Package org.ejml.dense.row

Class CommonOps_DDRM

java.lang.Object
    org.ejml.dense.row.CommonOps_DDRM

```
public class CommonOps_DDRM
extends Object
```

Common matrix operations are contained here. Which specific underlying algorithm is used is not specified just the out come of the operation. Nor should calls to these functions reply on the underlying implementation. Which algorithm is used can depend on the matrix being passed in.

For more exotic and specialized generic operations see SpecializedOps_DDRM.

See Also:

MatrixMatrixMult_DDRM, MatrixVectorMult_DDRM, SpecializedOps_DDRM, MatrixFeatures_DDRM

Constructor Summary

Constructors

Constructor	Description
CommonOps_DDRM()	

Method Summary

All Methods Static Methods Concrete Methods

Modifier and Type	Method	Description
static void	abs(DMatrixD1 a)	Performs absolute value of a matrix: a = abs(a) a_{ij} = abs(a_{ij})
static void	abs(DMatrixD1 a, DMatrixD1 c)	Performs absolute value of a matrix: c = abs(a) c_{ij} = abs(a_{ij})
static <T extends DMatrixD1> T	add(double alpha, T a, double beta, T b, T output)	Performs the following operation: c = α * a + β * b c_{ij} = α * a_{ij} + β * b_{ij}

Modifier and Type	Method	Description
static <T extends DMatrixD1> T	add(double alpha, T a, T b, T output)	Performs the following operation: c = α * a + b $c_{ij} = \alpha * a_{ij} + b_{ij}$
static void	add(DMatrixD1 a, double val)	Performs an in-place scalar addition: a = a + val $a_{ij} = a_{ij} + val$
static <T extends DMatrixD1> T	add(T a, double val, T output)	Performs scalar addition: c = a + val $c_{ij} = a_{ij} + val$
static <T extends DMatrixD1> T	add(T a, double beta, T b, T output)	Performs the following operation: c = a + β * b $c_{ij} = a_{ij} + \beta * b_{ij}$
static <T extends DMatrixD1> T	add(T a, T b, T output)	Performs the following operation: c = a + b $c_{ij} = a_{ij} + b_{ij}$
static void	addEquals(DMatrixD1 a, double beta, DMatrixD1 b)	Performs the following operation: a = a + β * b $a_{ij} = a_{ij} + \beta * b_{ij}$
static void	addEquals(DMatrixD1 a, DMatrixD1 b)	Performs the following operation: a = a + b $a_{ij} = a_{ij} + b_{ij}$
static DMatrixRMaj	apply(DMatrixRMaj input, DOperatorUnary func)	
static DMatrixRMaj	apply(DMatrixRMaj input, DOperatorUnary func, @Nullable DMatrixRMaj output)	This applies a given unary function on every value stored in the matrix
static void	changeSign(DMatrixD1 a)	Changes the sign of every element in the matrix. $a_{ij} = -a_{ij}$
static <T extends DMatrixD1> T	changeSign(T input, T output)	Changes the sign of every element in the matrix. $output_{ij} = -input_{ij}$

Modifier and Type	Method	Description
static DMatrixRMaj[]	columnsToVector(DMatrixRMaj A, @Nullable DMatrixRMaj[] v)	Converts the columns in a matrix into a set of vectors.
static DMatrixRMaj	concatColumns(DMatrixRMaj a, DMatrixRMaj b, @Nullable DMatrixRMaj output)	output = [a , b]
static DMatrixRMaj	concatColumnsMulti (DMatrixRMaj... m)	Concatinates all the matrices together along their columns.
static void	concatRows(DMatrixRMaj a, DMatrixRMaj b, DMatrixRMaj output)	output = [a ; b]
static DMatrixRMaj	concatRowsMulti (DMatrixRMaj... m)	Concatinates all the matrices together along their columns.
static int	countTrue(BMatrixRMaj A)	Counts the number of elements in A which are true
static double	det(DMatrixRMaj mat)	Returns the determinant of the matrix.
static DMatrixRMaj	diag(double... diagEl)	Creates a new square matrix whose diagonal elements are specified by diagEl and all the other elements are zero. a_{ij} = 0 if i ≤ j a_{ij} = diag[i] if i = j
static DMatrixRMaj	diag(@Nullable DMatrixRMaj ret, int width, double... diagEl)	
static DMatrixRMaj	diagR(int numRows, int numCols, double... diagEl)	Creates a new rectangular matrix whose diagonal elements are specified by diagEl and all the other elements are zero. a_{ij} = 0 if i ≤ j a_{ij} = diag[i] if i = j
static void	divide(double alpha, DMatrixD1 a)	Performs an in-place element by element scalar division with the scalar on top. $a_{ij} = \alpha / a_{ij}$
static <T extends DMatrixD1> T	divide(double alpha, T input, T output)	Performs an element by element scalar division with the scalar on top. $b_{ij} = \alpha / a_{ij}$

Modifier and Type	Method	Description
static void	divide(DMatrixD1 a, double alpha)	Performs an in-place element by element scalar division with the scalar on bottom. $a_{ij} = a_{ij}/\alpha$
static <T extends DMatrixD1> T	divide(T input, double alpha, T output)	Performs an element by element scalar division with the scalar on botton. $b_{ij} = a_{ij}/\alpha$
static void	divideCols(DMatrixRMaj A, double[] values)	Divides every element in column i by value[i].
static void	divideRows(double[] values, DMatrixRMaj A)	Divides every element in row i by value[i].
static void	divideRowsCols(double[] diagA, int offsetA, DMatrixRMaj B, double[] diagC, int offsetC)	Equivalent to multiplying a matrix B by the inverse of two diagonal matrices.
static double	dot(DMatrixD1 a, DMatrixD1 b)	Computes the dot product or inner product between two vectors.
static void	elementDiv(DMatrixD1 A, DMatrixD1 B)	Performs the an element by element division operation: $a_{ij} = a_{ij} / b_{ij}$
static <T extends DMatrixD1> T	elementDiv(T A, T B, T output)	Performs the an element by element division operation: $c_{ij} = a_{ij} / b_{ij}$
static <T extends DMatrixD1> T	elementExp(T A, T output)	Element-wise exp operation $c_{ij} = \text{Math.exp}(a_{ij})$
static BMatrixRMaj	elementLessThan(DMatrixRMaj A, double value, BMatrixRMaj output)	Applies the > operator to each element in A.
static BMatrixRMaj	elementLessThan(DMatrixRMaj A, DMatrixRMaj B, BMatrixRMaj output)	Applies the < operator to each element in A.
static BMatrixRMaj	elementLessThanOrEqual (DMatrixRMaj A, double value, BMatrixRMaj output)	Applies the ≥ operator to each element in A.

Modifier and Type	Method	Description
static BMatrixRMaj	elementLessThanOrEqual (DMatrixRMaj A, DMatrixRMaj B, BMatrixRMaj output)	Applies the A ≤ B operator to each element.
static <T extends DMatrixD1> T	elementLog(T A, T output)	Element-wise log operation $c_{ij} = \text{Math.log}(a_{ij})$
static double	elementMax(DMatrixD1 a)	Returns the value of the element in the matrix that has the largest value. Max{ a_{ij} } for all i and j
static double	elementMax(DMatrixD1 a, ElementLocation loc)	Returns the value of the element in the matrix that has the largest value. Max{ a_{ij} } for all i and j
static double	elementMaxAbs(DMatrixD1 a)	Returns the absolute value of the element in the matrix that has the largest absolute value. Max{ $\|a_{ij}\|$ } for all i and j
static double	elementMaxAbs(DMatrixD1 a, ElementLocation loc)	Returns the absolute value of the element in the matrix that has the largest absolute value. Max{ $\|a_{ij}\|$ } for all i and j
static double	elementMin(DMatrixD1 a)	Returns the value of the element in the matrix that has the minimum value. Min{ a_{ij} } for all i and j
static double	elementMin(DMatrixD1 a, ElementLocation loc)	Returns the value of the element in the matrix that has the minimum value. Min{ a_{ij} } for all i and j
static double	elementMinAbs(DMatrixD1 a)	Returns the absolute value of the element in the matrix that has the smallest absolute value. Min{ $\|a_{ij}\|$ } for all i and j

Modifier and Type	Method	Description
static double	elementMinAbs(DMatrixD1 a, ElementLocation loc)	Returns the absolute value of the element in the matrix that has the smallest absolute value. Min{ $\|a_{ij}\|$ } for all i and j
static BMatrixRMaj	elementMoreThan(DMatrixRMaj A, double value, BMatrixRMaj output)	Applies the > operator to each element in A.
static BMatrixRMaj	elementMoreThanOrEqual (DMatrixRMaj A, double value, BMatrixRMaj output)	Applies the ≥ operator to each element in A.
static void	elementMult(DMatrixD1 A, DMatrixD1 B)	Performs the an element by element multiplication operation: $a_{ij} = a_{ij} * b_{ij}$
static <T extends DMatrixD1> T	elementMult(T A, T B, T output)	Performs the an element by element multiplication operation: $c_{ij} = a_{ij} * b_{ij}$
static <T extends DMatrixD1> T	elementPower(double a, T B, T output)	Element-wise power operation $c_{ij} = a \wedge b_{ij}$
static <T extends DMatrixD1> T	elementPower(T A, double b, T output)	Element-wise power operation $c_{ij} = a_{ij} \wedge b$
static <T extends DMatrixD1> T	elementPower(T A, T B, T output)	Element-wise power operation $c_{ij} = a_{ij} \wedge b_{ij}$
static DMatrixRMaj	elements(DMatrixRMaj A, BMatrixRMaj marked, @Nullable DMatrixRMaj output)	Returns a row matrix which contains all the elements in A which are flagged as true in 'marked'
static double	elementSum(DMatrixD1 mat)	Computes the sum of all the elements in the matrix: sum(i=1:m , j=1:n ; a_{ij})
static double	elementSumAbs(DMatrixD1 mat)	Computes the sum of the absolute value all the elements in the matrix: sum(i=1:m , j=1:n ; $\|a_{ij}\|$)

Modifier and Type	Method	Description
static void	extract(DMatrix src, int srcY0, int srcY1, int srcX0, int srcX1, DMatrix dst)	Extract where the destination is reshaped to match the extracted region
static void	extract(DMatrix src, int srcY0, int srcY1, int srcX0, int srcX1, DMatrix dst, int dstY0, int dstX0)	Extracts a submatrix from 'src' and inserts it in a submatrix in 'dst'.
static void	extract(DMatrix src, int srcY0, int srcX0, DMatrix dst)	Extracts a submatrix from 'src' and inserts it in a submatrix in 'dst'.
static DMatrixRMaj	extract(DMatrixRMaj src, int[] rows, int rowsSize, int[] cols, int colsSize, @Nullable DMatrixRMaj dst)	Extracts out a matrix from source given a sub matrix with arbitrary rows and columns specified in two array lists
static DMatrixRMaj	extract(DMatrixRMaj src, int[] indexes, int length, @Nullable DMatrixRMaj dst)	Extracts the elements from the source matrix by their 1D index.
static DMatrixRMaj	extract(DMatrixRMaj src, int srcY0, int srcY1, int srcX0, int srcX1)	Creates a new matrix which is the specified submatrix of 'src'
static DMatrixRMaj	extractColumn(DMatrixRMaj a, int column, @Nullable DMatrixRMaj out)	Extracts the column from a matrix.
static DMatrixRMaj	extractDiag(DMatrixRMaj src, @Nullable DMatrixRMaj dst)	Extracts the diagonal elements 'src' write it to the 'dst' vector.
static DMatrixRMaj	extractRow(DMatrixRMaj a, int row, @Nullable DMatrixRMaj out)	Extracts the row from a matrix.
static void	fill(DMatrixD1 a, double value)	Sets every element in the matrix to the specified value. a_{ij} = value
static DMatrixRMaj	identity(int width)	Creates an identity matrix of the specified size. a_{ij} = 0 if i ≠ j a_{ij} = 1 if i = j
static DMatrixRMaj	identity(int numRows, int numCols)	Creates a rectangular matrix which is zero except along the diagonals.
static void	insert(DMatrixRMaj src, DMatrixRMaj dst, int[] rows, int rowsSize, int[] cols, int colsSize)	Inserts into the specified elements of dst the source matrix.

Modifier and Type	Method	Description
static void	insert(DMatrix src, DMatrix dest, int destY0, int destX0)	Inserts matrix 'src' into matrix 'dest' with the (0,0) of src at (row,col) in dest.
static boolean	invert(DMatrixRMaj mat)	Performs a matrix inversion operation on the specified matrix and stores the results in the same matrix. $a = a^{-1}$
static boolean	invert(DMatrixRMaj mat, DMatrixRMaj result)	Performs a matrix inversion operation that does not modify the original and stores the results in another matrix.
static boolean	invertSPD(DMatrixRMaj mat, DMatrixRMaj result)	Matrix inverse for symmetric positive definite matrices.
static DMatrixRMaj	kron(DMatrixRMaj A, DMatrixRMaj B, @Nullable DMatrixRMaj C)	The Kronecker product of two matrices is defined as: $C_{ij} = a_{ij}B$ where C_{ij} is a sub matrix inside of $C \in \Re^{m*k \times n*l}$, $A \in \Re^{m \times n}$, and $B \in \Re^{k \times l}$.
static DMatrixRMaj	maxCols(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Finds the element with the minimum value along column in the input matrix and returns the results in a vector: $b_j = \min(i=1:m ; a_{ij})$
static DMatrixRMaj	maxRows(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Finds the element with the maximum value along each row in the input matrix and returns the results in a vector: $b_j = \max(i=1:n ; a_{ji})$
static DMatrixRMaj	minCols(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Finds the element with the minimum value along column in the input matrix and returns the results in a vector: $b_j = \min(i=1:m ; a_{ij})$
static DMatrixRMaj	minRows(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Finds the element with the minimum value along each row in the input matrix and returns the results in a vector: $b_j = \min(i=1:n ; a_{ji})$

Modifier and Type	Method	Description
static <T extends DMatrix1Row> T	mult(double alpha, T a, T b, T output)	Performs the following operation: c = α * a * b $c_{ij} = \alpha \sum_{k=1:n} \{ * a_{ik} * b_{kj}\}$
static <T extends DMatrix1Row> T	mult(T a, T b, T output)	Performs the following operation: c = a * b $c_{ij} = \sum_{k=1:n} \{ a_{ik} * b_{kj}\}$
static void	multAdd(double alpha, DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: c = c + α * a * b $c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ik} * b_{kj}\}$
static void	multAdd(DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: c = c + a * b $c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ik} * b_{kj}\}$
static void	multAddTransA(double alpha, DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + \alpha * a^T * b$ $c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ki} * b_{kj}\}$
static void	multAddTransA(DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + a^T * b$ $c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ki} * b_{kj}\}$
static void	multAddTransAB(double alpha, DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + \alpha * a^T * b^T$ $c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ki} * b_{jk}\}$
static void	multAddTransAB(DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + a^T * b^T$ $c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ki} * b_{jk}\}$
static void	multAddTransB(double alpha, DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + \alpha * a * b^T$ $c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ik} * b_{jk}\}$

Modifier and Type	Method	Description
static void	multAddTransB(DMatrix1Row a, DMatrix1Row b, DMatrix1Row c)	Performs the following operation: $c = c + a * b^T$ $c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$
static void	multCols(DMatrixRMaj A, double[] values)	Multiplies every element in column i by value[i].
static <T extends DMatrix1Row> T	multInner(T a, T output)	Computes the matrix multiplication inner product: $c = a^T * a$ $c_{ij} = \sum_{k=1:n} \{ a_{ki} * a_{kj} \}$
static <T extends DMatrix1Row> T	multOuter(T a, T output)	Computes the matrix multiplication outer product: $c = a * a^T$ $c_{ij} = \sum_{k=1:m} \{ a_{ik} * a_{jk} \}$
static void	multRows(double[] values, DMatrixRMaj A)	Multiplies every element in row i by value[i].
static <T extends DMatrix1Row> T	multTransA(double alpha, T a, T b, T output)	Performs the following operation: $c = \alpha * a^T * b$ $c_{ij} = \alpha \sum_{k=1:n} \{ a_{ki} * b_{kj} \}$
static <T extends DMatrix1Row> T	multTransA(T a, T b, T output)	Performs the following operation: $c = a^T * b$ $c_{ij} = \sum_{k=1:n} \{ a_{ki} * b_{kj} \}$
static <T extends DMatrix1Row> T	multTransAB(double alpha, T a, T b, T output)	Performs the following operation: $c = \alpha * a^T * b^T$ $c_{ij} = \alpha \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$
static <T extends DMatrix1Row> T	multTransAB(T a, T b, T output)	Performs the following operation: $c = a^T * b^T$ $c_{ij} = \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$
static <T extends DMatrix1Row> T	multTransB(double alpha, T a, T b, T output)	Performs the following operation: $c = \alpha * a * b^T$ $c_{ij} = \alpha \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$

Modifier and Type	Method	Description
static <T extends DMatrix1Row> T	multTransB(T a, T b, T output)	Performs the following operation: $c = a * b^T$ $c_{ij} = \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$
static DMatrixRMaj	permuteRowInv(int[] pinv, DMatrixRMaj input, DMatrixRMaj output)	Applies the row permutation specified by the vector to the input matrix and save the results in the output matrix.
static void	pinv(DMatrixRMaj A, DMatrixRMaj invA)	Computes the Moore-Penrose pseudo-inverse: $pinv(A) = (A^TA)^{-1} A^T$ or $pinv(A) = A^T(AA^T)^{-1}$
static void	removeColumns(DMatrixRMaj A, int col0, int col1)	Removes columns from the matrix.
static DMatrixRMaj[]	rowsToVector(DMatrixRMaj A, @Nullable DMatrixRMaj[] v)	Converts the rows in a matrix into a set of vectors.
static DMatrixRMaj	rref(DMatrixRMaj A, int numUnknowns, @Nullable DMatrixRMaj reduced)	Puts the augmented system matrix into reduced row echelon form (RREF) using Gauss-Jordan elimination with row (partial) pivots.
static void	scale(double alpha, DMatrixD1 a)	Performs an in-place element by element scalar multiplication. $a_{ij} = \alpha * a_{ij}$
static void	scale(double alpha, DMatrixD1 a, DMatrixD1 b)	Performs an element by element scalar multiplication. $b_{ij} = \alpha * a_{ij}$
static void	scaleCol(double alpha, DMatrixRMaj A, int col)	In-place scaling of a column in A
static void	scaleRow(double alpha, DMatrixRMaj A, int row)	In-place scaling of a row in A
static void	setIdentity(DMatrix1Row mat)	Sets all the diagonal elements equal to one and everything else equal to zero.
static boolean	solve(DMatrixRMaj a, DMatrixRMaj b, DMatrixRMaj x)	Solves for x in the following equation: $A*x = b$

Modifier and Type	Method	Description
static boolean	solveSPD(DMatrixRMaj A, DMatrixRMaj b, DMatrixRMaj x)	Linear solver for systems which are symmetric positive definite. A*x = b
static <T extends DMatrixD1> T	subtract(double val, T a, T output)	Performs matrix scalar subtraction: c = val - a c_{ij} = val - a_{ij}
static <T extends DMatrixD1> T	subtract(T a, double val, T output)	Performs matrix scalar subtraction: c = a - val c_{ij} = a_{ij} - val
static <T extends DMatrixD1> T	subtract(T a, T b, T output)	Performs the following subtraction operation: c = a - b c_{ij} = a_{ij} - b_{ij}
static void	subtractEquals(DMatrixD1 a, DMatrixD1 b)	Performs the following subtraction operation: a = a - b a_{ij} = a_{ij} - b_{ij}
static DMatrixRMaj	sumCols(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Computes the sum of each column in the input matrix and returns the results in a vector: b_j = sum(i=1:m ; a_{ij})
static DMatrixRMaj	sumRows(DMatrixRMaj input, @Nullable DMatrixRMaj output)	Computes the sum of each row in the input matrix and returns the results in a vector: b_j = sum(i=1:n ; a_{ji})
static void	symmLowerToFull(DMatrixRMaj A)	Given a symmetric matrix which is represented by a lower triangular matrix convert it back into a full symmetric matrix.
static void	symmUpperToFull(DMatrixRMaj A)	Given a symmetric matrix which is represented by a lower triangular matrix convert it back into a full symmetric matrix.

Modifier and Type	Method	Description
static double	trace(DMatrix1Row a)	This computes the trace of the matrix: trace = $\sum_{i=1:n}$ { a_{ii} } where n = min(numRows,numCols)
static void	transpose(DMatrixRMaj mat)	Performs an "in-place" transpose.
static DMatrixRMaj	transpose(DMatrixRMaj A, @Nullable DMatrixRMaj A_tran)	Transposes matrix 'a' and stores the results in 'b': $b_{ij} = a_{ji}$ where 'b' is the transpose of 'a'.

Methods inherited from class java.lang.Object

clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait

Constructor Details

CommonOps_DDRM

```
public CommonOps_DDRM()
```

Method Details

mult

```
public static <T extends DMatrix1Row> T mult(T a,
                                             T b,
                                             @Nullable
                                             T output)
```

Performs the following operation:

c = a * b

$c_{ij} = \sum_{k=1:n}$ { a_{ik} * b_{kj}}

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

mult

```
public static <T extends DMatrix1Row> T mult(double alpha,
                                             T a,
                                             T b,
                                             @Nullable
                                             T output)
```

Performs the following operation:

c = α * a * b

$c_{ij} = \alpha \sum_{k=1:n} \{ * a_{ik} * b_{kj}\}$

Parameters:

`alpha` - Scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransA

```
public static <T extends DMatrix1Row> T multTransA(T a,
                                                   T b,
                                                   @Nullable
                                                   T output)
```

Performs the following operation:

$c = a^T * b$

$c_{ij} = \sum_{k=1:n} \{ a_{ki} * b_{kj}\}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransA

```
public static <T extends DMatrix1Row> T multTransA(double alpha,
                                                   T a,
                                                   T b,
                                                   @Nullable
                                                   T output)
```

Performs the following operation:

$c = \alpha * a^T * b$

$c_{ij} = \alpha \sum_{k=1:n} \{ a_{ki} * b_{kj} \}$

Parameters:

`alpha` - Scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransB

```
public static <T extends DMatrix1Row> T multTransB(T a,
                                                   T b,
                                                   @Nullable
                                                   T output)
```

Performs the following operation:

$c = a * b^T$
$c_{ij} = \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransB

```
public static <T extends DMatrix1Row> T multTransB(double alpha,
                                                   T a,
                                                   T b,
                                                   @Nullable
                                                   T output)
```

Performs the following operation:

$c = \alpha * a * b^T$
$c_{ij} = \alpha \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$

Parameters:

`alpha` - Scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransAB

```
public static <T extends DMatrix1Row> T multTransAB(T a,
                                                    T b,
                                                    @Nullable
                                                    T output)
```

Performs the following operation:

$c = a^T * b^T$
$c_{ij} = \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

multTransAB

```
public static <T extends DMatrix1Row> T multTransAB(double alpha,
                                                    T a,
                                                    T b,
                                                    @Nullable
                                                    T output)
```

Performs the following operation:

$c = \alpha * a^T * b^T$
$c_{ij} = \alpha \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$

Parameters:

`alpha` - Scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

dot

```
public static double dot(DMatrixD1 a,
                         DMatrixD1 b)
```

Computes the dot product or inner product between two vectors. If the two vectors are columns vectors then it is defined as:
`dot(a,b) = a<sup>T</sup> * b`
If the vectors are column or row or both is ignored by this function.

Parameters:

`a` - Vector

b - Vector

Returns:

Dot product of the two vectors

multInner

```
public static <T extends DMatrix1Row> T multInner(T a,
                                                  @Nullable
                                                  T output)
```

Computes the matrix multiplication inner product:

$c = a^T * a$

$c_{ij} = \sum_{k=1:n} \{ a_{ki} * a_{kj} \}$

Is faster than using a generic matrix multiplication by taking advantage of symmetry. For vectors there is an even faster option, see `VectorVectorMult_DDRM.innerProd(DMatrixD1, DMatrixD1)`

Parameters:

a - The matrix being multiplied. Not modified.

output - Where the results of the operation are stored. Modified.

multOuter

```
public static <T extends DMatrix1Row> T multOuter(T a,
                                                  @Nullable
                                                  T output)
```

Computes the matrix multiplication outer product:

$c = a * a^T$

$c_{ij} = \sum_{k=1:m} \{ a_{ik} * a_{jk} \}$

Is faster than using a generic matrix multiplication by taking advantage of symmetry.

Parameters:

a - The matrix being multiplied. Not modified.

output - Where the results of the operation are stored. Modified.

multAdd

```
public static void multAdd(DMatrix1Row a,
                           DMatrix1Row b,
                           DMatrix1Row c)
```

Performs the following operation:

$c = c + a * b$
$c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ik} * b_{kj} \}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`c` - Where the results of the operation are stored. Modified.

multAdd

```
public static void multAdd(double alpha,
                           DMatrix1Row a,
                           DMatrix1Row b,
                           DMatrix1Row c)
```

Performs the following operation:

$c = c + \alpha * a * b$
$c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ik} * b_{kj} \}$

Parameters:

`alpha` - scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`c` - Where the results of the operation are stored. Modified.

multAddTransA

```
public static void multAddTransA(DMatrix1Row a,
                                 DMatrix1Row b,
                                 DMatrix1Row c)
```

Performs the following operation:

$c = c + a^T * b$
$c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ki} * b_{kj} \}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`c` - Where the results of the operation are stored. Modified.

multAddTransA

```
public static void multAddTransA(double alpha,
                                 DMatrix1Row a,
                                 DMatrix1Row b,
                                 DMatrix1Row c)
```

Performs the following operation:

$c = c + \alpha * a^T * b$
$c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ki} * b_{kj} \}$

Parameters:

`alpha` - scaling factor

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`c` - Where the results of the operation are stored. Modified.

multAddTransB

```
public static void multAddTransB(DMatrix1Row a,
                                 DMatrix1Row b,
                                 DMatrix1Row c)
```

Performs the following operation:

$c = c + a * b^T$
$c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$

Parameters:

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

`c` - Where the results of the operation are stored. Modified.

multAddTransB

```
public static void multAddTransB(double alpha,
                                 DMatrix1Row a,
                                 DMatrix1Row b,
                                 DMatrix1Row c)
```

Performs the following operation:

$c = c + \alpha * a * b^T$
$c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ik} * b_{jk} \}$

Parameters:

`alpha` - Scaling factor.

`a` - The left matrix in the multiplication operation. Not modified.

`b` - The right matrix in the multiplication operation. Not modified.

c - Where the results of the operation are stored. Modified.

multAddTransAB

```
public static void multAddTransAB(DMatrix1Row a,
                                   DMatrix1Row b,
                                   DMatrix1Row c)
```

Performs the following operation:

$c = c + a^T * b^T$
$c_{ij} = c_{ij} + \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$

Parameters:

a - The left matrix in the multiplication operation. Not Modified.

b - The right matrix in the multiplication operation. Not Modified.

c - Where the results of the operation are stored. Modified.

multAddTransAB

```
public static void multAddTransAB(double alpha,
                                   DMatrix1Row a,
                                   DMatrix1Row b,
                                   DMatrix1Row c)
```

Performs the following operation:

$c = c + \alpha * a^T * b^T$
$c_{ij} = c_{ij} + \alpha * \sum_{k=1:n} \{ a_{ki} * b_{jk} \}$

Parameters:

alpha - Scaling factor.

a - The left matrix in the multiplication operation. Not Modified.

b - The right matrix in the multiplication operation. Not Modified.

c - Where the results of the operation are stored. Modified.

solve

```
public static boolean solve(DMatrixRMaj a,
                            DMatrixRMaj b,
                            DMatrixRMaj x)
```

Solves for x in the following equation:

A*x = b

If the system could not be solved then false is returned. If it returns true that just means the algorithm finished operating, but the results could still be bad because 'A' is singular or nearly singular.

If repeat calls to solve are being made then one should consider using `LinearSolverFactory_DDRM` instead.

It is ok for 'b' and 'x' to be the same matrix.

Parameters:

`a` - A matrix that is m by n. Not modified.

`b` - A matrix that is n by k. Not modified.

`x` - A matrix that is m by k. Modified.

Returns:

true if it could invert the matrix false if it could not.

solveSPD

```
public static boolean solveSPD(DMatrixRMaj A,
                               DMatrixRMaj b,
                               DMatrixRMaj x)
```

Linear solver for systems which are symmetric positive definite.
A*x = b

Parameters:

`A` - A matrix that is n by n and SPD. Not modified.

`b` - A matrix that is n by k. Not modified.

`x` - A matrix that is n by k. Modified.

Returns:

true if it could invert the matrix false if it could not.

See Also:

`UnrolledCholesky_DDRM`, `LinearSolverFactory_DDRM`

transpose

```
public static void transpose(DMatrixRMaj mat)
```

Performs an "in-place" transpose.

For square matrices the transpose is truly in-place and does not require additional memory. For non-square matrices, internally a temporary matrix is declared and `transpose(DMatrixRMaj, DMatrixRMaj)` is invoked.

Parameters:

`mat` - The matrix that is to be transposed. Modified.

transpose

```
public static DMatrixRMaj transpose(DMatrixRMaj A,
                                    @Nullable
                                    @Nullable DMatrixRMaj A_tran)
```

Transposes matrix 'a' and stores the results in 'b':

$b_{ij} = a_{ji}$
where 'b' is the transpose of 'a'.

Parameters:

`A` - The original matrix. Not modified.

`A_tran` - Where the transpose is stored. If null a new matrix is created. Modified.

Returns:

The transposed matrix.

trace

```
public static double trace(DMatrix1Row a)
```

This computes the trace of the matrix:

$\text{trace} = \sum_{i=1:n} \{ a_{ii} \}$
where n = min(numRows,numCols)

Parameters:

`a` - A square matrix. Not modified.

det

```
public static double det(DMatrixRMaj mat)
```

Returns the determinant of the matrix. If the inverse of the matrix is also needed, then using `LUDecomposition_F64` directly (or any similar algorithm) can be more efficient.

Parameters:

`mat` - The matrix whose determinant is to be computed. Not modified.

Returns:

The determinant.

invert

```
public static boolean invert(DMatrixRMaj mat)
```

Performs a matrix inversion operation on the specified matrix and stores the results in the same matrix.

$a = a^{-1}$

If the algorithm could not invert the matrix then false is returned. If it returns true that just means the algorithm finished. The results could still be bad because the matrix is singular or nearly singular.

Parameters:

`mat` - The matrix that is to be inverted. Results are stored here. Modified.

Returns:

true if it could invert the matrix false if it could not.

invert

```
public static boolean invert(DMatrixRMaj mat,
                             DMatrixRMaj result)
```

Performs a matrix inversion operation that does not modify the original and stores the results in another matrix. The two matrices must have the same dimension.

$b = a^{-1}$

If the algorithm could not invert the matrix then false is returned. If it returns true that just means the algorithm finished. The results could still be bad because the matrix is singular or nearly singular.

For medium to large matrices there might be a slight performance boost to using `LinearSolverFactory_DDRM` instead.

Parameters:

`mat` - The matrix that is to be inverted. Not modified.

`result` - Where the inverse matrix is stored. Modified.

Returns:

true if it could invert the matrix false if it could not.

invertSPD

```
public static boolean invertSPD(DMatrixRMaj mat,
                                DMatrixRMaj result)
```

Matrix inverse for symmetric positive definite matrices. For small matrices an unrolled cholesky is used. Otherwise a standard decomposition.

Parameters:

`mat` - (Input) SPD matrix

`result` - (Output) Inverted matrix.

Returns:

true if it could invert the matrix false if it could not.

See Also:

`UnrolledCholesky_DDRM`, `LinearSolverFactory_DDRM.chol(int)`

pinv

```
public static void pinv(DMatrixRMaj A,
                        DMatrixRMaj invA)
```

Computes the Moore-Penrose pseudo-inverse:

pinv(A) = $(A^T A)^{-1} A^T$
or
pinv(A) = $A^T (A A^T)^{-1}$

Internally it uses `SolvePseudoInverseSvd_DDRM` to compute the inverse. For performance reasons, this should only be used when a matrix is singular or nearly singular.

Parameters:

`A` - A m by n Matrix. Not modified.

`invA` - Where the computed pseudo inverse is stored. n by m. Modified.

columnsToVector

```
public static DMatrixRMaj[] columnsToVector(DMatrixRMaj A,
                                            @Nullable
                                            @Nullable DMatrixRMaj[] v)
```

Converts the columns in a matrix into a set of vectors.

Parameters:

`A` - Matrix. Not modified.

Returns:

An array of vectors.

rowsToVector

```
public static DMatrixRMaj[] rowsToVector(DMatrixRMaj A,
                                         @Nullable
                                         @Nullable DMatrixRMaj[] v)
```

Converts the rows in a matrix into a set of vectors.

Parameters:

`A` - Matrix. Not modified.

Returns:

An array of vectors.

setIdentity

```
public static void setIdentity(DMatrix1Row mat)
```

Sets all the diagonal elements equal to one and everything else equal to zero. If this is a square matrix then it will be an identity matrix.

Parameters:

`mat` - A square matrix.

See Also:

`identity(int)`

identity

```
public static DMatrixRMaj identity(int width)
```

Creates an identity matrix of the specified size.

$a_{ij} = 0$ if $i \neq j$
$a_{ij} = 1$ if $i = j$

Parameters:

`width` - The width and height of the identity matrix.

Returns:

A new instance of an identity matrix.

identity

```
public static DMatrixRMaj identity(int numRows,
                                   int numCols)
```

Creates a rectangular matrix which is zero except along the diagonals.

Parameters:

`numRows` - Number of rows in the matrix.

`numCols` - NUmber of columns in the matrix.

Returns:

A matrix with diagonal elements equal to one.

diag

```
public static DMatrixRMaj diag(double... diagEl)
```

Creates a new square matrix whose diagonal elements are specified by diagEl and all the other elements are zero.

$a_{ij} = 0$ if $i \leq j$
$a_{ij} = \text{diag}[i]$ if $i = j$

Parameters:

`diagEl` - Contains the values of the diagonal elements of the resulting matrix.

Returns:

A new matrix.

See Also:

`diagR(int, int, double...)`

diag

```
public static DMatrixRMaj diag(@Nullable
                               @Nullable DMatrixRMaj ret,
                               int width,
                               double... diagEl)
```

See Also:

`diag(double...)`

diagR

```
public static DMatrixRMaj diagR(int numRows,
                                int numCols,
                                double... diagEl)
```

Creates a new rectangular matrix whose diagonal elements are specified by diagEl and all the other elements are zero.

$a_{ij} = 0$ if $i \leq j$
$a_{ij} = \text{diag}[i]$ if $i = j$

Parameters:

`numRows` - Number of rows in the matrix.

`numCols` - Number of columns in the matrix.

`diagEl` - Contains the values of the diagonal elements of the resulting matrix.

Returns:

A new matrix.

See Also:

`diag(double...)`

kron

```
public static DMatrixRMaj kron(DMatrixRMaj A,
                               DMatrixRMaj B,
                               @Nullable
                               @Nullable DMatrixRMaj C)
```

The Kronecker product of two matrices is defined as:
$C_{ij} = a_{ij}B$
where C_{ij} is a sub matrix inside of $C \in \Re^{m*k \times n*l}$, $A \in \Re^{m \times n}$, and $B \in \Re^{k \times l}$.

Parameters:

`A` - The left matrix in the operation. Not modified.

`B` - The right matrix in the operation. Not modified.

`C` - Where the results of the operation are stored. Nullable. Modified.

extract

```
public static void extract(DMatrix src,
                           int srcY0,
                           int srcY1,
                           int srcX0,
                           int srcX1,
                           DMatrix dst,
                           int dstY0,
                           int dstX0)
```

Extracts a submatrix from 'src' and inserts it in a submatrix in 'dst'.

$s_{i-y0\ ,\ j-x0} = o_{ij}$ for all $y0 \le i < y1$ and $x0 \le j < x1$

where 's_{ij}' is an element in the submatrix and 'o_{ij}' is an element in the original matrix.

Parameters:

`src` - The original matrix which is to be copied. Not modified.

`srcX0` - Start column.

`srcX1` - Stop column+1.

`srcY0` - Start row.

`srcY1` - Stop row+1.

`dst` - Where the submatrix are stored. Modified.

`dstY0` - Start row in dst.

`dstX0` - start column in dst.

extract

```
public static void extract(DMatrix src,
                           int srcY0,
                           int srcY1,
                           int srcX0,
                           int srcX1,
                           DMatrix dst)
```

Extract where the destination is reshaped to match the extracted region

Parameters:

`src` - The original matrix which is to be copied. Not modified.

`srcX0` - Start column.

`srcX1` - Stop column+1.

`srcY0` - Start row.

`srcY1` - Stop row+1.

`dst` - Where the submatrix are stored. Modified.

extract

```
public static void extract(DMatrix src,
                           int srcY0,
                           int srcX0,
                           DMatrix dst)
```

Extracts a submatrix from 'src' and inserts it in a submatrix in 'dst'. Uses the shape of dst to determine the size of the matrix extracted.

Parameters:

`src` - The original matrix which is to be copied. Not modified.

`srcY0` - Start row in src.

`srcX0` - Start column in src.

`dst` - Where the matrix is extracted into.

extract

```
public static DMatrixRMaj extract(DMatrixRMaj src,
                                  int srcY0,
                                  int srcY1,
                                  int srcX0,
                                  int srcX1)
```

Creates a new matrix which is the specified submatrix of 'src'

$s_{i-yo\ ,\ j-xo} = o_{ij}$ for all $yo \leq i < y1$ and $xo \leq j < x1$

where 's_{ij}' is an element in the submatrix and 'o_{ij}' is an element in the original matrix.

Parameters:

`src` - The original matrix which is to be copied. Not modified.

`srcX0` - Start column.

`srcX1` - Stop column+1.

`srcY0` - Start row.

`srcY1` - Stop row+1.

Returns:

Extracted submatrix.

extract

```
public static DMatrixRMaj extract(DMatrixRMaj src,
                                  int[] rows,
                                  int rowsSize,
                                  int[] cols,
                                  int colsSize,
                                  @Nullable
                                  @Nullable DMatrixRMaj dst)
```

Extracts out a matrix from source given a sub matrix with arbitrary rows and columns specified in two array lists

Parameters:

`src` - Source matrix. Not modified.

`rows` - array of row indexes

`rowsSize` - maximum element in row array

`cols` - array of column indexes

`colsSize` - maximum element in column array

`dst` - output matrix. Must be correct shape.

extract

```
public static DMatrixRMaj extract(DMatrixRMaj src,
                                  int[] indexes,
                                  int length,
                                  @Nullable
                                  @Nullable DMatrixRMaj dst)
```

Extracts the elements from the source matrix by their 1D index.

Parameters:

`src` - Source matrix. Not modified.

`indexes` - array of row indexes

`length` - maximum element in row array

`dst` - output matrix. Must be a vector of the correct length.

insert

```
public static void insert(DMatrixRMaj src,
                          DMatrixRMaj dst,
                          int[] rows,
                          int rowsSize,
                          int[] cols,
                          int colsSize)
```

Inserts into the specified elements of dst the source matrix.

```
for i in len(rows):
  for j in len(cols):
     dst(rows[i],cols[j]) = src(i,j)
```

Parameters:

`src` - Source matrix. Not modified.

`dst` - output matrix. Must be correct shape.

`rows` - array of row indexes.

`rowsSize` - maximum element in row array

`cols` - array of column indexes

`colsSize` - maximum element in column array

extractDiag

```
public static DMatrixRMaj extractDiag(DMatrixRMaj src,
                                       @Nullable
                                       @Nullable DMatrixRMaj dst)
```

Extracts the diagonal elements 'src' write it to the 'dst' vector. 'dst' can either be a row or column vector.

Parameters:

`src` - Matrix whose diagonal elements are being extracted. Not modified.

`dst` - A vector the results will be written into. Modified.

extractRow

```
public static DMatrixRMaj extractRow(DMatrixRMaj a,
                                     int row,
                                     @Nullable
                                     @Nullable DMatrixRMaj out)
```

Extracts the row from a matrix.

Parameters:

`a` - Input matrix

`row` - Which row is to be extracted

`out` - output. Storage for the extracted row. If null then a new vector will be returned.

Returns:

The extracted row.

extractColumn

```
public static DMatrixRMaj extractColumn(DMatrixRMaj a,
                                        int column,
                                        @Nullable
                                        @Nullable DMatrixRMaj out)
```

Extracts the column from a matrix.

Parameters:

`a` - Input matrix

`column` - Which column is to be extracted

`out` - output. Storage for the extracted column. If null then a new vector will be returned.

Returns:

The extracted column.

removeColumns

```
public static void removeColumns(DMatrixRMaj A,
                                 int col0,
                                 int col1)
```

Removes columns from the matrix.

Parameters:

`A` - Matrix. Modified

`col0` - First column

`col1` - Last column, inclusive.

insert

```
public static void insert(DMatrix src,
                          DMatrix dest,
                          int destY0,
                          int destX0)
```

Inserts matrix 'src' into matrix 'dest' with the (0,0) of src at (row,col) in dest. This is equivalent to calling extract(src,0,src.numRows,0,src.numCols,dest,destY0,destX0).

Parameters:

`src` - matrix that is being copied into dest. Not modified.

`dest` - Where src is being copied into. Modified.

`destY0` - Start row for the copy into dest.

`destX0` - Start column for the copy into dest.

elementMax

```
public static double elementMax(DMatrixD1 a)
```

Returns the value of the element in the matrix that has the largest value.

Max{ a_{ij} } for all i and j

Parameters:

`a` - A matrix. Not modified.

Returns:

The max element value of the matrix.

elementMax

```
public static double elementMax(DMatrixD1 a,
                                ElementLocation loc)
```

Returns the value of the element in the matrix that has the largest value.

Max{ a_{ij} } for all i and j

Parameters:

`a` - A matrix. Not modified.

`loc` - (Output) Location of selected element.

Returns:

The max element value of the matrix.

elementMaxAbs

```
public static double elementMaxAbs(DMatrixD1 a)
```

Returns the absolute value of the element in the matrix that has the largest absolute value.

Max{ $|a_{ij}|$ } for all i and j

Parameters:

`a` - A matrix. Not modified.

Returns:

The max abs element value of the matrix.

elementMaxAbs

```
public static double elementMaxAbs(DMatrixD1 a,
                                   ElementLocation loc)
```

Returns the absolute value of the element in the matrix that has the largest absolute value.

Max{ $|a_{ij}|$ } for all i and j

Parameters:

`a` - A matrix. Not modified.

`loc` - (Output) Location of element element.

Returns:

The max abs element value of the matrix.

elementMin

```
public static double elementMin(DMatrixD1 a)
```

Returns the value of the element in the matrix that has the minimum value.

Min{ a_{ij} } for all i and j

Parameters:

`a` - A matrix. Not modified.

Returns:

The value of element in the matrix with the minimum value.

elementMin

```
public static double elementMin(DMatrixD1 a,
                                ElementLocation loc)
```

Returns the value of the element in the matrix that has the minimum value.

Min{ a_{ij} } for all i and j

Parameters:

`a` - A matrix. Not modified.

`loc` - (Output) Location of selected element.

Returns:

The value of element in the matrix with the minimum value.

elementMinAbs

```
public static double elementMinAbs(DMatrixD1 a)
```

Returns the absolute value of the element in the matrix that has the smallest absolute value.

Min{ $|a_{ij}|$ } for all i and j

Parameters:

`a` - A matrix. Not modified.

Returns:

The max element value of the matrix.

elementMinAbs

```
public static double elementMinAbs(DMatrixD1 a,
                                   ElementLocation loc)
```

Returns the absolute value of the element in the matrix that has the smallest absolute value.

Min{ $|a_{ij}|$ } for all i and j

Parameters:

`a` - (Input) A matrix. Not modified.

`loc` - (Output) Location of selected element.

Returns:

The max element value of the matrix.

elementMult

```
public static void elementMult(DMatrixD1 A,
                               DMatrixD1 B)
```

Performs the an element by element multiplication operation:

$a_{ij} = a_{ij} * b_{ij}$

Parameters:

`A` - The left matrix in the multiplication operation. Modified.

`B` - The right matrix in the multiplication operation. Not modified.

elementMult

```
public static <T extends DMatrixD1> T elementMult(T A,
                                                  T B,
                                                  @Nullable
                                                  T output)
```

Performs the an element by element multiplication operation:

$c_{ij} = a_{ij} * b_{ij}$

Parameters:

`A` - The left matrix in the multiplication operation. Not modified.

`B` - The right matrix in the multiplication operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

elementDiv

```
public static void elementDiv(DMatrixD1 A,
                              DMatrixD1 B)
```

Performs the an element by element division operation:

$a_{ij} = a_{ij} / b_{ij}$

Parameters:

`A` - The left matrix in the division operation. Modified.

`B` - The right matrix in the division operation. Not modified.

elementDiv

```
public static <T extends DMatrixD1> T elementDiv(T A,
                                                  T B,
                                                  @Nullable
                                                  T output)
```

Performs the an element by element division operation:

$c_{ij} = a_{ij} / b_{ij}$

Parameters:

`A` - The left matrix in the division operation. Not modified.

`B` - The right matrix in the division operation. Not modified.

`output` - Where the results of the operation are stored. Modified.

elementSum

```
public static double elementSum(DMatrixD1 mat)
```

Computes the sum of all the elements in the matrix:

$\text{sum}(i=1{:}m\ ,\ j=1{:}n\ ;\ a_{ij})$

Parameters:

`mat` - An m by n matrix. Not modified.

Returns:

The sum of the elements.

elementSumAbs

```
public static double elementSumAbs(DMatrixD1 mat)
```

Computes the sum of the absolute value all the elements in the matrix:

$\text{sum}(i=1{:}m\ ,\ j=1{:}n\ ;\ |a_{ij}|)$

Parameters:

`mat` - An m by n matrix. Not modified.

Returns:

The sum of the absolute value of each element.

elementPower

```
public static <T extends DMatrixD1> T elementPower(T A,
                                                   T B,
                                                   @Nullable
                                                   T output)
```

Element-wise power operation
$c_{ij} = a_{ij} \wedge b_{ij}$

Parameters:

`A` - left side

`B` - right side

`output` - output (modified)

elementPower

```
public static <T extends DMatrixD1> T elementPower(double a,
                                                   T B,
                                                   @Nullable
                                                   T output)
```

Element-wise power operation
$c_{ij} = a \wedge b_{ij}$

Parameters:

`a` - left scalar

`B` - right side

`output` - output (modified)

elementPower

```
public static <T extends DMatrixD1> T elementPower(T A,
                                                   double b,
                                                   @Nullable
                                                   T output)
```

Element-wise power operation
$c_{ij} = a_{ij} \wedge b$

Parameters:

`A` - left side

`b` - right scalar

`output` - output (modified)

elementLog

```
public static <T extends DMatrixD1> T elementLog(T A,
                                                 @Nullable
                                                 T output)
```

Element-wise log operation
$c_{ij} = \text{Math.log}(a_{ij})$

Parameters:

`A` - (input) A matrix

`output` - (input/output) Storage for results. can be null. (modified)

Returns:

The results

elementExp

```
public static <T extends DMatrixD1> T elementExp(T A,
                                                 @Nullable
                                                 T output)
```

Element-wise exp operation
$c_{ij} = \text{Math.exp}(a_{ij})$

Parameters:

`A` - (input) A matrix

`output` - (input/output) Storage for results. can be null. (modified)

Returns:

The results

multRows

```
public static void multRows(double[] values,
                            DMatrixRMaj A)
```

Multiplies every element in row i by value[i].

Parameters:

`values` - array. Not modified.

`A` - Matrix. Modified.

divideRows

```
public static void divideRows(double[] values,
                              DMatrixRMaj A)
```

Divides every element in row i by value[i].

Parameters:

`values` - array. Not modified.

`A` - Matrix. Modified.

multCols

```
public static void multCols(DMatrixRMaj A,
                            double[] values)
```

Multiplies every element in column i by value[i].

Parameters:

`A` - Matrix. Modified.

`values` - array. Not modified.

divideCols

```
public static void divideCols(DMatrixRMaj A,
                              double[] values)
```

Divides every element in column i by value[i].

Parameters:

`A` - Matrix. Modified.

`values` - array. Not modified.

divideRowsCols

```
public static void divideRowsCols(double[] diagA,
                                  int offsetA,
                                  DMatrixRMaj B,
                                  double[] diagC,
                                  int offsetC)
```

Equivalent to multiplying a matrix B by the inverse of two diagonal matrices. B = inv(A)*B*inv(C), where A=diag(a) and C=diag(c).

Parameters:

`diagA` - Array of length offsteA + B.numRows

`offsetA` - First index in A

`B` - Rectangular matrix

`diagC` - Array of length indexC + B.numCols

`offsetC` - First index in C

sumRows

```
public static DMatrixRMaj sumRows(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Computes the sum of each row in the input matrix and returns the results in a vector:

$b_j = sum(i=1:n ; a_{ji})$

Parameters:

`input` - INput matrix whose rows are summed.

`output` - Optional storage for output. Reshaped into a column. Modified.

Returns:

Vector containing the sum of each row in the input.

minRows

```
public static DMatrixRMaj minRows(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Finds the element with the minimum value along each row in the input matrix and returns the results in a vector:

$b_j = min(i=1:n ; a_{ji})$

Parameters:

`input` - Input matrix

`output` - Optional storage for output. Reshaped into a column. Modified.

Returns:

Vector containing the sum of each row in the input.

maxRows

```
public static DMatrixRMaj maxRows(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Finds the element with the maximum value along each row in the input matrix and returns the results in a vector:

$b_j = max(i=1:n ; a_{ji})$

Parameters:

`input` - Input matrix

`output` - Optional storage for output. Reshaped into a column. Modified.

Returns:

Vector containing the sum of each row in the input.

sumCols

```
public static DMatrixRMaj sumCols(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Computes the sum of each column in the input matrix and returns the results in a vector:

$b_j = sum(i=1:m ; a_{ij})$

Parameters:

`input` - Input matrix

`output` - Optional storage for output. Reshaped into a row vector. Modified.

Returns:

Vector containing the sum of each column

minCols

```
public static DMatrixRMaj minCols(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Finds the element with the minimum value along column in the input matrix and returns the results in a vector:

$b_j = min(i=1:m ; a_{ij})$

Parameters:

`input` - Input matrix

`output` - Optional storage for output. Reshaped into a row vector. Modified.

Returns:

Vector containing the minimum of each column

maxCols

```
public static DMatrixRMaj maxCols(DMatrixRMaj input,
                                  @Nullable
                                  @Nullable DMatrixRMaj output)
```

Finds the element with the minimum value along column in the input matrix and returns the results in a vector:

$b_j = \min(i=1:m ; a_{ij})$

Parameters:

`input` - Input matrix

`output` - Optional storage for output. Reshaped into a row vector. Modified.

Returns:

Vector containing the maximum of each column

addEquals

```
public static void addEquals(DMatrixD1 a,
                             DMatrixD1 b)
```

Performs the following operation:

$a = a + b$
$a_{ij} = a_{ij} + b_{ij}$

Parameters:

`a` - (input/output) A Matrix. Modified.

`b` - (input) A Matrix. Not modified.

addEquals

```
public static void addEquals(DMatrixD1 a,
                             double beta,
                             DMatrixD1 b)
```

Performs the following operation:

$a = a + \beta * b$
$a_{ij} = a_{ij} + \beta * b_{ij}$

Parameters:

`beta` - The number that matrix 'b' is multiplied by.

`a` - (input/output) A Matrix. Modified.

`b` - (input) A Matrix. Not modified.

add

```
public static <T extends DMatrixD1> T add(T a,
                                          T b,
                                          @Nullable
                                          T output)
```

Performs the following operation:

$c = a + b$
$c_{ij} = a_{ij} + b_{ij}$

Matrix C can be the same instance as Matrix A and/or B.

Parameters:

`a` - A Matrix. Not modified.

`b` - A Matrix. Not modified.

`output` - (output) A Matrix where the results are stored. Can be null. Modified.

Returns:

The results.

add

```
public static <T extends DMatrixD1> T add(T a,
                                          double beta,
                                          T b,
                                          @Nullable
                                          T output)
```

Performs the following operation:

$c = a + \beta * b$
$c_{ij} = a_{ij} + \beta * b_{ij}$

Matrix C can be the same instance as Matrix A and/or B.

Parameters:

`a` - A Matrix. Not modified.

`beta` - Scaling factor for matrix b.

`b` - A Matrix. Not modified.

`output` - (output) A Matrix where the results are stored. Can be null. Modified.

Returns:

The results.

add

```
public static <T extends DMatrixD1> T add(double alpha,
                                          T a,
                                          double beta,
                                          T b,
                                          @Nullable
                                          T output)
```

Performs the following operation:

$c = \alpha * a + \beta * b$
$c_{ij} = \alpha * a_{ij} + \beta * b_{ij}$

Matrix C can be the same instance as Matrix A and/or B.

Parameters:

`alpha` - A scaling factor for matrix a.

`a` - A Matrix. Not modified.

`beta` - A scaling factor for matrix b.

`b` - A Matrix. Not modified.

`output` - (output) A Matrix where the results are stored. Can be null. Modified.

Returns:

The results.

add

```
public static <T extends DMatrixD1> T add(double alpha,
                                          T a,
                                          T b,
                                          T output)
```

Performs the following operation:

$c = \alpha * a + b$
$c_{ij} = \alpha * a_{ij} + b_{ij}$

Matrix C can be the same instance as Matrix A and/or B.

Parameters:

`alpha` - A scaling factor for matrix a.

`a` - A Matrix. Not modified.

`b` - A Matrix. Not modified.

`output` - (output) A Matrix where the results are stored. Can be null. Modified.

Returns:

The results.

add

```
public static void add(DMatrixD1 a,
                       double val)
```

Performs an in-place scalar addition:

$a = a + val$
$a_{ij} = a_{ij} + val$

Parameters:

`a` - A matrix. Modified.

`val` - The value that's added to each element.

add

```
public static <T extends DMatrixD1> T add(T a,
                                          double val,
                                          T output)
```

Performs scalar addition:

c = a + val
$c_{ij} = a_{ij} + val$

Parameters:

`a` - A matrix. Not modified.

`val` - The value that's added to each element.

`output` - (output) Storage for results. Can be null. Modified.

Returns:

The resulting matrix

subtract

```
public static <T extends DMatrixD1> T subtract(T a,
                                               double val,
                                               @Nullable
                                               T output)
```

Performs matrix scalar subtraction:

c = a - val
$c_{ij} = a_{ij} - val$

Parameters:

`a` - (input) A matrix. Not modified.

`val` - (input) The value that's subtracted to each element.

`output` - (output) Storage for results. Can be null. Modified.

Returns:

The resulting matrix

subtract

```
public static <T extends DMatrixD1> T subtract(double val,
                                               T a,
                                               @Nullable
                                               T output)
```

Performs matrix scalar subtraction:

c = val - a
$c_{ij} = val - a_{ij}$

Parameters:

`val` - (input) The value that's subtracted to each element.

`a` - (input) A matrix. Not modified.

`output` - (output) Storage for results. Can be null. Modified.

Returns:

The resulting matrix

subtractEquals

```
public static void subtractEquals(DMatrixD1 a,
                                  DMatrixD1 b)
```

Performs the following subtraction operation:

a = a - b
$a_{ij} = a_{ij} - b_{ij}$

Parameters:

`a` - (input) A Matrix. Modified.

`b` - (input) A Matrix. Not modified.

subtract

```
public static <T extends DMatrixD1> T subtract(T a,
                                               T b,
                                               @Nullable
                                               T output)
```

Performs the following subtraction operation:

c = a - b
$c_{ij} = a_{ij} - b_{ij}$

Matrix C can be the same instance as Matrix A and/or B.

Parameters:

`a` - (input) A Matrix. Not modified.

`b` - (input) A Matrix. Not modified.

`output` - (output) A Matrix. Can be null. Modified.

Returns:

The resulting matrix

scale

```
public static void scale(double alpha,
                         DMatrixD1 a)
```

Performs an in-place element by element scalar multiplication.

$a_{ij} = \alpha * a_{ij}$

Parameters:

`a` - The matrix that is to be scaled. Modified.

`alpha` - the amount each element is multiplied by.

scale

```
public static void scale(double alpha,
                         DMatrixD1 a,
                         DMatrixD1 b)
```

Performs an element by element scalar multiplication.

$b_{ij} = \alpha * a_{ij}$

Parameters:

`alpha` - the amount each element is multiplied by.

`a` - The matrix that is to be scaled. Not modified.

`b` - Where the scaled matrix is stored. Modified.

scaleRow

```
public static void scaleRow(double alpha,
                            DMatrixRMaj A,
                            int row)
```

In-place scaling of a row in A

Parameters:

`alpha` - scale factor

`A` - matrix

`row` - which row in A

scaleCol

```
public static void scaleCol(double alpha,
                            DMatrixRMaj A,
                            int col)
```

In-place scaling of a column in A

Parameters:

`alpha` - scale factor

`A` - matrix

`col` - which row in A

divide

```
public static void divide(double alpha,
                          DMatrixD1 a)
```

Performs an in-place element by element scalar division with the scalar on top.

$a_{ij} = \alpha / a_{ij}$

Parameters:

`a` - (input/output) The matrix whose elements are divide the scalar. Modified.

`alpha` - top value in division

divide

```
public static void divide(DMatrixD1 a,
                          double alpha)
```

Performs an in-place element by element scalar division with the scalar on bottom.

$a_{ij} = a_{ij} / \alpha$

Parameters:

`a` - (input/output) The matrix whose elements are to be divided. Modified.

`alpha` - the amount each element is divided by.

divide

```
public static <T extends DMatrixD1> T divide(double alpha,
                                             T input,
                                             T output)
```

Performs an element by element scalar division with the scalar on top.

$b_{ij} = \alpha / a_{ij}$

Parameters:

`alpha` - The numerator.

`input` - The matrix whose elements are the divisor. Not modified.

`output` - Where the results are stored. Modified. Can be null.

Returns:

The resulting matrix

divide

```
public static <T extends DMatrixD1> T divide(T input,
                                             double alpha,
                                             @Nullable
                                             T output)
```

Performs an element by element scalar division with the scalar on botton.

$b_{ij} = a_{ij} / \alpha$

Parameters:

`input` - The matrix whose elements are to be divided. Not modified.

`alpha` - the amount each element is divided by.

`output` - Where the results are stored. Modified. Can be null.

Returns:

The resulting matrix

changeSign

```
public static void changeSign(DMatrixD1 a)
```

Changes the sign of every element in the matrix.

$a_{ij} = -a_{ij}$

Parameters:

`a` - A matrix. Modified.

changeSign

```
public static <T extends DMatrixD1> T changeSign(T input,
                                                 @Nullable
                                                 T output)
```

Changes the sign of every element in the matrix.

$output_{ij} = -input_{ij}$

Parameters:

`input` - A matrix. Modified.

fill

```
public static void fill(DMatrixD1 a,
                        double value)
```

Sets every element in the matrix to the specified value.

a_{ij} = value

Parameters:

`a` - A matrix whose elements are about to be set. Modified.

`value` - The value each element will have.

rref

```
public static DMatrixRMaj rref(DMatrixRMaj A,
                               int numUnknowns,
                               @Nullable
                               @Nullable DMatrixRMaj reduced)
```

Puts the augmented system matrix into reduced row echelon form (RREF) using Gauss-Jordan elimination with row (partial) pivots. A matrix is said to be in RREF is the following conditions are true:

1. If a row has non-zero entries, then the first non-zero entry is 1. This is known as the leading one.
2. If a column contains a leading one then all other entries in that column are zero.
3. If a row contains a leading 1, then each row above contains a leading 1 further to the left.

[1] Page 19 in, Otter Bretscherm "Linear Algebra with Applications" Prentice-Hall Inc, 1997

Parameters:

`A` - Input matrix. Unmodified.

`numUnknowns` - Number of unknowns/columns that are reduced. Set to -1 to default to A.numCols, which works for most applications.

`reduced` - Storage for reduced echelon matrix. If null then a new matrix is returned. Modified.

Returns:

Reduced echelon form of A

See Also:

`RrefGaussJordanRowPivot_DDRM`

elementLessThan

```
public static BMatrixRMaj elementLessThan(DMatrixRMaj A,
                                          double value,
                                          BMatrixRMaj output)
```

Applies the > operator to each element in A. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrx

`value` - value each element is compared against

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elementLessThanOrEqual

```
public static BMatrixRMaj elementLessThanOrEqual(DMatrixRMaj A,
                                                 double value,
                                                 BMatrixRMaj output)
```

Applies the ≥ operator to each element in A. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrix

`value` - value each element is compared against

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elementMoreThan

```
public static BMatrixRMaj elementMoreThan(DMatrixRMaj A,
                                          double value,
                                          BMatrixRMaj output)
```

Applies the > operator to each element in A. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrix

`value` - value each element is compared against

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elementMoreThanOrEqual

```
public static BMatrixRMaj elementMoreThanOrEqual(DMatrixRMaj A,
                                                 double value,
                                                 BMatrixRMaj output)
```

Applies the ≥ operator to each element in A. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrix

`value` - value each element is compared against

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elementLessThan

```
public static BMatrixRMaj elementLessThan(DMatrixRMaj A,
                                          DMatrixRMaj B,
                                          BMatrixRMaj output)
```

Applies the < operator to each element in A. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrix

`B` - Input matrix

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elementLessThanOrEqual

```
public static BMatrixRMaj elementLessThanOrEqual(DMatrixRMaj A,
                                                 DMatrixRMaj B,
                                                 BMatrixRMaj output)
```

Applies the A ≤ B operator to each element. Results are stored in a boolean matrix.

Parameters:

`A` - Input matrix

`B` - Input matrix

`output` - (Optional) Storage for results. Can be null. Is reshaped.

Returns:

Boolean matrix with results

elements

```
public static DMatrixRMaj elements(DMatrixRMaj A,
                                   BMatrixRMaj marked,
                                   @Nullable
                                   @Nullable DMatrixRMaj output)
```

Returns a row matrix which contains all the elements in A which are flagged as true in 'marked'

Parameters:

`A` - Input matrix

`marked` - Input matrix marking elements in A

`output` - Storage for output row vector. Can be null. Will be reshaped.

Returns:

Row vector with marked elements

countTrue

```
public static int countTrue(BMatrixRMaj A)
```

Counts the number of elements in A which are true

Parameters:

`A` - input matrix

Returns:

number of true elements

concatColumns

```
public static DMatrixRMaj concatColumns(DMatrixRMaj a,
                                        DMatrixRMaj b,
                                        @Nullable
                                        @Nullable DMatrixRMaj output)
```

output = [a , b]

concatColumnsMulti

```
public static DMatrixRMaj concatColumnsMulti(DMatrixRMaj... m)
```

Concatinates all the matrices together along their columns. If the rows do not match the upper elements are set to zero.

A = [m[0] , ... , m[n-1]]

Parameters:

`m` - Set of matrices

Returns:

Resulting matrix

concatRows

```
public static void concatRows(DMatrixRMaj a,
                              DMatrixRMaj b,
                              DMatrixRMaj output)
```

output = [a ; b]

concatRowsMulti

```
public static DMatrixRMaj concatRowsMulti(DMatrixRMaj... m)
```

Concatinates all the matrices together along their columns. If the rows do not match the upper elements are set to zero.

A = [m[0] ; ... ; m[n-1]]

Parameters:

`m` - Set of matrices

Returns:

Resulting matrix

permuteRowInv

```
public static DMatrixRMaj permuteRowInv(int[] pinv,
                                        DMatrixRMaj input,
                                        DMatrixRMaj output)
```

Applies the row permutation specified by the vector to the input matrix and save the results in the output matrix. output[perm[j],:] = input[j,:]

Parameters:

`pinv` - (Input) Inverse permutation vector. Specifies new order of the rows.

`input` - (Input) Matrix which is to be permuted

`output` - (Output) Matrix which has the permutation stored in it. Is reshaped.

abs

```
public static void abs(DMatrixD1 a,
                       DMatrixD1 c)
```

Performs absolute value of a matrix:

c = abs(a)
$c_{ij} = abs(a_{ij})$

Parameters:

`a` - A matrix. Not modified.

`c` - A matrix. Modified.

abs

```
public static void abs(DMatrixD1 a)
```

Performs absolute value of a matrix:

a = abs(a)
a_{ij} = abs(a_{ij})

Parameters:

`a` - A matrix. Modified.

symmLowerToFull

```
public static void symmLowerToFull(DMatrixRMaj A)
```

Given a symmetric matrix which is represented by a lower triangular matrix convert it back into a full symmetric matrix.

Parameters:

`A` - (Input) Lower triangular matrix (Output) symmetric matrix

symmUpperToFull

```
public static void symmUpperToFull(DMatrixRMaj A)
```

Given a symmetric matrix which is represented by a lower triangular matrix convert it back into a full symmetric matrix.

Parameters:

`A` - (Input) Lower triangular matrix (Output) symmetric matrix

apply

```
public static DMatrixRMaj apply(DMatrixRMaj input,
                                DOperatorUnary func,
                                @Nullable
                                @Nullable DMatrixRMaj output)
```

This applies a given unary function on every value stored in the matrix

```
output[i,j] = func(input[i,j])
```

A and B can be the same instance.

Parameters:

`input` - (Input) input matrix. Not modified

`func` - Unary function accepting a double

`output` - (Output) Matrix. Can be same instance as A. Modified.

Returns:

The output matrix

apply

```
public static DMatrixRMaj apply(DMatrixRMaj input,
                                DOperatorUnary func)
```