</i> == false: LDB = P, TDB = N</li> <li>• <i>lside</i> == false &amp; <i>trans</i> == true: LDB = P, TDB = M</li> <li>• <i>lside</i> == false &amp; <i>trans</i> == false: LDB = M, TDB = P</li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the resulting M-by-N matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 329 of file linalg\_core.f90.

**4.4.2.3** subroutine `linalg_core::diag_mtx_mult_mtx2` ( `logical`, intent(in) *lside*, `real(dp)`, intent(in) *alpha*, `real(dp)`, dimension(:), intent(in) *a*, `real(dp)`, dimension(:, :), intent(inout) *b*, `class(errors)`, intent(inout), optional, target *err* ) [private]

Computes the matrix operation:  $B = \alpha * A * \text{op}(B)$ , or  $B = \alpha * \text{op}(B) * A$ .

#### Parameters

in	<i>lside</i>	Set to true to apply matrix A from the left; else, set to false to apply matrix A from the left.
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of A where $K = \text{MIN}(M, P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N, P)$ .
in	<i>b</i>	On input, the M-by-N matrix B. On output, the resulting M-by-N matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 496 of file linalg\_core.f90.

**4.4.2.4** subroutine `linalg_core::diag_mtx_mult_mtx3` ( `logical`, intent(in) *lside*, `logical`, intent(in) *trans*, `real(dp)` *alpha*, `complex(dp)`, dimension(:), intent(in) *a*, `real(dp)`, dimension(:, :), intent(in) *b*, `real(dp)` *beta*, `complex(dp)`, dimension(:, :), intent(inout) *c*, `class(errors)`, intent(inout), optional, target *err* ) [private]

Computes the matrix operation:  $C = \alpha * A * \text{op}(B) + \beta * C$ , or  $C = \alpha * \text{op}(B) * A + \beta * C$ , where A and C are complex-valued.

#### Parameters

in	<i>lside</i>	Set to true to apply matrix A from the left; else, set to false to apply matrix A from the left.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^T$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of A where $K = \text{MIN}(M, P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N, P)$ .
in	<i>b</i>	The LDB-by-TDB matrix B where (LDB = leading dimension of B, and TDB = trailing dimension of B): <ul style="list-style-type: none"> <li><i>lside</i> == true &amp; <i>trans</i> == true: LDB = N, TDB = P</li> <li><i>lside</i> == true &amp; <i>trans</i> == false: LDB = P, TDB = N</li> <li><i>lside</i> == false &amp; <i>trans</i> == true: LDB = P, TDB = M</li> <li><i>lside</i> == false &amp; <i>trans</i> == false: LDB = M, TDB = P</li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the resulting M-by-N matrix.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>
-----	-----	--

Definition at line 579 of file linalg\_core.f90.

**4.4.2.5** subroutine linalg\_core::diag\_mtx\_mult\_mtx4 ( logical, intent(in) *lside*, logical, intent(in) *trans*, real(dp) *alpha*, complex(dp), dimension(:), intent(in) *a*, complex(dp), dimension(:, :), intent(in) *b*, real(dp) *beta*, complex(dp), dimension(:, :), intent(inout) *c*, class(errors), intent(inout), optional, target *err* ) [private]

Computes the matrix operation:  $C = \alpha * A * \text{op}(B) + \beta * C$ , or  $C = \alpha * \text{op}(B) * A + \beta * C$ , where *A*, *B*, and *C* are complex-valued.

## Parameters

in	<i>lside</i>	Set to true to apply matrix <i>A</i> from the left; else, set to false to apply matrix <i>A</i> from the right.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^{**T}$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of <i>A</i> where $K = \text{MIN}(M, P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N, P)$ .
in	<i>b</i>	<p>The LDB-by-TDB matrix <i>B</i> where:</p> <ul style="list-style-type: none"> <li><i>lside</i> == true &amp; <i>trans</i> == true: LDA = N, TDB = P</li> <li><i>lside</i> == true &amp; <i>trans</i> == false: LDA = P, TDB = N</li> <li><i>lside</i> == false &amp; <i>trans</i> == true: LDA = P, TDB = M</li> <li><i>lside</i> == false &amp; <i>trans</i> == false: LDA = M, TDB = P</li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix <i>C</i> . On output, the resulting M-by-N matrix.
out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 755 of file linalg\_core.f90.

**4.4.2.6** subroutine linalg\_core::mtx\_mult\_mtx ( logical, intent(in) *transa*, logical, intent(in) *transb*, real(dp), intent(in) *alpha*, real(dp), dimension(:, :), intent(in) *a*, real(dp), dimension(:, :), intent(in) *b*, real(dp), intent(in) *beta*, real(dp), dimension(:, :), intent(inout) *c*, class(errors), intent(inout), optional, target *err* ) [private]

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

## Parameters

in	<i>transa</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false for $\text{op}(A) = A$ .
in	<i>transb</i>	Set to true if $\text{op}(B) = B^{**T}$ ; else, set to false for $\text{op}(B) = B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	If <i>transa</i> is set to true, an K-by-M matrix; else, if <i>transa</i> is set to false, an M-by-K matrix.
in	<i>b</i>	If <i>transb</i> is set to true, an N-by-K matrix; else, if <i>transb</i> is set to false, a K-by-N matrix.
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the M-by-N result.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the BLAS routine DGEMM.

Definition at line 85 of file linalg\_core.f90.

**4.4.2.7** subroutine linalg\_core::mtx\_mult\_vec ( logical, intent(in) *trans*, real(dp), intent(in) *alpha*, real(dp), dimension(:,:), intent(in) *a*, real(dp), dimension(:), intent(in) *b*, real(dp), intent(in) *beta*, real(dp), dimension(:), intent(inout) *c*, class(errors), intent(inout), optional, target *err* ) [private]

Performs the matrix-vector operation:  $c = \alpha * \text{op}(A) * b + \beta * c$ .

## Parameters

in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false for $\text{op}(A) = A$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	The M-by-N matrix A.
in	<i>b</i>	If <i>trans</i> is set to true, an M-element array; else, if <i>trans</i> is set to false, an N-element array.
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, if <i>trans</i> is set to true, an N-element array; else, if <i>trans</i> is set to false, an M-element array. On output, the results of the operation.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the BLAS routine DGEMV.

Definition at line 179 of file linalg\_core.f90.

**4.4.2.8** `integer(i32) function, public linalg_core::mtx_rank ( real(dp), dimension(:,:), intent(inout) a, real(dp), intent(in), optional tol, real(dp), dimension(:), intent(out), optional, pointer work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err )`

Computes the rank of a matrix.

#### Parameters

in, out	<i>a</i>	On input, the M-by-N matrix of interest. On output, the contents of the matrix are overwritten.
in	<i>tol</i>	An optional input, that if supplied, overrides the default tolerance on singular values such that singular values less than this tolerance are treated as zero. The default tolerance is: $\text{MAX}(M, N) * \text{EPS} * \text{MAX}(S)$ . If the supplied value is less than the smallest value that causes an overflow if inverted, the tolerance reverts back to its default value, and the operation continues; however, a warning message is issued.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

#### See Also

- [Wolfram MathWorld](#)

Definition at line 968 of file linalg\_core.f90.

**4.4.2.9** `subroutine, public linalg_core::rank1_update ( real(dp), intent(in) alpha, real(dp), dimension(:), intent(in) x, real(dp), dimension(:), intent(in) y, real(dp), dimension(:,:), intent(inout) a, class(errors), intent(inout), optional, target err )`

Performs the rank-1 update to matrix *A* such that:  $A = \alpha * X * Y^T + A$ , where *A* is an M-by-N matrix, *alpha* is a scalar, *X* is an M-element array, and *N* is an N-element array.

#### Parameters

in	<i>alpha</i>	The scalar multiplier.
in	<i>x</i>	An M-element array.

## Parameters

<code>in</code>	<code>y</code>	An N-element array.
<code>in, out</code>	<code>a</code>	On input, the M-by-N matrix to update. On output, the updated M-by-N matrix.
<code>out</code>	<code>err</code>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if the size of <code>a</code> does not match with <code>x</code> and <code>y</code>.</li> </ul>

## Notes

This routine is based upon the BLAS routine DGER.

Definition at line 257 of file `linalg_core.f90`.

**4.4.2.10** `subroutine, public linalg_core::recip_mult_array ( real(dp), intent(in) a, real(dp), dimension(:), intent(inout) x )`

Multiplies a vector by the reciprocal of a real scalar.

## Parameters

<code>in</code>	<code>a</code>	The scalar which is used to divide each component of <code>x</code> . The value must be $\geq 0$ , or the subroutine will divide by zero.
<code>in, out</code>	<code>x</code>	The vector.

## Notes

This routine is based upon the LAPACK routine DRSCL.

Definition at line 1239 of file `linalg_core.f90`.

**4.4.2.11** `subroutine, public linalg_core::swap ( real(dp), dimension(:), intent(inout) x, real(dp), dimension(:), intent(inout) y, class(errors), intent(inout), optional, target err )`

Swaps the contents of two arrays.

## Parameters

<code>in, out</code>	<code>x</code>	One of the N-element arrays.
<code>in, out</code>	<code>y</code>	The other N-element array.
<code>out</code>	<code>err</code>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <code>x</code> and <code>y</code> are not the same size.</li> </ul>

Definition at line 1193 of file linalg\_core.f90.

#### 4.4.2.12 pure real(dp) function, public linalg\_core::trace ( real(dp), dimension(:, :), intent(in) *x* )

Computes the trace of a matrix (the sum of the main diagonal elements).

##### Parameters

in	<i>x</i>	The matrix on which to operate.
----	----------	---------------------------------

##### Returns

The trace of *x*.

Definition at line 912 of file linalg\_core.f90.

#### 4.4.2.13 subroutine, public linalg\_core::tri\_mtx\_mult ( logical, intent(in) *upper*, real(dp), intent(in) *alpha*, real(dp), dimension(:, :), intent(in) *a*, real(dp), intent(in) *beta*, real(dp), dimension(:, :), intent(inout) *b*, class(errors), intent(inout), optional, target *err* )

Computes the triangular matrix operation:  $B = \alpha * A^{**T} * A + \beta * B$ , or  $B = \alpha * A * A^{**T} + \beta * B$ , where *A* is a triangular matrix.

##### Parameters

in	<i>upper</i>	Set to true if matrix <i>A</i> is upper triangular, and $B = \alpha * A^{**T} * A + \beta * B$ is to be calculated; else, set to false if <i>A</i> is lower triangular, and $B = \alpha * A * A^{**T} + \beta * B$ is to be computed.
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	The N-by-N triangular matrix. Notice, if <i>upper</i> is true only the upper triangular portion of this matrix is referenced; else, if <i>upper</i> is false, only the lower triangular portion of this matrix is referenced.
in	<i>beta</i>	A scalar multiplier.
in, out	<i>b</i>	On input, the N-by-N matrix <i>B</i> . On output, the N-by-N solution matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> </ul>

Definition at line 1316 of file linalg\_core.f90.

## 4.5 linalg\_eigen Module Reference

### [linalg\\_eigen](#)



## Data Types

- interface [eigen](#)

*Computes the eigenvalues, and optionally the eigenvectors, of a matrix.*

## Functions/Subroutines

- subroutine [eigen\\_symm](#) (vecs, a, vals, work, olwork, err)

*Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.*

- subroutine [eigen\\_asymm](#) (a, vals, vecs, work, olwork, err)

*Computes the eigenvalues, and optionally the right eigenvectors of a square matrix.*

- subroutine [eigen\\_gen](#) (a, b, alpha, beta, vecs, work, olwork, err)

*Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A*X = \lambda B*X$ .*

### 4.5.1 Detailed Description

#### [linalg\\_eigen](#)

#### Purpose

Provides routines for computing the eigenvalues and eigenvectors of matrices.

### 4.5.2 Function/Subroutine Documentation

**4.5.2.1** subroutine `linalg_eigen::eigen_asymm` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *a*, `complex(dp)`, `dimension(:)`, `intent(out)` *vals*, `complex(dp)`, `dimension(:, :)`, `intent(out)`, optional *vecs*, `real(dp)`, `dimension(:)`, `intent(out)`, optional, pointer *work*, `integer(i32)`, `intent(out)`, optional *olwork*, `class(errors)`, `intent(inout)`, optional, target *err* ) [private]

Computes the eigenvalues, and optionally the right eigenvectors of a square matrix.

#### Parameters

in, out	<i>a</i>	On input, the N-by-N matrix on which to operate. On output, the contents of this matrix are overwritten.
out	<i>vals</i>	An N-element array containing the eigenvalues of the matrix. The eigenvalues are not sorted.
out	<i>vecs</i>	An optional N-by-N matrix, that if supplied, signals to compute the right eigenvectors (one per column). If not provided, only the eigenvalues will be computed.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> </ul>
Generated by Doxygen		<ul style="list-style-type: none"> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGEEV.

Definition at line 185 of file linalg\_eigen.f90.

```
4.5.2.2 subroutine linalg_eigen::eigen_gen ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:,:), intent(inout)
      b, complex(dp), dimension(:,:), intent(out) alpha, real(dp), dimension(:,:), intent(out), optional beta, complex(dp),
      dimension(:,:), intent(out), optional vecs, real(dp), dimension(:,:), intent(out), optional, pointer work, integer(i32),
      intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]
```

Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A \cdot X = \lambda B \cdot X$ .

## Parameters

in, out	<i>a</i>	On input, the N-by-N matrix A. On output, the contents of this matrix are overwritten.
in, out	<i>b</i>	On input, the N-by-N matrix B. On output, the contents of this matrix are overwritten.
out	<i>alpha</i>	An N-element array that, if <i>beta</i> is not supplied, contains the eigenvalues. If <i>beta</i> is supplied however, the eigenvalues must be computed as ALPHA / BETA. This however, is not as trivial as it seems as it is entirely possible, and likely, that ALPHA / BETA can overflow or underflow. With that said, the values in ALPHA will always be less than and usually comparable with the NORM(A).
out	<i>beta</i>	An optional N-element array that if provided forces <i>alpha</i> to return the numerator, and this array contains the denominator used to determine the eigenvalues as ALPHA / BETA. If used, the values in this array will always be less than and usually comparable with the NORM(B).
out	<i>vecs</i>	An optional N-by-N matrix, that if supplied, signals to compute the right eigenvectors (one per column). If not provided, only the eigenvalues will be computed.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>• LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• LA_CONVERGENCE_ERROR: Occurs if the algorithm failed to converge.</li> </ul>

## Usage

As an example, consider the eigenvalue problem arising from a mechanical system of masses and springs such that the masses are described by a mass matrix *M*, and the arrangement of springs are described by a stiffness matrix *K*.

```
! This is an example illustrating the use of the eigenvalue and eigenvector
! routines to solve a free vibration problem of 3 masses connected by springs.
```

```

!
!      k1      k2      k3      k4
!  |---|---|---|---|
!  |---|---|---|---|
!
! As illustrated above, the system consists of 3 masses connected by springs.
! Spring k1 and spring k4 connect the end masses to ground. The equations of
! motion for this system are as follows.
!
!  | m1  0   0 | |x1"|  | k1+k2  -k2      0 | |x1|  |0|
!  |  0  m2  0 | |x2"| + | -k2  k2+k3   -k3 | |x2| = |0|
!  |  0   0  m3| |x3"|  |  0   -k3   k3+k4| |x3|  |0|
!
! Notice: x1" = the second time derivative of x1.
program example
  use linalg_constants, only : dp, i32
  use linalg_eigen
  implicit none

  ! Define the model parameters
  real(dp), parameter :: pi = 3.14159265359d0
  real(dp), parameter :: m1 = 0.5d0
  real(dp), parameter :: m2 = 2.5d0
  real(dp), parameter :: m3 = 0.75d0
  real(dp), parameter :: k1 = 5.0d6
  real(dp), parameter :: k2 = 10.0d6
  real(dp), parameter :: k3 = 10.0d6
  real(dp), parameter :: k4 = 5.0d6

  ! Local Variables
  integer(i32) :: i, j
  real(dp) :: m(3,3), k(3,3), natfreq(3)
  complex(dp) :: vals(3), modeshapes(3,3)

  ! Define the mass matrix
  m = reshape([m1, 0.0d0, 0.0d0, 0.0d0, m2, 0.0d0, 0.0d0, 0.0d0, m3], [3, 3])

  ! Define the stiffness matrix
  k = reshape([k1 + k2, -k2, 0.0d0, -k2, k2 + k3, -k3, 0.0d0, -k3, k3 + k4], &
    [3, 3])

  ! Compute the eigenvalues and eigenvectors.
  call eigen(k, m, vals, vecs = modeshapes)

  ! Compute the natural frequency values, and return them with units of Hz.
  ! Notice, all eigenvalues and eigenvectors are real for this example.
  natfreq = sqrt(real(vals)) / (2.0d0 * pi)

  ! Display the natural frequency and mode shape values. Notice, the eigen
  ! routine does not necessarily sort the values.
  print '(A)', "Modal Information (Not Sorted):"
  do i = 1, size(natfreq)
    print '(AI0AF8.4A)', "Mode ", i, ": (", natfreq(i), " Hz)"
    print '(F10.3)', (real(modeshapes(j,i)), j = 1, size(natfreq))
  end do
end program

```

## Notes

This routine utilizes the LAPACK routine DGGEV.

Definition at line 465 of file linalg\_eigen.f90.

**4.5.2.3** subroutine linalg\_eigen::eigen\_symm ( logical, intent(in) *vecs*, real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:,:), intent(out) *vals*, real(dp), dimension(:,:), intent(out), optional, pointer *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.

## Parameters

<i>in</i>	<i>vecs</i>	Set to true to compute the eigenvectors as well as the eigenvalues; else, set to false to just compute the eigenvalues.
-----------	-------------	---

## Parameters

in, out	<i>a</i>	On input, the N-by-N symmetric matrix on which to operate. On output, and if <i>vecs</i> is set to true, the matrix will contain the eigenvectors (one per column) corresponding to each eigenvalue in <i>vals</i> . If <i>vecs</i> is set to false, the lower triangular portion of the matrix is overwritten.
out	<i>vals</i>	An N-element array that will contain the eigenvalues sorted into ascending order.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs if the algorithm failed to converge.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DSYEV.

Definition at line 68 of file `linalg_eigen.f90`.

## 4.6 linalg\_factor Module Reference

### `linalg_factor`

## Data Types

- interface `form_lu`  
*Extracts the L and U matrices from the condensed [L\U] storage format used by the `lu_factor`.*
- interface `form_qr`  
*Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.*
- interface `mult_qr`  
*Multiplies a general matrix by the orthogonal matrix Q from a QR factorization.*
- interface `mult_rz`  
*Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization.*
- interface `qr_factor`  
*Computes the QR factorization of an M-by-N matrix.*

## Functions/Subroutines

- subroutine, public [lu\\_factor](#) (a, ipvt, err)  
*Computes the LU factorization of an M-by-N matrix.*
- subroutine [form\\_lu\\_all](#) (lu, ipvt, u, p, err)  
*Extracts the L, U, and P matrices from the output of the [lu\\_factor](#) routine.*
- subroutine [form\\_lu\\_only](#) (lu, u, err)  
*Extracts the L, and U matrices from the output of the [lu\\_factor](#) routine.*
- subroutine [qr\\_factor\\_no\\_pivot](#) (a, tau, work, olwork, err)  
*Computes the QR factorization of an M-by-N matrix without pivoting.*
- subroutine [qr\\_factor\\_pivot](#) (a, tau, jpvt, work, olwork, err)  
*Computes the QR factorization of an M-by-N matrix with column pivoting such that  $A * P = Q * R$ .*
- subroutine [form\\_qr\\_no\\_pivot](#) (r, tau, q, work, olwork, err)  
*Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.*
- subroutine [form\\_qr\\_pivot](#) (r, tau, pvt, q, p, work, olwork, err)  
*Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.*
- subroutine [mult\\_qr\\_mtx](#) (lside, trans, a, tau, c, work, olwork, err)  
*Multiplies a general matrix by the orthogonal matrix Q from a QR factorization such that:  $C = op(Q) * C$ , or  $C = C * op(Q)$ .*
- subroutine [mult\\_qr\\_vec](#) (trans, a, tau, c, work, olwork, err)  
*Multiplies a vector by the orthogonal matrix Q from a QR factorization such that:  $C = op(Q) * C$ .*
- subroutine, public [qr\\_rank1\\_update](#) (q, r, u, v, work, err)  
*Computes the rank 1 update to an M-by-N QR factored matrix A ( $M \geq N$ ) where  $A = Q * R$ , and  $A1 = A + U * V^{**T}$  such that  $A1 = Q1 * R1$ .*
- subroutine, public [cholesky\\_factor](#) (a, upper, err)  
*Computes the Cholesky factorization of a symmetric, positive definite matrix.*
- subroutine, public [cholesky\\_rank1\\_update](#) (r, u, work, err)  
*Computes the rank 1 update to a Cholesky factored matrix (upper triangular).*
- subroutine, public [cholesky\\_rank1\\_downdate](#) (r, u, work, err)  
*Computes the rank 1 downdate to a Cholesky factored matrix (upper triangular).*
- subroutine, public [rz\\_factor](#) (a, tau, work, olwork, err)  
*Factors an upper trapezoidal matrix by means of orthogonal transformations such that  $A = R * Z = (R \ 0) * Z$ . Z is an orthogonal matrix of dimension N-by-N, and R is an M-by-M upper triangular matrix.*
- subroutine [mult\\_rz\\_mtx](#) (lside, trans, l, a, tau, c, work, olwork, err)  
*Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization such that:  $C = op(Z) * C$ , or  $C = C * op(Z)$ .*
- subroutine [mult\\_rz\\_vec](#) (trans, l, a, tau, c, work, olwork, err)  
*Multiplies a vector by the orthogonal matrix Z from an RZ factorization such that:  $C = op(Z) * C$ .*
- subroutine, public [svd](#) (a, s, u, vt, work, olwork, err)  
*Computes the singular value decomposition of a matrix A. The SVD is defined as:  $A = U * S * V^{**T}$ , where U is an M-by-M orthogonal matrix, S is an M-by-N diagonal matrix, and V is an N-by-N orthogonal matrix.*

### 4.6.1 Detailed Description

#### [linalg\\_factor](#)

##### Purpose

Provides a set of matrix factorization routines.

## 4.6.2 Function/Subroutine Documentation

**4.6.2.1** subroutine, public linalg\_factor::cholesky\_factor ( real(dp), dimension(:,:), intent(inout) *a*, logical, intent(in), optional *upper*, class(errors), intent(inout), optional, target *err* )

Computes the Cholesky factorization of a symmetric, positive definite matrix.

### Parameters

in, out	<i>a</i>	On input, the N-by-N matrix to factor. On output, the factored matrix is returned in either the upper or lower triangular portion of the matrix, dependent upon the value of <i>upper</i> .
in	<i>upper</i>	An optional input that, if specified, provides control over whether the factorization is computed as $A = U^{**T} * U$ (set to true), or as $A = L * L^{**T}$ (set to false). The default value is true such that $A = U^{**T} * U$ .
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>a</i> is not square.</li> <li>LA_MATRIX_FORMAT_ERROR: Occurs if <i>a</i> is not positive definite.</li> </ul>

### Usage

To solve a system of N equations of N unknowns using Cholesky factorization, the following code will suffice. Notice, the system of equations must be positive definite.

```
! Solve the system: A*X = B, where A is an N-by-N matrix, and B and X are
! N-by-NRHS in size.

! Variables
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b
logical :: upper

! Initialize A and B...

! Specify that we're using the upper portion of A (remember positive
! definite matrices are symmetric)
upper = .true.

! Compute the factorization of A.
call cholesky_factor(a, upper)

! Solve A*X = B for X - Note: X overwrites B.
call solve_cholesky(upper, a, b)
```

### Notes

This routine utilizes the LAPACK routine DPOTRF.

Definition at line 1308 of file linalg\_factor.f90.

**4.6.2.2** subroutine, public linalg\_factor::cholesky\_rank1\_downdate ( real(dp), dimension(:,:), intent(inout) *r*, real(dp), dimension(:), intent(inout) *u*, real(dp), dimension(:), intent(out), optional, target *work*, class(errors), intent(inout), optional, target *err* )

Computes the rank 1 downdate to a Cholesky factored matrix (upper triangular).

## Parameters

<code>in, out</code>	<code>r</code>	On input, the N-by-N upper triangular matrix R. On output, the updated matrix R1.
<code>in, out</code>	<code>u</code>	On input, the N-element update vector U. On output, the rotation sines used to transform R to R1.
<code>out</code>	<code>work</code>	An optional argument that if supplied prevents local memory allocation. If provided, the array must have at least N elements. Additionally, this workspace array is used to contain the rotation cosines used to transform R to R1.
<code>out</code>	<code>err</code>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_MATRIX_FORMAT_ERROR</code>: Occurs if the downdated matrix is not positive definite.</li> <li>• <code>LA_SINGULAR_MATRIX_ERROR</code>: Occurs if <code>r</code> is singular.</li> </ul>

## Notes

This routine utilizes the QRUPDATE routine DCH1DN.

## See Also

[Source](#)

Definition at line 1494 of file `linalg_factor.f90`.

**4.6.2.3** subroutine, public `linalg_factor::cholesky_rank1_update` ( `real(dp)`, `dimension(:, :)`, `intent(inout) r`, `real(dp)`, `dimension(:)`, `intent(inout) u`, `real(dp)`, `dimension(:)`, `intent(out)`, optional, target `work`, `class(errors)`, `intent(inout)`, optional, target `err` )

Computes the rank 1 update to a Cholesky factored matrix (upper triangular).

## Parameters

<code>in, out</code>	<code>r</code>	On input, the N-by-N upper triangular matrix R. On output, the updated matrix R1.
<code>in, out</code>	<code>u</code>	On input, the N-element update vector U. On output, the rotation sines used to transform R to R1.
<code>out</code>	<code>work</code>	An optional argument that if supplied prevents local memory allocation. If provided, the array must have at least N elements. Additionally, this workspace array is used to contain the rotation cosines used to transform R to R1.
<code>out</code>	<code>err</code>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the QRUPDATE routine DCH1UP.

## See Also

[Source](#)

Definition at line 1400 of file linalg\_factor.f90.

```
4.6.2.4 subroutine linalg_factor::form_lu_all ( real(dp), dimension(:, :), intent(inout) lu, integer(i32), dimension(:), intent(in) ipvt,
      real(dp), dimension(:, :), intent(out) u, real(dp), dimension(:, :), intent(out) p, class(errors), intent(inout), optional, target
      err ) [private]
```

Extracts the L, U, and P matrices from the output of the [lu\\_factor](#) routine.

## Parameters

in, out	<i>lu</i>	On input, the N-by-N matrix as output by <a href="#">lu_factor</a> . On output, the N-by-N lower triangular matrix L.
in	<i>ipvt</i>	The N-element pivot array as output by <a href="#">lu_factor</a> .
out	<i>u</i>	An N-by-N matrix where the U matrix will be written.
out	<i>p</i>	An N-by-N matrix where the row permutation matrix will be written.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Remarks

This routine allows extraction of the actual "L", "U", and "P" matrices of the decomposition. To use these matrices to solve the system  $A \cdot X = B$ , the following approach is used.

1. First, solve the linear system:  $L \cdot Y = P \cdot B$  for Y.
2. Second, solve the linear system:  $U \cdot X = Y$  for X.

Notice, as both L and U are triangular in structure, the above equations can be solved by forward and backward substitution.

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 211 of file linalg\_factor.f90.

```
4.6.2.5 subroutine linalg_factor::form_lu_only ( real(dp), dimension(:, :), intent(inout) lu, real(dp), dimension(:, :), intent(out) u,
      class(errors), intent(inout), optional, target err ) [private]
```

Extracts the L, and U matrices from the output of the [lu\\_factor](#) routine.



## Parameters

in, out	<i>lu</i>	On input, the N-by-N matrix as output by <a href="#">lu_factor</a> . On output, the N-by-N lower triangular matrix L.
out	<i>u</i>	An N-by-N matrix where the U matrix will be written.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 288 of file `linalg_factor.f90`.

**4.6.2.6** subroutine `linalg_factor::form_qr_no_pivot` ( `real(dp), dimension(:, :), intent(inout) r`, `real(dp), dimension(:, :), intent(in) tau`, `real(dp), dimension(:, :), intent(out) q`, `real(dp), dimension(:, :), intent(out), optional, target work`, `integer(i32), intent(out), optional olwork`, `class(errors), intent(inout), optional, target err` ) [private]

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

## Parameters

in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that M > N, Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 (Q = [Q1, Q2]) as the factorization can be written as Q * R = [Q1, Q2] * [R1; 0].
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORGQR.

Definition at line 694 of file `linalg_factor.f90`.

4.6.2.7 subroutine `linalg_factor::form_qr_pivot` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *r*, `real(dp)`, `dimension(:)`, `intent(in)` *tau*, `integer(i32)`, `dimension(:)`, `intent(in)` *pvt*, `real(dp)`, `dimension(:, :)`, `intent(out)` *q*, `real(dp)`, `dimension(:, :)`, `intent(out)` *p*, `real(dp)`, `dimension(:)`, `intent(out)`, optional, target *work*, `integer(i32)`, `intent(out)`, optional *olwork*, `class(errors)`, `intent(inout)`, optional, target *err* ) [`private`]

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

#### Parameters

in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
in	<i>pvt</i>	An N-element column pivot array as returned by the QR factorization.
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that $M > N$ , Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 ( $Q = [Q1, Q2]$ ) as the factorization can be written as $Q * R = [Q1, Q2] * [R1; 0]$ .
out	<i>p</i>	An N-by-N matrix where the pivot matrix will be written.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

#### Notes

This routine utilizes the LAPACK routine DORGQR.

Definition at line 825 of file `linalg_factor.f90`.

4.6.2.8 subroutine, public `linalg_factor::lu_factor` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *a*, `integer(i32)`, `dimension(:)`, `intent(out)` *ipvt*, `class(errors)`, `intent(inout)`, optional, target *err* )

Computes the LU factorization of an M-by-N matrix.

#### Parameters

in, out	<i>a</i>	On input, the M-by-N matrix on which to operate. On output, the LU factored matrix in the form $[L \backslash U]$ where the unit diagonal elements of L are not stored.
---------	----------	---

## Parameters

out	<i>ipvt</i>	An MIN(M, N)-element array used to track row-pivot operations. The array stored pivot information such that row <i>l</i> is interchanged with row IPVT( <i>l</i> ).
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>ipvt</i> is not sized appropriately.</li> <li>LA_SINGULAR_MATRIX_ERROR: Occurs as a warning if <i>a</i> is found to be singular.</li> </ul>

## Usage

To solve a system of *N* equations of *N* unknowns using LU factorization, the following code will suffice.

```
! Solve the system: A*X = B, where A is an N-by-N matrix, and B and X are
! N-by-NRHS in size.

! Variables
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b

! Define the array used to track row pivots.
integer(i32), dimension(n) :: pvt

! Initialize A and B...

! Compute the LU factorization of A. On output, A contains [L\U].
call lu_factor(a, pvt)

! Solve A*X = B for X - Note: X overwrites B.
call solve_lu(a, pvt, b)
```

## Notes

This routine utilizes the LAPACK routine DGETRF.

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 127 of file linalg\_factor.f90.

**4.6.2.9** subroutine linalg\_factor::mult\_qr\_mtx ( logical, intent(in) *lside*, logical, intent(in) *trans*, real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:), intent(in) *tau*, real(dp), dimension(:,:), intent(inout) *c*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Multiplies a general matrix by the orthogonal matrix *Q* from a QR factorization such that:  $C = \text{op}(Q) * C$ , or  $C = C * \text{op}(Q)$ .

## Parameters

in	<i>lside</i>	Set to true to apply <i>Q</i> or $Q^{**T}$ from the left; else, set to false to apply <i>Q</i> or $Q^{**T}$ from the right.
in	<i>trans</i>	Set to true to apply $Q^{**T}$ ; else, set to false.

## Parameters

in	<i>a</i>	On input, an LDA-by-K matrix containing the elementary reflectors output from the QR factorization. If <i>lside</i> is set to true, LDA = M, and $M \geq K \geq 0$ ; else, if <i>lside</i> is set to false, LDA = N, and $N \geq K \geq 0$ . Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of each elementary reflector defined in <i>a</i> .
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the product of the orthogonal matrix Q and the original matrix C.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORMQR.

Definition at line 927 of file `linalg_factor.f90`.

**4.6.2.10** `subroutine linalg_factor::mult_qr_vec ( logical, intent(in) trans, real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :), intent(in) tau, real(dp), dimension(:, :), intent(inout) c, real(dp), dimension(:, :), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Multiplies a vector by the orthogonal matrix Q from a QR factorization such that:  $C = \text{op}(Q) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Q^{**T}$ ; else, set to false.
in	<i>a</i>	On input, an M-by-K matrix containing the elementary reflectors output from the QR factorization. Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of each elementary reflector defined in <i>a</i> .
in, out	<i>c</i>	On input, the M-element vector C. On output, the product of the orthogonal matrix Q and the original vector C.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.

## Parameters

out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
-----	------------	--

## Notes

This routine is based upon the LAPACK routine DORM2R.

Definition at line 1054 of file `linalg_factor.f90`.

**4.6.2.11** subroutine `linalg_factor::mult_rz_mtx` ( logical, intent(in) *lside*, logical, intent(in) *trans*, integer(i32), intent(in) *l*, real(dp), dimension(:, :), intent(inout) *a*, real(dp), dimension(:), intent(in) *tau*, real(dp), dimension(:, :), intent(inout) *c*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [*private*]

Multiplies a general matrix by the orthogonal matrix *Z* from an RZ factorization such that:  $C = \text{op}(Z) * C$ , or  $C = C * \text{op}(Z)$ .

## Parameters

in	<i>lside</i>	Set to true to apply <i>Z</i> or <i>Z**T</i> from the left; else, set to false to apply <i>Z</i> or <i>Z**T</i> from the right.
in	<i>trans</i>	Set to true to apply <i>Z**T</i> ; else, set to false.
in	<i>l</i>	The number of columns in matrix <i>a</i> containing the meaningful part of the Householder vectors. If <i>lside</i> is true, $M \geq L \geq 0$ ; else, if <i>lside</i> is false, $N \geq L \geq 0$ .
in, out	<i>a</i>	On input the K-by-LTA matrix <i>Z</i> , where $LTA = M$ if <i>lside</i> is true; else, $LTA = N$ if <i>lside</i> is false. The <i>l</i> -th row must contain the Householder vector in the last <i>k</i> rows. Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of the elementary reflectors, where $M \geq K \geq 0$ if <i>lside</i> is true; else, $N \geq K \geq 0$ if <i>lside</i> is false.
in, out	<i>c</i>	On input, the M-by-N matrix <i>C</i> . On output, the product of the orthogonal matrix <i>Z</i> and the original matrix <i>C</i> .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
Generated by Doxygen		

## Notes

This routine utilizes the LAPACK routine DORMRZ.

Definition at line 1745 of file linalg\_factor.f90.

**4.6.2.12** `subroutine linalg_factor::mult_rz_vec ( logical, intent(in) trans, integer(i32), intent(in) l, real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:), intent(in) tau, real(dp), dimension(:), intent(inout) c, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Multiplies a vector by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Z^{**T}$ ; else, set to false.
in	<i>l</i>	The number of columns in matrix <i>a</i> containing the meaningful part of the Householder vectors. If <i>lside</i> is true, $M \geq L \geq 0$ ; else, if <i>lside</i> is false, $N \geq L \geq 0$ .
in, out	<i>a</i>	On input the K-by-LTA matrix Z, where LTA = M if <i>lside</i> is true; else, LTA = N if <i>lside</i> is false. The l-th row must contain the Householder vector in the last k rows. Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of the elementary reflectors, where $M \geq K \geq 0$ if <i>lside</i> is true; else, $N \geq K \geq 0$ if <i>lside</i> is false.
in, out	<i>c</i>	On input, the M-element array C. On output, the product of the orthogonal matrix Z and the original array C.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORMRZ.

Definition at line 1887 of file linalg\_factor.f90.

**4.6.2.13** `subroutine linalg_factor::qr_factor_no_pivot ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:), intent(out) tau, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Computes the QR factorization of an M-by-N matrix without pivoting.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <i>tau</i> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>tau</i> or <i>work</i> are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Remarks

QR factorization without pivoting is best suited to solving an overdetermined system in least-squares terms, or to solve a normally defined system. To solve an underdetermined system, it is recommended to use either LQ factorization, or a column-pivoting based QR factorization.

## Usage

To solve a system of M equations of N unknowns using QR factorization, the following code will suffice assuming  $M \geq N$ .

```
! Solve the system: A*X = B in a least-squares sense, where A is an
! M-by-N matrix, B is an M-by-NRHS matrix, and X is an N-by-NRHS matrix.

! Variables
real(dp), dimension(m, n) :: a
real(dp), dimension(m, nrhs) :: b, qtb
real(dp), dimension(n, nrhs) :: x
real(dp), dimension(n) :: tau
real(dp), dimension(m, m) :: q

! Initialize A and B...

! Compute the QR factorization. We're intentionally not forming the full
! Q matrix, but instead storing it in terms of its elementary reflector
! components in the sub-diagonal portions of A, and the corresponding
! scalar factors in TAU. Additionally, we'll let the algorithm allocate
! it's own workspace array; therefore, the call to factor A is:
call qr_factor(a, tau)

! Solve A*X = B for X. The first N rows of B are used to store X.
call solve_qr(a, tau, b)

! Also note, we could form Q and R explicitly. Then solution of the
! system of equations can be found. First we form Q and R.
call form_qr(a, tau, q) ! Forms Q, and R is stored in A

! Since we now have Q and R, we seek a solution to the equation:
! Q*R*X = B, but Q is an orthogonal matrix (i.e. Q**T = inv(Q)).
! Then: R*X = Q**T * B, and R is upper triangular; therefore, back
! substitution will suffice for a solution procedure.
!
! Next, compute Q**T * B, and store in QTB.
call mtv_mult(.true., .false., 1.0d0, q, b, 0.0d0, qtb)

! Copy the first N rows of Q**T * B into X for the solution process.
```

```

! Notice, only the first N rows are needed as rows N+1:M are all zero in
! matrix R.
x = qtb(1:n,nrhs)

! Compute the solution and store in X
call solve_triangular_system(.true., .true., .false., .true., 1.0d0, &
    a(1:n,1:n), x)

```

## Notes

This routine utilizes the LAPACK routine DGEQRF.

Definition at line 425 of file linalg\_factor.f90.

**4.6.2.14** subroutine linalg\_factor::qr\_factor\_pivot ( real(dp), dimension(:, :), intent(inout) *a*, real(dp), dimension(:), intent(out) *tau*, integer(i32), dimension(:), intent(inout) *jpvt*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Computes the QR factorization of an M-by-N matrix with column pivoting such that  $A * P = Q * R$ .

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <i>tau</i> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
in, out	<i>jpvt</i>	On input, an N-element array that if JPVT(I) .ne. 0, the I-th column of A is permuted to the front of $A * P$ ; if JPVT(I) = 0, the I-th column of A is a free column. On output, if JPVT(I) = K, then the I-th column of $A * P$ was the K-th column of A.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Usage

To solve a system of M equations of N unknowns using QR factorization, the following code will suffice for any M and N.

```

! Solve the least-squares (M >= N), or the underdetermined (M < N)
! problem A*X = B, where A is an M-by-N matrix, B is an M-by-NRHS matrix,
! and X is an N-by-NRHS matrix. In the underdetermined case, or the
! case where the rank of matrix A is less than N, the solution obtained
! contains the fewest possible non-zero entries.

! Variables

```



```

real(dp), dimension(m, n) :: a
real(dp), dimension(n, nrhs) :: b
real(dp), dimension(k) :: tau ! k = min(m, n)
real(dp), dimension(m, m) :: q
real(dp), dimension(n, n) :: p
integer(i32), dimension(n) :: pvt

! Initialize A and B...

! Allow all columns to be free.
pvt = 0

! Compute the QR factorization. We're intentionally not forming the full
! Q matrix, but instead storing it in terms of its elementary reflector
! components in the sub-diagonal portions of A, and the corresponding
! scalar factors in TAU. Additionally, we'll let the algorithm allocate
! it's own workspace array; therefore, the call to factor A is:
call qr_factor(a, tau, pvt)

! Solve A*X = B for X. If M > N, the first N rows of B are used to store
! X. If M < N, the input matrix B must be N-by-NRHS, and only the first
! M rows are used for the actual matrix B. The remaining N-M rows
! can contain whatever as they are not referenced until they are
! overwritten by the N-by-NRHS solution matrix X.
call solve_qr(a, tau, pvt, b)

! Notice, if the explicit Q matrix from the factorization is desired,
! the form_qr routine works similarly as in the no-pivot case;
! however, the permutation matrix P is also constructed. The call would
! be as follows. Also, as with the no-pivot algorithm, the matrix R is
! stored in matrix A.
call form_qr(a, tau, pvt, q, p)

! Solution can proceed as per typical, but with a full Q matrix. Also
! note, the problem is of the form: A*P = Q*R. Solution is straight
! forward, as with the no-pivot case; however, if M < N, then R is upper
! trapezoidal, and must be appropriately partitioned to solve. The rank
! of matrix r should be considered when applying the partition.

```

## Notes

This routine utilizes the LAPACK routine DGEQP3.

Definition at line 579 of file linalg\_factor.f90.

**4.6.2.15** subroutine, public linalg\_factor::qr\_rank1\_update ( real(dp), dimension(:, :), intent(inout) *q*, real(dp), dimension(:, :), intent(inout) *r*, real(dp), dimension(:), intent(inout) *u*, real(dp), dimension(:), intent(inout) *v*, real(dp), dimension(:), intent(out), optional, target *work*, class(errors), intent(inout), optional, target *err* )

Computes the rank 1 update to an M-by-N QR factored matrix A ( $M \geq N$ ) where  $A = Q * R$ , and  $A1 = A + U * V^{**T}$  such that  $A1 = Q1 * R1$ .

## Parameters

in, out	<i>q</i>	On input, the original M-by-K orthogonal matrix Q. On output, the updated matrix Q1.
in, out	<i>r</i>	On input, the M-by-N matrix R. On output, the updated matrix R1.
in, out	<i>u</i>	On input, the M-element U update vector. On output, the original content of the array is overwritten.
in, out	<i>v</i>	On input, the N-element V update vector. On output, the original content of the array is overwritten.
out	<i>work</i>	An optional argument that if supplied prevents local memory allocation. If provided, the array must have at least 2*K elements.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
-----	-----	--

## Remarks

Notice,  $K$  must either be equal to  $M$ , or to  $N$ . In the event that  $K = N$ , only the submatrix  $Qa$  is updated. This is appropriate as the QR factorization for an overdetermined system can be written as follows:

$$A = Q * R = [Qa, Qb] * \begin{bmatrix} Ra \\ 0 \end{bmatrix}$$

Note:  $Ra$  is upper triangular of dimension  $N$ -by- $N$ .

## Notes

This routine utilizes the QRUPDATE routine DQR1UP.

## See Also

[Source](#)

Definition at line 1180 of file `linalg_factor.f90`.

**4.6.2.16** subroutine, public `linalg_factor::rz_factor` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *a*, `real(dp)`, `dimension(:)`, `intent(out)` *tau*, `real(dp)`, `dimension(:)`, `intent(out)`, optional, `target` *work*, `integer(i32)`, `intent(out)`, optional *olwork*, `class(errors)`, `intent(inout)`, optional, `target` *err* )

Factors an upper trapezoidal matrix by means of orthogonal transformations such that  $A = R * Z = (R \ 0) * Z$ .  $Z$  is an orthogonal matrix of dimension  $N$ -by- $N$ , and  $R$  is an  $M$ -by- $M$  upper triangular matrix.

## Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ upper trapezoidal matrix to factor. On output, the leading $M$ -by- $M$ upper triangular part of the matrix contains the upper triangular matrix $R$ , and elements $N-L+1$ to $N$ of the first $M$ rows of $A$ , with the array <code>tau</code> , represent the orthogonal matrix $Z$ as a product of $M$ elementary reflectors.
out	<i>tau</i>	An $M$ -element array used to store the scalar factors of the elementary reflectors.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
-----	-----	--

## Further Details

The factorization is obtained by Householder's method. The  $k$ th transformation matrix,  $Z(k)$ , which is used to introduce zeros into the  $(m - k + 1)$ th row of  $A$ , is given in the form

$$Z(k) = \begin{pmatrix} I & 0 \\ 0 & T(k) \end{pmatrix},$$

where

$$T(k) = I - \tau u(k) u(k)^* T, \quad u(k) = \begin{pmatrix} 1 \\ 0 \\ z(k) \end{pmatrix},$$

$\tau$  is a scalar and  $z(k)$  is an  $l$  element vector.  $\tau$  and  $z(k)$  are chosen to annihilate the elements of the  $k$ th row of  $A_2$ .

The scalar  $\tau$  is returned in the  $k$ th element of  $TAU$  and the vector  $u(k)$  in the  $k$ th row of  $A_2$ , such that the elements of  $z(k)$  are in  $a(k, l + 1)$ , ...,  $a(k, n)$ . The elements of  $R$  are returned in the upper triangular part of  $A_1$ .

$Z$  is given by

$$Z = Z(1) * Z(2) * \dots * Z(m).$$

## Notes

This routine is based upon the LAPACK routine DTZRZF.

## See Also

- [LAPACK Users Manual](#)

Definition at line 1633 of file `linalg_factor.f90`.

**4.6.2.17** `subroutine, public linalg_factor::svd ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(out) s, real(dp), dimension(:, :), intent(out), optional u, real(dp), dimension(:, :), intent(out), optional vt, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err )`

Computes the singular value decomposition of a matrix  $A$ . The SVD is defined as:  $A = U * S * V^* T$ , where  $U$  is an  $M$ -by- $M$  orthogonal matrix,  $S$  is an  $M$ -by- $N$  diagonal matrix, and  $V$  is an  $N$ -by- $N$  orthogonal matrix.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to factor. The matrix is overwritten on output.
out	<i>s</i>	A MIN(M, N)-element array containing the singular values of <i>a</i> sorted in descending order.
out	<i>u</i>	An optional argument, that if supplied, is used to contain the orthogonal matrix U from the decomposition. The matrix U contains the left singular vectors, and can be either M-by-M (all left singular vectors are computed), or M-by-MIN(M,N) (only the first MIN(M, N) left singular vectors are computed).
out	<i>vt</i>	An optional argument, that if supplied, is used to contain the transpose of the N-by-N orthogonal matrix V. The matrix V contains the right singular vectors.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

## Usage

```

! Decompose matrix the M-by-N matrix A such that A = U * S * V**T with
! M >= N.

! Variables
real(dp), dimension(m, n) :: a
real(dp), dimension(m, m) :: u
real(dp), dimension(n, n) :: vt
real(dp), dimension(n) :: s

! Initialize A...

! Compute the SVD of A. On output, S contains the MIN(M,N) singular
! values of A in descending order, U contains the left singular vectors
! (one per column), and VT contains the right singular vectors (one per
! row).
call svd(a, s, u, vt)

! Note: If M > N, then we can make U M-by-N, and compute the N
! left singular vectors of A, as there are at most N singular values
! of A. Also, if M < N, then there are at most M singular values of A,
! and as such, the length of the array s should be m.

```

## Notes

This routine utilizes the LAPACK routine DGESVD.

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 2046 of file `linalg_factor.f90`.

## 4.7 linalg\_solve Module Reference

### linalg\_solve

#### Data Types

- interface [solve\\_cholesky](#)  
*Solves a system of Cholesky factored equations.*
- interface [solve\\_least\\_squares](#)  
*Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns.*
- interface [solve\\_least\\_squares\\_full](#)  
*Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns, but uses a full orthogonal factorization of the system.*
- interface [solve\\_least\\_squares\\_svd](#)  
*Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .*
- interface [solve\\_lu](#)  
*Solves a system of LU-factored equations.*
- interface [solve\\_qr](#)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns.*
- interface [solve\\_triangular\\_system](#)  
*Solves a triangular system of equations.*

#### Functions/Subroutines

- subroutine [solve\\_tri\\_mtx](#) (lside, upper, trans, nounit, alpha, a, b, err)  
*Solves one of the matrix equations:  $op(A) * X = alpha * B$ , or  $X * op(A) = alpha * B$ , where  $A$  is a triangular matrix.*
- subroutine [solve\\_tri\\_vec](#) (upper, trans, nounit, a, x, err)  
*Solves the system of equations:  $op(A) * X = B$ , where  $A$  is a triangular matrix.*
- subroutine [solve\\_lu\\_mtx](#) (a, ipvt, b, err)  
*Solves a system of LU-factored equations.*
- subroutine [solve\\_lu\\_vec](#) (a, ipvt, b, err)  
*Solves a system of LU-factored equations.*
- subroutine [solve\\_qr\\_no\\_pivot\\_mtx](#) (a, tau, b, work, olwork, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where  $M \geq N$ .*
- subroutine [solve\\_qr\\_no\\_pivot\\_vec](#) (a, tau, b, work, olwork, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where  $M \geq N$ .*
- subroutine [solve\\_qr\\_pivot\\_mtx](#) (a, tau, jpvt, b, work, olwork, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where the QR factorization made use of column pivoting.*
- subroutine [solve\\_qr\\_pivot\\_vec](#) (a, tau, jpvt, b, work, olwork, err)  
*Solves a system of  $M$  QR-factored equations of  $N$  unknowns where the QR factorization made use of column pivoting.*
- subroutine [solve\\_cholesky\\_mtx](#) (upper, a, b, err)  
*Solves a system of Cholesky factored equations.*
- subroutine [solve\\_cholesky\\_vec](#) (upper, a, b, err)  
*Solves a system of Cholesky factored equations.*
- subroutine, public [mtx\\_inverse](#) (a, iwork, work, olwork, err)  
*Computes the inverse of a square matrix.*
- subroutine, public [mtx\\_pinverse](#) (a, ainvt, tol, work, olwork, err)

Computes the Moore-Penrose pseudo-inverse of a  $M$ -by- $N$  matrix using the singular value decomposition of the matrix.

- subroutine `solve_least_squares_mtx` (`a`, `b`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.
- subroutine `solve_least_squares_vec` (`a`, `b`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.
- subroutine `solve_least_squares_mtx_pvt` (`a`, `b`, `ipvt`, `arnk`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a complete orthogonal factorization of matrix  $A$ .
- subroutine `solve_least_squares_vec_pvt` (`a`, `b`, `ipvt`, `arnk`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a complete orthogonal factorization of matrix  $A$ .
- subroutine `solve_least_squares_mtx_svd` (`a`, `b`, `arnk`, `s`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .
- subroutine `solve_least_squares_vec_svd` (`a`, `b`, `arnk`, `s`, `work`, `olwork`, `err`)  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

#### 4.7.1 Detailed Description

##### `linalg_solve`

##### Purpose

Provides a set of routines for solving systems of linear equations.

#### 4.7.2 Function/Subroutine Documentation

- 4.7.2.1 subroutine, public `linalg_solve::mtx_inverse` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` `a`, `integer(i32)`, `dimension(:)`, `intent(out)`, optional, target `iwork`, `real(dp)`, `dimension(:)`, `intent(out)`, optional, target `work`, `integer(i32)`, `intent(out)`, optional `olwork`, `class(errors)`, `intent(inout)`, optional, target `err` )

Computes the inverse of a square matrix.

##### Parameters

in, out	<code>a</code>	On input, the $N$ -by- $N$ matrix to invert. On output, the inverted matrix.
out	<code>iwork</code>	An optional $N$ -element integer workspace array.
out	<code>work</code>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<code>olwork</code>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <code>a</code> is not square. Will also occur if incorrectly sized workspace arrays are provided.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_SINGULAR_MATRIX_ERROR</code>: Occurs if the input matrix is singular.</li> </ul>
-----	-----	---

## Usage

```

! The following example illustrates how to solve a system of linear
! equations by matrix inversion. Notice, this is not a preferred
! solution technique (use LU factorization instead), but is merely a
! means of illustrating how to compute the inverse of a square matrix.

! Variables
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b, x

! Initialize A and B...

! Compute the inverse of A. The inverse will overwrite the original
! matrix.
call mtx_inverse(a)

! Solve A*X = B as X = inv(A) * B.
x = matmul(a, b)

```

## Notes

This routine utilizes the LAPACK routines DGETRF to perform an LU factorization of the matrix, and DGETRI to invert the LU factored matrix.

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 1240 of file linalg\_solve.f90.

**4.7.2.2** subroutine, public `linalg_solve::mtx_pinverse` ( `real(dp), dimension(:, :), intent(inout) a`, `real(dp), dimension(:, :), intent(out) ain`, `real(dp), intent(in), optional tol`, `real(dp), dimension(:, :), intent(out), optional target work`, `integer(i32), intent(out), optional olwork`, `class(errors), intent(inout), optional, target err` )

Computes the Moore-Penrose pseudo-inverse of a M-by-N matrix using the singular value decomposition of the matrix.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to invert. The matrix is overwritten on output.
out	<i>ain</i>	The N-by-M matrix where the pseudo-inverse of <i>a</i> will be written.

## Parameters

in	<i>tol</i>	An optional input, that if supplied, overrides the default tolerance on singular values such that singular values less than this tolerance are forced to have a reciprocal of zero, as opposed to $1/S(I)$ . The default tolerance is: $\text{MAX}(M, N) * \text{EPS} * \text{MAX}(S)$ . If the supplied value is less than a value that causes an overflow, the tolerance reverts back to its default value, and the operation continues; however, a warning message is issued.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

## Usage

```

! Use the pseudo-inverse to obtain a least-squares solution to the
! overdetermined problem  $A \cdot X = B$ , where A is an M-by-N matrix ( $M \geq N$ ),
! B is an M-by-NRHS matrix, and X is an N-by-NRHS matrix.

! Variables
real(dp), dimension(m, n) :: a
real(dp), dimension(n, m) :: ainvs
real(dp), dimension(m, nrhs) :: b
real(dp), dimension(n, nrhs) :: x

! Initialize A, and B...

! Compute the pseudo-inverse of A. Let the subroutine allocate its
! own workspace array.
call mtv_pinverse(a, ainvs)

! Compute  $X = AINV * B$  to obtain the solution.
x = matmul(ainvs, b)

```

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)
- [MathWorks](#)

Definition at line 1400 of file `linalg_solve.f90`.

**4.7.2.3** `subroutine linalg_solve::solve_cholesky_mtx ( logical, intent(in) upper, real(dp), dimension(:, :), intent(in) a, real(dp), dimension(:, :), intent(inout) b, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of Cholesky factored equations.



## Parameters

in	<i>upper</i>	Set to true if the original matrix A was factored such that $A = U^{**T} * U$ ; else, set to false if the factorization of A was $A = L^{**T} * L$ .
in	<i>a</i>	The N-by-N Cholesky factored matrix.
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side matrix B. On output, the solution matrix X.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DPOTRS.

Definition at line 1073 of file linalg\_solve.f90.

```
4.7.2.4 subroutine linalg_solve::solve_cholesky_vec ( logical, intent(in) upper, real(dp), dimension(:, :), intent(in) a, real(dp),
dimension(:, :), intent(inout) b, class(errors), intent(inout), optional, target err ) [private]
```

Solves a system of Cholesky factored equations.

## Parameters

in	<i>upper</i>	Set to true if the original matrix A was factored such that $A = U^{**T} * U$ ; else, set to false if the factorization of A was $A = L^{**T} * L$ .
in	<i>a</i>	The N-by-N Cholesky factored matrix.
in, out	<i>b</i>	On input, the N-element right-hand-side vector B. On output, the solution vector X.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DPOTRS.

Definition at line 1139 of file linalg\_solve.f90.

```
4.7.2.5 subroutine linalg_solve::solve_least_squares_mtx ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :),
intent(inout) b, real(dp), dimension(:, :), intent(out), optional, target work, integer(i32), intent(out), optional olwork,
class(errors), intent(inout), optional, target err ) [private]
```

Solves the overdetermined or underdetermined system ( $A * X = B$ ) of M equations of N unknowns using a QR or LQ factorization of the matrix A. Notice, it is assumed that matrix A has full rank.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix A. On output, if $M \geq N$ , the QR factorization of A in the form as output by <code>qr_factor</code> ; else, if $M < N$ , the LQ factorization of A.
in, out	<i>b</i>	If $M \geq N$ , the M-by-NRHS matrix B. On output, the first N rows contain the N-by-NRHS solution matrix X. If $M < N$ , an N-by-NRHS matrix with the first M rows containing the matrix B. On output, the N-by-NRHS solution matrix X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_INVALID_OPERATION_ERROR</code>: Occurs if <i>a</i> is not of full rank.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELS.

Definition at line 1568 of file `linalg_solve.f90`.

```
4.7.2.6 subroutine linalg_solve::solve_least_squares_mtx_pvt ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :),
intent(inout) b, integer(i32), dimension(:), intent(inout), optional, target ipvt, integer(i32), intent(out), optional arnk,
real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors),
intent(inout), optional, target err ) [private]
```

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of M equations of N unknowns using a complete orthogonal factorization of matrix A.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix A. On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the M-by-NRHS matrix B. On output, the first N rows contain the N-by-NRHS solution matrix X. If $M < N$ , an N-by-NRHS matrix with the first M rows containing the matrix B. On output, the N-by-NRHS solution matrix X.
out	<i>ipvt</i>	An optional input that on input, an N-element array that if <code>IPVT(I) .ne. 0</code> , the I-th column of A is permuted to the front of $A \cdot P$ ; if <code>IPVT(I) = 0</code> , the I-th column of A is a free column. On output, if <code>IPVT(I) = K</code> , then the I-th column of $A \cdot P$ was the K-th column of A. If not supplied, memory is allocated internally, and IPVT is set to all zeros such that all columns are treated as free.
out	<i>arnk</i>	An optional output, that if provided, will return the rank of <i>a</i> .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .

## Parameters

out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSY.

Definition at line 1786 of file linalg\_solve.f90.

```
4.7.2.7 subroutine linalg_solve::solve_least_squares_mtx_svd ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:,:),
intent(inout) b, integer(i32), intent(out), optional arnk, real(dp), dimension(:), intent(out) s, real(dp), dimension(:),
intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target
err ) [private]
```

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

## Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the $M$ -by- $N$ -RHS matrix $B$ . On output, the first $N$ rows contain the $N$ -by- $N$ -RHS solution matrix $X$ . If $M < N$ , an $N$ -by- $N$ -RHS matrix with the first $M$ rows containing the matrix $B$ . On output, the $N$ -by- $N$ -RHS solution matrix $X$ .
out	<i>arnk</i>	An optional output, that if provided, will return the rank of $a$ .
out	<i>s</i>	A $\text{MIN}(M, N)$ -element array that on output contains the singular values of $a$ in descending order. Notice, the condition number of $a$ can be determined by $S(1) / S(\text{MIN}(M, N))$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>
Generated by Doxygen		

## Notes

This routine utilizes the LAPACK routine DGELSS.

Definition at line 2076 of file linalg\_solve.f90.

```
4.7.2.8 subroutine linalg_solve::solve_least_squares_vec ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:),
intent(inout) b, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork,
class(errors), intent(inout), optional, target err ) [private]
```

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.

## Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, if $M \geq N$ , the QR factorization of $A$ in the form as output by <code>qr_factor</code> ; else, if $M < N$ , the LQ factorization of $A$ .
in, out	<i>b</i>	If $M \geq N$ , the $M$ -element array $B$ . On output, the first $N$ elements contain the $N$ -element solution array $X$ . If $M < N$ , an $N$ -element array with the first $M$ elements containing the array $B$ . On output, the $N$ -element solution array $X$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_INVALID_OPERATION_ERROR: Occurs if <i>a</i> is not of full rank.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELS.

Definition at line 1674 of file linalg\_solve.f90.

```
4.7.2.9 subroutine linalg_solve::solve_least_squares_vec_pvt ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:),
intent(inout) b, integer(i32), dimension(:), intent(inout), optional, target ipvt, integer(i32), intent(out), optional arnk,
real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors),
intent(inout), optional, target err ) [private]
```

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a complete orthogonal factorization of matrix  $A$ .

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix A. On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the M-element array B. On output, the first N elements contain the N-element solution array X. If $M < N$ , an N-element array with the first M elements containing the array B. On output, the N-element solution array X.
out	<i>ipvt</i>	An optional input that on input, an N-element array that if $IPVT(l) \neq 0$ , the l-th column of A is permuted to the front of $A * P$ ; if $IPVT(l) = 0$ , the l-th column of A is a free column. On output, if $IPVT(l) = K$ , then the l-th column of $A * P$ was the K-th column of A. If not supplied, memory is allocated internally, and IPVT is set to all zeros such that all columns are treated as free.
out	<i>arnk</i>	An optional output, that if provided, will return the rank of <i>a</i> .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSY.

Definition at line 1932 of file linalg\_solve.f90.

**4.7.2.10** `subroutine linalg_solve::solve_least_squares_vec_svd ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:), intent(inout) b, integer(i32), intent(out), optional arnk, real(dp), dimension(:), intent(out) s, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves the overdetermined or underdetermined system ( $A * X = B$ ) of M equations of N unknowns using a singular value decomposition of matrix A.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix A. On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the M-by-NRHS matrix B. On output, the first N rows contain the N-by-NRHS solution matrix X. If $M < N$ , an N-by-NRHS matrix with the first M rows containing the matrix B. On output, the N-by-NRHS solution matrix X.
out	<i>arnk</i>	An optional output, that if provided, will return the rank of <i>a</i> .
out	<i>s</i>	A $\min(M, N)$ -element array that on output contains the singular values of <i>a</i> in descending order. Notice, the condition number of <i>a</i> can be determined by $S(1) / S(\min(M, N))$ .

## Parameters

out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSS.

Definition at line 2204 of file `linalg_solve.f90`.

**4.7.2.11** `subroutine linalg_solve::solve_lu_mtx ( real(dp), dimension(:, :), intent(in) a, integer(i32), dimension(:), intent(in) ipvt, real(dp), dimension(:, :), intent(inout) b, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of LU-factored equations.

## Parameters

in	<i>a</i>	The N-by-N LU factored matrix as output by <code>lu_factor</code> .
in	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side matrix. On output, the N-by-NRHS solution matrix.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

The routine is based upon the LAPACK routine DGETRS.

Definition at line 351 of file `linalg_solve.f90`.

4.7.2.12 subroutine `linalg_solve::solve_lu_vec` ( `real(dp)`, `dimension(:, :)`, `intent(in)` *a*, `integer(i32)`, `dimension(:)`, `intent(in)` *ipvt*, `real(dp)`, `dimension(:)`, `intent(inout)` *b*, `class(errors)`, `intent(inout)`, optional, `target err` ) [`private`]

Solves a system of LU-factored equations.

#### Parameters

in	<i>a</i>	The N-by-N LU factored matrix as output by <code>lu_factor</code> .
in	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
in, out	<i>b</i>	On input, the N-element right-hand-side array. On output, the N-element solution array.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

#### Notes

The routine is based upon the LAPACK routine DGETRS.

Definition at line 412 of file `linalg_solve.f90`.

4.7.2.13 subroutine `linalg_solve::solve_qr_no_pivot_mtx` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *a*, `real(dp)`, `dimension(:)`, `intent(in)` *tau*, `real(dp)`, `dimension(:, :)`, `intent(inout)` *b*, `real(dp)`, `dimension(:)`, `intent(out)`, optional, `target work`, `integer(i32)`, `intent(out)`, optional *olwork*, `class(errors)`, `intent(inout)`, optional, `target err` ) [`private`]

Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .

#### Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are restored. Notice, M must be greater than or equal to N.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>b</i>	On input, the M-by-NRHS right-hand-side matrix. On output, the first N columns are overwritten by the solution matrix X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon a subset of the LAPACK routine DGELS.

Definition at line 487 of file linalg\_solve.f90.

**4.7.2.14** `subroutine linalg_solve::solve_qr_no_pivot_vec ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(in) tau, real(dp), dimension(:), intent(inout) b, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of  $M$  QR-factored equations of  $N$  unknowns where  $M \geq N$ .

## Parameters

in	<i>a</i>	On input, the $M$ -by- $N$ QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are restored. Notice, $M$ must be greater than or equal to $N$ .
in	<i>tau</i>	A $\min(M, N)$ -element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>b</i>	On input, the $M$ -element right-hand-side vector. On output, the first $N$ elements are overwritten by the solution vector $X$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon a subset of the LAPACK routine DGELS.

Definition at line 602 of file linalg\_solve.f90.

**4.7.2.15** `subroutine linalg_solve::solve_qr_pivot_mtx ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(in) tau, integer(i32), dimension(:), intent(in) jpvt, real(dp), dimension(:, :), intent(inout) b, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of  $M$  QR-factored equations of  $N$  unknowns where the QR factorization made use of column pivoting.



## Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are altered.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>jpvt</i>	An N-element array, as output by <code>qr_factor</code> , used to track the column pivots.
in	<i>b</i>	On input, the MAX(M, N)-by-NRHS matrix where the first M rows contain the right-hand-side matrix B. On output, the first N rows are overwritten by the solution matrix X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon a subset of the LAPACK routine DGELSY.

Definition at line 715 of file `linalg_solve.f90`.

```
4.7.2.16 subroutine linalg_solve::solve_qr_pivot_vec ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(in)
tau, integer(i32), dimension(:), intent(in) jpvt, real(dp), dimension(:), intent(inout) b, real(dp), dimension(:), intent(out),
optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err )
[private]
```

Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.

## Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are altered.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>jpvt</i>	An N-element array, as output by <code>qr_factor</code> , used to track the column pivots.
in	<i>b</i>	On input, the MAX(M, N)-element array where the first M elements contain the right-hand-side vector B. On output, the first N elements are overwritten by the solution vector X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
-----	-----	--

## Notes

This routine is based upon a subset of the LAPACK routine DGELSY.

Definition at line 902 of file linalg\_solve.f90.

**4.7.2.17** subroutine linalg\_solve::solve\_tri\_mtx ( logical, intent(in) *lside*, logical, intent(in) *upper*, logical, intent(in) *trans*, logical, intent(in) *nounit*, real(dp), intent(in) *alpha*, real(dp), dimension(:,:), intent(in) *a*, real(dp), dimension(:,:), intent(inout) *b*, class(errors), intent(inout), optional, target *err* ) [private]

Solves one of the matrix equations:  $\text{op}(A) * X = \alpha * B$ , or  $X * \text{op}(A) = \alpha * B$ , where  $A$  is a triangular matrix.

## Parameters

in	<i>lside</i>	Set to true to solve $\text{op}(A) * X = \alpha * B$ ; else, set to false to solve $X * \text{op}(A) = \alpha * B$ .
in	<i>upper</i>	Set to true if $A$ is an upper triangular matrix; else, set to false if $A$ is a lower triangular matrix.
in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false if $\text{op}(A) = A$ .
in	<i>nounit</i>	Set to true if $A$ is not a unit-diagonal matrix (ones on every diagonal element); else, set to false if $A$ is a unit-diagonal matrix.
in	<i>alpha</i>	The scalar multiplier to $B$ .
in	<i>a</i>	If <i>lside</i> is true, the $M$ -by- $M$ triangular matrix on which to operate; else, if <i>lside</i> is false, the $N$ -by- $N$ triangular matrix on which to operate.
in, out	<i>b</i>	On input, the $M$ -by- $N$ right-hand-side. On output, the $M$ -by- $N$ solution.
out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>a</i> is not square, or if the sizes of <i>a</i> and <i>b</i> are not compatible.</li> </ul>

## Usage

To solve a triangular system of  $N$  equations of  $N$  unknowns  $A * X = B$ , where  $A$  is an  $N$ -by- $N$  upper triangular matrix, and  $B$  and  $X$  are  $N$ -by- $N$  matrices, the following code will suffice.

```
! Solve the system: A*X = B, where A is an upper triangular N-by-N
```

```

! matrix, and B and X are N-by-NRHS in size.

! Variables
integer(i32) :: info
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b

! Initialize A and B...

! Solve A*X = B for X - Note: X overwrites B.
call solve_triangular_system(.true., .true., .false., .true., &
    1.0d0, a, b)

```

## Notes

This routine is based upon the BLAS routine DTRSM.

Definition at line 169 of file linalg\_solve.f90.

**4.7.2.18** subroutine linalg\_solve::solve\_tri\_vec ( logical, intent(in) *upper*, logical, intent(in) *trans*, logical, intent(in) *nounit*, real(dp), dimension(:,:), intent(in) *a*, real(dp), dimension(:), intent(inout) *x*, class(errors), intent(inout), optional, target *err* ) [private]

Solves the system of equations:  $\text{op}(A) * X = B$ , where  $A$  is a triangular matrix.

## Parameters

in	<i>upper</i>	Set to true if $A$ is an upper triangular matrix; else, set to false if $A$ is a lower triangular matrix.
in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false if $\text{op}(A) = A$ .
in	<i>nounit</i>	Set to true if $A$ is not a unit-diagonal matrix (ones on every diagonal element); else, set to false if $A$ is a unit-diagonal matrix.
in	<i>a</i>	The N-by-N triangular matrix.
in, out	<i>x</i>	On input, the N-element right-hand-side array. On output, the N-element solution array.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <math>a</math> is not square, or if the sizes of <math>a</math> and <math>b</math> are not compatible.</li> </ul>

## Usage

To solve a triangular system of  $N$  equations of  $N$  unknowns  $A*X = B$ , where  $A$  is an N-by-N upper triangular matrix, and  $B$  and  $X$  are N-element arrays, the following code will suffice.

```

! Solve the system: A*X = B, where A is an upper triangular N-by-N
! matrix, and B and X are N-elements in size.

! Variables
integer(i32) :: info
real(dp), dimension(n, n) :: a
real(dp), dimension(n) :: b

! Initialize A and B...

! Solve A*X = B for X - Note: X overwrites B.
call solve_triangular_system(.true., .false., a, b)

```

## Notes

This routine is based upon the BLAS routine DTRSV.

Definition at line 275 of file linalg\_solve.f90.

## 4.8 linalg\_sorting Module Reference

### [linalg\\_sorting](#)

#### Data Types

- interface [sort](#)  
*Sorts an array.*

#### Functions/Subroutines

- subroutine [sort\\_dbl\\_array](#) (x, ascend)  
*Sorts an array.*
- subroutine [sort\\_dbl\\_array\\_ind](#) (x, ind, ascend, err)  
*Sorts an array.*
- subroutine [sort\\_cmplx\\_array](#) (x, ascend)  
*Sorts an array.*
- subroutine [sort\\_cmplx\\_array\\_ind](#) (x, ind, ascend, err)  
*Sorts an array.*
- subroutine [sort\\_eigen\\_cmplx](#) (vals, vecs, ascend, err)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*
- subroutine [sort\\_eigen\\_dbl](#) (vals, vecs, ascend, err)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*
- recursive subroutine [qsort\\_dbl\\_ind](#) (ascend, x, ind)  
*A recursive quick sort algorithm.*
- subroutine [dbl\\_partition\\_ind](#) (ascend, x, ind, marker)  
*A routine to perform the partitioning necessary for the quick sort algorithm.*
- recursive subroutine [qsort\\_cmplx](#) (ascend, x)  
*A recursive quick sort algorithm.*
- subroutine [cmplx\\_partition](#) (ascend, x, marker)  
*A routine to perform the partitioning necessary for the quick sort algorithm.*
- recursive subroutine [qsort\\_cmplx\\_ind](#) (ascend, x, ind)  
*A recursive quick sort algorithm.*
- subroutine [cmplx\\_partition\\_ind](#) (ascend, x, ind, marker)  
*A routine to perform the partitioning necessary for the quick sort algorithm.*

### 4.8.1 Detailed Description

#### [linalg\\_sorting](#)

#### Purpose

Provides sorting routines.

## 4.8.2 Function/Subroutine Documentation

4.8.2.1 subroutine `linalg_sorting::cmplx_partition` ( `logical`, `intent(in)` *ascend*, `complex(dp)`, `dimension(:)`, `intent(inout)` *x*, `integer(i32)`, `intent(out)` *marker* ) [`private`]

A routine to perform the partitioning necessary for the quick sort algorithm.

### Parameters

<code>in</code>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<code>in, out</code>	<i>x</i>	On input, the array to sort. On output, the sorted array.
<code>out</code>	<i>marker</i>	The partitioning marker.

### Remarks

As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 578 of file `linalg_sorting.f90`.

4.8.2.2 subroutine `linalg_sorting::cmplx_partition_ind` ( `logical`, `intent(in)` *ascend*, `complex(dp)`, `dimension(:)`, `intent(inout)` *x*, `integer(i32)`, `dimension(:)`, `intent(inout)` *ind*, `integer(i32)`, `intent(out)` *marker* ) [`private`]

A routine to perform the partitioning necessary for the quick sort algorithm.

### Parameters

<code>in</code>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<code>in, out</code>	<i>x</i>	On input, the array to sort. On output, the sorted array.
<code>in, out</code>	<i>ind</i>	On input, a tracking array of the same length as <i>x</i> . On output, the same array, but shuffled to match the sorting order of <i>x</i> .
<code>out</code>	<i>marker</i>	The partitioning marker.

### Remarks

As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 703 of file `linalg_sorting.f90`.

**4.8.2.3** subroutine `linalg_sorting::dbl_partition_ind` ( logical, intent(in) *ascend*, real(dp), dimension(:), intent(inout) *x*, integer(i32), dimension(:), intent(inout) *ind*, integer(i32), intent(out) *marker* ) [private]

A routine to perform the partitioning necessary for the quick sort algorithm.

#### Parameters

in	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, a tracking array of the same length as <i>x</i> . On output, the same array, but shuffled to match the sorting order of <i>x</i> .
out	<i>marker</i>	The partitioning marker.

#### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95)

Definition at line 452 of file `linalg_sorting.f90`.

**4.8.2.4** recursive subroutine `linalg_sorting::qsort_cmplx` ( logical, intent(in) *ascend*, complex(dp), dimension(:), intent(inout) *x* ) [private]

A recursive quick sort algorithm.

#### Parameters

in	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.

#### Remarks

As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

#### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95)

Definition at line 545 of file `linalg_sorting.f90`.

**4.8.2.5** recursive subroutine `linalg_sorting::qsort_cmplx_ind` ( logical, intent(in) *ascend*, complex(dp), dimension(:), intent(inout) *x*, integer(i32), dimension(:), intent(inout) *ind* ) [private]

A recursive quick sort algorithm.

## Parameters

<i>in</i>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<i>in, out</i>	<i>x</i>	On input, the array to sort. On output, the sorted array.
<i>in, out</i>	<i>ind</i>	On input, a tracking array of the same length as <i>x</i> . On output, the same array, but shuffled to match the sorting order of <i>x</i> .

## Remarks

As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

## Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95)

Definition at line 666 of file linalg\_sorting.f90.

**4.8.2.6** recursive subroutine `linalg_sorting::qsort_dbl_ind` ( *logical*, intent(in) *ascend*, *real(dp)*, dimension(:), intent(inout) *x*, integer(i32), dimension(:), intent(inout) *ind* ) [private]

A recursive quick sort algorithm.

## Parameters

<i>in</i>	<i>ascend</i>	Set to true to sort in ascending order; else, false to sort in descending order.
<i>in, out</i>	<i>x</i>	On input, the array to sort. On output, the sorted array.
<i>in, out</i>	<i>ind</i>	On input, a tracking array of the same length as <i>x</i> . On output, the same array, but shuffled to match the sorting order of <i>x</i> .

## Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 419 of file linalg\_sorting.f90.

**4.8.2.7** subroutine `linalg_sorting::sort_cmplx_array` ( *complex(dp)*, dimension(:), intent(inout) *x*, *logical*, intent(in), optional *ascend* ) [private]

Sorts an array.

## Parameters

<i>in, out</i>	<i>x</i>	On input, the array to sort. On output, the sorted array.
<i>in</i>	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.

**Remarks**

This routine utilizes a quick sort algorithm. As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

**Notes**

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 156 of file linalg\_sorting.f90.

**4.8.2.8** subroutine linalg\_sorting::sort\_cmplx\_array\_ind ( complex(dp), dimension(:), intent(inout) x, integer(i32), dimension(:), intent(inout) ind, logical, intent(in), optional *ascend*, class(errors), intent(inout), optional, target *err* ) [private]

Sorts an array.

**Parameters**

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, an integer array. On output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>ind</i> is not sized to match <i>x</i>.</li> </ul>

**Remarks**

This routine utilizes a quick sort algorithm. As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

**Notes**

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 205 of file linalg\_sorting.f90.

**4.8.2.9** subroutine linalg\_sorting::sort\_dbl\_array ( real(dp), dimension(:), intent(inout) x, logical, intent(in), optional *ascend* ) [private]

Sorts an array.



## Parameters

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.

## Remarks

The routine utilizes a quick sort algorithm unless the size of the array is less than or equal to 20. For such small arrays an insertion sort algorithm is utilized.

## Notes

This routine utilizes the LAPACK routine DLASRT.

Definition at line 47 of file linalg\_sorting.f90.

```
4.8.2.10 subroutine linalg_sorting::sort_dbl_array_ind ( real(dp), dimension(:), intent(inout) x, integer(i32), dimension(:),
               intent(inout) ind, logical, intent(in), optional ascend, class(errors), intent(inout), optional, target err ) [private]
```

Sorts an array.

## Parameters

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, an integer array. On output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>ind</i> is not sized to match <i>x</i>.</li> </ul>

## Remarks

This routine utilizes a quick sort algorithm explained at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 97 of file linalg\_sorting.f90.

```
4.8.2.11 subroutine linalg_sorting::sort_eigen_cmplx ( complex(dp), dimension(:), intent(inout) vals, complex(dp),
               dimension(:,:), intent(inout) vecs, logical, intent(in), optional ascend, class(errors), intent(inout), optional, target err )
               [private]
```

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

**Parameters**

<i>in, out</i>	<i>vals</i>	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
<i>in, out</i>	<i>vecs</i>	On input, an N-by-N matrix containing the eigenvectors associated with <i>vals</i> (one vector per column). On output, the sorted eigenvector matrix.
<i>in</i>	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
<i>out</i>	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <i>vecs</i> is not sized to match <i>vals</i>.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if there is insufficient memory available to complete this operation.</li> </ul>

Definition at line 267 of file `linalg_sorting.f90`.

```
4.8.2.12 subroutine linalg_sorting::sort_eigen_dbl ( real(dp), dimension(:), intent(inout) vals, real(dp), dimension(:,:),
            intent(inout) vecs, logical, intent(in), optional ascend, class(errors), intent(inout), optional, target err )
            [private]
```

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

**Parameters**

<i>in, out</i>	<i>vals</i>	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
<i>in, out</i>	<i>vecs</i>	On input, an N-by-N matrix containing the eigenvectors associated with <i>vals</i> (one vector per column). On output, the sorted eigenvector matrix.
<i>in</i>	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
<i>out</i>	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <i>vecs</i> is not sized to match <i>vals</i>.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if there is insufficient memory available to complete this operation.</li> </ul>

Definition at line 345 of file `linalg_sorting.f90`.

## Chapter 5

# Data Type Documentation

### 5.1 linalg\_core::diag\_mtx\_mult Interface Reference

Multiplies a diagonal matrix with another matrix or array.

#### Private Member Functions

- subroutine [diag\\_mtx\\_mult\\_mtx](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ .*
- subroutine [diag\\_mtx\\_mult\\_mtx2](#) (lside, alpha, a, b, err)  
*Computes the matrix operation:  $B = \alpha * A * op(B)$ , or  $B = \alpha * op(B) * A$ .*
- subroutine [diag\\_mtx\\_mult\\_mtx3](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ , where  $A$  and  $C$  are complex-valued.*
- subroutine [diag\\_mtx\\_mult\\_mtx4](#) (lside, trans, alpha, a, b, beta, c, err)  
*Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ , where  $A$ ,  $B$ , and  $C$  are complex-valued.*

#### 5.1.1 Detailed Description

Multiplies a diagonal matrix with another matrix or array.

Definition at line 46 of file linalg\_core.f90.

#### 5.1.2 Member Function/Subroutine Documentation

5.1.2.1 subroutine linalg\_core::diag\_mtx\_mult::diag\_mtx\_mult\_mtx ( logical, intent(in) lside, logical, intent(in) trans, real(dp) alpha, real(dp), dimension(:), intent(in) a, real(dp), dimension(:, :), intent(in) b, real(dp) beta, real(dp), dimension(:, :), intent(inout) c, class(errors), intent(inout), optional, target err ) [private]

Computes the matrix operation:  $C = \alpha * A * op(B) + \beta * C$ , or  $C = \alpha * op(B) * A + \beta * C$ .

## Parameters

in	<i>lside</i>	Set to true to apply matrix A from the left; else, set to false to apply matrix A from the left.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^{**T}$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of A where $K = \text{MIN}(M,P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N,P)$ .
in	<i>b</i>	The LDB-by-TDB matrix B where (LDB = leading dimension of B, and TDB = trailing dimension of B): <ul style="list-style-type: none"> <li><code>lside == true &amp; trans == true</code>: LDB = N, TDB = P</li> <li><code>lside == true &amp; trans == false</code>: LDB = P, TDB = N</li> <li><code>lside == false &amp; trans == true</code>: LDB = P, TDB = M</li> <li><code>lside == false &amp; trans == false</code>: LDB = M, TDB = P</li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the resulting M-by-N matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li><code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 329 of file `linalg_core.f90`.

5.1.2.2 subroutine `linalg_core::diag_mtx_mult::diag_mtx_mult_mtx2` ( `logical`, intent(in) *lside*, `real(dp)`, intent(in) *alpha*, `real(dp)`, dimension(:), intent(in) *a*, `real(dp)`, dimension(:, :), intent(inout) *b*, `class(errors)`, intent(inout), optional, target *err* ) [private]

Computes the matrix operation:  $B = \alpha * A * \text{op}(B)$ , or  $B = \alpha * \text{op}(B) * A$ .

## Parameters

in	<i>lside</i>	Set to true to apply matrix A from the left; else, set to false to apply matrix A from the left.
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A K-element array containing the diagonal elements of A where $K = \text{MIN}(M,P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N,P)$ .
in	<i>b</i>	On input, the M-by-N matrix B. On output, the resulting M-by-N matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li><code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 496 of file `linalg_core.f90`.

5.1.2.3 subroutine `linalg_core::diag_mtx_mult::diag_mtx_mult_mtx3` ( `logical, intent(in) lside`, `logical, intent(in) trans`, `real(dp) alpha`, `complex(dp), dimension(:), intent(in) a`, `real(dp), dimension(:, :), intent(in) b`, `real(dp) beta`, `complex(dp), dimension(:, :), intent(inout) c`, `class(errors), intent(inout), optional, target err` ) [private]

Computes the matrix operation:  $C = \alpha * A * \text{op}(B) + \beta * C$ , or  $C = \alpha * \text{op}(B) * A + \beta * C$ , where  $A$  and  $C$  are complex-valued.

#### Parameters

in	<i>lside</i>	Set to true to apply matrix $A$ from the left; else, set to false to apply matrix $A$ from the left.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^{**T}$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A $K$ -element array containing the diagonal elements of $A$ where $K = \text{MIN}(M, P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N, P)$ .
in	<i>b</i>	The LDB-by-TDB matrix $B$ where (LDB = leading dimension of $B$ , and TDB = trailing dimension of $B$ ): <ul style="list-style-type: none"> <li><i>lside</i> == true &amp; <i>trans</i> == true: LDB = <math>N</math>, TDB = <math>P</math></li> <li><i>lside</i> == true &amp; <i>trans</i> == false: LDB = <math>P</math>, TDB = <math>N</math></li> <li><i>lside</i> == false &amp; <i>trans</i> == true: LDB = <math>P</math>, TDB = <math>M</math></li> <li><i>lside</i> == false &amp; <i>trans</i> == false: LDB = <math>M</math>, TDB = <math>P</math></li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the $M$ -by- $N$ matrix $C$ . On output, the resulting $M$ -by- $N$ matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 579 of file `linalg_core.f90`.

5.1.2.4 subroutine `linalg_core::diag_mtx_mult::diag_mtx_mult_mtx4` ( `logical, intent(in) lside`, `logical, intent(in) trans`, `real(dp) alpha`, `complex(dp), dimension(:), intent(in) a`, `complex(dp), dimension(:, :), intent(in) b`, `real(dp) beta`, `complex(dp), dimension(:, :), intent(inout) c`, `class(errors), intent(inout), optional, target err` ) [private]

Computes the matrix operation:  $C = \alpha * A * \text{op}(B) + \beta * C$ , or  $C = \alpha * \text{op}(B) * A + \beta * C$ , where  $A$ ,  $B$ , and  $C$  are complex-valued.

#### Parameters

in	<i>lside</i>	Set to true to apply matrix $A$ from the left; else, set to false to apply matrix $A$ from the left.
in	<i>trans</i>	Set to true if $\text{op}(B) == B^{**T}$ ; else, set to false if $\text{op}(B) == B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	A $K$ -element array containing the diagonal elements of $A$ where $K = \text{MIN}(M, P)$ if <i>lside</i> is true; else, if <i>lside</i> is false, $K = \text{MIN}(N, P)$ .

## Parameters

in	<i>b</i>	<p>The LDB-by-TDB matrix B where:</p> <ul style="list-style-type: none"> <li>• <code>lside == true &amp; trans == true</code>: LDA = N, TDB = P</li> <li>• <code>lside == true &amp; trans == false</code>: LDA = P, TDB = N</li> <li>• <code>lside == false &amp; trans == true</code>: LDA = P, TDB = M</li> <li>• <code>lside == false &amp; trans == false</code>: LDA = M, TDB = P</li> </ul>
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the resulting M-by-N matrix.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 755 of file `linalg_core.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_core.f90`

## 5.2 lapack::DLAMCH Interface Reference

### Public Member Functions

- `real(dp)` function **dlamch** (cmach)

#### 5.2.1 Detailed Description

Definition at line 14 of file `lapack.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/lapack.f90`

## 5.3 linalg\_eigen::eigen Interface Reference

Computes the eigenvalues, and optionally the eigenvectors, of a matrix.

## Private Member Functions

- subroutine [eigen\\_symm](#) (vecs, a, vals, work, olwork, err)  
*Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.*
- subroutine [eigen\\_asymm](#) (a, vals, vecs, work, olwork, err)  
*Computes the eigenvalues, and optionally the right eigenvectors of a square matrix.*
- subroutine [eigen\\_gen](#) (a, b, alpha, beta, vecs, work, olwork, err)  
*Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A*X = \lambda*B*X$ .*

### 5.3.1 Detailed Description

Computes the eigenvalues, and optionally the eigenvectors, of a matrix.

#### See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)
- [LAPACK Users Manual](#)

Definition at line 24 of file linalg\_eigen.f90.

### 5.3.2 Member Function/Subroutine Documentation

**5.3.2.1** subroutine `linalg_eigen::eigen::eigen_asymm` ( `real(dp)`, `dimension(:,:)`, `intent(inout)` `a`, `complex(dp)`, `dimension(:)`, `intent(out)` `vals`, `complex(dp)`, `dimension(:,:)`, `intent(out)`, optional `vecs`, `real(dp)`, `dimension(:)`, `intent(out)`, optional, pointer `work`, `integer(i32)`, `intent(out)`, optional `olwork`, `class(errors)`, `intent(inout)`, optional, target `err` ) `[private]`

Computes the eigenvalues, and optionally the right eigenvectors of a square matrix.

#### Parameters

in, out	<code>a</code>	On input, the N-by-N matrix on which to operate. On output, the contents of this matrix are overwritten.
out	<code>vals</code>	An N-element array containing the eigenvalues of the matrix. The eigenvalues are not sorted.
out	<code>vecs</code>	An optional N-by-N matrix, that if supplied, signals to compute the right eigenvectors (one per column). If not provided, only the eigenvalues will be computed.
out	<code>work</code>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<code>olwork</code>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<code>err</code>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
Generated by Doxygen		<ul style="list-style-type: none"> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGEEV.

Definition at line 185 of file linalg\_eigen.f90.

```
5.3.2.2 subroutine linalg_eigen::eigen::eigen_gen ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:,:), intent(inout)
b, complex(dp), dimension(:,:), intent(out) alpha, real(dp), dimension(:,:), intent(out), optional beta, complex(dp),
dimension(:,:), intent(out), optional vecs, real(dp), dimension(:,:), intent(out), optional, pointer work, integer(i32),
intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]
```

Computes the eigenvalues, and optionally the right eigenvectors of a square matrix assuming the structure of the eigenvalue problem is  $A \cdot X = \lambda B \cdot X$ .

## Parameters

in, out	<i>a</i>	On input, the N-by-N matrix A. On output, the contents of this matrix are overwritten.
in, out	<i>b</i>	On input, the N-by-N matrix B. On output, the contents of this matrix are overwritten.
out	<i>alpha</i>	An N-element array that, if <i>beta</i> is not supplied, contains the eigenvalues. If <i>beta</i> is supplied however, the eigenvalues must be computed as ALPHA / BETA. This however, is not as trivial as it seems as it is entirely possible, and likely, that ALPHA / BETA can overflow or underflow. With that said, the values in ALPHA will always be less than and usually comparable with the NORM(A).
out	<i>beta</i>	An optional N-element array that if provided forces <i>alpha</i> to return the numerator, and this array contains the denominator used to determine the eigenvalues as ALPHA / BETA. If used, the values in this array will always be less than and usually comparable with the NORM(B).
out	<i>vecs</i>	An optional N-by-N matrix, that if supplied, signals to compute the right eigenvectors (one per column). If not provided, only the eigenvalues will be computed.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>LA_CONVERGENCE_ERROR: Occurs if the algorithm failed to converge.</li> </ul>

## Usage

As an example, consider the eigenvalue problem arising from a mechanical system of masses and springs such that the masses are described by a mass matrix *M*, and the arrangement of springs are described by a stiffness matrix *K*.

```
! This is an example illustrating the use of the eigenvalue and eigenvector
! routines to solve a free vibration problem of 3 masses connected by springs.
```



```

!
!      k1      k2      k3      k4
!  |---|---|---|---|
!  |---|---|---|---|
!
! As illustrated above, the system consists of 3 masses connected by springs.
! Spring k1 and spring k4 connect the end masses to ground. The equations of
! motion for this system are as follows.
!
!  | m1  0   0 | |x1"|  | k1+k2  -k2      0 | |x1|  |0|
!  |  0  m2  0 | |x2"| + | -k2  k2+k3   -k3 | |x2| = |0|
!  |  0   0  m3| |x3"|  |  0   -k3   k3+k4| |x3|  |0|
!
! Notice: x1" = the second time derivative of x1.
program example
  use linalg_constants, only : dp, i32
  use linalg_eigen
  implicit none

  ! Define the model parameters
  real(dp), parameter :: pi = 3.14159265359d0
  real(dp), parameter :: m1 = 0.5d0
  real(dp), parameter :: m2 = 2.5d0
  real(dp), parameter :: m3 = 0.75d0
  real(dp), parameter :: k1 = 5.0d6
  real(dp), parameter :: k2 = 10.0d6
  real(dp), parameter :: k3 = 10.0d6
  real(dp), parameter :: k4 = 5.0d6

  ! Local Variables
  integer(i32) :: i, j
  real(dp) :: m(3,3), k(3,3), natfreq(3)
  complex(dp) :: vals(3), modeshapes(3,3)

  ! Define the mass matrix
  m = reshape([m1, 0.0d0, 0.0d0, 0.0d0, m2, 0.0d0, 0.0d0, 0.0d0, m3], [3, 3])

  ! Define the stiffness matrix
  k = reshape([k1 + k2, -k2, 0.0d0, -k2, k2 + k3, -k3, 0.0d0, -k3, k3 + k4], &
    [3, 3])

  ! Compute the eigenvalues and eigenvectors.
  call eigen(k, m, vals, vecs = modeshapes)

  ! Compute the natural frequency values, and return them with units of Hz.
  ! Notice, all eigenvalues and eigenvectors are real for this example.
  natfreq = sqrt(real(vals)) / (2.0d0 * pi)

  ! Display the natural frequency and mode shape values. Notice, the eigen
  ! routine does not necessarily sort the values.
  print '(A)', "Modal Information (Not Sorted):"
  do i = 1, size(natfreq)
    print '(AI0AF8.4A)', "Mode ", i, ": (", natfreq(i), " Hz)"
    print '(F10.3)', (real(modeshapes(j,i)), j = 1, size(natfreq))
  end do
end program

```

## Notes

This routine utilizes the LAPACK routine DGGEV.

Definition at line 465 of file linalg\_eigen.f90.

**5.3.2.3** subroutine linalg\_eigen::eigen::eigen\_symm ( logical, intent(in) *vecs*, real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:,:), intent(out) *vals*, real(dp), dimension(:,:), intent(out), optional, pointer *work*, integer(i32), intent(out), optional *o/work*, class(errors), intent(inout), optional, target *err* ) [private]

Computes the eigenvalues, and optionally the eigenvectors of a real, symmetric matrix.

## Parameters

<i>in</i>	<i>vecs</i>	Set to true to compute the eigenvectors as well as the eigenvalues; else, set to false to just compute the eigenvalues.
-----------	-------------	---

## Parameters

in, out	<i>a</i>	On input, the N-by-N symmetric matrix on which to operate. On output, and if <i>vecs</i> is set to true, the matrix will contain the eigenvectors (one per column) corresponding to each eigenvalue in <i>vals</i> . If <i>vecs</i> is set to false, the lower triangular portion of the matrix is overwritten.
out	<i>vals</i>	An N-element array that will contain the eigenvalues sorted into ascending order.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_CONVERGENCE_ERROR</code>: Occurs if the algorithm failed to converge.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DSYEV.

Definition at line 68 of file `linalg_eigen.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_eigen.f90`

## 5.4 linalg\_factor::form\_lu Interface Reference

Extracts the L and U matrices from the condensed [L\U] storage format used by the [lu\\_factor](#).

### Private Member Functions

- subroutine [form\\_lu\\_all](#) (lu, ipvt, u, p, err)  
*Extracts the L, U, and P matrices from the output of the [lu\\_factor](#) routine.*
- subroutine [form\\_lu\\_only](#) (lu, u, err)  
*Extracts the L, and U matrices from the output of the [lu\\_factor](#) routine.*

#### 5.4.1 Detailed Description

Extracts the L and U matrices from the condensed [L\U] storage format used by the [lu\\_factor](#).

Definition at line 31 of file `linalg_factor.f90`.

## 5.4.2 Member Function/Subroutine Documentation

**5.4.2.1** subroutine `linalg_factor::form_lu::form_lu_all` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *lu*, `integer(i32)`, `dimension(:)`, `intent(in)` *ipvt*, `real(dp)`, `dimension(:, :)`, `intent(out)` *u*, `real(dp)`, `dimension(:, :)`, `intent(out)` *p*, `class(errors)`, `intent(inout)`, `optional`, `target` *err* ) [`private`]

Extracts the L, U, and P matrices from the output of the `lu_factor` routine.

### Parameters

<code>in, out</code>	<i>lu</i>	On input, the N-by-N matrix as output by <code>lu_factor</code> . On output, the N-by-N lower triangular matrix L.
<code>in</code>	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
<code>out</code>	<i>u</i>	An N-by-N matrix where the U matrix will be written.
<code>out</code>	<i>p</i>	An N-by-N matrix where the row permutation matrix will be written.
<code>out</code>	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

### Remarks

This routine allows extraction of the actual "L", "U", and "P" matrices of the decomposition. To use these matrices to solve the system  $A \cdot X = B$ , the following approach is used.

1. First, solve the linear system:  $L \cdot Y = P \cdot B$  for Y.
2. Second, solve the linear system:  $U \cdot X = Y$  for X.

Notice, as both L and U are triangular in structure, the above equations can be solved by forward and backward substitution.

### See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 211 of file `linalg_factor.f90`.

**5.4.2.2** subroutine `linalg_factor::form_lu::form_lu_only` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *lu*, `real(dp)`, `dimension(:, :)`, `intent(out)` *u*, `class(errors)`, `intent(inout)`, `optional`, `target` *err* ) [`private`]

Extracts the L, and U matrices from the output of the `lu_factor` routine.

### Parameters

<code>in, out</code>	<i>lu</i>	On input, the N-by-N matrix as output by <code>lu_factor</code> . On output, the N-by-N lower triangular matrix L.
<code>out</code>	<i>u</i>	An N-by-N matrix where the U matrix will be written.
<code>out</code> Generated by Doxygen	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

Definition at line 288 of file linalg\_factor.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/linalg/src/linalg\_factor.f90

## 5.5 linalg\_factor::form\_qr Interface Reference

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

### Private Member Functions

- subroutine [form\\_qr\\_no\\_pivot](#) (r, tau, q, work, olwork, err)  
*Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.*
- subroutine [form\\_qr\\_pivot](#) (r, tau, pvt, q, p, work, olwork, err)  
*Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.*

### 5.5.1 Detailed Description

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

#### See Also

- [Wikipedia](#)
- [LAPACK Users Manual](#)

Definition at line 55 of file linalg\_factor.f90.

### 5.5.2 Member Function/Subroutine Documentation

**5.5.2.1** subroutine linalg\_factor::form\_qr::form\_qr\_no\_pivot ( real(dp), dimension(:,:), intent(inout) *r*, real(dp), dimension(:), intent(in) *tau*, real(dp), dimension(:,:), intent(out) *q*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

#### Parameters

in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
---------	----------	--

## Parameters

in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that $M > N$ , Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 ( $Q = [Q1, Q2]$ ) as the factorization can be written as $Q * R = [Q1, Q2] * [R1; 0]$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>• LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORGQR.

Definition at line 694 of file linalg\_factor.f90.

**5.5.2.2** subroutine linalg\_factor::form\_qr::form\_qr\_pivot ( real(dp), dimension(:,:), intent(inout) *r*, real(dp), dimension(:), intent(in) *tau*, integer(i32), dimension(:), intent(in) *pvt*, real(dp), dimension(:,:), intent(out) *q*, real(dp), dimension(:,:), intent(out) *p*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Forms the full M-by-M orthogonal matrix Q from the elementary reflectors returned by the base QR factorization algorithm.

## Parameters

in, out	<i>r</i>	On input, an M-by-N matrix where the elements below the diagonal contain the elementary reflectors generated from the QR factorization. On and above the diagonal, the matrix contains the matrix R. On output, the elements below the diagonal are zeroed such that the remaining matrix is simply the M-by-N matrix R.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of each elementary reflector defined in <i>r</i> .
in	<i>pvt</i>	An N-element column pivot array as returned by the QR factorization.
out	<i>q</i>	An M-by-M matrix where the full orthogonal matrix Q will be written. In the event that $M > N$ , Q may be supplied as M-by-N, and therefore only return the useful submatrix Q1 ( $Q = [Q1, Q2]$ ) as the factorization can be written as $Q * R = [Q1, Q2] * [R1; 0]$ .
out	<i>p</i>	An N-by-N matrix where the pivot matrix will be written.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .

## Parameters

out	olwork	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	err	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORGQR.

Definition at line 825 of file `linalg_factor.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_factor.f90`

## 5.6 linalg\_core::mtx\_mult Interface Reference

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

### Private Member Functions

- subroutine `mtx_mult_mtx` (`transa`, `transb`, `alpha`, `a`, `b`, `beta`, `c`, `err`)  
*Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .*
- subroutine `mtx_mult_vec` (`trans`, `alpha`, `a`, `b`, `beta`, `c`, `err`)  
*Performs the matrix-vector operation:  $c = \alpha * \text{op}(A) * b + \beta * c$ .*

### 5.6.1 Detailed Description

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

Definition at line 39 of file `linalg_core.f90`.

### 5.6.2 Member Function/Subroutine Documentation

**5.6.2.1** subroutine `linalg_core::mtx_mult::mtx_mult_mtx` ( `logical`, intent(in) *transa*, `logical`, intent(in) *transb*, `real(dp)`, intent(in) *alpha*, `real(dp)`, dimension(:,:) intent(in) *a*, `real(dp)`, dimension(:,:) intent(in) *b*, `real(dp)`, intent(in) *beta*, `real(dp)`, dimension(:,:) intent(inout) *c*, `class(errors)`, intent(inout), optional, target *err* ) [private]

Performs the matrix operation:  $C = \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ .

## Parameters

in	<i>transa</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false for $\text{op}(A) = A$ .
in	<i>transb</i>	Set to true if $\text{op}(B) = B^{**T}$ ; else, set to false for $\text{op}(B) = B$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	If <i>transa</i> is set to true, an K-by-M matrix; else, if <i>transa</i> is set to false, an M-by-K matrix.
in	<i>b</i>	If <i>transb</i> is set to true, an N-by-K matrix; else, if <i>transb</i> is set to false, a K-by-N matrix.
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the M-by-N result.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the BLAS routine DGEMM.

Definition at line 85 of file linalg\_core.f90.

**5.6.2.2** subroutine linalg\_core::mtx\_mult::mtx\_mult\_vec ( logical, intent(in) *trans*, real(dp), intent(in) *alpha*, real(dp), dimension(:, :), intent(in) *a*, real(dp), dimension(:), intent(in) *b*, real(dp), intent(in) *beta*, real(dp), dimension(:), intent(inout) *c*, class(errors), intent(inout), optional, target *err* ) [private]

Performs the matrix-vector operation:  $c = \alpha * \text{op}(A) * b + \beta * c$ .

## Parameters

in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false for $\text{op}(A) = A$ .
in	<i>alpha</i>	A scalar multiplier.
in	<i>a</i>	The M-by-N matrix A.
in	<i>b</i>	If <i>trans</i> is set to true, an M-element array; else, if <i>trans</i> is set to false, an N-element array.
in	<i>beta</i>	A scalar multiplier.
in, out	<i>c</i>	On input, if <i>trans</i> is set to true, an N-element array; else, if <i>trans</i> is set to false, an M-element array. On output, the results of the operation.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

This routine utilizes the BLAS routine DGEMV.

Definition at line 179 of file `linalg_core.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_core.f90`

## 5.7 `linalg_factor::mult_qr` Interface Reference

Multiplies a general matrix by the orthogonal matrix  $Q$  from a QR factorization.

### Private Member Functions

- subroutine `mult_qr_mtx` (`lside`, `trans`, `a`, `tau`, `c`, `work`, `olwork`, `err`)  
*Multiplies a general matrix by the orthogonal matrix  $Q$  from a QR factorization such that:  $C = \text{op}(Q) * C$ , or  $C = C * \text{op}(Q)$ .*
- subroutine `mult_qr_vec` (`trans`, `a`, `tau`, `c`, `work`, `olwork`, `err`)  
*Multiplies a vector by the orthogonal matrix  $Q$  from a QR factorization such that:  $C = \text{op}(Q) * C$ .*

### 5.7.1 Detailed Description

Multiplies a general matrix by the orthogonal matrix  $Q$  from a QR factorization.

Definition at line 63 of file `linalg_factor.f90`.

### 5.7.2 Member Function/Subroutine Documentation

**5.7.2.1** subroutine `linalg_factor::mult_qr::mult_qr_mtx` ( `logical`, intent(in) `lside`, `logical`, intent(in) `trans`, `real(dp)`, dimension(:,:), intent(inout) `a`, `real(dp)`, dimension(:), intent(in) `tau`, `real(dp)`, dimension(:,:), intent(inout) `c`, `real(dp)`, dimension(:), intent(out), optional, target `work`, `integer(i32)`, intent(out), optional `olwork`, `class(errors)`, intent(inout), optional, target `err` ) [`private`]

Multiplies a general matrix by the orthogonal matrix  $Q$  from a QR factorization such that:  $C = \text{op}(Q) * C$ , or  $C = C * \text{op}(Q)$ .

#### Parameters

in	<code>lside</code>	Set to true to apply $Q$ or $Q^{**T}$ from the left; else, set to false to apply $Q$ or $Q^{**T}$ from the right.
in	<code>trans</code>	Set to true to apply $Q^{**T}$ ; else, set to false.
in	<code>a</code>	On input, an LDA-by-K matrix containing the elementary reflectors output from the QR factorization. If <code>lside</code> is set to true, $\text{LDA} = M$ , and $M \geq K \geq 0$ ; else, if <code>lside</code> is set to false, $\text{LDA} = N$ , and $N \geq K \geq 0$ . Notice, the contents of this matrix are restored on exit.
in	<code>tau</code>	A K-element array containing the scalar factors of each elementary reflector defined in <code>a</code> .
in, out	<code>c</code>	On input, the M-by-N matrix $C$ . On output, the product of the orthogonal matrix $Q$ and the original matrix $C$ .
out	<code>work</code>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .



## Parameters

out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORMQR.

Definition at line 927 of file `linalg_factor.f90`.

**5.7.2.2** subroutine `linalg_factor::mult_qr::mult_qr_vec` ( logical, intent(in) *trans*, real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:), intent(in) *tau*, real(dp), dimension(:), intent(inout) *c*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Multiplies a vector by the orthogonal matrix *Q* from a QR factorization such that:  $C = \text{op}(Q) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Q^{**T}$ ; else, set to false.
in	<i>a</i>	On input, an M-by-K matrix containing the elementary reflectors output from the QR factorization. Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of each elementary reflector defined in <i>a</i> .
in, out	<i>c</i>	On input, the M-element vector <i>C</i> . On output, the product of the orthogonal matrix <i>Q</i> and the original vector <i>C</i> .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon the LAPACK routine DORM2R.

Definition at line 1054 of file linalg\_factor.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/linalg/src/linalg\_factor.f90

## 5.8 linalg\_factor::mult\_rz Interface Reference

Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization.

### Private Member Functions

- subroutine [mult\\_rz\\_mtx](#) (lside, trans, l, a, tau, c, work, olwork, err)  
*Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ , or  $C = C * \text{op}(Z)$ .*
- subroutine [mult\\_rz\\_vec](#) (trans, l, a, tau, c, work, olwork, err)  
*Multiplies a vector by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ .*

### 5.8.1 Detailed Description

Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization.

Definition at line 71 of file linalg\_factor.f90.

### 5.8.2 Member Function/Subroutine Documentation

**5.8.2.1** subroutine linalg\_factor::mult\_rz::mult\_rz\_mtx ( logical, intent(in) lside, logical, intent(in) trans, integer(i32), intent(in) l, real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(in) tau, real(dp), dimension(:, :), intent(inout) c, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]

Multiplies a general matrix by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ , or  $C = C * \text{op}(Z)$ .

#### Parameters

in	lside	Set to true to apply Z or Z**T from the left; else, set to false to apply Z or Z**T from the right.
in	trans	Set to true to apply Z**T; else, set to false.
in	l	The number of columns in matrix a containing the meaningful part of the Householder vectors. If lside is true, $M \geq L \geq 0$ ; else, if lside is false, $N \geq L \geq 0$ .
in, out	a	On input the K-by-LTA matrix Z, where LTA = M if lside is true; else, LTA = N if lside is false. The l-th row must contain the Householder vector in the last k rows. Notice, the contents of this matrix are restored on exit.

## Parameters

in	<i>tau</i>	A K-element array containing the scalar factors of the elementary reflectors, where $M \geq K \geq 0$ if <i>lside</i> is true; else, $N \geq K \geq 0$ if <i>lside</i> is false.
in, out	<i>c</i>	On input, the M-by-N matrix C. On output, the product of the orthogonal matrix Z and the original matrix C.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DORMRZ.

Definition at line 1745 of file linalg\_factor.f90.

**5.8.2.2** subroutine linalg\_factor::mult\_rz::mult\_rz\_vec ( logical, intent(in) *trans*, integer(i32), intent(in) *l*, real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:), intent(in) *tau*, real(dp), dimension(:), intent(inout) *c*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Multiplies a vector by the orthogonal matrix Z from an RZ factorization such that:  $C = \text{op}(Z) * C$ .

## Parameters

in	<i>trans</i>	Set to true to apply $Z^*T$ ; else, set to false.
in	<i>l</i>	The number of columns in matrix <i>a</i> containing the meaningful part of the Householder vectors. If <i>lside</i> is true, $M \geq L \geq 0$ ; else, if <i>lside</i> is false, $N \geq L \geq 0$ .
in, out	<i>a</i>	On input the K-by-LTA matrix Z, where LTA = M if <i>lside</i> is true; else, LTA = N if <i>lside</i> is false. The <i>l</i> -th row must contain the Householder vector in the last <i>k</i> rows. Notice, the contents of this matrix are restored on exit.
in	<i>tau</i>	A K-element array containing the scalar factors of the elementary reflectors, where $M \geq K \geq 0$ if <i>lside</i> is true; else, $N \geq K \geq 0$ if <i>lside</i> is false.
in, out	<i>c</i>	On input, the M-element array C. On output, the product of the orthogonal matrix Z and the original array C.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
-----	-----	--

## Notes

This routine utilizes the LAPACK routine DORMRZ.

Definition at line 1887 of file `linalg_factor.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_factor.f90`

## 5.9 linalg\_factor::qr\_factor Interface Reference

Computes the QR factorization of an M-by-N matrix.

### Private Member Functions

- subroutine `qr_factor_no_pivot` (a, tau, work, olwork, err)  
*Computes the QR factorization of an M-by-N matrix without pivoting.*
- subroutine `qr_factor_pivot` (a, tau, jpvt, work, olwork, err)  
*Computes the QR factorization of an M-by-N matrix with column pivoting such that  $A * P = Q * R$ .*

### 5.9.1 Detailed Description

Computes the QR factorization of an M-by-N matrix.

#### See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)
- [LAPACK Users Manual](#)

Definition at line 43 of file `linalg_factor.f90`.

### 5.9.2 Member Function/Subroutine Documentation

**5.9.2.1** subroutine `linalg_factor::qr_factor::qr_factor_no_pivot` ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:), intent(out) tau, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]

Computes the QR factorization of an M-by-N matrix without pivoting.

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <i>tau</i> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>tau</i> or <i>work</i> are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Remarks

QR factorization without pivoting is best suited to solving an overdetermined system in least-squares terms, or to solve a normally defined system. To solve an underdetermined system, it is recommended to use either LQ factorization, or a column-pivoting based QR factorization.

## Usage

To solve a system of M equations of N unknowns using QR factorization, the following code will suffice assuming  $M \geq N$ .

```
! Solve the system: A*X = B in a least-squares sense, where A is an
! M-by-N matrix, B is an M-by-NRHS matrix, and X is an N-by-NRHS matrix.

! Variables
real(dp), dimension(m, n) :: a
real(dp), dimension(m, nrhs) :: b, qtb
real(dp), dimension(n, nrhs) :: x
real(dp), dimension(n) :: tau
real(dp), dimension(m, m) :: q

! Initialize A and B...

! Compute the QR factorization. We're intentionally not forming the full
! Q matrix, but instead storing it in terms of its elementary reflector
! components in the sub-diagonal portions of A, and the corresponding
! scalar factors in TAU. Additionally, we'll let the algorithm allocate
! it's own workspace array; therefore, the call to factor A is:
call qr_factor(a, tau)

! Solve A*X = B for X. The first N rows of B are used to store X.
call solve_qr(a, tau, b)

! Also note, we could form Q and R explicitly. Then solution of the
! system of equations can be found. First we form Q and R.
call form_qr(a, tau, q) ! Forms Q, and R is stored in A

! Since we now have Q and R, we seek a solution to the equation:
! Q*R*X = B, but Q is an orthogonal matrix (i.e. Q**T = inv(Q)).
! Then: R*X = Q**T * B, and R is upper triangular; therefore, back
! substitution will suffice for a solution procedure.
!
! Next, compute Q**T * B, and store in QTB.
call mtv_mult(.true., .false., 1.0d0, q, b, 0.0d0, qtb)

! Copy the first N rows of Q**T * B into X for the solution process.
```

```

! Notice, only the first N rows are needed as rows N+1:M are all zero in
! matrix R.
x = qtb(1:n,nrhs)

! Compute the solution and store in X
call solve_triangular_system(.true., .true., .false., .true., 1.0d0, &
a(1:n,1:n), x)

```

## Notes

This routine utilizes the LAPACK routine DGEQRF.

Definition at line 425 of file linalg\_factor.f90.

**5.9.2.2** subroutine linalg\_factor::qr\_factor::qr\_factor\_pivot ( real(dp), dimension(:, :), intent(inout) *a*, real(dp), dimension(:), intent(out) *tau*, integer(i32), dimension(:), intent(inout) *jpvt*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Computes the QR factorization of an M-by-N matrix with column pivoting such that  $A * P = Q * R$ .

## Parameters

in, out	<i>a</i>	On input, the M-by-N matrix to factor. On output, the elements on and above the diagonal contain the MIN(M, N)-by-N upper trapezoidal matrix R (R is upper triangular if $M \geq N$ ). The elements below the diagonal, along with the array <i>tau</i> , represent the orthogonal matrix Q as a product of elementary reflectors.
out	<i>tau</i>	A MIN(M, N)-element array used to store the scalar factors of the elementary reflectors.
in, out	<i>jpvt</i>	On input, an N-element array that if JPVT(I) .ne. 0, the I-th column of A is permuted to the front of $A * P$ ; if JPVT(I) = 0, the I-th column of A is a free column. On output, if JPVT(I) = K, then the I-th column of $A * P$ was the K-th column of A.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Usage

To solve a system of M equations of N unknowns using QR factorization, the following code will suffice for any M and N.

```

! Solve the least-squares (M >= N), or the underdetermined (M < N)
! problem A*X = B, where A is an M-by-N matrix, B is an M-by-NRHS matrix,
! and X is an N-by-NRHS matrix. In the underdetermined case, or the
! case where the rank of matrix A is less than N, the solution obtained
! contains the fewest possible non-zero entries.

! Variables

```

```

real(dp), dimension(m, n) :: a
real(dp), dimension(n, nrhs) :: b
real(dp), dimension(k) :: tau ! k = min(m, n)
real(dp), dimension(m, m) :: q
real(dp), dimension(n, n) :: p
integer(i32), dimension(n) :: pvt

! Initialize A and B...

! Allow all columns to be free.
pvt = 0

! Compute the QR factorization. We're intentionally not forming the full
! Q matrix, but instead storing it in terms of its elementary reflector
! components in the sub-diagonal portions of A, and the corresponding
! scalar factors in TAU. Additionally, we'll let the algorithm allocate
! it's own workspace array; therefore, the call to factor A is:
call qr_factor(a, tau, pvt)

! Solve A*X = B for X. If M > N, the first N rows of B are used to store
! X. If M < N, the input matrix B must be N-by-NRHS, and only the first
! M rows are used for the actual matrix B. The remaining N-M rows
! can contain whatever as they are not referenced until they are
! overwritten by the N-by-NRHS solution matrix X.
call solve_qr(a, tau, pvt, b)

! Notice, if the explicit Q matrix from the factorization is desired,
! the form_qr routine works similarly as in the no-pivot case;
! however, the permutation matrix P is also constructed. The call would
! be as follows. Also, as with the no-pivot algorithm, the matrix R is
! stored in matrix A.
call form_qr(a, tau, pvt, q, p)

! Solution can proceed as per typical, but with a full Q matrix. Also
! note, the problem is of the form: A*P = Q*R. Solution is straight
! forward, as with the no-pivot case; however, if M < N, then R is upper
! trapezoidal, and must be appropriately partitioned to solve. The rank
! of matrix r should be considered when applying the partition.

```

## Notes

This routine utilizes the LAPACK routine DGEQP3.

Definition at line 579 of file linalg\_factor.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/linalg/src/linalg\_factor.f90

## 5.10 linalg\_solve::solve\_cholesky Interface Reference

Solves a system of Cholesky factored equations.

### Private Member Functions

- subroutine [solve\\_cholesky\\_mtx](#) (upper, a, b, err)  
*Solves a system of Cholesky factored equations.*
- subroutine [solve\\_cholesky\\_vec](#) (upper, a, b, err)  
*Solves a system of Cholesky factored equations.*

### 5.10.1 Detailed Description

Solves a system of Cholesky factored equations.

Definition at line 83 of file linalg\_solve.f90.

### 5.10.2 Member Function/Subroutine Documentation

5.10.2.1 subroutine `linalg_solve::solve_cholesky::solve_cholesky_mtx` ( `logical, intent(in) upper`, `real(dp), dimension(:, :)`, `intent(in) a`, `real(dp), dimension(:, :)`, `intent(inout) b`, `class(errors)`, `intent(inout), optional, target err` ) `[private]`

Solves a system of Cholesky factored equations.

#### Parameters

<code>in</code>	<code>upper</code>	Set to true if the original matrix A was factored such that $A = U^{**T} * U$ ; else, set to false if the factorization of A was $A = L^{**T} * L$ .
<code>in</code>	<code>a</code>	The N-by-N Cholesky factored matrix.
<code>in, out</code>	<code>b</code>	On input, the N-by-NRHS right-hand-side matrix B. On output, the solution matrix X.
<code>out</code>	<code>err</code>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

#### Notes

This routine utilizes the LAPACK routine DPOTRS.

Definition at line 1073 of file `linalg_solve.f90`.

5.10.2.2 subroutine `linalg_solve::solve_cholesky::solve_cholesky_vec` ( `logical, intent(in) upper`, `real(dp), dimension(:, :)`, `intent(in) a`, `real(dp), dimension(:, :)`, `intent(inout) b`, `class(errors)`, `intent(inout), optional, target err` ) `[private]`

Solves a system of Cholesky factored equations.

#### Parameters

<code>in</code>	<code>upper</code>	Set to true if the original matrix A was factored such that $A = U^{**T} * U$ ; else, set to false if the factorization of A was $A = L^{**T} * L$ .
<code>in</code>	<code>a</code>	The N-by-N Cholesky factored matrix.
<code>in, out</code>	<code>b</code>	On input, the N-element right-hand-side vector B. On output, the solution vector X.
<code>out</code>	<code>err</code>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input array sizes are incorrect.</li> </ul>

#### Notes

This routine utilizes the LAPACK routine DPOTRS.

Definition at line 1139 of file `linalg_solve.f90`.

The documentation for this interface was generated from the following file:



- `/home/jason/Documents/Code/linalg/src/linalg_solve.f90`

## 5.11 linalg\_solve::solve\_least\_squares Interface Reference

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns.

### Private Member Functions

- subroutine `solve_least_squares_mtx` ( $a$ ,  $b$ ,  $work$ ,  $olwork$ ,  $err$ )  
*Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.*
- subroutine `solve_least_squares_vec` ( $a$ ,  $b$ ,  $work$ ,  $olwork$ ,  $err$ )  
*Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.*

### 5.11.1 Detailed Description

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns.

Definition at line 91 of file `linalg_solve.f90`.

### 5.11.2 Member Function/Subroutine Documentation

5.11.2.1 subroutine `linalg_solve::solve_least_squares::solve_least_squares_mtx` ( `real(dp)`, `dimension(:, :)`, `intent(inout)`  $a$ , `real(dp)`, `dimension(:, :)`, `intent(inout)`  $b$ , `real(dp)`, `dimension(:)`, `intent(out)`, optional, target  $work$ , `integer(i32)`, `intent(out)`, optional  $olwork$ , `class(errors)`, `intent(inout)`, optional, target  $err$  ) [`private`]

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.

#### Parameters

<code>in, out</code>	$a$	On input, the $M$ -by- $N$ matrix $A$ . On output, if $M \geq N$ , the QR factorization of $A$ in the form as output by <code>qr_factor</code> ; else, if $M < N$ , the LQ factorization of $A$ .
<code>in, out</code>	$b$	If $M \geq N$ , the $M$ -by- $N$ -RHS matrix $B$ . On output, the first $N$ rows contain the $N$ -by- $N$ -RHS solution matrix $X$ . If $M < N$ , an $N$ -by- $N$ -RHS matrix with the first $M$ rows containing the matrix $B$ . On output, the $N$ -by- $N$ -RHS solution matrix $X$ .
<code>out</code>	$work$	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
<code>out</code>	$olwork$	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for $work$ , and returns without performing any actual calculations.
<code>out</code>	$err$	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> </ul>
Generated by Doxygen		<ul style="list-style-type: none"> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_INVALID_OPERATION_ERROR</code>: Occurs if <math>a</math> is not of full rank.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELS.

Definition at line 1568 of file linalg\_solve.f90.

**5.11.2.2** subroutine `linalg_solve::solve_least_squares::solve_least_squares_vec` ( `real(dp)`, `dimension(:, :)`, `intent(inout)` *a*, `real(dp)`, `dimension(:)`, `intent(inout)` *b*, `real(dp)`, `dimension(:)`, `intent(out)`, optional, target *work*, `integer(i32)`, `intent(out)`, optional *olwork*, `class(errors)`, `intent(inout)`, optional, target *err* ) [`private`]

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a QR or LQ factorization of the matrix  $A$ . Notice, it is assumed that matrix  $A$  has full rank.

## Parameters

<code>in, out</code>	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, if $M \geq N$ , the QR factorization of $A$ in the form as output by <code>qr_factor</code> ; else, if $M < N$ , the LQ factorization of $A$ .
<code>in, out</code>	<i>b</i>	If $M \geq N$ , the $M$ -element array $B$ . On output, the first $N$ elements contain the $N$ -element solution array $X$ . If $M < N$ , an $N$ -element array with the first $M$ elements containing the array $B$ . On output, the $N$ -element solution array $X$ .
<code>out</code>	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
<code>out</code>	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
<code>out</code>	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <code>LA_INVALID_OPERATION_ERROR</code>: Occurs if <i>a</i> is not of full rank.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELS.

Definition at line 1674 of file linalg\_solve.f90.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_solve.f90`

## 5.12 linalg\_solve::solve\_least\_squares\_full Interface Reference

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns, but uses a full orthogonal factorization of the system.

## Private Member Functions

- subroutine `solve_least_squares_mtx_pvt` (*a*, *b*, *ipvt*, *arnk*, *work*, *olwork*, *err*)  
Solves the overdetermined or underdetermined system ( $A * X = B$ ) of *M* equations of *N* unknowns using a complete orthogonal factorization of matrix *A*.
- subroutine `solve_least_squares_vec_pvt` (*a*, *b*, *ipvt*, *arnk*, *work*, *olwork*, *err*)  
Solves the overdetermined or underdetermined system ( $A * X = B$ ) of *M* equations of *N* unknowns using a complete orthogonal factorization of matrix *A*.

### 5.12.1 Detailed Description

Solves the overdetermined or underdetermined system ( $A * X = B$ ) of *M* equations of *N* unknowns, but uses a full orthogonal factorization of the system.

Definition at line 100 of file `linalg_solve.f90`.

### 5.12.2 Member Function/Subroutine Documentation

**5.12.2.1** subroutine `linalg_solve::solve_least_squares_full::solve_least_squares_mtx_pvt` ( *real*(*dp*), *dimension*(:,:), *intent*(*inout*) *a*, *real*(*dp*), *dimension*(:,:), *intent*(*inout*) *b*, *integer*(*i32*), *dimension*(:), *intent*(*inout*), *optional*, *target* *ipvt*, *integer*(*i32*), *intent*(*out*), *optional* *arnk*, *real*(*dp*), *dimension*(:), *intent*(*out*), *optional*, *target* *work*, *integer*(*i32*), *intent*(*out*), *optional* *olwork*, *class*(*errors*), *intent*(*inout*), *optional*, *target* *err* ) [private]

Solves the overdetermined or underdetermined system ( $A * X = B$ ) of *M* equations of *N* unknowns using a complete orthogonal factorization of matrix *A*.

#### Parameters

<i>in, out</i>	<i>a</i>	On input, the M-by-N matrix A. On output, the matrix is overwritten by the details of its complete orthogonal factorization.
<i>in, out</i>	<i>b</i>	If $M \geq N$ , the M-by-NRHS matrix B. On output, the first N rows contain the N-by-NRHS solution matrix X. If $M < N$ , an N-by-NRHS matrix with the first M rows containing the matrix B. On output, the N-by-NRHS solution matrix X.
<i>out</i>	<i>ipvt</i>	An optional input that on input, an N-element array that if $IPVT(I) \neq 0$ , the I-th column of A is permuted to the front of $A * P$ ; if $IPVT(I) = 0$ , the I-th column of A is a free column. On output, if $IPVT(I) = K$ , then the I-th column of $A * P$ was the K-th column of A. If not supplied, memory is allocated internally, and IPVT is set to all zeros such that all columns are treated as free.
<i>out</i>	<i>arnk</i>	An optional output, that if provided, will return the rank of <i>a</i> .
<i>out</i>	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
<i>out</i>	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
<i>out</i>	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSY.

Definition at line 1786 of file linalg\_solve.f90.

**5.12.2.2** subroutine linalg\_solve::solve\_least\_squares\_full::solve\_least\_squares\_vec\_pvt ( real(dp), dimension(:, :), intent(inout) *a*, real(dp), dimension(:), intent(inout) *b*, integer(i32), dimension(:), intent(inout), optional, target *ipvt*, integer(i32), intent(out), optional *arnk*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a complete orthogonal factorization of matrix  $A$ .

## Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the $M$ -element array $B$ . On output, the first $N$ elements contain the $N$ -element solution array $X$ . If $M < N$ , an $N$ -element array with the first $M$ elements containing the array $B$ . On output, the $N$ -element solution array $X$ .
out	<i>ipvt</i>	An optional input that on input, an $N$ -element array that if $IPVT(l) \neq 0$ , the $l$ -th column of $A$ is permuted to the front of $A \cdot P$ ; if $IPVT(l) = 0$ , the $l$ -th column of $A$ is a free column. On output, if $IPVT(l) = K$ , then the $l$ -th column of $A \cdot P$ was the $K$ -th column of $A$ . If not supplied, memory is allocated internally, and $IPVT$ is set to all zeros such that all columns are treated as free.
out	<i>arnk</i>	An optional output, that if provided, will return the rank of $a$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSY.

Definition at line 1932 of file linalg\_solve.f90.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_solve.f90`

## 5.13 linalg\_solve::solve\_least\_squares\_svd Interface Reference

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

### Private Member Functions

- subroutine [solve\\_least\\_squares\\_mtx\\_svd](#) ( $a$ ,  $b$ ,  $arnk$ ,  $s$ ,  $work$ ,  $olwork$ ,  $err$ )  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .
- subroutine [solve\\_least\\_squares\\_vec\\_svd](#) ( $a$ ,  $b$ ,  $arnk$ ,  $s$ ,  $work$ ,  $olwork$ ,  $err$ )  
Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

### 5.13.1 Detailed Description

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

Definition at line 109 of file `linalg_solve.f90`.

### 5.13.2 Member Function/Subroutine Documentation

**5.13.2.1** subroutine `linalg_solve::solve_least_squares_svd::solve_least_squares_mtx_svd` ( `real(dp)`, `dimension(:, :)`, `intent(inout) a`, `real(dp)`, `dimension(:, :)`, `intent(inout) b`, `integer(i32)`, `intent(out)`, optional `arnk`, `real(dp)`, `dimension(:, :)`, `intent(out) s`, `real(dp)`, `dimension(:, :)`, `intent(out)`, optional, target `work`, `integer(i32)`, `intent(out)`, optional `olwork`, `class(errors)`, `intent(inout)`, optional, target `err` ) [`private`]

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

#### Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the $M$ -by- $N$ -RHS matrix $B$ . On output, the first $N$ rows contain the $N$ -by- $N$ -RHS solution matrix $X$ . If $M < N$ , an $N$ -by- $N$ -RHS matrix with the first $M$ rows containing the matrix $B$ . On output, the $N$ -by- $N$ -RHS solution matrix $X$ .
out	<i>arnk</i>	An optional output, that if provided, will return the rank of $a$ .
out	<i>s</i>	A $\text{MIN}(M, N)$ -element array that on output contains the singular values of $a$ in descending order. Notice, the condition number of $a$ can be determined by $S(1) / S(\text{MIN}(M, N))$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.

## Parameters

out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>
-----	------------	--

## Notes

This routine utilizes the LAPACK routine DGELSS.

Definition at line 2076 of file linalg\_solve.f90.

**5.13.2.2** subroutine linalg\_solve::solve\_least\_squares\_svd::solve\_least\_squares\_vec\_svd ( real(dp), dimension(:,:), intent(inout) *a*, real(dp), dimension(:), intent(inout) *b*, integer(i32), intent(out), optional *arnk*, real(dp), dimension(:), intent(out) *s*, real(dp), dimension(:), intent(out), optional, target *work*, integer(i32), intent(out), optional *olwork*, class(errors), intent(inout), optional, target *err* ) [private]

Solves the overdetermined or underdetermined system ( $A \cdot X = B$ ) of  $M$  equations of  $N$  unknowns using a singular value decomposition of matrix  $A$ .

## Parameters

in, out	<i>a</i>	On input, the $M$ -by- $N$ matrix $A$ . On output, the matrix is overwritten by the details of its complete orthogonal factorization.
in, out	<i>b</i>	If $M \geq N$ , the $M$ -by- $NRHS$ matrix $B$ . On output, the first $N$ rows contain the $N$ -by- $NRHS$ solution matrix $X$ . If $M < N$ , an $N$ -by- $NRHS$ matrix with the first $M$ rows containing the matrix $B$ . On output, the $N$ -by- $NRHS$ solution matrix $X$ .
out	<i>arnk</i>	An optional output, that if provided, will return the rank of $a$ .
out	<i>s</i>	A $\min(M, N)$ -element array that on output contains the singular values of $a$ in descending order. Notice, the condition number of $a$ can be determined by $S(1) / S(\min(M, N))$ .
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>
		<ul style="list-style-type: none"> <li>• <b>LA_CONVERGENCE_ERROR</b>: Occurs as a warning if the QR iteration process could not converge to a zero value.</li> </ul>

## Notes

This routine utilizes the LAPACK routine DGELSS.

Definition at line 2204 of file linalg\_solve.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/linalg/src/linalg\_solve.f90

## 5.14 linalg\_solve::solve\_lu Interface Reference

Solves a system of LU-factored equations.

### Private Member Functions

- subroutine [solve\\_lu\\_mtx](#) (a, ipvt, b, err)  
*Solves a system of LU-factored equations.*
- subroutine [solve\\_lu\\_vec](#) (a, ipvt, b, err)  
*Solves a system of LU-factored equations.*

### 5.14.1 Detailed Description

Solves a system of LU-factored equations.

## Usage

To solve a system of N equations of N unknowns using LU factorization, the following code will suffice.

```
! Solve the system: A*X = B, where A is an N-by-N matrix, and B and X are
! N-by-NRHS in size.

! Variables
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b

! Define the array used to track row pivots.
integer(i32), dimension(n) :: pvt

! Initialize A and B...

! Compute the LU factorization of A. On output, A contains [L\U].
call lu_factor(a, pvt)

! Solve A*X = B for X - Note: X overwrites B.
call solve_lu(a, pvt, b)
```

## See Also

- [Wikipedia](#)
- [Wolfram MathWorld](#)

Definition at line 63 of file linalg\_solve.f90.

### 5.14.2 Member Function/Subroutine Documentation

**5.14.2.1** subroutine `linalg_solve::solve_lu::solve_lu_mtx` ( `real(dp), dimension(:, :), intent(in) a`, `integer(i32), dimension(:), intent(in) ipvt`, `real(dp), dimension(:, :), intent(inout) b`, `class(errors), intent(inout), optional, target err` )  
[private]

Solves a system of LU-factored equations.

## Parameters

in	<i>a</i>	The N-by-N LU factored matrix as output by <code>lu_factor</code> .
in	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
in, out	<i>b</i>	On input, the N-by-NRHS right-hand-side matrix. On output, the N-by-NRHS solution matrix.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

The routine is based upon the LAPACK routine DGETRS.

Definition at line 351 of file `linalg_solve.f90`.

```
5.14.2.2 subroutine linalg_solve::solve_lu::solve_lu_vec ( real(dp), dimension(:,:), intent(in) a, integer(i32), dimension(:),
               intent(in) ipvt, real(dp), dimension(:), intent(inout) b, class(errors), intent(inout), optional, target err ) [private]
```

Solves a system of LU-factored equations.

## Parameters

in	<i>a</i>	The N-by-N LU factored matrix as output by <code>lu_factor</code> .
in	<i>ipvt</i>	The N-element pivot array as output by <code>lu_factor</code> .
in, out	<i>b</i>	On input, the N-element right-hand-side array. On output, the N-element solution array.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input array sizes are incorrect.</li> </ul>

## Notes

The routine is based upon the LAPACK routine DGETRS.

Definition at line 412 of file `linalg_solve.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_solve.f90`

## 5.15 linalg\_solve::solve\_qr Interface Reference

Solves a system of M QR-factored equations of N unknowns.



## Private Member Functions

- subroutine [solve\\_qr\\_no\\_pivot\\_mtx](#) (a, tau, b, work, olwork, err)  
*Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .*
- subroutine [solve\\_qr\\_no\\_pivot\\_vec](#) (a, tau, b, work, olwork, err)  
*Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .*
- subroutine [solve\\_qr\\_pivot\\_mtx](#) (a, tau, jpvt, b, work, olwork, err)  
*Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.*
- subroutine [solve\\_qr\\_pivot\\_vec](#) (a, tau, jpvt, b, work, olwork, err)  
*Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.*

### 5.15.1 Detailed Description

Solves a system of M QR-factored equations of N unknowns.

#### See Also

- [Wikipedia](#)
- [LAPACK Users Manual](#)

Definition at line 74 of file linalg\_solve.f90.

### 5.15.2 Member Function/Subroutine Documentation

**5.15.2.1** subroutine linalg\_solve::solve\_qr::solve\_qr\_no\_pivot\_mtx ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :), intent(in) tau, real(dp), dimension(:, :), intent(inout) b, real(dp), dimension(:, :), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]

Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .

#### Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by qr_factor. On output, the contents of this matrix are restored. Notice, M must be greater than or equal to N.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by qr_factor.
in	<i>b</i>	On input, the M-by-NRHS right-hand-side matrix. On output, the first N columns are overwritten by the solution matrix X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <i>olwork</i> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <i>work</i> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <b>LA_ARRAY_SIZE_ERROR</b>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <b>LA_OUT_OF_MEMORY_ERROR</b>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
Generated	by Doxygen	

## Notes

This routine is based upon a subset of the LAPACK routine DGELS.

Definition at line 487 of file linalg\_solve.f90.

**5.15.2.2** `subroutine linalg_solve::solve_qr::solve_qr_no_pivot_vec ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :), intent(in) tau, real(dp), dimension(:, :), intent(inout) b, real(dp), dimension(:, :), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of M QR-factored equations of N unknowns where  $M \geq N$ .

## Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are restored. Notice, M must be greater than or equal to N.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>b</i>	On input, the M-element right-hand-side vector. On output, the first N elements are overwritten by the solution vector X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if any of the input arrays are not sized appropriately.</li> <li>LA_OUT_OF_MEMORY_ERROR: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon a subset of the LAPACK routine DGELS.

Definition at line 602 of file linalg\_solve.f90.

**5.15.2.3** `subroutine linalg_solve::solve_qr::solve_qr_pivot_mtx ( real(dp), dimension(:, :), intent(inout) a, real(dp), dimension(:, :), intent(in) tau, integer(i32), dimension(:, :), intent(in) jpvt, real(dp), dimension(:, :), intent(inout) b, real(dp), dimension(:, :), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.

## Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are altered.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>jpvt</i>	An N-element array, as output by <code>qr_factor</code> , used to track the column pivots.
in	<i>b</i>	On input, the MAX(M, N)-by-NRHS matrix where the first M rows contain the right-hand-side matrix B. On output, the first N rows are overwritten by the solution matrix X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>

## Notes

This routine is based upon a subset of the LAPACK routine DGELSY.

Definition at line 715 of file `linalg_solve.f90`.

**5.15.2.4** `subroutine linalg_solve::solve_qr::solve_qr_pivot_vec ( real(dp), dimension(:,:), intent(inout) a, real(dp), dimension(:), intent(in) tau, integer(i32), dimension(:), intent(in) jpvt, real(dp), dimension(:), intent(inout) b, real(dp), dimension(:), intent(out), optional, target work, integer(i32), intent(out), optional olwork, class(errors), intent(inout), optional, target err ) [private]`

Solves a system of M QR-factored equations of N unknowns where the QR factorization made use of column pivoting.

## Parameters

in	<i>a</i>	On input, the M-by-N QR factored matrix as returned by <code>qr_factor</code> . On output, the contents of this matrix are altered.
in	<i>tau</i>	A MIN(M, N)-element array containing the scalar factors of the elementary reflectors as returned by <code>qr_factor</code> .
in	<i>jpvt</i>	An N-element array, as output by <code>qr_factor</code> , used to track the column pivots.
in	<i>b</i>	On input, the MAX(M, N)-element array where the first M elements contain the right-hand-side vector B. On output, the first N elements are overwritten by the solution vector X.
out	<i>work</i>	An optional input, that if provided, prevents any local memory allocation. If not provided, the memory required is allocated within. If provided, the length of the array must be at least <code>olwork</code> .
out	<i>olwork</i>	An optional output used to determine workspace size. If supplied, the routine determines the optimal size for <code>work</code> , and returns without performing any actual calculations.

## Parameters

<code>out</code>	<code>err</code>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if any of the input arrays are not sized appropriately.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if local memory must be allocated, and there is insufficient memory available.</li> </ul>
------------------	------------------	--

## Notes

This routine is based upon a subset of the LAPACK routine DGELSY.

Definition at line 902 of file `linalg_solve.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_solve.f90`

## 5.16 linalg\_solve::solve\_triangular\_system Interface Reference

Solves a triangular system of equations.

### Private Member Functions

- subroutine `solve_tri_mtx` (`lside`, `upper`, `trans`, `nounit`, `alpha`, `a`, `b`, `err`)  
*Solves one of the matrix equations:  $op(A) * X = \alpha * B$ , or  $X * op(A) = \alpha * B$ , where  $A$  is a triangular matrix.*
- subroutine `solve_tri_vec` (`upper`, `trans`, `nounit`, `a`, `x`, `err`)  
*Solves the system of equations:  $op(A) * X = B$ , where  $A$  is a triangular matrix.*

### 5.16.1 Detailed Description

Solves a triangular system of equations.

Definition at line 29 of file `linalg_solve.f90`.

### 5.16.2 Member Function/Subroutine Documentation

**5.16.2.1** subroutine `linalg_solve::solve_triangular_system::solve_tri_mtx` ( `logical`, intent(in) *lside*, `logical`, intent(in) *upper*, `logical`, intent(in) *trans*, `logical`, intent(in) *nounit*, `real(dp)`, intent(in) *alpha*, `real(dp)`, dimension(:,:) *a*, `real(dp)`, dimension(:,:) *b*, `class(errors)`, intent(inout) *err* ) [private]

Solves one of the matrix equations:  $op(A) * X = \alpha * B$ , or  $X * op(A) = \alpha * B$ , where  $A$  is a triangular matrix.

## Parameters

in	<i>lside</i>	Set to true to solve $\text{op}(A) * X = \alpha * B$ ; else, set to false to solve $X * \text{op}(A) = \alpha * B$ .
in	<i>upper</i>	Set to true if A is an upper triangular matrix; else, set to false if A is a lower triangular matrix.
in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false if $\text{op}(A) = A$ .
in	<i>nounit</i>	Set to true if A is not a unit-diagonal matrix (ones on every diagonal element); else, set to false if A is a unit-diagonal matrix.
in	<i>alpha</i>	The scalar multiplier to B.
in	<i>a</i>	If <i>lside</i> is true, the M-by-M triangular matrix on which to operate; else, if <i>lside</i> is false, the N-by-N triangular matrix on which to operate.
in, out	<i>b</i>	On input, the M-by-N right-hand-side. On output, the M-by-N solution.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>a</i> is not square, or if the sizes of <i>a</i> and <i>b</i> are not compatible.</li> </ul>

## Usage

To solve a triangular system of N equations of N unknowns  $A * X = B$ , where A is an N-by-N upper triangular matrix, and B and X are N-by-NRHS matrices, the following code will suffice.

```
! Solve the system: A*X = B, where A is an upper triangular N-by-N
! matrix, and B and X are N-by-NRHS in size.

! Variables
integer(i32) :: info
real(dp), dimension(n, n) :: a
real(dp), dimension(n, nrhs) :: b

! Initialize A and B...

! Solve A*X = B for X - Note: X overwrites B.
call solve_triangular_system(.true., .true., .false., .true., &
    1.0d0, a, b)
```

## Notes

This routine is based upon the BLAS routine DTRSM.

Definition at line 169 of file linalg\_solve.f90.

**5.16.2.2** subroutine linalg\_solve::solve\_triangular\_system::solve\_tri\_vec ( logical, intent(in) *upper*, logical, intent(in) *trans*, logical, intent(in) *nounit*, real(dp), dimension(:, :), intent(in) *a*, real(dp), dimension(:), intent(inout) *x*, class(errors), intent(inout), optional, target *err* ) [private]

Solves the system of equations:  $\text{op}(A) * X = B$ , where A is a triangular matrix.

## Parameters

in	<i>upper</i>	Set to true if A is an upper triangular matrix; else, set to false if A is a lower triangular matrix.
in	<i>trans</i>	Set to true if $\text{op}(A) = A^{**T}$ ; else, set to false if $\text{op}(A) = A$ .

## Parameters

in	<i>nounit</i>	Set to true if A is not a unit-diagonal matrix (ones on every diagonal element); else, set to false if A is a unit-diagonal matrix.
in	<i>a</i>	The N-by-N triangular matrix.
in, out	<i>x</i>	On input, the N-element right-hand-side array. On output, the N-element solution array.
out	<i>err</i>	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>a</i> is not square, or if the sizes of <i>a</i> and <i>b</i> are not compatible.</li> </ul>

## Usage

To solve a triangular system of N equations of N unknowns  $A \cdot X = B$ , where A is an N-by-N upper triangular matrix, and B and X are N-element arrays, the following code will suffice.

```
! Solve the system: A*X = B, where A is an upper triangular N-by-N
! matrix, and B and X are N-elements in size.

! Variables
integer(i32) :: info
real(dp), dimension(n, n) :: a
real(dp), dimension(n) :: b

! Initialize A and B...

! Solve A*X = B for X - Note: X overwrites B.
call solve_triangular_system(.true., .false., a, b)
```

## Notes

This routine is based upon the BLAS routine DTRSV.

Definition at line 275 of file linalg\_solve.f90.

The documentation for this interface was generated from the following file:

- /home/jason/Documents/Code/linalg/src/linalg\_solve.f90

## 5.17 linalg\_sorting::sort Interface Reference

Sorts an array.

## Private Member Functions

- subroutine `sort_dbl_array` (`x`, `ascend`)  
*Sorts an array.*
- subroutine `sort_dbl_array_ind` (`x`, `ind`, `ascend`, `err`)  
*Sorts an array.*
- subroutine `sort_cmplx_array` (`x`, `ascend`)  
*Sorts an array.*
- subroutine `sort_cmplx_array_ind` (`x`, `ind`, `ascend`, `err`)  
*Sorts an array.*
- subroutine `sort_eigen_cmplx` (`vals`, `vecs`, `ascend`, `err`)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*
- subroutine `sort_eigen_dbl` (`vals`, `vecs`, `ascend`, `err`)  
*A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.*

### 5.17.1 Detailed Description

Sorts an array.

Definition at line 18 of file `linalg_sorting.f90`.

### 5.17.2 Member Function/Subroutine Documentation

5.17.2.1 subroutine `linalg_sorting::sort::sort_cmplx_array` ( `complex(dp)`, `dimension(:)`, `intent(inout) x`, `logical`, `intent(in)`, optional `ascend` ) [`private`]

Sorts an array.

#### Parameters

<code>in, out</code>	<code>x</code>	On input, the array to sort. On output, the sorted array.
<code>in</code>	<code>ascend</code>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.

#### Remarks

This routine utilizes a quick sort algorithm. As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

#### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 156 of file `linalg_sorting.f90`.

5.17.2.2 **subroutine** `linalg_sorting::sort::sort_cmplx_array_ind` ( `complex(dp)`, `dimension(:)`, `intent(inout)` *x*, `integer(i32)`, `dimension(:)`, `intent(inout)` *ind*, `logical`, `intent(in)`, optional *ascend*, `class(errors)`, `intent(inout)`, optional, target *err* )  
[private]

Sorts an array.

#### Parameters

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, an integer array. On output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>ind</i> is not sized to match <i>x</i>.</li> </ul>

#### Remarks

This routine utilizes a quick sort algorithm. As this routine operates on complex valued items, the complex values are sorted based upon the real component of the number.

#### Notes

This implementation is a slight modification of the code presented at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 205 of file `linalg_sorting.f90`.

5.17.2.3 **subroutine** `linalg_sorting::sort::sort_dbl_array` ( `real(dp)`, `dimension(:)`, `intent(inout)` *x*, `logical`, `intent(in)`, optional *ascend* ) [private]

Sorts an array.

#### Parameters

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.

#### Remarks

The routine utilizes a quick sort algorithm unless the size of the array is less than or equal to 20. For such small arrays an insertion sort algorithm is utilized.



## Notes

This routine utilizes the LAPACK routine DLASRT.

Definition at line 47 of file linalg\_sorting.f90.

**5.17.2.4** subroutine linalg\_sorting::sort::sort\_dbl\_array\_ind ( real(dp), dimension(:), intent(inout) *x*, integer(i32), dimension(:), intent(inout) *ind*, logical, intent(in), optional *ascend*, class(errors), intent(inout), optional, target *err* ) [private]

Sorts an array.

## Parameters

in, out	<i>x</i>	On input, the array to sort. On output, the sorted array.
in, out	<i>ind</i>	On input, an integer array. On output, the contents of this array are shifted in the same order as that of <i>x</i> as a means of tracking the sorting operation. It is often useful to set this array to an ascending group of values (1, 2, ... n) such that this array tracks the original positions of the sorted array. Such an array can then be used to align other arrays. This array must be the same size as <i>x</i> .
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
out	<i>err</i>	An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows. <ul style="list-style-type: none"> <li>LA_ARRAY_SIZE_ERROR: Occurs if <i>ind</i> is not sized to match <i>x</i>.</li> </ul>

## Remarks

This routine utilizes a quick sort algorithm explained at [http://www.fortran.com/qsort\\_c.f95](http://www.fortran.com/qsort_c.f95).

Definition at line 97 of file linalg\_sorting.f90.

**5.17.2.5** subroutine linalg\_sorting::sort::sort\_eigen\_cmplx ( complex(dp), dimension(:), intent(inout) *vals*, complex(dp), dimension(:,), intent(inout) *vecs*, logical, intent(in), optional *ascend*, class(errors), intent(inout), optional, target *err* ) [private]

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

## Parameters

in, out	<i>vals</i>	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
in, out	<i>vecs</i>	On input, an N-by-N matrix containing the eigenvectors associated with <i>vals</i> (one vector per column). On output, the sorted eigenvector matrix.
in	<i>ascend</i>	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.

## Parameters

out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <code>vecs</code> is not sized to match <code>vals</code>.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if there is insufficient memory available to complete this operation.</li> </ul>
-----	-----	---

Definition at line 267 of file `linalg_sorting.f90`.

5.17.2.6 subroutine `linalg_sorting::sort::sort_eigen_dbl` ( `real(dp)`, `dimension(:)`, `intent(inout) vals`, `real(dp)`, `dimension(:, :)`, `intent(inout) vecs`, `logical`, `intent(in)`, optional `ascend`, `class(errors)`, `intent(inout)`, optional, target `err` )  
`[private]`

A sorting routine specifically tailored for sorting of eigenvalues and their associated eigenvectors using a quick-sort approach.

## Parameters

in, out	vals	On input, an N-element array containing the eigenvalues. On output, the sorted eigenvalues.
in, out	vecs	On input, an N-by-N matrix containing the eigenvectors associated with <code>vals</code> (one vector per column). On output, the sorted eigenvector matrix.
in	ascend	An optional input that, if specified, controls if the the array is sorted in an ascending order (default), or a descending order.
out	err	<p>An optional errors-based object that if provided can be used to retrieve information relating to any errors encountered during execution. If not provided, a default implementation of the errors class is used internally to provide error handling. Possible errors and warning messages that may be encountered are as follows.</p> <ul style="list-style-type: none"> <li>• <code>LA_ARRAY_SIZE_ERROR</code>: Occurs if <code>vecs</code> is not sized to match <code>vals</code>.</li> <li>• <code>LA_OUT_OF_MEMORY_ERROR</code>: Occurs if there is insufficient memory available to complete this operation.</li> </ul>

Definition at line 345 of file `linalg_sorting.f90`.

The documentation for this interface was generated from the following file:

- `/home/jason/Documents/Code/linalg/src/linalg_sorting.f90`

# Index

- cholesky\_factor
  - linalg\_factor, [44](#)
- cholesky\_factor\_c
  - linalg\_c\_binding, [9](#)
- cholesky\_rank1\_downdate
  - linalg\_factor, [44](#)
- cholesky\_rank1\_downdate\_c
  - linalg\_c\_binding, [10](#)
- cholesky\_rank1\_update
  - linalg\_factor, [45](#)
- cholesky\_rank1\_update\_c
  - linalg\_c\_binding, [10](#)
- cmplx\_partition
  - linalg\_sorting, [75](#)
- cmplx\_partition\_ind
  - linalg\_sorting, [75](#)
- cmtx\_mult\_c
  - linalg\_c\_binding, [11](#)
- dbl\_partition\_ind
  - linalg\_sorting, [75](#)
- det
  - linalg\_core, [31](#)
- det\_c
  - linalg\_c\_binding, [11](#)
- diag\_cmtx\_mult\_left\_c
  - linalg\_c\_binding, [12](#)
- diag\_cmtx\_mult\_right\_c
  - linalg\_c\_binding, [12](#)
- diag\_mtx\_mult\_cmplx\_left\_c
  - linalg\_c\_binding, [13](#)
- diag\_mtx\_mult\_cmplx\_right\_c
  - linalg\_c\_binding, [13](#)
- diag\_mtx\_mult\_left\_c
  - linalg\_c\_binding, [13](#)
- diag\_mtx\_mult\_mtx
  - linalg\_core, [32](#)
  - linalg\_core::diag\_mtx\_mult, [81](#)
- diag\_mtx\_mult\_mtx2
  - linalg\_core, [33](#)
  - linalg\_core::diag\_mtx\_mult, [82](#)
- diag\_mtx\_mult\_mtx3
  - linalg\_core, [33](#)
  - linalg\_core::diag\_mtx\_mult, [82](#)
- diag\_mtx\_mult\_mtx4
  - linalg\_core, [34](#)
  - linalg\_core::diag\_mtx\_mult, [83](#)
- disg\_mtx\_mult\_right\_c
  - linalg\_c\_binding, [14](#)

- eigen\_asymm
  - linalg\_eigen, [39](#)
  - linalg\_eigen::eigen, [85](#)
- eigen\_asymm\_c
  - linalg\_c\_binding, [14](#)
- eigen\_gen
  - linalg\_eigen, [40](#)
  - linalg\_eigen::eigen, [86](#)
- eigen\_gen\_c
  - linalg\_c\_binding, [15](#)
- eigen\_symm
  - linalg\_eigen, [41](#)
  - linalg\_eigen::eigen, [87](#)
- eigen\_symm\_c
  - linalg\_c\_binding, [15](#)
- form\_lu\_all
  - linalg\_factor, [46](#)
  - linalg\_factor::form\_lu, [89](#)
- form\_lu\_c
  - linalg\_c\_binding, [16](#)
- form\_lu\_only
  - linalg\_factor, [46](#)
  - linalg\_factor::form\_lu, [89](#)
- form\_qr\_c
  - linalg\_c\_binding, [16](#)
- form\_qr\_no\_pivot
  - linalg\_factor, [47](#)
  - linalg\_factor::form\_qr, [90](#)
- form\_qr\_pivot
  - linalg\_factor, [47](#)
  - linalg\_factor::form\_qr, [91](#)
- form\_qr\_pivot\_c
  - linalg\_c\_binding, [17](#)
- lapack, [7](#)
- lapack::DLAMCH, [84](#)
- linalg\_c\_binding, [7](#)
  - cholesky\_factor\_c, [9](#)
  - cholesky\_rank1\_downdate\_c, [10](#)
  - cholesky\_rank1\_update\_c, [10](#)
  - cmtx\_mult\_c, [11](#)
  - det\_c, [11](#)
  - diag\_cmtx\_mult\_left\_c, [12](#)
  - diag\_cmtx\_mult\_right\_c, [12](#)
  - diag\_mtx\_mult\_cmplx\_left\_c, [13](#)
  - diag\_mtx\_mult\_cmplx\_right\_c, [13](#)
  - diag\_mtx\_mult\_left\_c, [13](#)
  - disg\_mtx\_mult\_right\_c, [14](#)
  - eigen\_asymm\_c, [14](#)

- eigen\_gen\_c, 15
- eigen\_symm\_c, 15
- form\_lu\_c, 16
- form\_qr\_c, 16
- form\_qr\_pivot\_c, 17
- lu\_factor\_c, 18
- mtx\_inverse\_c, 18
- mtx\_mult\_c, 19
- mtx\_pinverse\_c, 19
- mtx\_rank\_c, 20
- mult\_qr\_c, 20
- mult\_rz\_c, 21
- qr\_factor\_c, 21
- qr\_factor\_pivot\_c, 22
- qr\_rank1\_update\_c, 23
- rank1\_update\_c, 23
- rz\_factor\_c, 23
- solve\_cholesky\_c, 24
- solve\_least\_squares\_c, 24
- solve\_lu\_c, 25
- solve\_qr\_c, 25
- solve\_qr\_pivot\_c, 26
- solve\_tri\_mtx\_c, 26
- sort\_cmplx\_ind\_c, 27
- sort\_dbl\_ind\_c, 27
- sort\_eigen\_cmplx\_c, 28
- sort\_eigen\_dbl\_c, 28
- svd\_c, 28
- swap\_c, 29
- trace\_c, 29
- linalg\_constants, 30
- linalg\_core, 30
  - det, 31
  - diag\_mtx\_mult\_mtx, 32
  - diag\_mtx\_mult\_mtx2, 33
  - diag\_mtx\_mult\_mtx3, 33
  - diag\_mtx\_mult\_mtx4, 34
  - mtx\_mult\_mtx, 34
  - mtx\_mult\_vec, 35
  - mtx\_rank, 36
  - rank1\_update, 36
  - recip\_mult\_array, 37
  - swap, 37
  - trace, 38
  - tri\_mtx\_mult, 38
- linalg\_core::diag\_mtx\_mult, 81
  - diag\_mtx\_mult\_mtx, 81
  - diag\_mtx\_mult\_mtx2, 82
  - diag\_mtx\_mult\_mtx3, 82
  - diag\_mtx\_mult\_mtx4, 83
- linalg\_core::mtx\_mult, 92
  - mtx\_mult\_mtx, 92
  - mtx\_mult\_vec, 93
- linalg\_eigen, 38
  - eigen\_asymm, 39
  - eigen\_gen, 40
  - eigen\_symm, 41
- linalg\_eigen::eigen, 84
  - eigen\_asymm, 85
  - eigen\_gen, 86
  - eigen\_symm, 87
- linalg\_factor, 42
  - cholesky\_factor, 44
  - cholesky\_rank1\_downdate, 44
  - cholesky\_rank1\_update, 45
  - form\_lu\_all, 46
  - form\_lu\_only, 46
  - form\_qr\_no\_pivot, 47
  - form\_qr\_pivot, 47
  - lu\_factor, 48
  - mult\_qr\_mtx, 49
  - mult\_qr\_vec, 50
  - mult\_rz\_mtx, 51
  - mult\_rz\_vec, 52
  - qr\_factor\_no\_pivot, 52
  - qr\_factor\_pivot, 54
  - qr\_rank1\_update, 55
  - rz\_factor, 56
  - svd, 57
- linalg\_factor::form\_lu, 88
  - form\_lu\_all, 89
  - form\_lu\_only, 89
- linalg\_factor::form\_qr, 90
  - form\_qr\_no\_pivot, 90
  - form\_qr\_pivot, 91
- linalg\_factor::mult\_qr, 94
  - mult\_qr\_mtx, 94
  - mult\_qr\_vec, 95
- linalg\_factor::mult\_rz, 96
  - mult\_rz\_mtx, 96
  - mult\_rz\_vec, 97
- linalg\_factor::qr\_factor, 98
  - qr\_factor\_no\_pivot, 98
  - qr\_factor\_pivot, 100
- linalg\_solve, 59
  - mtx\_inverse, 60
  - mtx\_pinverse, 61
  - solve\_cholesky\_mtx, 62
  - solve\_cholesky\_vec, 63
  - solve\_least\_squares\_mtx, 63
  - solve\_least\_squares\_mtx\_pvt, 64
  - solve\_least\_squares\_mtx\_svd, 65
  - solve\_least\_squares\_vec, 66
  - solve\_least\_squares\_vec\_pvt, 66
  - solve\_least\_squares\_vec\_svd, 67
  - solve\_lu\_mtx, 68
  - solve\_lu\_vec, 68
  - solve\_qr\_no\_pivot\_mtx, 69
  - solve\_qr\_no\_pivot\_vec, 70
  - solve\_qr\_pivot\_mtx, 70
  - solve\_qr\_pivot\_vec, 71
  - solve\_tri\_mtx, 72
  - solve\_tri\_vec, 73
- linalg\_solve::solve\_cholesky, 101
  - solve\_cholesky\_mtx, 102
  - solve\_cholesky\_vec, 102

- linalg\_solve::solve\_least\_squares, 103
  - solve\_least\_squares\_mtx, 103
  - solve\_least\_squares\_vec, 104
- linalg\_solve::solve\_least\_squares\_full, 104
  - solve\_least\_squares\_mtx\_pvt, 105
  - solve\_least\_squares\_vec\_pvt, 106
- linalg\_solve::solve\_least\_squares\_svd, 107
  - solve\_least\_squares\_mtx\_svd, 107
  - solve\_least\_squares\_vec\_svd, 108
- linalg\_solve::solve\_lu, 109
  - solve\_lu\_mtx, 109
  - solve\_lu\_vec, 110
- linalg\_solve::solve\_qr, 110
  - solve\_qr\_no\_pivot\_mtx, 111
  - solve\_qr\_no\_pivot\_vec, 112
  - solve\_qr\_pivot\_mtx, 112
  - solve\_qr\_pivot\_vec, 113
- linalg\_solve::solve\_triangular\_system, 114
  - solve\_tri\_mtx, 114
  - solve\_tri\_vec, 115
- linalg\_sorting, 74
  - cmplx\_partition, 75
  - cmplx\_partition\_ind, 75
  - dbl\_partition\_ind, 75
  - qsort\_cmplx, 76
  - qsort\_cmplx\_ind, 76
  - qsort\_dbl\_ind, 77
  - sort\_cmplx\_array, 77
  - sort\_cmplx\_array\_ind, 78
  - sort\_dbl\_array, 78
  - sort\_dbl\_array\_ind, 79
  - sort\_eigen\_cmplx, 79
  - sort\_eigen\_dbl, 80
- linalg\_sorting::sort, 116
  - sort\_cmplx\_array, 117
  - sort\_cmplx\_array\_ind, 117
  - sort\_dbl\_array, 118
  - sort\_dbl\_array\_ind, 119
  - sort\_eigen\_cmplx, 119
  - sort\_eigen\_dbl, 120
- lu\_factor
  - linalg\_factor, 48
- lu\_factor\_c
  - linalg\_c\_binding, 18
- mtx\_inverse
  - linalg\_solve, 60
- mtx\_inverse\_c
  - linalg\_c\_binding, 18
- mtx\_mult\_c
  - linalg\_c\_binding, 19
- mtx\_mult\_mtx
  - linalg\_core, 34
  - linalg\_core::mtx\_mult, 92
- mtx\_mult\_vec
  - linalg\_core, 35
  - linalg\_core::mtx\_mult, 93
- mtx\_pinverse
  - linalg\_solve, 61
- mtx\_pinverse\_c
  - linalg\_c\_binding, 19
- mtx\_rank
  - linalg\_core, 36
- mtx\_rank\_c
  - linalg\_c\_binding, 20
- mult\_qr\_c
  - linalg\_c\_binding, 20
- mult\_qr\_mtx
  - linalg\_factor, 49
  - linalg\_factor::mult\_qr, 94
- mult\_qr\_vec
  - linalg\_factor, 50
  - linalg\_factor::mult\_qr, 95
- mult\_rz\_c
  - linalg\_c\_binding, 21
- mult\_rz\_mtx
  - linalg\_factor, 51
  - linalg\_factor::mult\_rz, 96
- mult\_rz\_vec
  - linalg\_factor, 52
  - linalg\_factor::mult\_rz, 97
- qr\_factor\_c
  - linalg\_c\_binding, 21
- qr\_factor\_no\_pivot
  - linalg\_factor, 52
  - linalg\_factor::qr\_factor, 98
- qr\_factor\_pivot
  - linalg\_factor, 54
  - linalg\_factor::qr\_factor, 100
- qr\_factor\_pivot\_c
  - linalg\_c\_binding, 22
- qr\_rank1\_update
  - linalg\_factor, 55
- qr\_rank1\_update\_c
  - linalg\_c\_binding, 23
- qsort\_cmplx
  - linalg\_sorting, 76
- qsort\_cmplx\_ind
  - linalg\_sorting, 76
- qsort\_dbl\_ind
  - linalg\_sorting, 77
- rank1\_update
  - linalg\_core, 36
- rank1\_update\_c
  - linalg\_c\_binding, 23
- recip\_mult\_array
  - linalg\_core, 37
- rz\_factor
  - linalg\_factor, 56
- rz\_factor\_c
  - linalg\_c\_binding, 23
- solve\_cholesky\_c
  - linalg\_c\_binding, 24
- solve\_cholesky\_mtx
  - linalg\_solve, 62

- `linalg_solve::solve_cholesky`, 102
- `solve_cholesky_vec`
  - `linalg_solve`, 63
  - `linalg_solve::solve_cholesky`, 102
- `solve_least_squares_c`
  - `linalg_c_binding`, 24
- `solve_least_squares_mtx`
  - `linalg_solve`, 63
  - `linalg_solve::solve_least_squares`, 103
- `solve_least_squares_mtx_pvt`
  - `linalg_solve`, 64
  - `linalg_solve::solve_least_squares_full`, 105
- `solve_least_squares_mtx_svd`
  - `linalg_solve`, 65
  - `linalg_solve::solve_least_squares_svd`, 107
- `solve_least_squares_vec`
  - `linalg_solve`, 66
  - `linalg_solve::solve_least_squares`, 104
- `solve_least_squares_vec_pvt`
  - `linalg_solve`, 66
  - `linalg_solve::solve_least_squares_full`, 106
- `solve_least_squares_vec_svd`
  - `linalg_solve`, 67
  - `linalg_solve::solve_least_squares_svd`, 108
- `solve_lu_c`
  - `linalg_c_binding`, 25
- `solve_lu_mtx`
  - `linalg_solve`, 68
  - `linalg_solve::solve_lu`, 109
- `solve_lu_vec`
  - `linalg_solve`, 68
  - `linalg_solve::solve_lu`, 110
- `solve_qr_c`
  - `linalg_c_binding`, 25
- `solve_qr_no_pivot_mtx`
  - `linalg_solve`, 69
  - `linalg_solve::solve_qr`, 111
- `solve_qr_no_pivot_vec`
  - `linalg_solve`, 70
  - `linalg_solve::solve_qr`, 112
- `solve_qr_pivot_c`
  - `linalg_c_binding`, 26
- `solve_qr_pivot_mtx`
  - `linalg_solve`, 70
  - `linalg_solve::solve_qr`, 112
- `solve_qr_pivot_vec`
  - `linalg_solve`, 71
  - `linalg_solve::solve_qr`, 113
- `solve_tri_mtx`
  - `linalg_solve`, 72
  - `linalg_solve::solve_triangular_system`, 114
- `solve_tri_mtx_c`
  - `linalg_c_binding`, 26
- `solve_tri_vec`
  - `linalg_solve`, 73
  - `linalg_solve::solve_triangular_system`, 115
- `sort_cmplx_array`
  - `linalg_sorting`, 77
- `linalg_sorting::sort`, 117
- `sort_cmplx_array_ind`
  - `linalg_sorting`, 78
  - `linalg_sorting::sort`, 117
- `sort_cmplx_ind_c`
  - `linalg_c_binding`, 27
- `sort_dbl_array`
  - `linalg_sorting`, 78
  - `linalg_sorting::sort`, 118
- `sort_dbl_array_ind`
  - `linalg_sorting`, 79
  - `linalg_sorting::sort`, 119
- `sort_dbl_ind_c`
  - `linalg_c_binding`, 27
- `sort_eigen_cmplx`
  - `linalg_sorting`, 79
  - `linalg_sorting::sort`, 119
- `sort_eigen_cmplx_c`
  - `linalg_c_binding`, 28
- `sort_eigen_dbl`
  - `linalg_sorting`, 80
  - `linalg_sorting::sort`, 120
- `sort_eigen_dbl_c`
  - `linalg_c_binding`, 28
- `svd`
  - `linalg_factor`, 57
- `svd_c`
  - `linalg_c_binding`, 28
- `swap`
  - `linalg_core`, 37
- `swap_c`
  - `linalg_c_binding`, 29
- `trace`
  - `linalg_core`, 38
- `trace_c`
  - `linalg_c_binding`, 29
- `tri_mtx_mult`
  - `linalg_core`, 38