

# MATRIX CLASS PACKAGE

## SUMMARY

Introduction .....	2
How to create a Matrix object? .....	2
What are the basic operations available in the package?.....	5
Checking equality between two matrixes:.....	5
addition of two matrixes: .....	5
Subtraction of two matrixes:.....	5
multiplication of a Matrix by a scalar:.....	5
multiplication of two matrixes: .....	6
More informations on function Matrix ::op .....	6
The methods of Matrix class .....	7
Conclusion.....	12

Akpé Aurelle Emmanuel Moïse Zinsou

## INTRODUCTION

Nowadays Matrix became an inescapable tool to solve a large set of problems of scientific order. From mathematics to physics and even IT, Matrixes have widely proved their efficiency. PHP is an extremely rich language which has a programmers community very active and which is moreover an absolutely free tool. However , like many other languages it doesn't support matrix calculations natively. Matrix class package allow handling all the types of matrix with real coefficients. This library permits to do, all the basic calculations on matrix and even more complex, like Determinant calculation, the search of eigenvalues and eigenvectors.

## HOW TO CREATE A MATRIX OBJECT?

Matrix class package use arrays, the only type of PHP which permit handling matrix.

To create a matrix  $A_{m \times n}$  you must proceed like that :

```
<?php $a=new Matrix(array(array(0,-1,2),array(0,0,-1),array(0,0,0))); ?>
```

To create a **line matrix**  $A_{1 \times n}$  you must proceed like that :

```
<?php $a=new Matrix(array(array(0,-1,2))); ?>
```

To create a **column matrix**  $A_{m \times 1}$  you must proceed like that :

```
<?php $a=new Matrix(array(array(0),array(-1),array(2))); ?>
```

To create a **diagonal matrix**  $A_{n \times n}$  you must proceed like that :

```
<?php $a= Matrix::diagonal(array(3,2,5,4));?>
```

To create a **scalar matrix**  $A_{n \times n}$  you must proceed like that:

```
<?php  
$a= Matrix::scalar(float $scalaire, int $n);  
  
//Example of identity matrix:  
  
$a= Matrix::scalar(1, 9);
```

```
//Example of scalar matrix :
```

```
$a= Matrix::scalar(pi(), 9);
```

```
?>
```

To create a **random vector A** you must proceed like that:

```
<?php
```

```
$a= Matrix:: random_vector(int($x),bool( $column =true),int($max=10)); //or
```

```
$a= Matrix::random_positive_vector(int($x), bool( $column =true),int($max=10));// for  
vectors with only positives coefficients
```

```
$a=Matrix::random_limitedfloat_vector(int($x),bool($column=true),int($max=10),int($limit  
ed=10)); // for vectors with limited number of digit after the float point
```

```
$a=Matrix::random_limitedpositivefloat_vector(int($x),bool($column=true),int($max=1  
0),int($limited=10)); // for vectors with only positives coefficients with limited  
number of digit after the float point
```

```
$a= Matrix::random_norm1_vector(int($x),$column=true)// to generate a vector of which  
its norm strictly equals to 1.
```

```
$a= Matrix::null_vector(int($x),$column=true)// to generate a vector of which its norm  
strictly equals to 0 ; a null vector.
```

```
$a= Matrix::K_vector(int($x),$column=true,float($k)) // to generate a vector of which its  
norm strictly equals to $k ;
```

**//NB:** Argument **\$x** is used for specify the number of coefficients; **\$max** permits to specify the maximum numerator and denominator in the generation of float number with unlimited digit. For the functions which contain argument **\$limited**, argument **\$max** is used for specify maximum number which may be at the numerator of coefficients while argument **\$limited** is used for specify denominator; **\$limited** is by default 10. That means that coefficients will have only one digit after the float point. Put it to 1 for example will generate integer coefficients exclusively. Argument **\$column** of functions is use to specify if user wish to have a line matrix (line vector) or column matrix (column vector).

```
//Example of random column vectors with 9 coefficients
```

```
$a= Matrix::randomvector(9);
```

```
$a= Matrix::random_limitedfloat_vector(9);
```

```
$a=Matrix::random_limitedfloat_vector(9,true,10,3) // could generate coefficients  
with unlimited digit after the float point... ;
```

?>

To create a **random matrix A<sub>[m][n]</sub>** you must proceed like that:

```
<?php
```

```
$a=Matrix::random_matrix(int($i),int($j),int($max=10),int($limited=10),string($type=null));  
// or  
  
$a=Matrix::random_limitedfloat_matrix(int($i),int($j),int($max=10),int($limited=10),string(  
$type=null));
```

**//NB:** Arguments **\$i et \$j** are used to specify respectively number of rows and the number of column of the random matrix. Argument **\$max** permits to specify the maximum numerator in the generation of float number with unlimited digit. Depending on function, **\$limited** is used for specify maximum denominator. **\$limited** is by default 10. That means that coefficients will have only one digit after the float point. Put it to 1 for example will generate integer coefficients exclusively. For the function `random_matrix()`, **\$limited** is used above all for specify the maximum bound of the interval where the denominator will be chosen. Argument **\$type** of functions is used for specify if user wish to have a matrix selected in the following set :

**\$type={square,uptri,lowtri,diagonal,uphessenberg,lowhessenberg,tridiagonal,rectangular}**. If no type is specify, a null matrix will be generated. It is preferable to use variant of function `random_vector` for random matrix with one dimension equal to 1 in order to not generate matrix with bad representation.

As the names are explicit we will only talk here about `lowtri` and `uptri`. `Lowtri` is for lower triangular and `uptri` is for upper triangular.

```
// example of random matrix A[m][n]  
  
$A= Matrix::random_limitedfloat_matrix(5,5,10,10,'diagonal') ;  
  
$A= Matrix::random_limitedfloat_matrix(7,5,10,10,'rectangular') ;?>
```

## WHAT ARE THE BASIC OPERATIONS AVAILABLE IN THE PACKAGE?

All the basic operations on matrix are supported in the package via the static method Matrix::OP. All the mathematical rules are respected and the different operations will return an answer only if they are mathematically defined otherwise the functions will return silently the Boolean false ...

### CHECKING EQUALITY BETWEEN TWO MATRIXES:

For testing equality between two matrixes:

```
<?php  
$a=Matrix::OP(MatrixObject,MatrixObject,'='); //return a Boolean  
?>
```

### ADDITION OF TWO MATRIXES:

To add two matrix :

```
<?php  
$a=Matrix::OP(MatrixObject,MatrixObject,'+');return a Matrix object  
?>
```

### SUBTRACTION OF TWO MATRIXES:

To subtract two matrixes:

```
<?php  
$a=Matrix::OP(MatrixObject,MatrixObject,'-'); //return a Matrix object,  
?>
```

### MULTIPLICATION OF A MATRIX BY A SCALAR:

For a scalar and a matrix product:

```
<?php  
$a=Matrix::OP(MatrixObject,scalar,'*');  
$a=Matrix::OP(scalar,MatrixObject,'*');  
?>
```

## MULTIPLICATION OF TWO MATRIXES:

For multiplication of two matrixes:

```
<?php
```

```
$a=Matrix::OP(MatrixObject,MatrixObject,'*'); //return either an int or a matrix  
object
```

## MORE INFORMATIONS ON FUNCTION MATRIX ::OP

Keep in mind that only the order of the two operands really count , the operator can then been placed anywhere during the method call

Example:

```
$a=Matrix::OP("*,MatrixObject,MatrixObject");
```

```
$a=Matrix::OP(MatrixObject,'*',MatrixObject);
```

```
$a=Matrix::OP(MatrixObject,MatrixObject,'*'); //This way is just a convention I  
have chosen to use in my code.
```

It is also possible to add or subtract a matrix object with a scalar .this operation consist according to the order of operandes in adding or subtracting a scalar to/from each coefficient of the matrix .

Example:

```
$a=Matrix::OP(MatrixObject ,5 ,'-'); !== $a=Matrix::OP(5,MatrixObject ,'-');
```

```
$a=Matrix::OP(MatrixObject ,5 ,'+'); !== $a=Matrix::OP(5,MatrixObject ,'+');
```

## THE METHODS OF MATRIX CLASS

**public function** \_\_construct(\$lines)

**public function** CholeskyLLTdcmp()

apply a LU factorization by Cholesky's method and return the matrice L such as  $L^*L^t = M$  where M is a square symmetric matrix defined positive handled and  $L^t$  the transposed matrix of L.

**public function** getdiag()

return an array of diagonal coefficients.

**public function** ludcmp2()

apply a LU factorization on the matrix and return an array containing the result. Index 'U' contains the upper triangular matrix and index 'L' contains the lower triangular matrix.

**public function** eigsubspace(**float**(\$x)) return the Eigen-subspace associated with a real eigenvalue \$x of matrix in process( algorithm based on Singular Values Decomposition. Resolve the linear system  $A'x=0$  ( $A \cdot xI=0$ ).)

**public function** Gaussian\_algo

Apply algorithm of elimination by permutation of Gauss Jordan and return an array containing at index 'matrix' the result (an identity matrix if the matrix is non singular), 'p' the number of permutations, and 'pivots' an array of the different chosen pivots.

**public function** getA1norm()

return A1 norm of the matrix.

**public function** getAinfiniteNorm()

return A infinite norm of the matrix.

**public function** getnorm()

return the Frobenius or Euclidian norm of the matrix.

**public function** gettrack()

return the track of the matrix.

**public function** getcolumns()

return an array of columns of the matrix.

**public function** getcomatrix()

return the cofactors matrix of the matrix.

**public function** getIdentity()

return the identity matrix corresponding to the matrix.

**public function** getrows()

return an array of the rows of the matrix.

**public function** getrank()

return the rank of the matrix....

**public function** getspacevectorsbasis()

return (a basis) a family of independent vectors which spanned the vectorial space of the matrix.The result is a matrix of which each column vector constitute a vector of the family.

**public function** GramschmidtWith\_Reorth()

apply a QR decomposition by the method classical Gram Schmidt with reorthogonalization...and return an array of matrix objects where index "Q" contains an orthogonal matrix Q and index « R » the triangular matrix R. The function doesn't work on PHP versions preceding version 5.3 because of absence of structure Goto.

**public function** householder\_elementarymatrixQ()

return an array of elementary matrix  $Q_k$  of householder of the matrix in process.

**public function** householder\_matrixQ()

return the orthogonal matrix Q stem from householder QR decomposition of the initial matrix.

**public function** householder\_matrixR()

return the triangular matrix R stem from householder QR decomposition of the initial matrix.

**public function** householder\_transformations()

apply a QR decomposition by the method of householder... and return an array of matrix objects where index "Q" contains an orthogonal matrix Q and index « R » the triangular matrix R.

**public function** is\_antisymmetric()

return true if the matrix is skewsymmetric ,false if not.

**public function** is\_column()

return true if the matrix is a column vector, false if not.

**public function** is\_diagonal

return true if the matrix is diagonal , false if not.

**public function** is\_idempotent()

return true if the matrix is idempotent , false if not.

**public function** is\_identity ()

return true if the matrix is identity matrix, false if not.

**public function** is\_heptadiagonal()

return true if the matrix is heptadiagonal , false if not.

**public function** is\_lowtriangular()

return true if the matrix is lower triangular , false if not.

**public function** is\_lowhessenberg()

return true if the matrix is a lower hessenberg matrix, false if not..

**public function** is\_line()

return true if the matrix is a line vector, false if not.

**public function** is\_null()

return true if the matrix is null matrix, false if not.

**public function** is\_orthogonal()

return true if the matrix is orthogonal , false if not.

**public function** is\_pentadiagonal()

return true if the matrix is pentadiagonal , false if not.

**public function** is\_reversible()

return true if the matrix is reversible, false if not.

**public function** is\_scalar()

return true if the matrix is scalar, false if not.

**public function** is\_square()

return true if the matrix is square , false if not.

**public function** is\_uptriangular()

return true if the matrix is upper triangular, false if not.

**public function** is\_uphessenberg()

return true if the matrix is upper hessenberg matrix , false if not.

**public function** is\_symmetric()

return true if the matrix is symmetric , false if not.

**public function** is\_triangularMatrix()

return true if the matrix is triangular, false if not.

**public function** is\_tridiagonalMatrix()

return true if the matrix is tridiagonal , false if not.

**public function** ludcmp(**bool**(\$mat=true))

apply a LU factorization on the matrix and return an array containing the result ,a unique matrix LU which one can easily separate( just as we did in method ludcmp2()) at index 'a', index 'd' contains a value of the sign in relation with the number of permutations and which permits to calculate easily the matrix DET and at least an array of pivots at index 'indx' which is used for solving linear systems of equations associated to the matrix. \$mat is used for specify if LU decomposition must be returned in matrix object form or not .It is by default the case.

**public function** lubksb( Matrixobject \$b)

resolve linear systems of equations associated to the matrix in process where \$b is the column vector of right side example :Ax=b.

**public function** getDet()

return the matrix DETERMINANT.

**public function** Exp() //approximant of Padé...

return matrix exponential

**public function** power((**int**)\$x)

return the power \$x of the matrix

**public function** maybe\_nilpotent((**int**)\$x)

return an array of which index « maybe\_nilpotent » contains true and index « index » contains the index of nilpotence if the matrix is nilpotent, and return simply false if not. If \$x is specified, the search will automatically stop at the power \$x+1 of matrix and will return simply false to mean that the null matrix didn't appears before the value \$x; this can be helpful when we want to limit the search, or extend it according to maximum execution time allocated via the php.ini files.

**public function** modifiedgramschmidt()

apply a QR decomposition by the modified Gram Schmidt method ...and return an array of matrix objects where index "Q" contains an orthogonal matrix Q and index « R » the triangular matrix R.

**public function** getsize()

return the number of coefficients of the matrix.

**public function** opposite()

return the matrix opposite to the matrix in process.

**public function** pseudoinverse()

return the pseudo-inverse of the matrix if it is rectangular and its inverse if it is square.

**public function** reversed()

return the matrix inverse by Gaussian elimination method

**public function** lu\_reversed()

return the matrix inverse by LU decomposition method.

**public function** svdcmp(**bool**(\$mat))

return an array containing the decomposition in singular values of the matrix in process.  
\$mat is used for specify if the results must be returned as matrix or a simple array.

**public function** svdbksb(**Matrix** \$b)

apply an algorithm for solve linear system of equations based on SVD. \$b is use to specify the right side member of equation. Example : Ax=b.

**public function** getformat()

return a string representing a textual format of matrix in terms of indices I and J ("i\*j").

**public function** transposed()

return the transposed of the matrix

**public function** tridiagonalization()

return the tridiagonalized form by the method of householder of the symmetric matrix in process.

**public function** triL(**int**(\$x))

return a lower triangular matrix of which coefficients are those of the matrix in process. Argument \$x is used for specify from which diagonal one wish to take values. By default \$x equals 0 .This means to take values from the main diagonal.

**public function** triU(**int**(\$x))

return a upper triangular matrix of which coefficients are those of the matrix in process. Argument \$x is use to specify from which diagonal one wish to take values. By default \$x equals 0 .This means to take values from the main diagonal.

**public function** uphessenbergformReduction (**bool** (\$mat))

return a reduced upper hessenberg form of the matrix in process. \$mat is used for specify whether or not result must be returned as matrix object.

**public function** getEigens()

apply QR algorithm with shift and reorthogonalization by the method of givens after reducing to upper hessenberg form and return an array containing the real and complex eigenvalues of the matrix. It is an effective algorithm on all types of matrix with real coefficients, symmetric or not which the library can handle

**public function** powerIteration(**(int)**\$n=25,**(float)**\$delta=1.e-13)

apply the power iteration algorithm and return the eigenvalue with the highest module and the associated eigenvector. This algorithm could trigger like an infinite loop if the highest value is complex. \$n, is use to specify number of iterations allowed before stopping and \$delta the degree of precision .

**public function** deflation(**(int)** \$\$n=null, **(int)** \$iter=25, **(float)** \$delta=1.e-13)

apply the power iteration algorithm combined to deflation and return an array of the eigenvalues of the matrix and the associated eigenvectors matrix. This algorithm could trigger like an infinite loop if the eigenvalues are complex. So it will be clever to use exclusively it in symmetric matrix case. However if the matrix is not symmetric and doesn't have complex eigenvalue, this function will be very useful to search all the Eigen vectors and values. One can limit the search to the n first eigenvalues by specifying it with argument \$n of the function. \$iter, is used for specify number of iterations allowed before stopping and \$delta the degree of precision.

**public function** inversepowerIteration(**(int)** \$n=30)

apply inverse power iteration algorithm and return the eigenvalue with the lowest module and the associated eigenvector. However the algorithm systematically choose value with the lowest module on top of 0 .But if there is too much negative low values and a too large gap separate them, this algorithm choose then the lowest in module of negative values . This algorithm could trigger like an infinite loop if the eigenvalues are complex. \$n, is used for specify number of iterations allowed before stopping and \$delta the degree of precision .

**public function** inverseandshift (**(int)** \$mu=0, **(int)** \$n=100, **(float)** \$delta=1.e-14)

apply inverse and shift algorithm and return the eigenvalue which is the closest to the chosen shift \$mu and the associated vector. By default \$mu is equal to 0. \$n, is used for specify number of iterations allowed before stopping and \$delta the degree of precision.

**Public static Function sign** (\$x ,\$y)

apply the result of the function sign(\$y) to \$x ;

The Matrix class also implements the following interfaces **ArrayAccess** , **countable** , **JsonSerializable** , **Iterator**

You can then use **count()**,**json\_encode()**, but also iterate on a matrix object with a **foreach** loop and also access indices with array access style **\$matrix[\$i][\$j]**;

## CONCLUSION

**Contact:** [leizmo@gmail.com](mailto:leizmo@gmail.com)

**TO BE CONTINUED...**