16.37 LINALG: Linear algebra package

This package provides a selection of functions that are useful in the world of linear algebra.

Author: Matt Rebbeck.

16.37.1 Introduction

This package provides a selection of functions that are useful in the world of linear algebra. These functions are described alphabetically in subsection 16.37.3 and are labelled 16.37.3.1 to 16.37.3.53. They can be classified into four sections(n.b: the numbers after the dots signify the function label in section 16.37.3).

Contributions to this package have been made by Walter Tietze (ZIB).

16.37.1.1 Basic matrix handling

16.37.1.2 Constructors

Functions that create matrices.

16.37.1.3 High level algorithms

There is a separate NORMFORM[1] package for computing the following matrix normal forms in REDUCE:

smithex, smithex_int, frobenius, ratjordan, jordansymbolic, jordan.

16.37.1.4 Predicates

Note on examples:

In the examples the matrix $\mathcal{A}$ will be

$$\mathcal{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Notation

Throughout $\mathcal{I}$ is used to indicate the identity matrix and $\mathcal{A}^T$ to indicate the transpose of the matrix $\mathcal{A}$.

16.37.2 Getting started

If you have not used matrices within REDUCE before then the following may be helpful.

Creating matrices

Initialisation of matrices takes the following syntax:

```
mat1 := mat((a,b,c),(d,e,f),(g,h,i));
```

will produce

$$mat1 := \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Getting at the entries

The (i, j)th entry can be accessed by:

```
mat1(i,j);
```

Loading the linear_algebra package

The package is loaded by:

```
load_package linalg;
```

16.37.3 What's available

16.37.3.1 add_columns, add_rows

Syntax:

`add_columns(`$\mathcal{A}$`,c1,c2,expr);`

$\mathcal{A}$	:-	a matrix.
$c1, c2$	:-	positive integers.
expr	:-	a scalar expression.

Synopsis:

`add_columns` replaces column $c2$ of $\mathcal{A}$ by
`expr * column(`$\mathcal{A}$`,c1) + column(`$\mathcal{A}$`,c2)`.
`add_rows` performs the equivalent task on the rows of $\mathcal{A}$.

Examples:

$$\texttt{add\_columns}(\mathcal{A},1,2,x) = \begin{pmatrix} 1 & x+2 & 3 \\ 4 & 4*x+5 & 6 \\ 7 & 7*x+8 & 9 \end{pmatrix}$$

$$\texttt{add\_rows}(\mathcal{A},2,3,5) = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 27 & 33 & 39 \end{pmatrix}$$

Related functions:
`add_to_columns`, `add_to_rows`, `mult_columns`, `mult_rows`.

16.37.3.2 add_rows

See: `add_columns`.

16.37.3.3 add_to_columns, add_to_rows

Syntax:
`add_to_columns(`$\mathcal{A}$`,column_list,expr);`

$\mathcal{A}$	:-	a matrix.
column_list	:-	a positive integer or a list of positive integers.
expr	:-	a scalar expression.

Synopsis:
`add_to_columns` adds expr to each column specified in column_list of $\mathcal{A}$.

`add_to_rows` performs the equivalent task on the rows of $\mathcal{A}$.

Examples:

$$\texttt{add\_to\_columns}(\mathcal{A},\{1,2\},10) = \begin{pmatrix} 11 & 12 & 3 \\ 14 & 15 & 6 \\ 17 & 18 & 9 \end{pmatrix}$$

$$\texttt{add\_to\_rows}(\mathcal{A},2,-x) = \begin{pmatrix} 1 & 2 & 3 \\ -x+4 & -x+5 & -x+6 \\ 7 & 8 & 9 \end{pmatrix}$$

Related functions:
`add_columns`, `add_rows`, `mult_rows`, `mult_columns`.

16.37.3.4 add_to_rows

See: `add_to_columns`.

16.37.3.5 augment_columns, stack_rows

Syntax:

`augment_columns(`$\mathcal{A}$`,column_list);`

$\mathcal{A}$	:-	a matrix.
column_list	:-	either a positive integer or a list of positive integers.

Synopsis:

`augment_columns` gets hold of the columns of $\mathcal{A}$ specified in column_list and sticks them together.
`stack_rows` performs the same task on rows of $\mathcal{A}$.

Examples:

$$\texttt{augment\_columns}(\mathcal{A},\{1,2\}) = \begin{pmatrix} cc1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}$$

$$\texttt{stack\_rows}(\mathcal{A},\{1,3\}) = \begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{pmatrix}$$

Related functions:

`get_columns`, `get_rows`, `sub_matrix`.

16.37.3.6 band_matrix

Syntax:

`band_matrix(expr_list,square_size);`

expr_list	:-	either a single scalar expression or a list of an odd number of scalar expressions.
square_size	:-	a positive integer.

Synopsis:

`band_matrix` creates a square matrix of dimension `square_size`. The diagonal consists of the middle expr of the `expr_list`. The expressions to the left of this fill the required number of sub-diagonals and the expressions to the right the super-diagonals.

Examples:

$$\texttt{band\_matrix}(\{x,y,z\},6) = \begin{pmatrix} y & z & 0 & 0 & 0 & 0 \\ x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \\ 0 & 0 & 0 & 0 & x & y \end{pmatrix}$$

Related functions:

`diagonal`.

16.37.3.7 block_matrix

Syntax:

`block_matrix(r,c,matrix_list);`

r, c	:-	positive integers.
`matrix_list`	:-	a list of matrices.

Synopsis:

`block_matrix` creates a matrix that consists of $r \times c$ matrices filled from the `matrix_list` row-wise.

Examples:

$$\mathcal{B} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \mathcal{C} = \begin{pmatrix} 5 \\ 5 \end{pmatrix},\ \mathcal{D} = \begin{pmatrix} 22 & 33 \\ 44 & 55 \end{pmatrix}$$

$$\texttt{block\_matrix}(2,3,\{\mathcal{B},\mathcal{C},\mathcal{D},\mathcal{D},\mathcal{C},\mathcal{B}\}) = \begin{pmatrix} 1 & 0 & 5 & 22 & 33 \\ 0 & 1 & 5 & 44 & 55 \\ 22 & 33 & 5 & 1 & 0 \\ 44 & 55 & 5 & 0 & 1 \end{pmatrix}$$

16.37.3.8 char_matrix

Syntax:

`char_matrix(`$\mathcal{A}, \lambda$`);`

$\mathcal{A}$	:-	a square matrix.
λ	:-	a symbol or algebraic expression.

Synopsis:

`char_matrix` creates the characteristic matrix $\mathcal{C}$ of $\mathcal{A}$. This is $\mathcal{C} = \lambda\mathcal{I} - \mathcal{A}$.

Examples:

$$\texttt{char\_matrix}(\mathcal{A}, x) = \begin{pmatrix} x-1 & -2 & -3 \\ -4 & x-5 & -6 \\ -7 & -8 & x-9 \end{pmatrix}$$

Related functions:
`char_poly`.

16.37.3.9 char_poly

Syntax:
`char_poly(`$\mathcal{A}, \lambda$`);`

$\mathcal{A}$:- a square matrix.
λ :- a symbol or algebraic expression.

Synopsis:
`char_poly` finds the characteristic polynomial of $\mathcal{A}$.

This is the determinant of $\lambda\mathcal{I} - \mathcal{A}$.

Examples:
`char_poly(`$\mathcal{A}, x$`)` $= x^3 - 15 * x^2 - 18 * x$

Related functions:
`char_matrix`.

16.37.3.10 cholesky

Syntax:
`cholesky(`$\mathcal{A}$`);`

$\mathcal{A}$:- a positive definite matrix containing numeric entries.

Synopsis:
`cholesky` computes the cholesky decomposition of $\mathcal{A}$.

It returns $\{\mathcal{L}, \mathcal{U}\}$ where $\mathcal{L}$ is a lower matrix, $\mathcal{U}$ is an upper matrix, $\mathcal{A} = \mathcal{L}\mathcal{U}$, and $\mathcal{U} = \mathcal{L}^T$.

Examples:

$$\mathcal{F} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\texttt{cholesky}(\mathcal{F}) = \left\{ \begin{pmatrix} 1 & 0 & 0 \\ 1 & \sqrt{2} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 0 & \sqrt{2} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \right\}$$

Related functions:
`lu_decom`.

16.37.3.11 coeff_matrix

Syntax:
`coeff_matrix({lin_eqn`$_1$`,lin_eqn`$_2$`, ...,lin_eqn`$_n$`});` [14]

lin_eqn$_1$,lin_eqn$_2$, ...,lin_eqn$_n$:- linear equations. Can be of the form *equation* = *number* or just *equation* which is equivalent to *equation* = *0*.

Synopsis:
`coeff_matrix` creates the coefficient matrix $\mathcal{C}$ of the linear equations. It returns $\{\mathcal{C}, \mathcal{X}, \mathcal{B}\}$ such that $\mathcal{C}\mathcal{X} = \mathcal{B}$.

Examples:

`coeff_matrix`$(\{x + y + 4 * z = 10, y + x - z = 20, x + y + 4\}) =$

$$\left\{ \begin{pmatrix} 4 & 1 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \begin{pmatrix} z \\ y \\ x \end{pmatrix}, \begin{pmatrix} 10 \\ 20 \\ -4 \end{pmatrix} \right\}$$

16.37.3.12 column_dim, row_dim

Syntax:
`column_dim(`$\mathcal{A}$`);`

$\mathcal{A}$:- a matrix.

Synopsis:
`column_dim` finds the column dimension of $\mathcal{A}$.
`row_dim` finds the row dimension of $\mathcal{A}$.

Examples:
`column_dim`($\mathcal{A}$) = 3

16.37.3.13 companion

Syntax:
`companion(poly,x);`

[14] If you're feeling lazy then the {}'s can be omitted.

`poly` :- a monic univariate polynomial in x.
x :- the variable.

Synopsis:

`companion` creates the companion matrix $\mathcal{C}$ of `poly`.

This is the square matrix of dimension n, where n is the degree of `poly` w.r.t. x. The entries of $\mathcal{C}$ are: $\mathcal{C}(i,n) = -\texttt{coeffn}(\texttt{poly}, x, i-1)$ for $i = 1, \ldots, n$, $\mathcal{C}(i, i-1) = 1$ for $i = 2, \ldots, n$ and the rest are 0.

Examples:

$$\texttt{companion}(x^4 + 17 * x^3 - 9 * x^2 + 11, x) = \begin{pmatrix} 0 & 0 & 0 & -11 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 9 \\ 0 & 0 & 1 & -17 \end{pmatrix}$$

Related functions:

`find_companion`.

16.37.3.14 copy_into

Syntax:

`copy_into(`$\mathcal{A}, \mathcal{B}$`,r,c);`

$\mathcal{A}, \mathcal{B}$:- matrices.
r, c :- positive integers.

Synopsis:

`copy_into` copies matrix $\mathcal{A}$ into $\mathcal{B}$ with $\mathcal{A}(1,1)$ at $\mathcal{B}(r,c)$.

Examples:

$$\mathcal{G} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\texttt{copy\_into}(\mathcal{A}, \mathcal{G}, 1, 2) = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 4 & 5 & 6 \\ 0 & 7 & 8 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Related functions:

`augment_columns`, `extend`, `matrix_augment`, `matrix_stack`, `stack_rows`, `sub_matrix`.

16.37.3.15 diagonal

Syntax:

`diagonal({mat`$_1$`,mat`$_2$`, ...,mat`$_n$`});`[15]

mat$_1$,mat$_2$,...,mat$_n$:- each can be either a scalar expr or a square matrix.

Synopsis:

`diagonal` creates a matrix that contains the input on the diagonal.

Examples:

$$\mathcal{H} = \begin{pmatrix} 66 & 77 \\ 88 & 99 \end{pmatrix}$$

$$\texttt{diagonal}(\{\mathcal{A}, x, \mathcal{H}\}) = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 66 & 77 \\ 0 & 0 & 0 & 0 & 88 & 99 \end{pmatrix}$$

Related functions:

`jordan_block`.

16.37.3.16 extend

Syntax:

`extend(`$\mathcal{A}$`,r,c,expr);`

$\mathcal{A}$:- a matrix.

r, c :- positive integers.

`expr` :- algebraic expression or symbol.

Synopsis:

`extend` returns a copy of $\mathcal{A}$ that has been extended by r rows and c columns. The new entries are made equal to `expr`.

Examples:

$$\texttt{extend}(\mathcal{A}, 1, 2, x) = \begin{pmatrix} 1 & 2 & 3 & x & x \\ 4 & 5 & 6 & x & x \\ 7 & 8 & 9 & x & x \\ x & x & x & x & x \end{pmatrix}$$

[15] If you're feeling lazy then the {}'s can be omitted.

Related functions:

`copy_into`, `matrix_augment`, `matrix_stack`, `remove_columns`, `remove_rows`.

16.37.3.17 find_companion

Syntax:

`find_companion(`$\mathcal{A}$`,x);`

$\mathcal{A}$:- a matrix.
x :- the variable.

Synopsis:

Given a companion matrix, `find_companion` finds the polynomial from which it was made.

Examples:

$$\mathcal{C} = \begin{pmatrix} 0 & 0 & 0 & -11 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 9 \\ 0 & 0 & 1 & -17 \end{pmatrix}$$

`find_companion`$(\mathcal{C}, x) = x^4 + 17 * x^3 - 9 * x^2 + 11$

Related functions:

`companion`.

16.37.3.18 get_columns, get_rows

Syntax:

`get_columns(`$\mathcal{A}$`,column_list);`

$\mathcal{A}$:- a matrix.
c :- either a positive integer or a list of positive integers.

Synopsis:

`get_columns` removes the columns of $\mathcal{A}$ specified in `column_list` and returns them as a list of column matrices.

`get_rows` performs the same task on the rows of $\mathcal{A}$.

Examples:

`get_columns`$(\mathcal{A}, \{1, 3\}) = \left\{ \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}, \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} \right\}$

`get_rows`$(\mathcal{A}, 2) = \left\{ \begin{pmatrix} 4 & 5 & 6 \end{pmatrix} \right\}$

Related functions:

`augment_columns`, `stack_rows`, `sub_matrix`.

16.37.3.19 get_rows

See: `get_columns`.

16.37.3.20 gram_schmidt

Syntax:

`gram_schmidt({vec`$_1$`,vec`$_2$`, ...,vec`$_n$`});` [16]

vec$_1$,vec$_2$,...,vec$_n$:- linearly-independent vectors. Each vector must be written as a list, eg:{1,0,0}.

Synopsis:

`gram_schmidt` performs the Gram-Schmidt orthonormalisation on the input vectors. It returns a list of orthogonal normalised vectors.

Examples:

`gram_schmidt({{1,0,0},{1,1,0},{1,1,1}})` = {{1,0,0},{0,1,0},{0,0,1}}

`gram_schmidt({{1,2},{3,4}})` $= \{\{\frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}}\}, \{\frac{2 * \sqrt{5}}{5}, \frac{-\sqrt{5}}{5}\}\}$

16.37.3.21 hermitian_tp

Syntax:

`hermitian_tp(`$\mathcal{A}$`);`

$\mathcal{A}$:- a matrix.

Synopsis:

`hermitian_tp` computes the hermitian transpose of $\mathcal{A}$.

This is a matrix in which the (i, j)th entry is the conjugate of the (j, i)th entry of $\mathcal{A}$.

Examples:

[16] If you're feeling lazy then the {}'s can be omitted.

$$\mathcal{J} = \begin{pmatrix} i+1 & i+2 & i+3 \\ 4 & 5 & 2 \\ 1 & i & 0 \end{pmatrix}$$

$$\texttt{hermitian\_tp}(\mathcal{J}) = \begin{pmatrix} -i+1 & 4 & 1 \\ -i+2 & 5 & -i \\ -i+3 & 2 & 0 \end{pmatrix}$$

Related functions:
`tp`[17].

16.37.3.22 hessian

Syntax:
```
hessian(expr,variable_list);
```

expr	:-	a scalar expression.
variable_list	:-	either a single variable or a list of variables.

Synopsis:
`hessian` computes the hessian matrix of expr w.r.t. the varibles in `variable_list`.

This is an $n \times n$ matrix where n is the number of variables and the (i, j)th entry is `df(expr,variable_list(i),variable_list(j))`.

Examples:

$$\texttt{hessian}(x * y * z + x^2, \{w, x, y, z\}) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & z & y \\ 0 & z & 0 & x \\ 0 & y & x & 0 \end{pmatrix}$$

Related functions:
`df`[18].

16.37.3.23 hilbert

Syntax:
```
hilbert(square_size,expr);
```

square_size	:-	a positive integer.
expr	:-	an algebraic expression.

[17] standard reduce call for the transpose of a matrix - see section 14.4.
[18] standard reduce call for differentiation - see section 7.8.

Synopsis:

`hilbert` computes the square hilbert matrix of dimension `square_size`.

This is the symmetric matrix in which the (i,j)th entry is $1/(i+j-$`expr`$)$.

Examples:

$$\texttt{hilbert}(3, y+x) = \begin{pmatrix} \frac{-1}{x+y-2} & \frac{-1}{x+y-3} & \frac{-1}{x+y-4} \\ \frac{-1}{x+y-3} & \frac{-1}{x+y-4} & \frac{-1}{x+y-5} \\ \frac{-1}{x+y-4} & \frac{-1}{x+y-5} & \frac{-1}{x+y-6} \end{pmatrix}$$

16.37.3.24 jacobian

Syntax:

`mat_jacobian(expr_list,variable_list);`

expr_list	:-	either a single algebraic expression or a list of algebraic expressions.
variable_list	:-	either a single variable or a list of variables.

Synopsis:

`mat_jacobian` computes the jacobian matrix of `expr_list` w.r.t. `variable_list`.

This is a matrix whose (i,j)th entry is `df(expr_list(i),variable_list(j))`. The matrix is $n \times m$ where n is the number of variables and m the number of expressions.

Examples:

`mat_jacobian({`$x^4, x*y^2, x*y*z^3$`},{`w,x,y,z`})` $=$

$$\begin{pmatrix} 0 & 4*x^3 & 0 & 0 \\ 0 & y^2 & 2*x*y & 0 \\ 0 & y*z^3 & x*z^3 & 3*x*y*z^2 \end{pmatrix}$$

Related functions:

`hessian`, `df`[19].

NOTE: The function `mat_jacobian` used to be called just "`jacobian`" however us of that name was in conflict with another Reduce package.

[19] standard reduce call for differentiation - see REDUCE User's Manual[2].

16.37.3.25 jordan_block

Syntax:

`jordan_block(expr,square_size);`

expr	:-	an algebraic expression or symbol.
square_size	:-	a positive integer.

Synopsis:

`jordan_block` computes the square jordan block matrix $\mathcal{J}$ of dimension `square_size`.

The entries of $\mathcal{J}$ are: $\mathcal{J}(i,i) =$ `expr` for $i = 1,\ldots,n$, $\mathcal{J}(i,i+1) = 1$ for $i = 1,\ldots,n-1$, and all other entries are 0.

Examples:

$$\texttt{jordan\_block(x,5)} = \begin{pmatrix} x & 1 & 0 & 0 & 0 \\ 0 & x & 1 & 0 & 0 \\ 0 & 0 & x & 1 & 0 \\ 0 & 0 & 0 & x & 1 \\ 0 & 0 & 0 & 0 & x \end{pmatrix}$$

Related functions:

`diagonal`, `companion`.

16.37.3.26 lu_decom

Syntax:

`lu_decom(`$\mathcal{A}$`);`

$\mathcal{A}$	:-	a matrix containing either numeric entries or imaginary entries with numeric coefficients.

Synopsis:

`lu_decom` performs LU decomposition on $\mathcal{A}$, ie: it returns $\{\mathcal{L},\mathcal{U}\}$ where $\mathcal{L}$ is a lower diagonal matrix, $\mathcal{U}$ an upper diagonal matrix and $\mathcal{A} = \mathcal{L}\mathcal{U}$.

Caution: The algorithm used can swap the rows of $\mathcal{A}$ during the calculation. This means that $\mathcal{L}\mathcal{U}$ does not equal $\mathcal{A}$ but a row equivalent of it. Due to this, `lu_decom` returns $\{\mathcal{L},\mathcal{U}$,vec$\}$. The call `convert(`$\mathcal{A}$`,vec)` will return the matrix that has been decomposed, ie: $\mathcal{L}\mathcal{U} =$ `convert(`$\mathcal{A}$`,vec)`.

Examples:

$$\mathcal{K} = \begin{pmatrix} 1 & 3 & 5 \\ -4 & 3 & 7 \\ 8 & 6 & 4 \end{pmatrix}$$

$$\texttt{lu} := \texttt{lu\_decom}(\mathcal{K}) = \left\{ \begin{pmatrix} 8 & 0 & 0 \\ -4 & 6 & 0 \\ 1 & 2.25 & 1.1251 \end{pmatrix}, \begin{pmatrix} 1 & 0.75 & 0.5 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{pmatrix}, [\,3\;2\;3\,] \right\}$$

$$\texttt{first lu} \star \texttt{second lu} = \begin{pmatrix} 8 & 6 & 4 \\ -4 & 3 & 7 \\ 1 & 3 & 5 \end{pmatrix}$$

$$\texttt{convert(}\mathcal{K}\texttt{,third lu)} = \begin{pmatrix} 8 & 6 & 4 \\ -4 & 3 & 7 \\ 1 & 3 & 5 \end{pmatrix}$$

$$\mathcal{P} = \begin{pmatrix} i+1 & i+2 & i+3 \\ 4 & 5 & 2 \\ 1 & i & 0 \end{pmatrix}$$

$$\texttt{lu} := \texttt{lu\_decom}(\mathcal{P}) = \left\{ \begin{pmatrix} 1 & 0 & 0 \\ 4 & -4*i+5 & 0 \\ i+1 & 3 & 0.41463*i+2.26829 \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} 1 & i & 0 \\ 0 & 1 & 0.19512*i+0.24390 \\ 0 & 0 & 1 \end{pmatrix}, [\,3\;2\;3\,] \right\}$$

$$\texttt{first lu} \star \texttt{second lu} = \begin{pmatrix} 1 & i & 0 \\ 4 & 5 & 2 \\ i+1 & i+2 & i+3 \end{pmatrix}$$

$$\texttt{convert}(\mathcal{P}, \texttt{thirdlu}) = \begin{pmatrix} 1 & i & 0 \\ 4 & 5 & 2 \\ i+1 & i+2 & i+3 \end{pmatrix}$$

Related functions:

`cholesky`.

16.37.3.27 make_identity

Syntax:

```
make_identity(square_size);
```

square_size :- a positive integer.

Synopsis:

`make_identity` creates the identity matrix of dimension square_size.

Examples:

$$\texttt{make\_identity}(4) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Related functions:

`diagonal`.

16.37.3.28 matrix_augment, matrix_stack

Syntax:

`matrix_augment({`$\texttt{mat}_1$`,`$\texttt{mat}_2$`, ...,`$\texttt{mat}_n$`});`[20]

mat_1,mat_2, ...,mat_n :- matrices.

Synopsis:

`matrix_augment` sticks the matrices in `matrix_list` together horizontally.

`matrix_stack` sticks the matrices in `matrix_list` together vertically.

Examples:

$$\texttt{matrix\_augment}(\{\mathcal{A}, \mathcal{A}\}) = \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 4 & 6 & 4 & 5 & 6 \\ 7 & 8 & 9 & 7 & 8 & 9 \end{pmatrix}$$

$$\texttt{matrix\_stack}(\{\mathcal{A}, \mathcal{A}\}) = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Related functions:

`augment_columns`, `stack_rows`, `sub_matrix`.

[20] If you're feeling lazy then the {}'s can be omitted.

16.37.3.29 matrixp

Syntax:

`matrixp(test_input);`

test_input :- anything you like.

Synopsis:

`matrixp` is a boolean function that returns t if the input is a matrix and nil otherwise.

Examples:

`matrixp`($\mathcal{A}$) = t

`matrixp`(doodlesackbanana) = nil

Related functions:

`squarep`, `symmetricp`.

16.37.3.30 matrix_stack

See: `matrix_augment`.

16.37.3.31 minor

Syntax:

`minor(`$\mathcal{A}$`,r,c);`

$\mathcal{A}$:- a matrix.

r, c :- positive integers.

Synopsis:

`minor` computes the (r, c)th minor of $\mathcal{A}$.

This is created by removing the rth row and the cth column from $\mathcal{A}$.

Examples:

$$\texttt{minor}(\mathcal{A}, 1, 3) = \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}$$

Related functions:

`remove_columns`, `remove_rows`.

16.37.3.32 mult_columns, mult_rows

Syntax:

`mult_columns(`$\mathcal{A}$`,column_list,expr);`

$\mathcal{A}$	:-	a matrix.
column_list	:-	a positive integer or a list of positive integers.
expr	:-	an algebraic expression.

Synopsis:

`mult_columns` returns a copy of $\mathcal{A}$ in which the columns specified in column_list have been multiplied by expr.

`mult_rows` performs the same task on the rows of $\mathcal{A}$.

Examples:

$$\texttt{mult\_columns}(\mathcal{A}, \{1,3\}, x) = \begin{pmatrix} x & 2 & 3*x \\ 4*x & 5 & 6*x \\ 7*x & 8 & 9*x \end{pmatrix}$$

$$\texttt{mult\_rows}(\mathcal{A}, 2, 10) = \begin{pmatrix} 1 & 2 & 3 \\ 40 & 50 & 60 \\ 7 & 8 & 9 \end{pmatrix}$$

Related functions:

`add_to_columns`, `add_to_rows`.

16.37.3.33 mult_rows

See: `mult_columns`.

16.37.3.34 pivot

Syntax:

`pivot(`$\mathcal{A}$`,r,c);`

$\mathcal{A}$	:-	a matrix.
r, c	:-	positive integers such that $\mathcal{A}(r,c) \neq 0$.

Synopsis:

`pivot` pivots $\mathcal{A}$ about its (r,c)th entry.

To do this, multiples of the r'th row are added to every other row in the matrix.

This means that the c'th column will be 0 except for the (r,c)'th entry.

Examples:

$$\texttt{pivot}(\mathcal{A}, 2, 3) = \begin{pmatrix} -1 & -0.5 & 0 \\ 4 & 5 & 6 \\ 1 & 0.5 & 0 \end{pmatrix}$$

Related functions:

`rows_pivot`.

16.37.3.35 pseudo_inverse

Syntax:

`pseudo_inverse(`$\mathcal{A}$`);`

$\mathcal{A}$:- a matrix containing only real numeric entries.

Synopsis:

`pseudo_inverse`, also known as the Moore-Penrose inverse, computes the pseudo inverse of $\mathcal{A}$.

Given the singular value decomposition of $\mathcal{A}$, i.e: $\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T$, then the pseudo inverse $\mathcal{A}^\dagger$ is defined by $\mathcal{A}^\dagger = \mathcal{V}\Sigma^\dagger\mathcal{U}^T$. For the diagonal matrix Σ, the pseudoinverse $\Sigma^\dagger$ is computed by taking the reciprocal of only the nonzero diagonal elements.

If $\mathcal{A}$ is square and non-singular, then $\mathcal{A}^\dagger = \mathcal{A}$. In general, however, $\mathcal{A}\mathcal{A}^\dagger\mathcal{A} = \mathcal{A}$, and $\mathcal{A}^\dagger\mathcal{A}\mathcal{A}^\dagger = \mathcal{A}^\dagger$.

Perhaps more importantly, $\mathcal{A}^\dagger$ solves the following least-squares problem: given a rectangular matrix $\mathcal{A}$ and a vector b, find the x minimizing $\|\mathcal{A}x - b\|_2$, and which, in addition, has minimum ℓ_2 (euclidean) Norm, $\|x\|_2$. This x is $\mathcal{A}^\dagger b$.

Examples:

$$\mathcal{R} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 8 & 7 & 6 \end{pmatrix}, \quad \text{pseudo\_inverse}(\mathcal{R}) = \begin{pmatrix} -0.2 & 0.1 \\ -0.05 & 0.05 \\ 0.1 & 0 \\ 0.25 & -0.05 \end{pmatrix}$$

Related functions:

`svd`.

16.37.3.36 random_matrix

Syntax:

```
random_matrix(r,c,limit);
```

r, c, limit :- positive integers.

Synopsis:

`random_matrix` creates an $r \times c$ matrix with random entries in the range $-\text{limit} < \text{entry} < \text{limit}$.

Switches:

`imaginary`	:-	if on, then matrix entries are $x + iy$ where $-\text{limit} < x, y < \text{limit}$.
`not_negative`	:-	if on then $0 <$ entry $<$ limit. In the imaginary case we have $0 < x, y <$ limit.
`only_integer`	:-	if on then each entry is an integer. In the imaginary case x, y are integers.
`symmetric`	:-	if on then the matrix is symmetric.
`upper_matrix`	:-	if on then the matrix is upper triangular.
`lower_matrix`	:-	if on then the matrix is lower triangular.

Examples:

$$\texttt{random\_matrix}(3,3,10) = \begin{pmatrix} -4.729721 & 6.987047 & 7.521383 \\ -5.224177 & 5.797709 & -4.321952 \\ -9.418455 & -9.94318 & -0.730980 \end{pmatrix}$$

```
on only_integer, not_negative, upper_matrix, imaginary;
```

$$\texttt{random\_matrix}(4,4,10) = \begin{pmatrix} 2*i+5 & 3*i+7 & 7*i+3 & 6 \\ 0 & 2*i+5 & 5*i+1 & 2*i+1 \\ 0 & 0 & 8 & i \\ 0 & 0 & 0 & 5*i+9 \end{pmatrix}$$

16.37.3.37 remove_columns, remove_rows

Syntax:

`remove_columns(`$\mathcal{A}$`,column_list);`

$\mathcal{A}$	:-	a matrix.
column_list	:-	either a positive integer or a list of positive integers.

Synopsis:

`remove_columns` removes the columns specified in column_list from $\mathcal{A}$.

`remove_rows` performs the same task on the rows of $\mathcal{A}$.

Examples:

$$\texttt{remove\_columns}(\mathcal{A}, 2) = \begin{pmatrix} 1 & 3 \\ 4 & 6 \\ 7 & 9 \end{pmatrix}$$

$$\texttt{remove\_rows}(\mathcal{A}, \{1,3\}) = \begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$$

Related functions:

`minor`.

16.37.3.38 remove_rows

See: `remove_columns`.

16.37.3.39 row_dim

See: `column_dim`.

16.37.3.40 rows_pivot

Syntax:

`rows_pivot(`$\mathcal{A}$`,r,c,{row_list});`

$\mathcal{A}$	:-	a matrix.
r,c	:-	positive integers such that $\mathcal{A}$(r,c) neq 0.
row_list	:-	positive integer or a list of positive integers.

Synopsis:

`rows_pivot` performs the same task as `pivot` but applies the pivot only to the rows specified in row_list.

Examples:

$$\mathcal{N} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$\texttt{rows\_pivot}(\mathcal{N}, 2, 3, \{4, 5\}) = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ -0.75 & 0 & 0.75 \\ -0.375 & 0 & 0.375 \end{pmatrix}$$

Related functions:

`pivot`.

16.37.3.41 simplex

Syntax:

`simplex(max/min,objective function,{linear inequalities},[{bounds`

max/min	:-	either max or min (signifying maximise and minimise).
objective function	:-	the function you are maximising or minimising.
linear inequalities	:-	the constraint inequalities. Each one must be of the form *sum of variables (<=, =, >=) number*.
bounds	:-	bounds on the variables as specified for the LP file format. Each bound is of one of the forms $l \leq v$, $v \leq u$, or $l \leq v \leq u$, where v is a variable and l, u are numbers or `infinity` or `-infinity`

Synopsis:

`simplex` applies the revised simplex algorithm to find the optimal(either maximum or minimum) value of the objective function under the linear inequality constraints.

It returns {optimal value,{ values of variables at this optimal}}.

The `{bounds}` argument is optional and admissible only when the switch `fastsimplex` is on, which is the default.

Without a `{bounds}` argument, the algorithm implies that all the variables are non-negative.

Examples:

```
simplex(max,x+y,{x>=10,y>=20,x+y<=25});

***** Error in simplex: Problem has no feasible solution.

simplex(max,10x+5y+5.5z,{5x+3z<=200,x+0.1y+0.5z<=12,
        0.1x+0.2y+0.3z<=9, 30x+10y+50z<=1500});

{525.0,{x=40.0,y=25.0,z=0}}
```

16.37.3.42 squarep

Syntax:

`squarep(`$\mathcal{A}$`);`

$\mathcal{A}$	:-	a matrix.

Synopsis:

`squarep` is a boolean function that returns t if the matrix is square and nil otherwise.

Examples:

$\mathcal{L} = \begin{pmatrix} 1 & 3 & 5 \end{pmatrix}$

`squarep`$(\mathcal{A}) =$ `t`

`squarep`$(\mathcal{L}) =$ `nil`

Related functions:

`matrixp`, `symmetricp`.

16.37.3.43 stack_rows

See: `augment_columns`.

16.37.3.44 sub_matrix

Syntax:

`sub_matrix(`$\mathcal{A}$`,row_list,column_list);`

$\mathcal{A}$	:-	a matrix.
row_list, column_list	:-	either a positive integer or a list of positive integers.

Synopsis:

`sub_matrix` produces the matrix consisting of the intersection of the rows specified in row_list and the columns specified in column_list.

Examples:

`sub_matrix`$(\mathcal{A}, \{1,3\}, \{2,3\}) = \begin{pmatrix} 2 & 3 \\ 8 & 9 \end{pmatrix}$

Related functions:

`augment_columns`, `stack_rows`.

16.37.3.45 svd (singular value decomposition)

Syntax:

`svd(`$\mathcal{A}$`);`

$\mathcal{A}$	:-	a matrix containing only real numeric entries.

Synopsis:

`svd` computes the singular value decomposition of $\mathcal{A}$. If A is an $m \times n$ real matrix of (column) rank r, `svd` returns the 3-element list $\{\mathcal{U}, \Sigma, \mathcal{V}\}$ where $\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T$.

Let $k = \min(m, n)$. Then U is $m \times k$, V is $n \times k$, and and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_k)$, where $\sigma_i \geq 0$ are the singular values of $\mathcal{A}$; only r of these are non-zero. The singular values are the non-negative square roots of the eigenvalues of $\mathcal{A}^T\mathcal{A}$.

$\mathcal{U}$ and $\mathcal{V}$ are such that $\mathcal{U}\mathcal{U}^T = \mathcal{V}\mathcal{V}^T = \mathcal{V}^T\mathcal{V} = \mathcal{I}_k$.

Note: there are a number of different definitions of SVD in the literature, in some of which Σ is square and U and V rectangular, as here, but in others U and V are square, and Σ is rectangular.

Examples:

$$\mathcal{Q} = \begin{pmatrix} 1 & 3 \\ -4 & 3 \\ 3 & 6 \end{pmatrix}$$

$$\texttt{svd}(\mathcal{Q}) = \left\{ \begin{pmatrix} 0.0236042 & 0.419897 \\ -0.969049 & 0.232684 \\ 0.245739 & 0.877237 \end{pmatrix}, \begin{pmatrix} 4.83288 & 0 \\ 0 & 7.52618 \end{pmatrix}, \begin{pmatrix} 0.959473 & 0.281799 \\ -0.281799 & 0.959473 \end{pmatrix} \right\}$$

$$\texttt{svd}(\texttt{TP}(\mathcal{Q})) = \left\{ \begin{pmatrix} 0.959473 & 0.281799 \\ -0.281799 & 0.959473 \end{pmatrix}, \begin{pmatrix} 4.83288 & 0 \\ 0 & 7.52618 \end{pmatrix}, \begin{pmatrix} 0.0236042 & 0.419897 \\ -0.969049 & 0.232684 \\ 0.245739 & 0.877237 \end{pmatrix} \right\}$$

16.37.3.46 swap_columns, swap_rows

Syntax:

`swap_columns(`$\mathcal{A}$`,c1,c2);`

$\mathcal{A}$	:-	a matrix.
c1,c1	:-	positive integers.

Synopsis:

`swap_columns` swaps column c1 of $\mathcal{A}$ with column c2.

`swap_rows` performs the same task on 2 rows of $\mathcal{A}$.

Examples:

$$\texttt{swap\_columns}(\mathcal{A},2,3) = \begin{pmatrix} 1 & 3 & 2 \\ 4 & 6 & 5 \\ 7 & 9 & 8 \end{pmatrix}$$

Related functions:
`swap_entries`.

16.37.3.47 swap_entries

Syntax:
`swap_entries(`$\mathcal{A}$`,{r1,c1},{r2,c2});`

$\mathcal{A}$	:-	a matrix.
r1,c1,r2,c2	:-	positive integers.

Synopsis:
`swap_entries` swaps $\mathcal{A}$(r1,c1) with $\mathcal{A}$(r2,c2).

Examples:

$$\texttt{swap\_entries}(\mathcal{A},\{1,1\},\{3,3\}) = \begin{pmatrix} 9 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}$$

Related functions:
`swap_columns`, `swap_rows`.

16.37.3.48 swap_rows

See: `swap_columns`.

16.37.3.49 symmetricp

Syntax:
`symmetricp(`$\mathcal{A}$`);`

$\mathcal{A}$	:-	a matrix.

Synopsis:
`symmetricp` is a boolean function that returns t if the matrix is symmetric and nil otherwise.

Examples:

$$\mathcal{M} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

`symmetricp`$(\mathcal{A}) =$ `nil`

`symmetricp`$(\mathcal{M}) =$ `t`

Related functions:

`matrixp`, `squarep`.

16.37.3.50 toeplitz

Syntax:

`toeplitz({expr`$_1$`,expr`$_2$`, ...,expr`$_n$`});` [21]

expr$_1$,expr$_2$, ...,expr$_n$:- algebraic expressions.

Synopsis:

`toeplitz` creates the toeplitz matrix from the expression list.

This is a square symmetric matrix in which the first expression is placed on the diagonal and the i'th expression is placed on the (i-1)'th sub and super diagonals.

It has dimension n where n is the number of expressions.

Examples:

`toeplitz`$(\{w, x, y, z\}) = \begin{pmatrix} w & x & y & z \\ x & w & x & y \\ y & x & w & x \\ z & y & x & w \end{pmatrix}$

16.37.3.51 triang_adjoint

Syntax:

`triang_adjoint(`$\mathcal{A}$`);`

$\mathcal{A}$:- a matrix.

Synopsis:

[21] If you're feeling lazy then the {}'s can be omitted.

`triang_adjoint` computes the triangularizing adjoint $\mathcal{F}$ of matrix $\mathcal{A}$ due to the algorithm of Arne Storjohann. $\mathcal{F}$ is lower triangular matrix and the resulting matrix $\mathcal{T}$ of $\mathcal{F} * \mathcal{A} = \mathcal{T}$ is upper triangular with the property that the i-th entry in the diagonal of $\mathcal{T}$ is the determinant of the principal i-th submatrix of the matrix $\mathcal{A}$.

Examples:

$$\texttt{triang\_adjoint}(\mathcal{A}) = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -3 & 6 & -3 \end{pmatrix}$$

$$\mathcal{F} * \mathcal{A} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{pmatrix}$$

16.37.3.52 Vandermonde

Syntax:

`vandermonde`({expr$_1$,expr$_2$, ...,expr$_n$}); [22]

expr$_1$,expr$_2$, ...,expr$_n$:- algebraic expressions.

Synopsis:

`Vandermonde` creates the Vandermonde matrix from the expression list. This is the square matrix in which the (i, j)th entry is $\text{expr}_i^{(j-1)}$. It has dimension n, where n is the number of expressions.

Examples:

$$\texttt{vandermonde}(\{x, 2 * y, 3 * z\}) = \begin{pmatrix} 1 & x & x^2 \\ 1 & 2 * y & 4 * y^2 \\ 1 & 3 * z & 9 * z^2 \end{pmatrix}$$

16.37.3.53 kronecker_product

Syntax:

`kronecker_product`(M_1, M_2)

M_1, M_2 :- Matrices

Synopsis:

`kronecker_product` creates a matrix containing the Kronecker product (also called `direct product` or `tensor product`) of its arguments.

[22] If you're feeling lazy then the {}'s can be omitted.

Examples:

```
a1 := mat((1,2),(3,4),(5,6))$
a2 := mat((1,1,1),(2,z,2),(3,3,3))$
kronecker_product(a1,a2);
```

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 2 & z & 2 & 4 & 2*z & 4 \\ 3 & 3 & 3 & 6 & 6 & 6 \\ 3 & 3 & 3 & 4 & 4 & 4 \\ 6 & 3*z & 6 & 8 & 4*z & 8 \\ 9 & 9 & 9 & 12 & 12 & 12 \\ 5 & 5 & 5 & 6 & 6 & 6 \\ 10 & 5*z & 10 & 12 & 6*z & 12 \\ 15 & 15 & 15 & 18 & 18 & 18 \end{pmatrix}$$

16.37.4 Fast Linear Algebra

By turning the `fast_la` switch on, the speed of the following functions will be increased:

add_columns	add_rows	augment_columns	column_dim
copy_into	make_identity	matrix_augment	matrix_stack
minor	mult_column	mult_row	pivot
remove_columns	remove_rows	rows_pivot	squarep
stack_rows	sub_matrix	swap_columns	swap_entries
swap_rows	symmetricp		

The increase in speed will be insignificant unless you are making a significant number(i.e: thousands) of calls. When using this switch, error checking is minimised. This means that illegal input may give strange error messages. Beware.

16.37.5 Acknowledgments

Many of the ideas for this package came from the Maple[3] Linalg package [4].

The algorithms for `cholesky`, `lu_decom`, and `svd` are taken from the book Linear Algebra - J.H. Wilkinson & C. Reinsch[5].

The `gram_schmidt` code comes from Karin Gatermann's Symmetry package[6] for REDUCE.

Bibliography

[1] Matt Rebbeck: NORMFORM: A REDUCE package for the computation of various matrix normal forms. ZIB, Berlin. (1993)

[2] Anthony C. Hearn: REDUCE User's Manual 3.6. RAND (1995)

[3] Bruce W. Char. . . [et al.]: Maple (Computer Program). Springer-Verlag (1991)

[4] Linalg - a linear algebra package for Maple[3].

[5] J. H. Wilkinson & C. Reinsch: Linear Algebra (volume II). Springer-Verlag (1971)

[6] Karin Gatermann: Symmetry: A REDUCE package for the computation of linear representations of groups. ZIB, Berlin. (1992)