

## NAME

Matrix

## SYNOPSIS

```
use Matrix;

use Matrix qw(:all);
```

## DESCRIPTION

Matrix class provides the following methods:

new, AddColumnValues, AddRowValues, Copy, GetColumnValues, GetColumnValuesAsColumnMatrix, GetColumnValuesAsRowMatrix, GetColumnValuesAsString, GetColumnValuesAsVector, GetDiagonalValues, GetDiagonalValuesAsColumnMatrix, GetDiagonalValuesAsRowMatrix, GetDiagonalValuesAsString, GetDiagonalValuesAsVector, GetDimension, GetMatrixValuesReference, GetNumOfColumns, GetNumOfRows, GetRowValues, GetRowValuesAsColumnMatrix, GetRowValuesAsRowMatrix, GetRowValuesAsString, GetRowValuesAsVector, GetSize, GetValue, IdentityMatrix, IsAntiSymmetric, IsBiDiagonal, IsDiagonal, IsIdentity, IsLeftTriangular, IsLowerBiDiagonal, IsLowerTriangular, IsLowerUniTriangular, IsMatrix, IsNegative, IsPositive, IsRightTriangular, IsSkewSymmetric, IsSquare, IsStrictlyLowerTriangular, IsStrictlyUpperTriangular, IsSymmetric, IsTriDiagonal, IsUnit, IsUnitLowerTriangular, IsUnitUpperTriangular, IsUpperBiDiagonal, IsUpperTriangular, IsUpperUniTriangular, NewFromColumns, NewFromDiagonal, NewFromRows, One, SetAllValues, SetColumnValues, SetDiagonalValues, SetMatrixPrintStyle, SetRowValues, SetValue, SetValuePrintFormat, StringifyMatrix, Transpose, UnitMatrix, Zero, ZeroMatrix

The following functions are available:

IsMatrix, IdentityMatrix, NewFromRows, NewFromColumns, NewFromDiagonal, UnitMatrix, ZeroMatrix

The following operators are overloaded:

```
" " bool !
@{}
+ - * / ** %
== != < <= > >=
neg
abs exp log sqrt cos sin
```

The matrix row and column indices start from zero.

## FUNCTIONS

new

```
$NewMatrix = $Matrix->new($NumOfRows, $NumOfCols);
```

Creates a new Matrix of size *NumOfRows* x *NumOfCol* and returns NewMatrix object.

AddColumnValues

```
$Matrix->AddColumnValues(@Values);
$Matrix->AddColumnValues(\@Values);
$Matrix->AddColumnValues($VectorObject);
$Matrix->AddColumnValues("Value1 Value2 Value3 ...");
```

Adds column values to *Matrix* using an array, reference to an array, another vector, or space delimited value string and returns *Matrix*.

AddRowValues

```
$Matrix->AddRowValues(@Values);
$Matrix->AddRowValues(\@Values);
$Matrix->AddRowValues($VectorObject);
$Matrix->AddRowValues("Value1 Value2 Value3 ...");
```

Adds row values to *Matrix* using an array, reference to an array, another vector, or space delimited value string and returns Matrix.

Copy

```
$NewMatrix = $Matrix->Copy();
```

Creates a copy of *Matrix* and returns NewMatrix.

### GetColumnValues

```
@Values = $Matrix->GetColumnValues($ColIndex);  
$ValueCount = $Matrix->GetColumnValues($ColIndex);
```

Returns an array containing column value specified using *ColIndex* with column index starting at 0. In scalar context, number of column values is returned.

### GetColumnValuesAsColumnMatrix

```
$ColumnMatrix = $Matrix->GetColumnValuesAsColumnMatrix($ColIndex);
```

Returns a new ColumnMatrix containing column values specified using *ColIndex* with column index starting at 0.

### GetColumnValuesAsRowMatrix

```
$RowMatrix = $Matrix->GetColumnValuesAsRowMatrix($ColIndex);
```

Returns a new RowMatrix containing column values specified using *ColIndex* with column index starting at 0.

### GetColumnValuesAsString

```
$ColumnValuesString = $Matrix->GetColumnValuesAsString($ColIndex);
```

Returns a space delimited ColumnValuesString column values specified using *ColIndex* with column index starting at 0.

### GetColumnValuesAsVector

```
$ColumnVector = $Matrix->GetColumnValuesAsVector($ColIndex);
```

Returns a new ColumnVector column values specified using *RowIndex* with column index starting at 0.

### GetDiagonalValues

```
@Values = $Matrix->GetDiagonalValues();  
$ValueCount = $Matrix->GetDiagonalValues();
```

Returns an array containing diagonal values. In scalar context, number of diagonal values is returned.

### GetDiagonalValuesAsColumnMatrix

```
$ColumnMatrix = $Matrix->GetDiagonalValuesAsColumnMatrix();
```

Returns a new ColumnMatrix containing diagonal values corresponding to *Matrix*.

### GetDiagonalValuesAsRowMatrix

```
$RowMatrix = $Matrix->GetDiagonalValuesAsRowMatrix();
```

Returns a new RowMatrix containing diagonal values corresponding to *Matrix*.

### GetDiagonalValuesAsString

```
$DiagonalValuesString = $Matrix->GetDiagonalValuesAsString();
```

Returns a space delimited DiagonalValuesString containing diagonal values corresponding to *Matrix*.

### GetDiagonalValuesAsVector

```
$DiagonalVector = $Matrix->GetDiagonalValuesAsVector();
```

Returns a new DiagonalVector containing diagonal values corresponding to *Matrix*.

### GetDimension

```
( $NumOfRows, $NumOfCols ) = $Matrix->GetDimension();
```

Returns size of *Matrix*.

### GetMatrixValuesReference

```
$ValuesRef = $Matrix->GetMatrixValuesReference();
```

Returns a reference to array containing rows and column values corresponding to *Matrix*.

**GetNumOfColumns**

```
$NumOfCols = $Matrix->GetNumOfColumns();
```

Returns NumOfCols in *Matrix*.

**GetNumOfRows**

```
$NumOfRows = $Matrix->GetNumOfRows();
```

Returns NumOfRows in *Matrix*.

**GetRowValues**

```
@Values = $Matrix->GetRowValues($RowIndex);  
$ValueCount = $Matrix->GetRowValues($RowIndex);
```

Returns an array containing row value specified using *RowIndex* with row index starting at 0. In scalar context, number of row values is returned.

**GetRowValuesAsColumnMatrix**

```
$ColumnMatrix = $Matrix->GetRowValuesAsColumnMatrix($RowIndex);
```

Returns a new ColumnMatrix containing row values specified using *RowIndex* with column index starting at 0.

**GetRowValuesAsRowMatrix**

```
$RowMatrix = $Matrix->GetRowValuesAsRowMatrix($RowIndex);
```

Returns a new RowMatrix containing row values specified using *RowIndex* with row index starting at 0.

**GetRowValuesAsString**

```
$RowValuesString = $Matrix->GetRowValuesAsString($RowIndex);
```

Returns a space delimited RowValuesString row values specified using *RowIndex* with row index starting at 0.

**GetRowValuesAsVector**

```
$RowVector = $Matrix->GetColumnValuesAsVector($RowIndex);
```

Returns a new RowVector row values specified using *RowIndex* with row index starting at 0.

**GetSize**

```
($NumOfRows, $NumOfCols) = $Matrix->GetSize();
```

Returns size of *Matrix*.

**GetValue**

```
$Value = $Matrix->GetValue($RowIndex, $ColIndex, [$SkipIndexCheck]);
```

Returns Value of *Matrix* element specified using *RowIndex* and *ColIndex* with indices starting at 0 with optional validation of specified index values.

**IdentityMatrix**

```
$NewIdentityMatrix = $Matrix->IdentityMatrix($NumOfRows, $NumOfCols);  
$NewIdentityMatrix = Matrix::IdentityMatrix($NumOfRows, $NumOfCols);  
$NewIdentityMatrix = Matrix::IdentityMatrix();
```

Creates a new IdentityMatrix of specified size *NumOfRows* x *NumOfCol* or of size 3 x 3 and returns NewIdentityMatrix object.

**IsAntiSymmetric**

```
$Status = $Matrix->IsAntiSymmetric();
```

Returns 1 or 0 based on whether *Matrix* is an anti symmetric matrix.

A matrix is an anti symmetric matrix:

- . It's a square matrix
- . Its elements are asymmetric with respect to main diagonal. In other words,

elements below the main diagonal are equal to the negative of elements above the main diagonal.

Transpose of an anti symmetric matrix equals the negative of the matrix.

#### IsBiDiagonal

```
$Status = $Matrix->IsBiDiagonal();
```

Returns 1 or 0 based on whether *Matrix* is upper or lower bidiagonal matrix.

#### IsDiagonal

```
$Status = $Matrix->IsDiagonal();
```

Returns 1 or 0 based on whether *Matrix* is a diagonal matrix.

A matrix is a diagonal matrix:

- . It's a square matrix
- . All its off-diagonal elements are zeros and its diagonal elements may or may not be zeros

#### IsIdentity

```
$Status = $Matrix->IsIdentity();
```

Returns 1 or 0 based on whether *Matrix* is an identity matrix.

#### IsLeftTriangular

```
$Status = $Matrix->IsLeftTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a left or lower matrix.

A matrix is a left triangular matrix:

- . It's a square matrix
- . All its entries above the main diagonal are zero

#### IsLowerBiDiagonal

```
$Status = $Matrix->IsLowerBiDiagonal();
```

Returns 1 or 0 based on whether *Matrix* is a lower bidiagonal matrix.

A matrix is a lower bidiagonal matrix:

- . It's a square matrix
- . All its main diagonal and lower diagonal elements are non-zeros and all its other elements are zeros

#### IsLowerTriangular

```
$Status = $Matrix->IsLowerTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a left or lower triangular matrix.

A matrix is a lower triangular matrix:

- . It's a square matrix
- . All its entries above the main diagonal are zero

#### IsLowerUniTriangular

```
$Status = $Matrix->IsLowerUniTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a lower triangular matrix.

#### IsMatrix

```
$Status = Matrix::IsMatrix($Object);
```

Returns 1 or 0 based on whether *Object* is a Matrix object.

#### IsNegative

```
$Status = $Matrix->IsNegative();
```

---

Returns 1 or 0 based on whether *Matrix* is a negative matrix containing only values less than or equal to zero.

#### IsPositive

```
$Status = $Matrix->IsPositive();
```

Returns 1 or 0 based on whether *Matrix* is a negative matrix containing only values greater than or equal to zero.

#### IsRightTriangular

```
$Status = $Matrix->IsRightTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a right or upper triangular matrix.

#### IsSkewSymmetric

```
$Status = $Matrix->IsSkewSymmetric();
```

Returns 1 or 0 based on whether *Matrix* is a skew or anti symmetric matrix.

#### IsSquare

```
$Status = $Matrix->IsSquare();
```

Returns 1 or 0 based on whether *Matrix* is a square matrix containing equal number of rows and columns.

#### IsStrictlyLowerTriangular

```
$Status = $Matrix->IsStrictlyLowerTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a strictly lower triangular matrix.

A matrix is a strictly lower triangular matrix:

- . It's a square matrix
- . All its entries on and above the main diagonal are zero

#### IsStrictlyUpperTriangular

```
$Status = $Matrix->IsStrictlyUpperTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a strictly upper triangular matrix.

A matrix is a strictly upper triangular matrix:

- . It's a square matrix
- . All its entries on and below the main diagonal are zero

#### IsSymmetric

```
$Status = $Matrix->IsSymmetric();
```

Returns 1 or 0 based on whether *Matrix* is a symmetric matrix.

A matrix is a symmetric matrix:

- . It's a square matrix
- . Its elements are symmetric with respect to main diagonal. In other words, elements below the main diagonal are equal to the elements above the main diagonal.

Transpose of a symmetric matrix equals the matrix itself.

#### IsTriDiagonal

```
$Status = $Matrix->IsTriDiagonal();
```

Returns 1 or 0 based on whether *Matrix* is a tridiagonal matrix.

A matrix is a tribidiagonal matrix:

- . It's a square matrix
- . All its main diagonal, upper diagonal, and lower diagonal elements are non-zeros and all its other elements are zeros

## IsUnit

```
$Status = $Matrix->IsUnit();
```

Returns 1 or 0 based on whether *Matrix* is a unit matrix.

A matrix is a unit matrix:

- . It's a square matrix
- . All its diagonal elements are ones and its off-diagonal elements are zeros

## IsUnitLowerTriangular

```
$Status = $Matrix->IsUnitLowerTriangular();
```

Returns 1 or 0 based on whether *Matrix* is an unit lower triangular matrix.

A matrix is an unit lower triangular matrix:

- . It's a square matrix
- . All its entries main diagonal are one
- . All its entries above the main diagonal are zero

## IsUnitUpperTriangular

```
$Status = $Matrix->IsUnitUpperTriangular();
```

Returns 1 or 0 based on whether *Matrix* is an unit upper triangular matrix.

A matrix is an unit upper triangular matrix:

- . It's a square matrix
- . All its entries main diagonal are one
- . All its entries below the main diagonal are zero

## IsUpperBiDiagonal

```
$Status = $Matrix->IsUpperBiDiagonal();
```

Returns 1 or 0 based on whether *Matrix* is an upper bidiagonal matrix.

A matrix is an upper bidiagonal matrix:

- . It's a square matrix
- . All its main diagonal and upper diagonal elements are non-zeros and all its other elements are zeros

## IsUpperTriangular

```
$Status = $Matrix->IsUpperTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a right or upper triangular matrix.

A matrix is an upper triangular matrix:

- . It's a square matrix
- . All its entries below the main diagonal are zero

## IsUpperUniTriangular

```
$Status = $Matrix->IsUpperUniTriangular();
```

Returns 1 or 0 based on whether *Matrix* is a right or upper triangular matrix.

## NewFromColumns

```
$NewMatrix = Matrix::NewFromColumns($Col1Vector, $Col2Vector, ...);
$NewMatrix = Matrix::NewFromColumns($Col1ValuesRef, $Col2ValuesRef, ...);
$NewMatrix = Matrix::NewFromColumns("Val1 Val2 ...", "Val1 Val2", ...);

$NewMatrix = $Matrix->NewFromColumns($Col1Vector, $Col2Vector, ...);
$NewMatrix = $Matrix->NewFromColumns($Col1ValuesRef, $Col2ValuesRef, ...);
$NewMatrix = $Matrix->NewFromColumns("Val1 Val2 ...", "Val1 Val2", ...);
```

Creates a new Matrix using specified column values and returns NewMatrix object.

The column values can be specified in one of the following formats:

- . List of vector objects
- . References to list of values
- . List of strings containing columns values delimited by space

Each column must contain the same number of values.

#### NewFromDiagonal

```
$NewMatrix = Matrix::NewFromDiagonal($DiagonalVector);  
$NewMatrix = Matrix::NewFromDiagonal($DiagonalValuesRef);  
$NewMatrix = Matrix::NewFromDiagonal("Val1 Val2 ...");  
  
$NewMatrix = Matrix->NewFromDiagonal($DiagonalVector);  
$NewMatrix = Matrix->NewFromDiagonal($DiagonalValuesRef);  
$NewMatrix = Matrix->NewFromDiagonal("Val1 Val2 ...");
```

Creates a new Matrix using specified diagonal values and returns NewMatrix object.

The column values can be specified in one of the following formats:

- . A vector object
- . Reference to list of values
- . Strings containing diagonal values delimited by space

#### NewFromRows

```
$NewMatrix = Matrix::NewFromRows($Row1Vector, $RowVector, ...);  
$NewMatrix = Matrix::NewFromRows($Row1ValuesRef, $Row2ValuesRef, ...);  
$NewMatrix = Matrix::NewFromRows("Val1 Val2 ...", "Val1 Val2", ...);  
  
$NewMatrix = $Matrix->NewFromRows($Row1Vector, $Row2Vector, ...);  
$NewMatrix = $Matrix->NewFromRows($Row1ValuesRef, $Row2ValuesRef, ...);  
$NewMatrix = $Matrix->NewFromRows("Val1 Val2 ...", "Val1 Val2", ...);
```

Creates a new Matrix using specified row values and returns NewMatrix object.

The row values can be specified in one of the following formats:

- . List of vector objects
- . References to list of values
- . List of strings containing columns values delimited by space

Each row must contain the same number of values.

#### One

```
$Matrix->One();
```

Sets values of all *Matrix* elements to 1 and returns *Matrix*.

#### SetAllValues

```
$Matrix->SetAllValues($Value);
```

Sets values of all *Matrix* elements to specified *Value* and returns *Matrix*.

#### SetColumnValues

```
$Matrix->SetColumnValues($ColIndex, @Values);  
$Matrix->SetColumnValues($ColIndex, \@Values);  
$Matrix->SetColumnValues($ColIndex, $VectorObject);  
$Matrix->SetColumnValues($ColIndex, "Value1 Value2 Value3 ...");
```

Sets column values of a specified *ColIndex* of *Matrix* using an array, reference to an array, another vector, or space delimited value string and returns *Matrix*.

#### SetDiagonalValues

```
$Matrix->SetDiagonalValues(@Values);  
$Matrix->SetDiagonalValues(\@Values);  
$Matrix->SetDiagonalValues($VectorObject);  
$Matrix->SetDiagonalValues("Value1 Value2 Value3 ...");
```

Sets values of the diagonal in square *Matrix* and returns *Matrix*.

### SetMatrixPrintStyle

```
$Matrix->SetMatrixPrintStyle($MatrixStyle);  
$Matrix::SetMatrixPrintStyle($MatrixStyle);
```

Sets print style for matrix rows for an individual object or the whole class during StringifyMatrix operation. Possible *MatrixStyle* values: *AllRowsInOneLine*, *OneRowPerLine*. Default: *AllRowsInOneLine*.

### SetRowValues

```
$Matrix->SetRowValues($ColIndex, @Values);  
$Matrix->SetRowValues($ColIndex, \@Values);  
$Matrix->SetRowValues($ColIndex, $VectorObject);  
$Matrix->SetRowValues($ColIndex, "Value1 Value2 Value3 ...");
```

Sets row values of a specified *RowIndex* of *Matrix* using an array, reference to an array, another vector, or space delimited value string and returns *Matrix*.

### SetValue

```
$Matrix->SetValue($RowIndex, $ColIndex, $Value, [$SkipIndexCheck]);
```

Sets Value of *Matrix* element specified using *RowIndex* and *ColIndex* with indices starting at 0 with optional validation of specified index values and return *Matrix*.

### SetValuePrintFormat

```
$Matrix->SetValuePrintFormat($ValueFormat);  
$Matrix::SetValuePrintFormat($ValueFormat);
```

Sets value print format for an individual object or the whole class during StringifyMatrix operation and returns *Matrix*.

### StringifyMatrix

```
$String = $Matrix->StringifyMatrix();
```

Returns a string containing information about *Matrix* object.

### Transpose

```
$Matrix->Transpose();
```

Transposes *Matrix* by swapping rows with columns and returns *Matrix*.

### UnitMatrix

```
$NewUnitMatrix = $Matrix::UnitMatrix($NumOfRows, $NumOfCols);  
$NewUnitMatrix = $Matrix::UnitMatrix();  
$NewUnitMatrix = $Matrix->UnitMatrix($NumOfRows, $NumOfCols);
```

Creates a new UnitMatrix of specified size *NumOfRows* x *NumOfCol* or of size 3 x 3 and returns NewUnitMatrix object.

### Zero

```
$Matrix->Zero();
```

Sets values of all *Matrix* elements to 0 and returns *Matrix*.

### ZeroMatrix

```
$NewZeroMatrix = $Matrix::ZeroMatrix($NumOfRows, $NumOfCols);  
$NewZeroMatrix = $Matrix::ZeroMatrix();  
$NewZeroMatrix = $Matrix->ZeroMatrix($NumOfRows, $NumOfCols);
```

Creates a new ZeroMatrix of specified size *NumOfRows* x *NumOfCol* or of size 3 x 3 and returns NewZeroMatrix object.

## AUTHOR

Manish Sud <msud@san.rr.com>

## SEE ALSO

Vector.pm



**COPYRIGHT**

Copyright (C) 2015 Manish Sud. All rights reserved.

This file is part of MayaChemTools.

MayaChemTools is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.