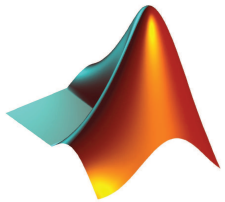
Initiation à Matlab : Langage Matriciel



Hervé Carfantan, IRAP

Herve.Carfantan@irap.omp.eu

http://userpages.irap.omp.eu/~hcarfantan

1. INTRODUCTION

❒ Matlab = abréviation de « *Matrix Laboratory* ».

Environnement informatique conçu pour le calcul matriciel

- facilité d'emploi,
- nombreuses possibilités d'affichages graphiques,
- convivialité,
- boites à outils (plus de 100 *toolboxes*):

  simulink, traitement du signal, traitement d'images, optimisation, équations différentielle, symbolique, contrôle…
- contributions d'autres utilisateurs :

  http://www.mathworks.fr/ et http://www.mathtools.net
- compatibilité octave (*freeware*) : http://www.octave.org

❒ Langage de programmation :

- interprété (non compilé : C, Fortran, Pascal...) ;
- non typé (variable = matrice)

  notion d'objet depuis la version 5 (non traité ici) ;
- interface possible avec C et Fortran ;
- transportable PC, Mac, Unix.

❒ Idéal pour le test et la réalisation de prototypes :

- gain en temps de développement par rapport au C/Fortran...
- perte en temps de calcul

❒ Parfaitement adapté au traitement de données et à l'enseignement.

Très utilisé à l'UPS, largement diffusé dans l'industrie

1. GÉNÉRALITÉS

1.1 Lancer Matlab

- PC : cliquer sur l'icône Matlab ou cliquer deux fois sur un fichier Matlab.

  Unix : taper matlab dans une fenêtre *shell*.
- Fenêtre « *command* »
- Taper les commandes Matlab (à la suite des chevrons **>>**).

  Rq : rappel des dernières commandes par ↑et ↓
  (et même **plot** puis ↑pour dernière commande débutant par **plot**).
- Pour quitter *quit* ou *exit*.

1.2.Langage interprété

```
>> 2+2
ans =   4
>> 2*sin(pi/4)
ans =   1.4142
```

- Résultat affiché et stocké dans la variable *ans*.
- Fonctions mathématiques usuelles :+, -, *, /, *sin*, *cos*, *exp*, *log*, *sqrt* ...
- Constantes : *pi*, *i* ...
- Tests : <, >, <=, <=, ==, ~= renvoie 1 si vrai, 0 si faux

1.3.Variables

```
>> x = pi/4
x =  0.7854
```

- Nom de variable : commence par une lettre (< 19 car).
- Attention Matlab voit les majuscules (x ≠ X).
- Point virgule pour ne pas afficher le résultat.
- Plusieurs commandes par ligne, séparées par un point virgule ou une virgule :

```
>> x = pi/2; y = sin(x);
```

- Ligne interrompue par « ... ».

1.4. Variables complexes

- Matlab indifférent aux nombres réels / complexes.
- `i` et `j` initialisés à la valeur complexe $\sqrt{-1}$: Ne pas les utiliser comme indice !

```
>> z = 3 + 2*i
z =  3.0000 + 2.0000i
```

- Fonctions usuelles prédéfinies dans Matlab :

  real, *imag*, *abs*, *angle* (en radian), *conj*.

```
>> r = abs(z)            % Module
r =    3.6056
>> theta = angle(z)      % Argument
theta =   0.5880         % En radian
>> y = r*exp(i*theta)
y =  3.0000 + 2.0000i
```


1.5. Vecteurs, matrices et leur manipulation

- Toute variable Matlab est une matrice.
- On peut entrer une matrice sous forme d'un tableau :

```
>> A = [ 1,2,3 ; 4,5,6 ; 7,8,9 ]
A =  1  2  3
     4  5  6
     7  8  9
```

- Éléments de matrice = expression quelconque de Matlab :

```
>> x = [ -1.3, sqrt(3), (1+2+3)*4/5 ]
x =  -1.3000     1.7321     4.8000
```


- Éléments de matrice référencés par leurs indices :

premier élément d'indice 1

```
>> x(2)
ans =  1.7321
>> x(5) = abs(x(1))
x =  -1.300  1.732  4.800  0.000  1.300
```

Rq : taille du vecteur *x* ajustée, éléments non précisés à 0.

Mais aussi :

```
>> indl = [3, 2, 1]; indc = [1, 3];
>> A(indl,indc)
ans =  7    9
       4    6
       1    3
```


- Fonctions *zeros*, *ones*, *eye* et *rand*, *randn* :

```
>> x=eye(2,3)    % eye comme I Identity
x =  1  0  0
     0  1  0
>> y=ones(1,5)
y = 1    1    1    1    1
>> z1=rand(1,3)       % aléatoire dans [0, 1]
z1 = 0.9501    0.2311    0.6068
>> z2=randn(1,3)      % aléatoire Gaussien
z1 = -0.4326   -1.6656   0.1253
```

- Taille d'une matrice *size*, *length* :

```
>> size(x)
ans =  2    3
>> size(x,2)   % Nombre de colonnes
ans =  3
```


- Ajout de lignes et de colonnes à une matrice :

```
>> r1 = [10, 11, 12];
>> A2 = [A ; r1]
A2 = 1    2    3
     4    5    6
     7    8    9
     10   11   12
>> r2 = [ 0 ; 0 ; 0];
>> A3 = [A, r2]
A3 = 1    2    3    0
     4    5    6    0
     7    8    9    0
```


1.6. L'opérateur « : »

- Opérateur d'énumération :

```
deb:pas:fin
```

construit le vecteur *[deb, deb+pas, deb+2*pas,... deb+n*pas]*

tel que *deb+n*pas* ≤ *fin* < *deb+(n+1)*pas*.

```
>> x = 0.5:0.1:0.85
x =    0.5000   0.6000  0.7000  0.8000
>> y = 5:-1:1
x =    5     4     3     2     1
```

pas d'incrémentation omis, 1 est pris par défaut :

```
>> x = 1:5
x =    1     2     3     4     5
```

- Sélection d'éléments d'un vecteur ou d'une matrice :

```
>> A(1,3)    % élément à la 1ère ligne 3ème col.
>> A(1,1:3)  % 3 premiers élts de la 1ère lig.
>> A(1,:)    % toute la 1ère ligne
>> A(:,3)    % toute la 3ème colonne
>> A(:)      % Matrice dans un vecteur colonne
>> x(1:2:10) % éléments d'indices impairs
```

et même (depuis la version 5) :

```
>> x(1:2:end)     % jusqu'au dernier
```


1.7. Chaînes de caractères

- Variables contenant des chaînes de caractères :

```
>> message = 'bienvenue sur Matlab';
>> message(4)
ans =    n
```

- Manipulations de même type que pour les vecteurs :

```
>> message = [message, ' version 7'];
message = bienvenue sur Matlab version 7
```

- Conversion de nombres en chaînes de caractères : *num2str*, *int2str*, *sprintf* :

```
>> message = ['pi vaut ', num2str(pi)]
message = pi vaut 3.1416
>> message = sprintf('pi vaut %f \n',pi)
message = pi vaut 3.141593
```


- Évaluation d'une chaîne de caractères en Matlab : *eval* et *feval* :

```
>> nom_var = 'x';
>> str = [ nom_var, '1 = sqrt(-1)']
str =    x1 = sqrt(-1)
>> eval(str)
x1 =     0.0000 + 1.0000i
>> nom_fonction = 'sin';
>> feval(nom_fonction,pi)
ans =  1.2246e-16   % sin(pi) ≠ 0 !
```


2. OPÉRATIONS MATRICIELLES

2.1 Opérations sur les matrices (au sens math.)

- Opérations usuelles définies de façon naturelle :

```
>> 2*A  % Multiplication par un scalaire
>> A+B  % Somme de deux matrices
>> A'   % Transposée conjuguée de A
>> A.'  % Transposée de A
>> A*B  % Multiplication matricielle
>> A^p  % Puissance matricielle
>> inv(A)        % Inverse d'une matrice
>> det(A)        % Déterminant de A
>> trace(A)      % Trace de A
>> [V,D] = eig(A)% Valeurs et Vecteurs propres
```


- Résolution de systèmes linéaires :

```
>> X = A\B        % solution de A*X = B
>> X = B/A        % solution de X*A = B
```

- Matrices creuses : gain en place mémoire et en coût de calcul.

```
>> A = eye(2); B = sparse(A);
>> A*[1;1];       % 4 multiplications
>> B*[1;1];       % 2 multiplications
```

- Construction de matrices particulières :

  vander (Vandermonde), *Toeplitz*, *Hankel*, *Hadamard*...

- Décomposition de matrices :

  eig (valeurs et vecteurs propres - *eigenvalues*), *svd* (valeurs singulières), *cholesky*, *QR*, *LU*…

2.2 Matrices = Suite de vecteurs colonnes

- Certaines fonctions agissent sur les vecteurs colonnes de la matrice :

```
>> min(A)      % Vecteur ligne des minima
               % des colonnes de A
>> max(A)      % Maxima
>> sum(A)      % Sommes
>> prod(A)     % Produits
>> cumsum(A)   % Sommes cumulées
>> cumprod(A)  % Produits cumulés
>> sort(A)     % Tri des colonnes
>> median(A)   % Valeurs médianes des col
>> mean(A)     % Moyennes des colonnes
>> std(A)      % Écarts-types des col
```

2.3 Matrices = Tableaux de valeurs

- Certaines fonctions agissent séparément sur chaque élément de la matrice :

```
>> A + 2     % Ajout à tous les éléments
>> A.*B      % Produit élément par élément
>> A.^3      % Puissance élt. par élt.
>> X = A./B  % Division élt. par élt.
```

Attention : Noter les différences entre `A*B` et `A.*B`…

```
>> exp(A); log(A); sqrt(A);
>> round(A);fix(A);floor(A);ceil(A); % Arrondis
>> sign(A);    % Signe
>> rem(A,2);   % Modulo
```

Pour calculer l'exponentiel, le logarithme ou la racine carrée matriciels : *expm*, *logm* et *sqrtm*.

- Tests sur les éléments des matrices

```
>> A>0; A==0          % Matrice binaire
>> [l,c] = find(A>0)  % Vecteurs d'indices
```

- Filtrage (*filter*), convolution (*conv*) (*filter2* et *conv2* en 2D).

- Transformation de Fourier rapide *fft*, transformation inverse *ifft*
  En dimension 2 (images) *fft2* et *ifft2*

  Remarque : fréquence sur [0, f_e], pour [-f_e/2, f_e/2] utiliser *fftshift et fftshift2*

2.4 Vecteur = coefficients d'un polynôme

```
>> p=[1 -6 -7.2 -2.7]    % P(z)=z^3-6z^2-7,2z-2,7
```

- *roots* racines du polynôme, *polyval* valeur du polynôme en certains points :

```
>> r=roots(p);
>> polyval(p,[ 1, exp(j*pi/4) ])
ans =   -14.9000      -8.4983 - 10.3841i
```

- *poly* :

```
>> p=poly(r)
p =    1.000   -6.000   -7.200   -2.700
```

Inverse de la fonction *roots* (racines →coefficients)

```
>> poly(A)
```

Polynôme caractéristique de la matrice A.

Remarque : produit de polynômes = convolution

2.5 Tableau de cellules

- Permet de stocker différent types de variables dans une seule

```
>> x{1,1} = 10;   x{2,1} = rand(5,1);
>> x{1,2} = 'MATLAB'; x{2,2} = {'hello';'bye'};
>> x
x =  [        10]    'MATLAB'
     [5x1 double]    {2x1 cell}
>> x{2,1}
ans = 0.8687  0.0844  0.3998  0.2599  0.8001
>> x{2,1}(4)
ans =  0.2599
>> x{2,2}
ans =   'hello'
        'bye'
```


- Concaténation de cellules :

```
>> y{1} = 'toto';
>> y{2} = 1:5;
>> z1 = [x, y']
z1 =   [        10]   'MATLAB'     'toto'
       [5x1 double]   {2x1 cell}   [1x5 double]
>> z1(2:3,:)
ans =  [5x4 double]   {2x1 cell  }
       'toto'         [1x5 double]
>> z2 = {x, y}
z2 = {2x2 cell}    {1x2 cell}
```


2.6 Structures

- Tableau avec accès par nom de champs

  S = STRUCT('Champ1',Valeur1,'Champ2',Valeur2,...)

Valeurs des champs : tableau de cellules de même dimension, ou une seule valeur

```
  s = struct('Nom',{'Einstein','Boop'},...
 'Prenom',{'Albert','Betty'},'Date',{1979,1930}) ;
>> s(1)
ans =  Nom: 'Einstein'
    Prenom: 'Albert'
       Date: 1979
>> s(1).Nom
ans = Einstein
```

fieldnames, getfield, setfield, isfield, orderfields, rmfield...

3. AFFICHAGES GRAPHIQUES ET ALPHANUMÉRIQUES

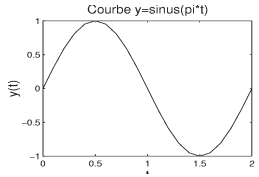
3.1 Affichage Alphanumérique

```
>> disp(message)
pi vaut 3.142
>> fprintf('pi vaut %.2f \n',pi)
pi vaut 3.14
>> rep=input('Valeur de lambda : ');
     % attend l'entrée d'une valeur.
Valeur de lambda : _
     % place l'entrée dans la variable rep
```


3.2 Affichages graphiques de courbes 1D

- Commandes *plot*, *title*, *xlabel*, *ylabel* :

```
>> t = 0:0.1:2;
>> y = sin(t*pi);
>> plot(t,y);  % Abscisse, Ordonnée
>> title('Courbe y(t) = sinus(pi*t)')
>> xlabel('t'); ylabel('y(t)')
```

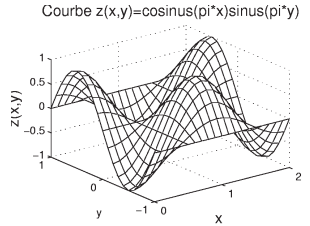


- Commandes *semilogx*, *semilogy*, *loglog*, *grid*, *zoom*…

3.3 Affichages graphiques de courbes 2D

- Commandes *mesh* et *meshgrid* :

```
>> x = 0:0.1:2;y = -1:0.1:1;
>> [X,Y] = meshgrid(x,y);
>> mesh(x,y,cos(pi*X).*sin(pi*Y))
```



- Mais aussi *meshc* et *contour*, *image*, *surf*, *surfc*, *view* ...

3.4 Affichage de plusieurs courbes

- Commandes *hold on* et *hold off*

- Commande *subplot* divise la fenêtre graphique :

```
subplot(nb_lig,nb_col,num_subdiv)
>> subplot(2,2,1)
>> plot(x,y)
>> subplot(2,2,2)
>> plot(x,y.^2)
```

(2,2,1)	(2,2,2)
(2,2,3)	(2,2,4)

- Commande *figure* crée une nouvelle fenêtre graphique.

- Déplacer les fenêtres, les agrandir…

4. ENVIRONNEMENT DE TRAVAIL, SCRIPTS, FONCTIONS

4.1. Répertoire de travail

- Répertoire courant *pwd*

- Le path de Matlab :

```
>> path(path,'c:\users\carfan\Matlab');
>> addpath c:\users\carfan\Matlab
```

- Commandes *cd* et *dir* ou *ls* (même syntaxe que sous MS-Dos et Unix).

- Définitions à l'initialisation dans le fichier « *startup.m* »

4.2. Sauvegarde et chargement de variables

- Sauvegarde au format Matlab « **.mat* » :

```
>> save nom_fichier nom_variables
```

Exemple :

```
>> save test.mat A x y
```

- Par défaut sauvegarde dans *matlab.mat*.
- Chargement :

```
load nom_fichier
```

- Commandes *who* et *whos* (informations sur les variables)
- Commandes *pack* et *clear* (gestion de la mémoire)
- Lecture de données de formats différents :
  readjpg, *readtif*, *readhdf*, *readbmp*, *readu8*, *fread*, *fscanf* ...
- Écriture de données :
  writedjpg, *writetif*, *writehdf*, *writebmp*, *fwriteu8*, *fwrite*, *fprintf*…

4.3 Scripts

Script = suite de commandes Matlab écrite dans un fichier.

- Ouvrir un fichier : menu « *file*, *new* » ou éditeur.
- Sauver le fichier avec une extension « .m » (*nom_fich*.m).
- Exécuter le script en tapant le nom du fichier sous Matlab (sans l'extension).
- Variables de l'espace de travail accessibles pour le script.
- Variables définies (ou modifiées) dans le script accessibles dans l'espace de travail.
- Commentaires à l'aide du caractère « % ».
- Indentation à l'aide des espaces blancs et tabulations.

4.4 Fonctions

Même utilisation que les fonctions prédéfinies dans Matlab.

- Première ligne d'une fonction :

```
function [ var_sorties, ...] = ...
       nom_fonction( var_entrée, ...)
```

- Exemple :

```
function y = sinuscardinal(x)
  z = sin(pi*x); % Variable de stockage
  y = z./x/pi;   % Résultat
                 % Attention en x=0 !
```

Remarque : x et y sont des vecteurs !

Fonction appelée par :

```
>> t = 0:.5:1; x = sinuscardinal(t)
x =  NaN    6.3662e-1    3.8980e-17
```


- Autre exemple :

```
function [mini, maxi] = minetmax(x)
    mini = min(x);
    maxi = max(x);
```

fonction appelée par :

```
>> M = rand(3), [minM, maxM] = minetmax(M)
M =  0.31652   0.29838   0.92061
     0.31583   0.54306   0.42533
     0.40139   0.12697   0.17610
minM =   0.31583   0.12697   0.17610
maxM =   0.40139   0.54306   0.92061
```


- Le nom de la fonction doit **impérativement** être identique au nom du fichier.
- Les nombres d'arguments en entrée et en sortie ne sont pas fixes et peuvent être récupérés par *nargin* et *nargout*.
- Variables de l'espace de travail non accessibles à la fonction sauf si elles sont fournies en variable d'entrée.
- Variables définies ou modifiées dans une fonction non accessibles dans l'espace de travail hormis variables de sortie.
- Possibilités de sous fonctions

Outils de Mise au point : « déboggage »

- Le plus simple : *keyboard*.
- Plus évolué : *dbstop*, *dbstep*, *dbcont*, *dbclear*...
  pour poser ou enlever un point d'arrêt, exécuter pas à pas ou poursuivre l'exécution.

⇒ éditeur/deboggeur intégré.

5. CONTRÔLES ET BOUCLES

5.1 Contrôle « *if* » :

```
if (expression logique)
     suite d'instructions 1;
else
     suite d'instructions 2;
end
```

Expression logique :

- Opérateurs logiques : et (&), ou (|), égal (==), supérieur (>,>=), inférieur (<,<=), non (~), ou exclusif (*xor*)...
- Fonctions logiques prédéfinies : *exist*, *any*, *find*, *isinf*, *isnan*...
- Toute expression Matlab, considérée comme fausse si nulle et vraie sinon (comme en C).

5.2 Boucle « *for* » :

```
for k=1:10
     suite d'instructions;
end
```

Remarque : ne pas utiliser `i=j=`$\sqrt{-1}$ comme indice de boucle,

Également :

```
for k=listk
```

avec **`listk`** vecteur quelconque.

Eviter au maximum les boucles dans Matlab (lent !).

- Exemple trivial :

```
for k=1:N, x(k) = k; end
```

est équivalent à : **`x = 1:N;`**

- Autre exemple :

```
temps = 0:0.1:10; f = 0.12 ;
for  k = 1:length(temps)
     x(k)=sin(2*pi*f*temps(k)) ;
end
```

est équivalent à : `x =sinus(2*pi*f*temps);`

- Autre exemple plus délicat :

```
r = 1:10;
A =[];    % initialisation de A à vide
for k=1:5, A = [A ; r]; end
```

peut être écrit :

```
A = ones(5,1)*r;
```

et même :

```
A = r(ones(5,1),:);
% Étudiez bien cette
% instruction
```


- Penser à utiliser les fonctions de Matlab :

```
maximum = max(A);
```

Au lieu de :

```
n = length(A(1,:)); % ou n = size(A,2) ;
for k=1:n, maximum(k) = max(A(:,k)); end
```

- Bien réfléchir avant de construire une boucle !

Essayer de penser en vectoriel !

5.3 Boucle « *while* » :

```
while (expression logique)
     suite d'instructions ;
end
```


6. AIDE EN LIGNE

- Commande *help*.
- Commande *help nom_répertoire*.
- Commande *help nom_fichier*.

- Créer son propre *help* :
  - pour les fonctions : `%` commentaires en début de fichier,
  - pour les répertoires : fichier « *Contents.m* ».

- Commande *lookfor*.

Remarques Importantes :

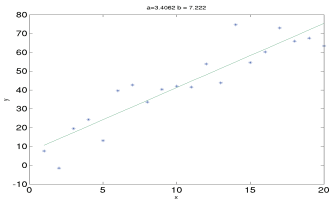
Il est vivement recommandé, avant d'écrire une fonction, de consulter la documentation ou l'aide en ligne afin de savoir s'il n'existe pas déjà une fonction similaire (et peut-être mieux écrite...).

De même, avant d'utiliser une fonction matlab pour la première fois, consulter la documentation ou l'aide en ligne pour vérifier qu'elle réalise bien ce que vous désirez.

7. EXEMPLES DE PROGRAMMES MATLAB

7.1 Estimation par moindres carrés

Problème :



On possède N mesures (x_k, y_k) (nécessairement bruitées mais le bruit à une moyenne nulle) censées correspondre à des points d'une droite. On va estimer la pente a et la valeur à l'origine b de la droite par moindres carrés c'est-à-dire minimisant : $\sum_{k=1}^{N}(y_k - ax_k - b)^2$

Solution :

En annulant les dérivées partielles par rapport à a et b, on trouve (un seul minimum) :

$$a = \frac{\sum_{k=1}^{N} x_k y_k - 1/N \left(\sum_{k=1}^{N} x_k\right)\left(\sum_{k=1}^{N} y_k\right)}{\sum_{k=1}^{N} x_k^2 - 1/N \left(\sum_{k=1}^{N} x_k\right)^2} \quad \text{et} \quad b = \frac{1}{N}\left(\sum_{k=1}^{N} y_k - a\sum_{k=1}^{N} x_k\right)$$

Programme Matlab :

```
% SIMULATION
% Vrai pente et valeur à l'origine
  a = pi;
  b = 10;
% Tirage d'un ensemble de mesures
  N = 20;
  x = 1:N;
  bruit = 10*randn(1,N);
  bruit = bruit - mean(bruit);
  y = a*x + b + bruit;
```



```
% ESTIMATION
% Estimation par moindres carrés de la
% pente et de la valeur à l'origine
  aest =  (sum(x.*y)-1/N*sum(x)*sum(y))/...
          (sum(x.^2)- 1/N*sum(x)^2);
% ou même
% aest =  (x*y' -1/N*sum(x)*sum(y))/...
%         (x*x' – 1/N*sum(x)^2);
  best = 1/N*(sum(y) -aest*sum(x));
```



```
% Affichage
  plot(x,y,'*',x,aest*x+best)
  xlabel('x')
  ylabel('y')
  title(['a=',num2str(aest),...
         ' b = ',num2str(best)])
```


7.2 Calcul de π par une méthode de Monte-Carlo

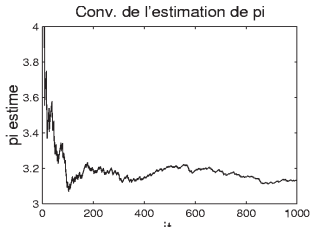
Problème :

On peut estimer la surface d'un quart de disque par une technique de type Monte-Carlo en tirant au hasard dans un carré de coté un et en faisant le rapport entre le nombre de points intérieurs au quart de cercle et le nombre total de point (on tend alors vers π/4). Cela se fait aisément avec Matlab :

```
>> N = 1000 ;
>> x = rand(1,N) ; y= rand(1,N) ;
>> pi_est = 4*sum(x.^2+y.^2<1)/N
pi_est  =  3.1280
```


Si l'on cherche à voir la convergence d'une telle méthode, on peut utiliser les sommes cumulées :

```
>> pi_est = 4*cumsum(x.^2+y.^2<1)./(1:N);
>> plot(pi_est);
>> xlabel('it.') ;ylabel('pi estime')
>> title('Conv. De l''estimation de pi')
```



7.3 Importance de la phase de la T.F.

Problème :

On peut mettre en évidence l'importance de la phase de la Transformée de Fourier d'une image en construisant une image z dont la TF a pour module le module de la TF d'une image x et pour phase, la phase de la TF d'une autre image y :

```
% Chargement de deux images
  load clown; x = X;
  load gatlin; y = X;
  lig = min(size(x,1),size(y,1));
  col = min(size(x,2),size(y,2));
  x=x(1:lig,1:col); y=y(1:lig,1:col);
```

```
% Construction de z dans Fourier
  X = fft2(x);
  Y = fft2(y);
  Z = abs(X).*exp(i*angle(Y));
  z = real(ifft2(Z));

% Affichage
  subplot(2,2,1); image(x) ; title('x')
  subplot(2,2,2); image(y) ; title('y')
  subplot(2,1,3) ; image(z) ;
  title('image module de x et phase de y')
  colormap('gray') ; axis('image')
```