

My Project

Generated by Doxygen 1.8.11

Contents

1	Todo List	1
2	Modules Index	3
2.1	Modules List	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	makeopticalelements Module Reference	7
4.1.1	Detailed Description	7
4.1.2	Function/Subroutine Documentation	7
4.1.2.1	ab <i>t</i> (<i>i</i> , <i>j</i> , <i>ft</i> , <i>nspec</i>)	7
4.1.2.2	alloc_temparrays(<i>n</i> space, <i>nspec</i>)	8
4.1.2.3	amp(<i>a</i>)	8
4.1.2.4	bb <i>d</i> (<i>i</i> , <i>j</i> , <i>ft</i> , <i>nspec</i>)	8
4.1.2.5	dealloc_temparrays	9
4.1.2.6	g4(<i>ft</i> , <i>nspec</i>)	9
4.1.2.7	make_bs(<i>n</i> space, <i>nspec</i> , <i>symp_mat</i> , <i>m</i> 1, <i>m</i> 2, <i>theta</i>)	9
4.1.2.8	make_sq(<i>n</i> space, <i>nspec</i> , <i>symp_mat</i> , <i>m</i> 1, <i>m</i> 2, <i>alpha</i> , <i>beta</i>)	9
4.1.3	Variable Documentation	10
4.1.3.1	ident	10
4.2	olis_f90stdlib Module Reference	10
4.2.1	Function/Subroutine Documentation	10
4.2.1.1	alloc_complex_eigenvecs(<i>matrix</i> , <i>eigenvals</i> , <i>u</i> , <i>v</i>)	10

4.2.1.2	<code>alloc_complex_svd(matrix, sigma, u, vt)</code>	11
4.2.1.3	<code>c_identity(n)</code>	11
4.2.1.4	<code>c_inv2(m_in)</code>	11
4.2.1.5	<code>complex_eigenvects(a, w, vl, vr)</code>	11
4.2.1.6	<code>complex_svd(a, sigma, u, vt)</code>	12
4.2.1.7	<code>complextrace(a)</code>	12
4.2.1.8	<code>expmatrix(matrix, n)</code>	12
4.2.1.9	<code>factorial(n)</code>	13
4.2.1.10	<code>matrixmul(x, n)</code>	13
4.2.1.11	<code>matrixnorm(c)</code>	13
4.2.1.12	<code>outerproduct(a, b)</code>	13
4.2.1.13	<code>printvectors(vect, desc, f)</code>	13
4.2.1.14	<code>randseed(seed)</code>	13
4.2.1.15	<code>tprod(a, b)</code>	14
4.2.2	Variable Documentation	14
4.2.2.1	<code>imaginary</code>	14
4.2.2.2	<code>pi</code>	14
5	File Documentation	15
5.1	<code>makeopticalelements.f90</code> File Reference	15
5.2	<code>num_hom.f90</code> File Reference	16
5.2.1	Function/Subroutine Documentation	16
5.2.1.1	<code>f(w1, w2, sig)</code>	16
5.2.1.2	<code>num_hom</code>	17
5.3	<code>olis_f90stdlib.f90</code> File Reference	17
Index		19

Chapter 1

Todo List

Subprogram [makeopticalelements::abt](#) (i, j, ft, nspec)

check this

Subprogram [makeopticalelements::bbd](#) (i, j, ft, nspec)

check this

Chapter 2

Modules Index

2.1 Modules List

Here is a list of all modules with brief descriptions:

makeopticalelements	Module for building symplectic matrices for optical elements	7
olis_f90stdlib	10

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

makeopticalelements.f90	15
num_hom.f90	16
olis_f90stdlib.f90	17

Chapter 4

Module Documentation

4.1 makeopticalelements Module Reference

module for building symplectic matrices for optical elements

Functions/Subroutines

- subroutine [make_bs](#) (nspace, nspec, symp_mat, m1, m2, theta)
makes beamsplitter symplectic matrix takes in an allocated matrix for the beamsplitter matrix to be written to uses the private ident_spec, spatial_work, n_work arrays
- subroutine [make_sq](#) (nspace, nspec, symp_mat, m1, m2, alpha, beta)
make symplectic squeezing matrix from exponentiated JSA a lot is broken...
- real(kind=dp) function [g4](#) (ft, nspec)
calculates g4 using matrix elements sum
- real(kind=dp) function [amp](#) (a)
returns the absolute value squared $|a|^2$
- complex(kind=dp) function [abt](#) (i, j, ft, nspec)
*calculates matrix elements Alpha-Beta^{**T} for $M = (A \ B) \ (B^* \ A^*)$ computes AB^{**T} and returns the i,j-th element*
- complex(kind=dp) function [bbd](#) (i, j, ft, nspec)
calculates the matrix elements $\text{Beta}^ \text{Beta}^{**H}$ for $M = (A \ B) \ (B^* \ A^*)$ computes $B^* B^{**H}$ (Hermitian conjg) and returns the i,j-th element*
- subroutine [alloc_temparrays](#) (nspace, nspec)
allocates temp arrays for matrices
- subroutine [dealloc_temparrays](#)

Variables

- real(kind=dp), public [ident](#)

4.1.1 Detailed Description

module for building symplectic matrices for optical elements

4.1.2 Function/Subroutine Documentation

4.1.2.1 complex(kind=dp) function `makeopticalelements::abt (integer i, integer j, complex(kind=dp), dimension(:,,:), intent(in), allocatable ft, integer nspec)`

calculates matrix elements Alpha-Beta^{**T} for $M = (A \ B) \ (B^* \ A^*)$ computes AB^{**T} and returns the i,j-th element

Parameters

<i>i</i>	input index 1
<i>j</i>	input index 2
<i>ft</i>	input symplectic transform matrix for the optical circuit
<i>nspec</i>	input number of spectral DOF

Todo check this

4.1.2.2 subroutine makeopticalelements::alloc_temparrays (integer, intent(in) *nspace*, integer, intent(in) *nspec*)

allocates temp arrays for matrices

Parameters

<i>nspace</i>	input
<i>nspec</i>	input allocates memory for ident_spec a spectral size matrix for tensor producing.

allocates mem for spatial_work, array size of spatial modes

allocates mem for n_work, work array size of alpha or beta in symplectic matrix

4.1.2.3 real(kind=dp) function makeopticalelements::amp (complex(kind=dp) *a*)

returns the absolute value squared $|a|^2$

Parameters

<i>a</i>	input complex number to be $ a ^2$
----------	------------------------------------

4.1.2.4 complex(kind=dp) function makeopticalelements::bbd (integer, intent(in) *i*, integer, intent(in) *j*, complex(kind=dp), dimension(:, :), intent(in), allocatable *ft*, integer, intent(in) *nspec*)

calculates the matrix elements $Beta * Beta^{**H}$ for $M = (A \ B) (B^* \ A^*)$ computes $B * B^{**H}$ (Hermitian conjg) and returns the i,j-th element

Parameters

<i>i</i>	input index 1
<i>j</i>	input index 2
<i>ft</i>	input symplectic transform matrix for the optical circuit
<i>nspec</i>	input number of spectral DOF

Todo check this

4.1.2.5 subroutine makeopticalelements::dealloc_temparrays ()

4.1.2.6 real(kind=dp) function makeopticalelements::g4 (complex(kind=dp), dimension(:, :), intent(in), allocatable ft, integer, intent(in) *nspec*)

calculates g4 using matrix elements sum

Parameters

<i>ft</i>	input is the full symplectic transform
<i>nspec</i>	input spectral DOF

4.1.2.7 subroutine makeopticalelements::make_bs (integer *nspace*, integer *nspec*, complex(kind=dp), dimension(:, :), allocatable *symp_mat*, integer *m1*, integer *m2*, real(kind=dp) *theta*)

makes beamsplitter symplectic matrix takes in an allocated matrix for the beamsplitter matrix to be written to uses the private ident_spec, spatial_work, n_work arrays

Parameters

<i>nspace</i>	is number of total spatial modes
<i>nspec</i>	is number of total spectral modes
<i>m_bs</i>	allocated n*n matrix for beamsplitter
<i>m1</i>	is spatial mode 1 for beam splitter
<i>m2</i>	is spatial mode 2 for beam splitter

4.1.2.8 subroutine makeopticalelements::make_sq (integer *nspace*, integer *nspec*, complex(kind=dp), dimension(:, :), allocatable *symp_mat*, integer *m1*, integer *m2*, complex(kind=dp), dimension(:, :), intent(inout) *alpha*, complex(kind=dp), dimension(:, :), intent(inout) *beta*)

make symplectic squeezing matrix from exponentiated JSA a lot is broken...

Note

only works if modes are consecutive

Note

alpha & beta are 2 spatial modes and all spectral modes dim 2*nspace*nspec

loop for alpha

check this is legal... full diag sq symp_mat(m1s:m1s+nspec, m1s+n:m1s+nspec+n)=beta(1:nspec, 1+nspec↵:2*nspec)

probably not legal symp_mat(m2s:m2s+nspec, m2s+n:m2s+nspec+n)=beta(nspec+1:2*nspec, 1:nspec)

loop for beta, offset to col+n

4.1.3 Variable Documentation

4.1.3.1 `real(kind=dp), public makeopticalelements::ident`

4.2 olis_f90stdlib Module Reference

Functions/Subroutines

- subroutine `alloc_complex_eigenvects` (matrix, eigenvals, u, v)
allocates eigenvals, u & v arrays for eigenvals & eigenvects
- subroutine `alloc_complex_svd` (matrix, sigma, u, vt)
allocates sigma (singular vals), u and vt for complexSVD allocates temp work arrays too
- subroutine `randseed` (seed)
generates random seed
- subroutine `printvectors` (vect, desc, f)
print formatted matrices can take optional args for labels or write directly to a file
- complex(kind=dp) function, dimension(2, 2) `outerproduct` (a, b)
outerproduct of two complex vectors, returns a complex matrix
- complex(kind=dp) function, dimension(n, n) `c_identity` (n)
makes complex identity matrix dim (nxn)
- complex(kind=dp) function, dimension(:, :), allocatable `tprod` (a, b)
tensor product for complex matrices aXb
- complex(kind=dp) function `complextrace` (a)
computes the trace of a complex matrix
- subroutine `complex_eigenvects` (a, w, vl, vr)
computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack
- subroutine `complex_svd` (a, sigma, u, vt)
computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack
- complex(kind=dp) function, dimension(2, 2) `c_inv2` (m_in)
inverse for a complex 2x2 matrix
- real(kind=dp) function `matrixnorm` (c)
computed Frobenius matrix norm of complex matrix using lapack zlange
- complex(kind=dp) function, dimension(size(matrix, 1), size(matrix, 2)) `expmatrix` (matrix, n)
- recursive complex(kind=dp) function, dimension(size(x, 1), size(x, 2)) `matrixmul` (x, n)
- recursive real(kind=dp) function `factorial` (n)

Variables

- real(kind=dp), parameter `pi` =4.0_dp*atan(1.0)
- complex(kind=dp), parameter `imaginary` =(0.0_dp, 1.0_dp)

4.2.1 Function/Subroutine Documentation

4.2.1.1 subroutine `olis_f90stdlib::alloc_complex_eigenvects` (complex(kind=dp), dimension(:, :), intent(in) *matrix*, complex(kind=dp), dimension(:, :), intent(inout), allocatable *eigenvals*, complex(kind=dp), dimension(:, :), intent(inout), allocatable *u*, complex(kind=dp), dimension(:, :), intent(inout), allocatable *v*)

allocates eigenvals, u & v arrays for eigenvals & eigenvects

allocated temp work arrays also

Author

Oliver Thomas August 2018

Parameters

<i>matrix</i>	input complex matrix
<i>eigenvals</i>	1d array for eigenvalues, is overwritten on exit
<i>u</i>	2d array of left eigenvectors
<i>v</i>	3d array of right eigenvectors

4.2.1.2 subroutine `olis_f90stdlib::alloc_complex_svd` (`complex(kind=dp), dimension(:,:), intent(in) matrix`, `real(kind=dp), dimension(:,:), intent(inout), allocatable sigma`, `complex(kind=dp), dimension(:,:), intent(inout), allocatable u`, `complex(kind=dp), dimension(:,:), intent(inout), allocatable vt`)

allocates sigma (singular vals), u and vt for complexSVD allocates temp work arrays too

Parameters

<i>matrix</i>	input complex matrix
<i>sigma</i>	real vector of singular values sorted in descending order
<i>u</i>	unitary matrix
<i>vt</i>	unitary matrix returns $V^{*}H$ NOT v

4.2.1.3 `complex(kind=dp) function, dimension(n,n) olis_f90stdlib::c_identity` (`integer, intent(in) n`)

makes complex identity matrix dim (nxn)

Parameters

<i>n</i>	input dimension
----------	-----------------

4.2.1.4 `complex(kind=dp) function, dimension(2,2) olis_f90stdlib::c_inv2` (`complex(kind=dp), dimension(2,2), intent(in) m_in`)

inverse for a complex 2x2 matrix

Parameters

<i>m_{in}</i>	is input complex 2x2 matrix
-----------------------	-----------------------------

4.2.1.5 subroutine `olis_f90stdlib::complex_eigenvects` (`complex(kind=dp), dimension(:,:), allocatable a`, `complex(kind=dp), dimension(:,:), allocatable w`, `complex(kind=dp), dimension(:,:), allocatable vl`, `complex(kind=dp), dimension(:,:), allocatable vr`)

computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack

Parameters

<i>a</i>	input allocatable complex matrix to be diagonalised
<i>w</i>	output allocatable complex 1d array containing eigenvals
<i>v</i> / <i>l</i>	output allocatable complex 2d array containing left eigenvectors
<i>v</i> / <i>r</i>	output allocatable complex 2d array containing right eigenvectors

Note

need to check this is optimised

4.2.1.6 subroutine `olis_f90stdlib::complex_svd` (`complex(kind=dp)`, `dimension(:, :)`, `intent(inout)`, allocatable *a*, `real(kind=dp)`, `dimension(:, :)`, allocatable *sigma*, `complex(kind=dp)`, `dimension(:, :)`, allocatable *u*, `complex(kind=dp)`, `dimension(:, :)`, allocatable *vt*)

computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack

Parameters

<i>a</i>	input allocatable complex matrix to be SVD'd
<i>sigma</i>	output allocatable complex 1d array containing ordered singular values
<i>u</i>	output allocatable complex 2d array containing u
<i>vt</i>	output allocatable complex 2d array containing v**H

Note

need to check this is optimised

4.2.1.7 `complex(kind=dp)` function `olis_f90stdlib::complextrace` (`complex(kind=dp)`, `dimension(:, :)` *a*)

computes the trace of a complex matrix

Parameters

<i>a</i>	is the complex matrix in
----------	--------------------------

4.2.1.8 `complex(kind=dp)` function, `dimension(size(matrix,1),size(matrix,2))` `olis_f90stdlib::expmatrix` (`complex(kind=dp)`, `dimension(:, :)` *matrix*, integer *n*)

Parameters

<i>n</i>	is the number of terms in taylor expansion to consider
----------	--

4.2.1.9 recursive real(kind=dp) function olis_f90stdlib::factorial (integer n)

4.2.1.10 recursive complex(kind=dp) function, dimension(size(x,1),size(x,2)) olis_f90stdlib::matrixmul (complex(kind=dp), dimension(:,:) x , integer n)

4.2.1.11 real(kind=dp) function olis_f90stdlib::matrixnorm (complex(kind=dp), dimension(:,:) c)

computed Frobenius matrix norm of complex matrix using lapack zlange

Parameters

c	input complex matrix
-----	----------------------

4.2.1.12 complex(kind=dp) function, dimension(2,2) olis_f90stdlib::outerproduct (complex(kind=dp), dimension(:), intent(in) a , complex(kind=dp), dimension(:), intent(in) b)

outerproduct of two complex vectors, returns a complex matrix

Parameters

a	is input vector 1, $ \text{ket}\rangle$
b	is input vector 2, $\langle\text{bra} $

4.2.1.13 subroutine olis_f90stdlib::printvectors (complex(kind=dp), dimension(:,:) intent(in) $vect$, character(len=*) intent(in), optional $desc$, integer intent(in), optional f)

print formatted matrices can take optional args for labels or write directly to a file

Parameters

$vect$	is the input complex matrix
$desc$	is the optional string to be written above the matrix
f	is the optional file output unit to write to, default is console

4.2.1.14 subroutine olis_f90stdlib::randseed (integer, dimension(:), allocatable $seed$)

generates random seed

Parameters

$seed$	is input allocatable 1d array
--------	-------------------------------

4.2.1.15 `complex(kind=dp) function, dimension(:,:), allocatable olis_f90stdlib::tprod (complex(kind=dp), dimension (:,:), intent(in) a, complex(kind=dp), dimension (:,:), intent(in) b)`

tensor product for complex matrices *aXb*

Parameters

<i>a</i>	complex matrix in
<i>b</i>	complex matrix in

4.2.2 Variable Documentation

4.2.2.1 `complex(kind=dp), parameter olis_f90stdlib::imaginary =(0.0_dp, 1.0_dp)`

4.2.2.2 `real(kind=dp), parameter olis_f90stdlib::pi =4.0_dp*atan(1.0)`

Chapter 5

File Documentation

5.1 makeopticalelements.f90 File Reference

Modules

- module [makeopticalelements](#)
module for building symplectic matrices for optical elements

Functions/Subroutines

- subroutine [makeopticalelements::make_bs](#) (nspace, nspec, symp_mat, m1, m2, theta)
makes beamsplitter symplectic matrix takes in an allocated matrix for the beamsplitter matrix to be written to uses the private ident_spec, spatial_work, n_work arrays
- subroutine [makeopticalelements::make_sq](#) (nspace, nspec, symp_mat, m1, m2, alpha, beta)
make symplectic squeezing matrix from exponentiated JSA a lot is broken...
- real(kind=dp) function [makeopticalelements::g4](#) (ft, nspec)
calculates g4 using matrix elements sum
- real(kind=dp) function [makeopticalelements::amp](#) (a)
returns the absolute value squared $|a|^2$
- complex(kind=dp) function [makeopticalelements::abt](#) (i, j, ft, nspec)
*calculates matrix elements $\text{Alpha} \cdot \text{Beta}^{**} T$ for $M = (A \ B) \ (B^* \ A^*)$ computes $AB^{**} T$ and returns the i,j -th element*
- complex(kind=dp) function [makeopticalelements::bbd](#) (i, j, ft, nspec)
*calculates the matrix elements $\text{Beta} \cdot \text{Beta}^{**} H$ for $M = (A \ B) \ (B^* \ A^*)$ computes $B \cdot B^{**} H$ (Hermitian conjg) and returns the i,j -th element*
- subroutine [makeopticalelements::alloc_temparrays](#) (nspace, nspec)
allocates temp arrays for matrices
- subroutine [makeopticalelements::dealloc_temparrays](#)

Variables

- real(kind=dp), public [makeopticalelements::ident](#)

5.2 num_hom.f90 File Reference

Functions/Subroutines

- program `num_hom`
program to compute matrix of a JSA
- complex(kind=dp) function `f` (`w1`, `w2`, `sig`)
JSA function taking two freq.

5.2.1 Function/Subroutine Documentation

5.2.1.1 complex(kind=dp) function `num_hom::f` (real(kind=dp), intent(in) `w1`, real(kind=dp), intent(in) `w2`, real(kind=dp), intent(in) `sig`)

JSA function taking two freq.

Note

files to write to
to make off diagonal for fmatrix `m_sq=0.0_dp` ! top right `m_sq(1:1*f_size, 3*f_size+1:4*f_size)=1` ! mid right `m_sq(1*f_size+1:2*f_size, 2*f_size+1:3*f_size)=2` ! mid left `m_sq(2*f_size+1:3*f_size, 1*f_size+1:2*f_size)=3` ! bot left `m_sq(3*f_size+1:4*f_size, 1:1*f_size)=4`
!h= 0.0 F_JSA F_JSA*T 0.0

`f_jsa = f_mat`

`M_sq = exp(i (0 H) (-H* 0)`

`M_sq = exp(i (0 0 0 F_JSA) (0 0 F_JSA**T 0) (0 -conjg(F_JSA) 0 0) (-F_JSA**H 0 0 0)`

Note

alpha beta are top left and top right of M $M = \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} B^* & A^* \end{pmatrix}$

Parameters

<i>alpha_size</i>	is 2*f_size as all spectral modes for 2 spatial
-------------------	---

Note

allocate for sq on modes 1&2

Parameters

<i>w1</i>	input signal freq
<i>w2</i>	input idler freq
<i>sig</i>	input variance

5.2.1.2 program num_hom ()

program to compute matrix of a JSA

5.3 olis_f90stdlib.f90 File Reference

Modules

- module [olis_f90stdlib](#)

Functions/Subroutines

- subroutine [olis_f90stdlib::alloc_complex_eigenvects](#) (matrix, eigenvals, u, v)
allocates eigenvals, u & v arrays for eigenvals & eigenvects
- subroutine [olis_f90stdlib::alloc_complex_svd](#) (matrix, sigma, u, vt)
allocates sigma (singular vals), u and vt for complexSVD allocates temp work arrays too
- subroutine [olis_f90stdlib::randseed](#) (seed)
generates random seed
- subroutine [olis_f90stdlib::printvectors](#) (vect, desc, f)
print formatted matrices can take optional args for labels or write directly to a file
- complex(kind=dp) function, dimension(2, 2) [olis_f90stdlib::outerproduct](#) (a, b)
outerproduct of two complex vectors, returns a complex matrix
- complex(kind=dp) function, dimension(n, n) [olis_f90stdlib::c_identity](#) (n)
makes complex identity matrix dim (nxn)
- complex(kind=dp) function, dimension(:, :), allocatable [olis_f90stdlib::tprod](#) (a, b)
tensor product for complex matrices aXb
- complex(kind=dp) function [olis_f90stdlib::complextrace](#) (a)
computes the trace of a complex matrix
- subroutine [olis_f90stdlib::complex_eigenvects](#) (a, w, vl, vr)
computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack
- subroutine [olis_f90stdlib::complex_svd](#) (a, sigma, u, vt)
computes the complex eigenvalues and eigenvectors overwrites matrix in, input eigenvalue array and eigenvector arrays uses the zgeev subroutine from lapack
- complex(kind=dp) function, dimension(2, 2) [olis_f90stdlib::c_inv2](#) (m_in)
inverse for a complex 2x2 matrix
- real(kind=dp) function [olis_f90stdlib::matrixnorm](#) (c)
computed Frobenius matrix norm of complex matrix using lapack zlange
- complex(kind=dp) function, dimension(size(matrix, 1), size(matrix, 2)) [olis_f90stdlib::expmatrix](#) (matrix, n)
- recursive complex(kind=dp) function, dimension(size(x, 1), size(x, 2)) [olis_f90stdlib::matrixmul](#) (x, n)
- recursive real(kind=dp) function [olis_f90stdlib::factorial](#) (n)

Variables

- real(kind=dp), parameter [olis_f90stdlib::pi](#) =4.0_dp*atan(1.0)
- complex(kind=dp), parameter [olis_f90stdlib::imaginary](#) =(0.0_dp, 1.0_dp)

Index

- abt
 - makeopticalelements, [7](#)
- alloc_complex_eigenvecs
 - olis_f90stdlib, [10](#)
- alloc_complex_svd
 - olis_f90stdlib, [11](#)
- alloc_temparrays
 - makeopticalelements, [8](#)
- amp
 - makeopticalelements, [8](#)
- bdd
 - makeopticalelements, [8](#)
- c_identity
 - olis_f90stdlib, [11](#)
- c_inv2
 - olis_f90stdlib, [11](#)
- complex_eigenvecs
 - olis_f90stdlib, [11](#)
- complex_svd
 - olis_f90stdlib, [12](#)
- complextrace
 - olis_f90stdlib, [12](#)
- dealloc_temparrays
 - makeopticalelements, [9](#)
- expmatrix
 - olis_f90stdlib, [12](#)
- f
 - num_hom.f90, [16](#)
- factorial
 - olis_f90stdlib, [12](#)
- g4
 - makeopticalelements, [9](#)
- ident
 - makeopticalelements, [10](#)
- imaginary
 - olis_f90stdlib, [14](#)
- make_bs
 - makeopticalelements, [9](#)
- make_sq
 - makeopticalelements, [9](#)
- makeopticalelements, [7](#)
 - abt, [7](#)
 - alloc_temparrays, [8](#)
 - amp, [8](#)
 - bdd, [8](#)
 - dealloc_temparrays, [9](#)
 - g4, [9](#)
 - ident, [10](#)
 - make_bs, [9](#)
 - make_sq, [9](#)
 - makeopticalelements.f90, [15](#)
 - matrixmul
 - olis_f90stdlib, [13](#)
 - matrixnorm
 - olis_f90stdlib, [13](#)
 - num_hom
 - num_hom.f90, [16](#)
 - num_hom.f90, [16](#)
 - f, [16](#)
 - num_hom, [16](#)
 - olis_f90stdlib, [10](#)
 - alloc_complex_eigenvecs, [10](#)
 - alloc_complex_svd, [11](#)
 - c_identity, [11](#)
 - c_inv2, [11](#)
 - complex_eigenvecs, [11](#)
 - complex_svd, [12](#)
 - complextrace, [12](#)
 - expmatrix, [12](#)
 - factorial, [12](#)
 - imaginary, [14](#)
 - matrixmul, [13](#)
 - matrixnorm, [13](#)
 - outerproduct, [13](#)
 - pi, [14](#)
 - printvectors, [13](#)
 - randseed, [13](#)
 - tprod, [13](#)
 - olis_f90stdlib.f90, [17](#)
 - outerproduct
 - olis_f90stdlib, [13](#)
 - pi
 - olis_f90stdlib, [14](#)
 - printvectors
 - olis_f90stdlib, [13](#)
 - randseed
 - olis_f90stdlib, [13](#)
 - tprod
 - olis_f90stdlib, [13](#)