16.62 SPARSE: Sparse Matrix Calculations

Author: Stephen Scowcroft

16.62.1 Introduction

A very powerful feature of REDUCE is the ease with which matrix calculations can be performed. This package extends the available matrix feature to enable calculations with sparse matrices. This package also provides a selection of functions that are useful in the world of linear algebra with respect to sparse matrices.

Loading the Package

The package is loaded by: `load_package sparse;`

16.62.2 Sparse Matrix Calculations

To extend the the syntax to this class of calculations we need to add an expression type `sparse`.

16.62.2.1 Sparse Variables

An identifier may be declared a sparse variable by the declaration `SPARSE`. The size of the sparse matrix must be declared explicitly in the matrix declaration. For example,

```
sparse aa(10,1),bb(200,200);
```

declares `AA` to be a 10 x 1 (column) sparse matrix and `Y` to be a 200 x 200 sparse matrix. The declaration `SPARSE` is similar to the declaration `MATRIX`. Once a symbol is declared to name a sparse matrix, it can not also be used to name an array, operator, procedure, or used as an ordinary variable. For more information see the Matrix Variables section (14.2).

16.62.2.2 Assigning Sparse Matrix Elements

Once a matix has been declared a sparse matrix all elements of the matrix are initialized to 0. Thus when a sparse matrix is initially referred to the message

```
"The matrix is dense, contains only zeros"
```

is returned. When printing out a matrix only the non-zero elements are printed. This is due to the fact that only the non-zero elements of the matrix are stored. To assign the elements of the declared matrix we use the following syntax. Assuming `AA` and `BB` have been declared as spasre matrices, we simply write,

```
aa(1,1):=10;
bb(100,150):=a;
```

etc. This then sets the element in the first row and first column to 10, or the element in the 100th row and 150th column to `a`.

16.62.2.3 Evaluating Sparse Matrix Elements

Once an element of a sparse matrix has been assingned, it may be referred to in standard array element notation. Thus `aa(2,1)` refers to the element in the second row and first column of the sparse matrix `AA`.

16.62.3 Sparse Matrix Expressions

These follow the normal rules of matrix algebra. Sums and products must be of compatible size; otherwise an error will result during evaluation. Similarly, only square matrices may be raised to a power. A negative power is computed as the inverse of the matrix raised to the corresponding positive power. For more information and the syntax for matrix algebra see the Matrix Expressions section (14.3).

16.62.4 Operators with Sparse Matrix Arguments

The operators in the Sparse Matix Package are the same as those in the Matrix Packge with the exception that the `NULLSPACE` operator is not defined. See section Operators with Matrix Arguments (14.4) for more details.

16.62.4.1 Examples

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

$$\mathcal{AA} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

```
det ppp;

135

trace ppp;

18

rank ppp;

4

spmateigen(ppp,eta);

{{eta - 1,1,

  spm(1,1) := arbcomplex(4)$
  },

 {eta - 3,1,

  spm(2,1) := arbcomplex(5)$
  },

 {eta - 5,1,

  spm(3,1) := arbcomplex(6)$
  },

 {eta - 9,1,

  spm(4,1) := arbcomplex(7)$
  }}
```

16.62.5 The Linear Algebra Package for Sparse Matrices

This package is an extension of the Linear Algebra Package for REDUCE described in section 16.37. These functions are described alphabetically in section 16.62.6. They can be classified into four sections(n.b: the numbers after the dots signify the function label in section 6).

16.62.5.1 Basic matrix handling

spadd_columns	...	16.62.6.1	spadd_rows	...	16.62.6.2
spadd_to_columns	...	16.62.6.3	spadd_to_rows	...	16.62.6.4
spaugment_columns	...	16.62.6.5	spchar_poly	...	16.62.6.9
spcol_dim	...	16.62.6.12	spcopy_into	...	16.62.6.14
spdiagonal	...	16.62.6.15	spextend	...	16.62.6.16
spfind_companion	...	16.62.6.17	spget_columns	...	16.62.6.18
spget_rows	...	16.62.6.19	sphermitian_tp	...	16.62.6.21
spmatrix_augment	...	16.62.6.27	spmatrix_stack	...	16.62.6.29
spminor	...	16.62.6.30	spmult_columns	...	16.62.6.31
spmult_rows	...	16.62.6.32	sppivot	...	16.62.6.33
spremove_columns	...	16.62.6.35	spremove_rows	...	16.62.6.36
sprow_dim	...	16.62.6.37	sprows_pivot	...	16.62.6.38
spstack_rows	...	16.62.6.41	spsub_matrix	...	16.62.6.42
spswap_columns	...	16.62.6.44	spswap_entries	...	16.62.6.45
spswap_rows	...	16.62.6.46			

16.62.5.2 Constructors

Functions that create sparse matrices.

spband_matrix	...	16.62.6.6	spblock_matrix	...	16.62.6.7
spchar_matrix	...	16.62.6.11	spcoeff_matrix	...	16.62.6.11
spcompanion	...	16.62.6.13	sphessian	...	16.62.6.22
spjacobian	...	16.62.6.23	spjordan_block	...	16.62.6.24
spmake_identity	...	16.62.6.26			

16.62.5.3 High level algorithms

spchar_poly	...	16.62.6.9	spcholesky	...	16.62.6.10
spgram_schmidt	...	16.62.6.20	splu_decom	...	16.62.6.25
sppseudo_inverse	...	16.62.6.34	spsvd	...	16.62.6.43

16.62.5.4 Predicates

matrixp	...	16.62.6.28	sparsematp	...	16.62.6.39
squarep	...	16.62.6.40	symmetricp	...	16.62.6.47

Note on examples:

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

$$\mathcal{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Unfortunately, due to restrictions of size, it is not practical to use "large" sparse matrices in the examples. As a result the examples shown may appear trivial, but they give an idea of how the functions work.

Notation

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

16.62.6 Available Functions

16.62.6.1 spadd_columns, spadd_rows

Syntax:

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

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

Synopsis:

`spadd_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:

$$\texttt{spadd\_columns}(\mathcal{A},1,2,x) = \begin{pmatrix} 1 & x & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

$$\texttt{spadd\_rows}(\mathcal{A},2,3,5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 25 & 9 \end{pmatrix}$$

Related functions:
`spadd_to_columns`, `spadd_to_rows`, `spmult_columns`, `spmult_rows`.

16.62.6.2 spadd_rows

See: `spadd_columns`.

16.62.6.3 spadd_to_columns, spadd_to_rows

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

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

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

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

Examples:

$$\texttt{spadd\_to\_columns}(\mathcal{A},\{1,2\},10) = \begin{pmatrix} 11 & 10 & 0 \\ 10 & 15 & 0 \\ 10 & 10 & 9 \end{pmatrix}$$

$$\texttt{spadd\_to\_rows}(\mathcal{A},2,-x) = \begin{pmatrix} 1 & 0 & 0 \\ -x & -x+5 & -x \\ 0 & 0 & 9 \end{pmatrix}$$

Related functions:
`spadd_columns`, `spadd_rows`, `spmult_rows`, `spmult_columns`.

16.62.6.4 spadd_to_rows

See: `spadd_to_columns`.

16.62.6.5 spaugment_columns, spstack_rows

Syntax:

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

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

Synopsis:

`spaugment_columns` gets hold of the columns of $\mathcal{A}$ specified in column_list and sticks them together.

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

Examples:

$$\texttt{spaugment\_columns}(\mathcal{A},\{1,2\}) = \begin{pmatrix} 1 & 0 \\ 0 & 5 \\ 0 & 0 \end{pmatrix}$$

$$\texttt{spstack\_rows}(\mathcal{A},\{1,3\}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Related functions:

`spget_columns`, `spget_rows`, `spsub_matrix`.

16.62.6.6 spband_matrix

Syntax:

`spband_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:

`spband_matrix` creates a sparse square matrix of dimension square_size.

Examples:

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

Related functions:

`spdiagonal`.

16.62.6.7 spblock_matrix

Syntax:

`spblock_matrix(r,c,matrix_list);`

r,c	:-	positive integers.
matrix_list	:-	a list of matrices of either sparse or matrix type.

Synopsis:

`spblock_matrix` creates a sparse matrix that consists of r by c matrices filled from the matrix_list row wise.

Examples:

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

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

16.62.6.8 spchar_matrix

Syntax:

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

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

Synopsis:

`spchar_matrix` creates the characteristic matrix $\mathcal{C}$ of $\mathcal{A}$.

This is $\mathcal{C} = \lambda * \mathcal{I} - \mathcal{A}$.

Examples:

$$\text{spchar\_matrix}(\mathcal{A},x) = \begin{pmatrix} x-1 & 0 & 0 \\ 0 & x-5 & 0 \\ 0 & 0 & x-9 \end{pmatrix}$$

Related functions:
`spchar_poly`.

16.62.6.9 spchar_poly

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

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

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

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

Examples:
`spchar_poly(`$\mathcal{A}$`,`x`)` $= x^3 - 15 * x^2 - 59 * x - 45$

Related functions:
`spchar_matrix`.

16.62.6.10 spcholesky

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

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

Synopsis:
`spcholesky` 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:

$$\mathcal{F} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

$$\texttt{cholesky}(\mathcal{F}) = \left\{ \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{5} & 0 \\ 0 & 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{5} & 0 \\ 0 & 0 & 3 \end{pmatrix} \right\}$$

Related functions:

splu_decom.

16.62.6.11 spcoeff_matrix

Syntax:

spcoeff_matrix({lin_eqn$_1$,lin_eqn$_2$, ...,lin_eqn$_n$});

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:

spcoeff_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:

$$\texttt{spcoeff\_matrix}(\{y - 20 * w = 10, y - z = 20, y + 4 + 3 * z, w + x + 50\}) =$$

$$\left\{ \begin{pmatrix} 1 & -20 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} y \\ w \\ z \\ x \end{pmatrix}, \begin{pmatrix} 10 \\ 20 \\ -4 \\ 50 \end{pmatrix} \right\}$$

16.62.6.12 spcol_dim, sprow_dim

Syntax:

column_dim($\mathcal{A}$);

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

Synopsis:

`spcol_dim` finds the column dimension of $\mathcal{A}$.
`sprow_dim` finds the row dimension of $\mathcal{A}$.

Examples:

`spcol_dim`($\mathcal{A}$) = 3

16.62.6.13 spcompanion

Syntax:

`spcompanion(poly,x);`

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

Synopsis:

`spcompanion` 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:

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

Related functions:

`spfind_companion`.

16.62.6.14 spcopy_into

Syntax:

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

$\mathcal{A}, \mathcal{B}$	:-	matrices of type sparse or matrix.
r,c	:-	positive integers.

Synopsis:

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

Examples:

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

$$\texttt{spcopy\_into}(\mathcal{A}, \mathcal{G}, 1, 2) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Related functions:
`spaugment_columns`, `spextend`, `spmatrix_augment`, `spmatrix_stack`, `spstack_rows`, `spsub_matrix`.

16.62.6.15 spdiagonal

Syntax:
`spdiagonal({mat`$_1$`,mat`$_2$`, ...,mat`$_n$`});`[32]

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

Synopsis:
`spdiagonal` creates a sparse matrix that contains the input on the diagonal.

Examples:

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

$$\texttt{spdiagonal}(\{\mathcal{A}, x, \mathcal{H}\}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 66 & 77 \\ 0 & 0 & 0 & 0 & 88 & 99 \end{pmatrix}$$

Related functions:
`spjordan_block`.

[32] The {}'s can be omitted.

16.62.6.16 spextend

Syntax:

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

$\mathcal{A}$	:-	a sparse matrix.
r,c	:-	positive integers.
expr	:-	algebraic expression or symbol.

Synopsis:

`spextend` 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:

$$\texttt{spextend}(\mathcal{A}, 1, 2, 0) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Related functions:

`spcopy_into`, `spmatrix_augment`, `spmatrix_stack`, `spremove_columns`, `spremove_rows`.

16.62.6.17 spfind_companion

Syntax:

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

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

Synopsis:

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

Examples:

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

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

Related functions:

`spcompanion`.

16.62.6.18 spget_columns, spget_rows

Syntax:

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

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

Synopsis:

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

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

Examples:

$$\texttt{spget\_columns}(\mathcal{A},\{1,3\}) = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 9 \end{pmatrix} \right\}$$

$$\texttt{spget\_rows}(\mathcal{A},2) = \left\{ \begin{pmatrix} 0 & 5 & 0 \end{pmatrix} \right\}$$

Related functions:

`spaugment_columns`, `spstack_rows`, `spsub_matrix`.

16.62.6.19 spget_rows

See: `spget_columns`.

16.62.6.20 spgram_schmidt

Syntax:

`spgram_schmidt({vec`$_1$`,vec`$_2$`, ...,vec`$_n$`});`

vec$_1$,vec$_2$, ...,vec$_n$	:-	linearly independent vectors. Each vector must be written as a list of predefined sparse (column) matrices, eg: sparse a(4,1);, a(1,1):=1;

Synopsis:

`spgram_schmidt` performs the gram_schmidt orthonormalisation on the input vectors.

It returns a list of orthogonal normalised vectors.

Examples:

Suppose a,b,c,d correspond to sparse matrices representing the following lists: {{1,0,0,0},{1,1,0,0},{1,1,1,0},{1,1,1,1}}.

`spgram_schmidt({{a},{b},{c},{d}}) =`

{{1,0,0,0},{0,1,0,0},{0,0,1,0},{0,0,0,1}}

16.62.6.21 sphermitian_tp

Syntax:

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

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

Synopsis:

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

Examples:

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

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

Related functions:

`tp`[33].

16.62.6.22 sphessian

Syntax:

`sphessian(expr,variable_list);`

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

Synopsis:

`sphessian` computes the hessian matrix of expr w.r.t. the variables in variable_list.

[33] standard reduce call for the transpose of a matrix - see section 14.4.

Examples:

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

16.62.6.23 spjacobian

Syntax:

`spjacobian(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:

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

Examples:

$$\texttt{spjacobian(}\{x^4,x*y^2,x*y*z^3\}\texttt{,}\{w,x,y,z\}\texttt{)} =$$

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

Related functions:

`sphessian`, `df`[34].

16.62.6.24 spjordan_block

Syntax:

`spjordan_block(expr,square_size);`

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

Synopsis:

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

[34] standard reduce call for differentiation - see 7.8.

Examples:

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

Related functions:
`spdiagonal`, `spcompanion`.

16.62.6.25 splu_decom

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

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

Synopsis:
`splu_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, `splu_decom` returns $\{\mathcal{L},\mathcal{U}$,vec$\}$. The call `spconvert(`$\mathcal{A}$`,vec)` will return the sparse matrix that has been decomposed, ie: $\mathcal{L}\mathcal{U} =$ `spconvert(`$\mathcal{A}$`,vec)`.

Examples:

$$\mathcal{K} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

$$\texttt{lu} := \texttt{splu\_decom}(\mathcal{K}) = \left\{ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, [\,1\;2\;3\,] \right\}$$

$$\texttt{first lu * second lu} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

$$\texttt{convert}(\mathcal{K},\texttt{third lu}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Related functions:
`spcholesky`.

16.62.6.26 spmake_identity

Syntax:
`spmake_identity(square_size);`

square_size :- a positive integer.

Synopsis:
`spmake_identity` creates the identity matrix of dimension square_size.

Examples:

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

Related functions:
`spdiagonal`.

16.62.6.27 spmatrix_augment, spmatrix_stack

Syntax:
`spmatrix_augment({mat`$_1$`,mat`$_2$`, ...,mat`$_n$`});`[35]

mat$_1$,mat$_2$, …,mat$_n$:- matrices.

[35]The { }'s can be omitted.

Synopsis:

`spmatrix_augment` joins the matrices in matrix_list together horizontally.

`spmatrix_stack` joins the matrices in matrix_list together vertically.

Examples:

$$\texttt{spmatrix\_augment}(\{\mathcal{A},\mathcal{A}\}) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 5 & 0 & 0 & 5 & 0 \\ 0 & 0 & 9 & 0 & 0 & 9 \end{pmatrix}$$

$$\texttt{spmatrix\_stack}(\{\mathcal{A},\mathcal{A}\}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \\ 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Related functions:

`spaugment_columns`, `spstack_rows`, `spsub_matrix`.

16.62.6.28 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 of type sparse or matrix and nil otherwise.

Examples:

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

`matrixp`(doodlesackbanana) = nil

Related functions:

`squarep`, `symmetricp`, `sparsematp`.

16.62.6.29 spmatrix_stack

See: `spmatrix_augment`.

16.62.6.30 spminor

Syntax:

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

$\mathcal{A}$	:-	a sparse matrix.
r,c	:-	positive integers.

Synopsis:

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

Examples:

$$\texttt{spminor}(\mathcal{A}, 1, 3) = \begin{pmatrix} 0 & 5 \\ 0 & 0 \end{pmatrix}$$

Related functions:

`spremove_columns`, `spremove_rows`.

16.62.6.31 spmult_columns, spmult_rows

Syntax:

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

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

Synopsis:

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

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

Examples:

$$\texttt{spmult\_columns}(\mathcal{A}, \{1, 3\}, x) = \begin{pmatrix} x & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 * x \end{pmatrix}$$

$$\texttt{spmult\_rows}(\mathcal{A}, 2, 10) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Related functions:

`spadd_to_columns`, `spadd_to_rows`.

16.62.6.32 spmult_rows

See: `spmult_columns`.

16.62.6.33 sppivot

Syntax:

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

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

Synopsis:

`sppivot` pivots $\mathcal{A}$ about it's (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.

Related functions:

`sprows_pivot`.

16.62.6.34 sppseudo_inverse

Syntax:

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

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

Synopsis:

`sppseudo_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:

$$\mathcal{R} = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 9 & 0 & 7 & 0 \end{pmatrix}$$

$$\texttt{sppseudo\_inverse}(\mathcal{R}) = \begin{pmatrix} -0.26 & 0.11 \\ 0 & 0 \\ 0.33 & 0 \\ 0.25 & -0.05 \end{pmatrix}$$

Related functions:
`spsvd`.

16.62.6.35 spremove_columns, spremove_rows

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

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

Synopsis:
`spremove_columns` removes the columns specified in column_list from $\mathcal{A}$.

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

Examples:

$$\texttt{spremove\_columns}(\mathcal{A}, 2) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 9 \end{pmatrix}$$

$$\texttt{spremove\_rows}(\mathcal{A}, \{1, 3\}) = \begin{pmatrix} 0 & 5 & 0 \end{pmatrix}$$

Related functions:
`spminor`.

16.62.6.36 spremove_rows

See: `spremove_columns`.

16.62.6.37 sprow_dim

See: `spcolumn_dim`.

16.62.6.38 sprows_pivot

Syntax:

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

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

Synopsis:

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

Related functions:

`sppivot`.

16.62.6.39 sparsematp

Syntax:

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

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

Synopsis:

`sparsematp` is a boolean function that returns t if the matrix is declared sparse and nil otherwise.

Examples:

$\mathcal{L}$:= `mat((1,2,3),(4,5,6),(7,8,9));`

$\texttt{sparsematp}(\mathcal{A}) = \texttt{t}$

$\texttt{sparsematp}(\mathcal{L}) = \texttt{nil}$

Related functions:

`matrixp`, `symmetricp`, `squarep`.

16.62.6.40 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:

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

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

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

Related functions:

`matrixp`, `symmetricp`, `sparsematp`.

16.62.6.41 spstack_rows

See: `spaugment_columns`.

16.62.6.42 spsub_matrix

Syntax:

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

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

Synopsis:

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

Examples:

$$\texttt{spsub\_matrix}(\mathcal{A}, \{1,3\}, \{2,3\}) = \begin{pmatrix} 5 & 0 \\ 0 & 9 \end{pmatrix}$$

Related functions:

spaugment_columns, spstack_rows.

16.62.6.43 spsvd (singular value decomposition)

Syntax:

spsvd($\mathcal{A}$);

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

Synopsis:

spsvd 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 $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$, 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:

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

$$\mathtt{svd}(\mathcal{Q}) = \left\{ \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1.0 & 0 \\ 0 & 5.0 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \right\}$$

16.62.6.44 spswap_columns, spswap_rows

Syntax:

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

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

Synopsis:

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

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

Examples:

$$\texttt{spswap\_columns}(\mathcal{A}, 2, 3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 5 \\ 0 & 9 & 0 \end{pmatrix}$$

Related functions:

spswap_entries.

16.62.6.45 swap_entries

Syntax:

spswap_entries($\mathcal{A}$,{r1,c1},{r2,c2});

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

Synopsis:

spswap_entries swaps $\mathcal{A}$(r1,c1) with $\mathcal{A}$(r2,c2).

Examples:

$$\texttt{spswap\_entries}(\mathcal{A}, \{1, 1\}, \{3, 3\}) = \begin{pmatrix} 9 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Related functions:

spswap_columns, spswap_rows.

16.62.6.46 spswap_rows

See: spswap_columns.

16.62.6.47 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:

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

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

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

Related functions:

`matrixp`, `squarep`, `sparsematp`.

16.62.7 Fast Linear Algebra

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

spadd_columns	spadd_rows	spaugment_columns	spcol_dim
spcopy_into	spmake_identity	spmatrix_augment	spmatrix_stack
spminor	spmult_column	spmult_row	sppivot
spremove_columns	spremove_rows	sprows_pivot	squarep
spstack_rows	spsub_matrix	spswap_columns	spswap_entries
spswap_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.

16.62.8 Acknowledgments

This package is an extention of the code from the Linear Algebra Package for REDUCE by Matt Rebbeck (cf. section 16.37).

The algorithms for `spcholesky`, `splu_decom`, and `spsvd` are taken from the book Linear Algebra - J.H. Wilkinson & C. Reinsch[3].

The `spgram_schmidt` code comes from Karin Gatermann's Symmetry package[4] for REDUCE.

Bibliography

[1] Matt Rebbeck: A Linear Algebra Package for REDUCE, ZIB , Berlin. (1994)

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

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

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