MATRIX CLASS PACKAGE

SOMMAIRE

Akpé Aurelle Emmanuel Moïse Zinsou

INTRODUCTION

De nos jours les matrices sont devenues un outil incontounable dans la résolution d'un grand nombre de problèmes d'ordre scientifique.Des mathématiques aux sciences physiques en passant par l'informatique,les matrices ont largement démontré leur efficacité. PHP est un langage extrêmement riche qui dispose d'une communauté de programmeurs très actifs. Cependant , à l'instar de beaucoup d'autres langages le support des calculs matriciels n'est pas natif. La présente librairie permet de réaliser, toutes les opérations basiques sur les matrices à coefficients réels mais également les plus complexes, comme le calcul des déterminants, la recherche de vecteurs propres et de valeurs propres aussi bien réelles que complexes.

COMMENT CREER UN OBJET MATRIX ?

Matrix class package utilise des arrays php, seul type du PHP pouvant permettre la manipulation de matrices.

Pour créer une matrice $\mathbf{A_{[m][n]}}$ il faut procéder comme suit :

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

Pour créer une **matrice ligne** $\mathbf{A_{[1][n]}}$ il faut procéder comme suit :

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

Pour créer une **matrice colonne** $\mathbf{A_{[m][1]}}$ il faut procéder comme suit :

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

Pour créer une **matrice diagonale** $\mathbf{A_{[n][n]}}$ il faut procéder comme suit :

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

Pour créer une **matrice scalaire** $\mathbf{A_{[n][n]}}$ il faut procéder comme suit :

```
<?php
```

```
$a= Matrix::scalar(float $scalaire, int $n);

//Exemple de matrice identité :

$a= Matrix::scalar(1, 9);

//Exemple de matrice scalaire :

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

?>
```

Pour créer un **vecteur aleatoire A** il faut procéder comme suit :

```
<?php

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

$a= Matrix::random_positive_vector(int($x), bool( $column =true),int($max=10));//
pour les vecteurs à coefficients positifs

$a=Matrix::random_limitedfloat_vector(int($x),bool($column=true),int($max=10),int($limit
ed=10)); // pour lesvecteurs à coefficients à nombre de chiffres limité après la virgule .

$a=Matrix::random_limitedpositivefloat_vector(int($x),bool($column=true),int($max=1
0),int($limited=10)); // pour les vecteurs à coefficients positifs à nombre de chiffres
limité après la virgule .

$a= Matrix::random_norm1_vector(int($x),$column=true)// pour générer un vecteur dont
la norme est strictement égale à 1.

$a= Matrix::null_vector(int($x),$column=true)// pour générer un vecteur dont la norme est
strictement égale à 0 donc tous les coefficients sont égaux à 0.

$a= Matrix::K_vector(int($x),$column=true,float($k))// pour générer un vecteur dont la
norme est strictement égale à $k.
```

//NB: l'argument **$x** sert à spécifier le nombre de coefficients et l'argument **$max** permet de spécifier le numérateur et le dénominateur maximal dans la génération des nombres flottants à nombre de chiffres illimités. Pour les fonctions contenant l'argument **$limited** , l'argument **$max** sert à spécifier le nombre maximal pouvant rester au numérateur des coefficients tandis que l'argument **$limited** sert à spécifier le dénominateur **;$limited** est par défaut à 10 ce qui signifie que les coefficients auront un chiffre après la virgule.le mettre à 1 par exemple donnerait des coefficients exclusivements entiers. L'argument **$column** des fonctions sert à spécifier si l'utilisateur souhaite avoir une matrice (vecteur) ligne ou une matrice(vecteur) colonne.

```
//Exemple de vecteurs colonnes aleatoires à 9 coefficients
```

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

$a= Matrix::random_limitedfloat_vector(9) ;

$a=Matrix::random_limitedfloat_vector(9,true,10,3)//pourrait générer des coefficients à nombre de chiffres illimité après la virgule … ;

?>

Pour créer une matrice **A[m][n]** **aléatoire** il faut procéder comme suit :

<?php

$a=Matrix::random_matrix(**int**($i),**int**($j),**int**($max=10),**int**($limited=10),**string**($type=**null**));
//ou encore

$a=Matrix::random_limitedfloat_matrix(**int**($i),**int**($j),**int**($max=10),**int**($limited=10),**string**($type=**null**));

//NB: les arguments **$i et $j** servent à spécifier respectivement le nombre de lignes et de colonnes de la matrice aléatoire.L'argument **$max** permet de spécifier le numérateur maximal dans la génération des nombres flottants à nombre de chiffres illimités.Selon la fonction , **$limited** sert à spécifier le nombre maximal pouvant rester au dénominateur **;$limited** est par défaut à 10 ce qui signifie que les coefficients auront un chiffre après la virgule.le mettre à 1 par exemple donnerait des coefficients exclusivements entiers.Pour la fonction random_matrix(), **$limited** spécifie surtout la borne maximale de l'interval où sera choisi le dénominateur. L'argument **$type** des fonctions sert à spécifier si l'utilisateur souhaite avoir une matrice choisie dans l'ensemble suivant : **$type={square,uptri,lowtri,diagonal,uphessenberg,lowhessenberg,tridiagonal,rectangular}** .Si aucun type n'est spécifié une matrice nulle sera retournée.Il est préférable d'utiliser les variantes de la fonction random vecteur pour les matrices aléatoires dont l'une des dimensions est égale à 1 afin de ne pas générer des matrices mal représentées.

Comme les noms sont explicites nous ne parlerons ici que de lowtri et uptri.Lowtri équivaut à triangulaire inférieure et uptri équivaut à triangulaire supérieure.

// exemple matrice A[m][n] aléatoire

$A= Matrix::random_limitedfloat_matrix(5,5,10,10,'diagonal') ;

$A= Matrix::random_limitedfloat_matrix(7,5,10,10,'rectangular') ;

?>

QUELLES SONT LES OPERATIONS MATRICIELLES DISPONIBLES DANS LE PACKAGE ?

Toutes les opérations élémentaires sur les matrices sont prises en charge dans le package via la méthode statique Matrix::OP(). Toutes les règles mathématiques sont respectées et les différentes opérations ne retourneront de réponses que si elles sont mathématiquement définies sinon les fonctions retourneront silencieusement le boolean false …

LA VERIFICATION D'EGALITE ENTRE DEUX MATRICES:

Pour tester l'égalité entre deux matrices :

```
<?php

$a=Matrix::OP(MatrixObject,MatrixObject,'=');//return a Boolean

?>
```

L'ADDITION DE DEUX MATRICES:

Pour additionner deux matrices :

```
<?php

$a=Matrix::OP(MatrixObject,MatrixObject,'+');return a Matrix object

?>
```

LA SOUSTRACTION DE DEUX MATRICES:

Pour soustraire deux matrices :

```
<?php

$a=Matrix::OP(MatrixObject,MatrixObject,'-') ;//return a Matrix object,

?>
```

La multiplication d'une matrice par un scalaire:

Pour une matrice par un scalaire :

```
<?php

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

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

LA MULTIPLICATION DE DEUX MATRICES:

Pour multiplier deux matrices entre elles :

```
<?php

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

PLUS D'INFOS SUR LA FONCTION MATRIX ::OP

Gardez à l'esprit que seul l'ordre des opérandes compte réellement.L'opérateur peut donc être placé n'importe où.

Example:

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

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

$a=Matrix::OP(MatrixObject,MatrixObject,'*');//ceci est une convention que j'ai volontairement dans mon code.
```

Il est également possible d'additionner ou de soustraire une matrix et un scalaire .L'operation consiste selon l'ordre des opérandes d'ajouter ou de soustraire un scalaire à chaque élement de la matrix.

Exemple:

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

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

LES DIFFERENTES METHODES DE LA CLASSE MATRIX

public function __construct(**$lines**)

public function CholeskyLLTdcmp()

effectue une factorisation LU par la méthode de Cholesky et retourne la matrice L telle que $L*L^t = M$ où M est la matrice carrée symétrique définie positive en cours de traitement et L^t la matrice transposée de L.

public function ludcmp2()

fait une factorisation LU de la matrice et retourne un tableau contenant le résultat.L'index 'U' contient la matrice triangulaire supérieure et l'index 'L' contient la matrice triangulaire inférieure.

public function eigsubspace(**float**($x))

retourne le sous espace propre associé à la valeur propre réelle $x de la matrice traitée(algorithme basé sur SVD.Résout le système A'x=0 (A-xI=0).)

public function Gaussian_algo()

Applique l'algorithme d'élimination par pivotage de Gauss Jordan et retourne un tableau contenant à l'index 'matrix' le résultat (une matrice identité si la matrice est inversible),'p' le nombre de pivotages ou de permutations, et 'pivots' un tableau des différents pivots choisis.

public function getA1norm()

retoune la norme A1 de la matrice.

public function getAinfinitenorm()

retourne la norme Ainfinie de la matrice

public function getnorm()

retourne la norme de frobenius ou norme euclidienne de la matrice

public function gettrack ()

retourne la trace de la matrice

public function getcolumns()

retourne un tableau des colonnes de la matrice

public function getcomatrix()

retourne la matrice des cofacteurs de la matrice

public function getIdentity ()

retourne la matrice identité correspondante de la matrice

public function getrows()

retourne un tableau des lignes de la matrice.

public function getrank()

retourne le rang de la matrice….

public function getspacevectorsbasis()

retourne (une base) une famille de vecteurs à la fois libre et génératrice d'un espace vectoriel de la matrice.Le résultat est sous forme de matrice dont chaque vecteur colonne constitue un vecteur de la famille.

public function GramschmidtWith_Reorth()

applique une décomposition QR par la méthode Gram schmidt classique avec reorthogonalisation…et retourne un tableau d'objet matrice où l'index "Q" a pour valeur la matrice Q orthogonale et l'index « R » la matrice triangulaire R.

public function householder_elementarymatrixQ()

retourne un tableau des matrices élémentaires Q_k de householder de la matrice en cours de traitement.

public function householder_matrixQ()

retourne la matrice Q orthogonale issue de la décomposition QR de la matrice initiale

public function householder_matrixR()

retourne la matrice triangulaire R issue de la décomposition QR de la matrice initiale

public function householder_transformations()

applique une décomposition QR par la méthode de householder…et retourne un tableau d'objet matrice où l'index "Q" a pour valeur la matrice Q orthogonale et l'index « R » la matrice triangulaire R.

public function is_antisymmetric()

retourne true si la matrice est antisymmetrique , false si non.

public function is_column()

retourne true si la matrice est un vecteur colonne ,false si non.

public function is_diagonal()

retourne true si la matrice est diagonale , false si non

public function is_idempotent()

retourne true si la matrice est idempotente ,false si non

public function is_identity()

retourne true si la matrice est une matrice identité,false si non.

public function is_heptadiagonal ()

retourne true si la matrice est heptadiagonale ,false si non

public function is_lowtriangular ()

retourne true si la matrice est triangulaire inférieure ,false si non.

public function is_lowhessenberg()

retourne true si la matrice est une matrice d'hessenberg inférieure ,false si non.

public function is_lineMatrix()

retourne true si la matrice est un vecteur ligne ,false si non

public function is_nullMatrix()

retourne true si la matrice est la matrice nulle,false si non

public function is_orthogonal()

retourne true si la matrice est orthogonnale ,false si non.

public function is_pentadiagonal()

retourne true si la matrice est pentadiagonale ,false si non.

public function is_reversible()

retourne true si la matrice est inversible,false si non.

public function is_scalar()

retourne true si la matrice est scalaire, false si non.

public function is_square()

retourne true si la matrice est carré ,false si non.

public function is_uptriangular ()

retourne true si la matrice est triangulaire supérieure, false si non.

public function is_uphessenberg()

retourne true si la matrice est une matrice d'hessenberg supérieure ,false si non.

public function is_symmetric()

retourne true si la matrice est symétrique ,false si non .

public function is_triangular()

retourne true si la matrice est triangulaire ,false si non.

public function is_tridiagonal()

retourne true si la matrice est tridiagonale ,false si non.

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

applique une décomposition LU à la matrice traitée et retourne un array contenant en fait les deux matrices LU qu'on peut facilement séparer(comme cela a été fait dans la déclaration de

la méthode factorization_LU)à l'index 'a', l'index 'd' porte la valeur représentant le signe en rapport au nombre de permutations et pouvant permettre de calculer facilement le déterminant et enfin un array des pivots à l'index 'indx' pouvant également servir dans la résolution du système d'équations linéaires associé à la matrice. $mat sert à spécifier si la décomposition LU doit être retournée sous forme de matrice ou non.C'est par défaut le cas.

public function lubksb(Matrix $b)

résout le système linéaire associé à la matrice en cours de traitement où b est le vecteur colonne du membre de droite exemple :Ax=b.

public function MatrixDet() retourne le determinant de la matrice.

public function Exp() //approximant of padé...

retoune l'exponentiel de la matrice

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

retourne la puissance $x de la matrice

public function maybe_nilpotent((**int**)**$x**)

retourne un tableau dont l'index « maybe_nilpotent » vaut true et l'index « index » vaut l'indice de nilpotence si la matrice est nilpotente,et retourne simplement false si non.Si $x est précisé , la recherche s'arrêtera automatiquement à la puissance $x de la matrice et retournera simplement false pour signifier que la matrice nulle n'est pas apparue avant cette valeur de $x ; ceci peut se révéler utile lorsqu'on veut limiter la recherche, ou l'étendre compte tenu du temps d'éxéxution maximum alloué via le php.ini.

public function modifiedgramschmidt()

applique une décomposition QR par la méthode de Gram schmidt modifiée…et retourne un tableau d'objet matrice où l'index "Q" a pour valeur la matrice Q orthogonale et l'index « R » la matrice triangulaire R.

public function getsize()

retourne le nombre de coefficients de la matrice

public function opposite ()

retourne la matrice opposée de la matrice

public function pseudoinverse()

retourne la pseudo inverse de la matrice si elle est rectangulaire et son inverse si elle est carrée.

public function reversed()

retourne la matrice inverse de la matrice par la méthode du pivot de Gauss.

public function lu_reversed()

retourne la matrice inverse de la matrice par la méthode de la décomposition LU.

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

retourne un array contenant la décomposition en valeurs singulières de la matrice traitée.$mat sert à spécifier si les résultats doivent être retournés sous forme de matrices ou de simples tableau.

public function svdbksb**(Matrix| |array** $b)

applique un algorithme de résolution de système d'équations linéaires basé sur la SVD.$b sert à spécifier le membre de droite de l' équation. exemple :Ax=b.

public function getformat()

retourne une chaine de caractères le format du tableau matriciel sous forme('m*n').

public function transposed()

retourne la matrice transposée de la matrice

public function tridiagonalization()

retourne la forme tridiagonalisée par la méthode de householder de la matrice symétrique en cours de traitement

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

retourne une matrice triangulaire inférieure dont les coefficients sont ceux de matrice en cours de traitement.L'argument $x permet de spécifier à partir de quelle diagonale on souhaite prendre les valeurs.Par défaut $x vaut 0 ce qui revient à prendre les valeurs à partir de la diagonale principale.

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

retourne une matrice triangulaire supérieure dont les coefficients sont ceux de la matrice en cours de traitement. L'argument $x permet de spécifier à partir de quelle diagonale on souhaite prendre les valeurs.Par défaut $x vaut 0 ce qui revient à prendre les valeurs à partir de la diagonale principale.

public function uphessenbergformReduction (**bool** ($mat)) retourne une forme réduite hessenberg supérieure de la matrice en cours de traitement. .$mat sert à spécifier si le résultat doit être retourné sous forme de matrice ou non.

public function getEigens() applique l'algorithme QR avec shift et reorthogonalisation par la méthode de givens après réduction à la forme hessenberg et retourne un tableau contenant les valeurs propres réelles et complexes de la matrice.C'est un algorithme efficace sur tous les types de matrices à coefficients réels ,symmétriques ou non que la librairie peut manipuler.

public function powerIteration((int)$n=25,(float) $delta=1.e-13)
applique l"algorithme de la puissance itérée et retourne la valeur propre qui est le plus grand en module et le vecteur propre associé.Cet algorithme déclenchera ce qui ressemble à une boucle infinie si la plus grande valeur propre est complexe . $n , représente ici le nombre d"itérations autorisées avant arrêt et $delta le degré de précision.
public function deflation((int)$$n=null, (int)$iter=25, (float) $delta=1.e-13)
applique l"algorithme de la puissance itérée conjuguée à la déflation et retourne un tableau des valeurs propres de la matrice et des vecteurs propres correspondants.Cet algorithme peut lancer ce qui ressemble à une boucle infinie si les valeurs propres de la matrice sont complexes.Aussi il vaudrait mieux l"utiliser dans les cas de matrices symmétriques exclusivement.Cependant si une matrice ne possède aucune valeur propre complexe ,elle peut se révéler utile pour la recherche des vecteurs propres. On peut limiter la recherche aux n premières valeurs propres en le spécifiant dans l"argument $n de la fonction. $iter, représente ici le nombre d"itérations autorisées avant arrêt et $delta le degré de précision de la recherche .

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

applique l"algorithme de l"iteration inverse et retourne la valeur propre qui est le plus faible en module et le vecteur propre associé.Cependant l"algorithme choisi systématiquement les valeurs de plus faible module en dessus de 0 .Toutefois si les plus faibles valeurs sont négatives et qu"un trop grand écart les sépare, l"algorithme choisi alors la plus faible en module des valeurs négatives.Cet algorithme peut lancer ce qui ressemble à une boucle infinie si les valeurs propres de la matrice sont complexes. $n , représente ici le nombre d"itérations autorisées avant arrêt.

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

applique l"algorithme de l"iteration inverse avec décalage et retourne la valeur propre dont le module est plus proche du shift $mu choisi et le vecteur propre associé.Par défaut $mu est égal à 0.$n, représente ici le nombre d"itérations autorisées avant arrêt et $delta le degré de précision de la recherche.

Public static function sign ($x ,$y) applique le retour de la fonction sgn($y) à $x ;

La classe Matrix implémente également les interfaces suivantes ArrayAccess , countable , JsonSerializable , Iterator

Vous pouvez donc utiliser count(),json_encode(), mais aussi itérer sur un objet Matrix avec une boucle foreach et également y accéder avec la notation crochets comme vous le feriez avec un tableau standard

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

A SUIVRE...

A SUIVRE...