Licence Sciences et Techniques
L2 MATH & PC & SI - parcours renforcé

UNIVERSITÉ DE TOULON

R31 – Initiation au traitement mathématique d'images avec Matlab/Octave

Recueil d'exercices corrigés et aide-mémoire.

Gloria Faccanoni

https://moodle.univ-tln.fr/course/view.php?id=5361

Année 2023 – 2024

Dernière mise-à-jour : Lundi 28 août 2023

Table des matières

Ce fascicule est un support pour le cours d'*initiation au traitement numérique d'images avec Matlab/Octave* de la deuxième année d'une Licence Scientifique.

R32 Modélisation physique – Traitement numérique d'images		
CM-TP	12h	4 séances de 3h

Séance 1 : TP : Rappels Matlab/Octave

Séance 2 : TP : concepts de base, manipulations élémentaires, résolution, [transformation du photomaton et du boulanger]

Séance 3 : TP : statistiques, modification du contraste, quantification

Séance 4 : TP : débruitage, détection des bords, images à couleur

CC

Gloria FACCANONI

IMATH Bâtiment M-117
Université de Toulon
Avenue de l'université
83957 LA GARDE - FRANCE

☎ 0033 (0)4 83 16 66 72

✉ gloria.faccanoni@univ-tln.fr
ℹ http://faccanoni.univ-tln.fr

Chapitre 1

Introduction à Octave/Matlab

Nous illustrerons les concepts vu en cours à l'aide de MATLAB (*MATrix LABoratory*), un environnement de programmation et de visualisation. Nous utiliserons aussi GNU Octave (en abrégé Octave) qui est un logiciel libre distribué sous licence GNU GPL. Octave est un interpréteur de haut niveau, compatible la plupart du temps avec MATLAB et possédant la majeure partie de ses fonctionnalités numériques. Dans ce chapitre, nous proposerons une introduction rapide à MATLAB et Octave. Le but de ce chapitre est de fournir suffisamment d'informations pour pouvoir tester les méthodes numériques vues dans ce polycopié. **Il n'est ni un manuel de Octave/Matlab ni une initiation à la programmation.**

1.1 Les environnements MATLAB et Octave

MATLAB et Octave sont des environnements intégrés pour le Calcul Scientifique et la visualisation. Ils sont écrits principalement en langage C et C++. MATLAB est distribué par la société *The MathWorks* (voir le site `www.mathworks.com`). Son nom vient de *MATrix LABoratory*, car il a été initialement développé pour le calcul matriciel. Octave, aussi connu sous le nom de GNU Octave (voir le site `www.octave.org`), est un logiciel distribué gratuitement. Vous pouvez le redistribuer et/ou le modifier selon les termes de la licence GNU *General Public License* (GPL) publiée par la *Free Software Foundation*.
Il existe des différences entre MATLAB et Octave, au niveau des environnements, des langages de programmation ou des *toolboxes* (collections de fonctions dédiées à un usage spécifique). Cependant, leur niveau de compatibilité est suffisant pour exécuter la plupart des programmes de ce cours indifféremment avec l'un ou l'autre. Quand ce n'est pas le cas – parce que les commandes n'ont pas la même syntaxe, parce qu'elles fonctionnent différemment ou encore parce qu'elles n'existent pas dans l'un des deux programmes – nous l'indiquons et expliquons comment procéder.
Nous utiliserons souvent dans la suite l'expression "commande MATLAB" : dans ce contexte, MATLAB doit être compris comme le langage utilisé par les deux programmes MATLAB et Octave. De même que MATLAB a ses toolboxes, Octave possède un vaste ensemble de fonctions disponibles à travers le projet Octave-forge. Ce dépôt de fonctions ne cesse de s'enrichir dans tous les domaines. Certaines fonctions que nous utilisons dans ce polycopié ne font pas partie du noyau d'Octave, toutefois, elles peuvent être téléchargées sur le site `octave.sourceforge.net`.

1.2 Installation(s) et version(s) en ligne

- La documentation et les sources d'Octave peuvent être téléchargées à l'adresse `https://www.gnu.org/software/octave/`.
  La version en ligne d'Octave est disponible ici `https://octave-online.net/`.
- L'université de Toulon propose aux étudiants la possibilité de le télécharger et de l'installer sur leur poste MATLAB. Toutes les informations sont ici `http://dsiun.univ-tln.fr/MATLAB.html`
  Par ailleurs, la version on line de MATLAB est disponible ici `https://fr.mathworks.com/products/matlab-online.html` Les étudiants et enseignants de l'université de Toulon peuvent s'y connecter avec leurs paramètres universitaires.

Une fois qu'on a installé MATLAB ou Octave, on peut accéder à l'environnement de travail, caractérisé par le symbole d'invite de commande `>>`. Il représente le prompt : cette marque visuelle indique que le logiciel est prêt à lire une commande. Il suffit de saisir à la suite une instruction puis d'appuyer sur la touche «Entrée».

1.3 Premiers pas

Lorsqu'on démarre Octave, une nouvelle fenêtre va s'ouvrir, c'est la fenêtre principale qui contient trois onglets : l'onglet "Fenêtre de commandes", l'onglet "Éditeur" et l'onglet "Documentation".

1.3.1 Fenêtre de commandes : mode interactif

L'onglet "Fenêtre de commandes" permet d'entrer directement des commandes et dès qu'on écrit une commande, Octave l'exécute et renvoie instantanément le résultat. L'invite de commande se compose de deux chevrons (>>) et représente le prompt : cette marque visuelle indique qu'Octave est prêt à lire une commande. Il suffit de saisir à la suite une instruction

puis d'appuyer sur la touche «Entrée». La console Octave fonctionne comme une simple calculatrice : on peut saisir une expression dont la valeur est renvoyée dès qu'on presse la touche «Entrée». Voici un exemple de résolution d'un système d'équations linéaires : [1]

```
>> A = [2 1 0; -1 2 2; 0 1 4];
>> b = [1; 2; 3];
>> soln = A\b
soln =
   0.25000
   0.50000
   0.62500
```

Ce mode interactif est très pratique pour rapidement tester des instructions et directement voir leurs résultats. Son utilisation reste néanmoins limitée à des programmes de quelques instructions. En effet, devoir à chaque fois retaper toutes les instructions s'avérera vite pénible.
Si on ferme Octave et qu'on le relance, comment faire en sorte que l'ordinateur se souvienne de ce que nous avons tapé ? On ne peut pas sauvegarder directement ce qui se trouve dans la onglet "Fenêtre de commandes", parce que cela comprendrait à la fois les commandes tapées et les réponses du système. Il faut alors avoir préalablement écrit un fichier avec uniquement les commandes qu'on a tapées et l'avoir enregistré sur l'ordinateur avec l'extension `.m`. Une fois cela fait, on demandera à Octave de lire ce fichier et exécuter son contenu, instruction par instruction, comme si on les avait tapées l'une après l'autre dans la Fenêtre de commandes. Ainsi plus tard on pourra ouvrir ce fichier et lancer Octave sans avoir à retaper toutes les commandes. Passons alors à l'onglet "Éditeur".

1.3.2 Éditeur : mode script

On voit qu'il n'y a rien dans cette nouvelle fenêtre (pas d'en-tête comme dans la "Fenêtre de commandes"). Ce qui veut dire que ce fichier est uniquement pour les commandes : Octave n'interviendra pas avec ses réponses lorsque on écrira le programme et ce tant que on ne le lui demandera pas. Ayant sauvé le programme dans un fichier avec l'**extension `.m`**, pour le faire tourner et afficher les résultats dans la "Fenêtre de commandes" il suffira d'appuyer sur la touche «F5». Si on a fait une faute de frappe, Octave le remarquera et demandera de corriger.
Maintenant qu'on a sauvé le programme, on est capable de le recharger.
Un fichier de script contient des instructions qui sont **lues et exécutées séquentiellement** par l'interpréteur d'Octave. Ce sont obligatoirement des fichiers au format texte. Copier par exemple les lignes suivantes dans un fichier appelé `first.m`

```
A = [2 1 0; -1 2 2; 0 1 4];
b = [1; 2; 3];
soln = A\b
```

Appuyer sur la touche «F5», cliquer sur "Changer de répertoire" et regarder le résultat dans l'onglet "Fenêtre de commandes". [2]

1.4 Notions de base

1.4.1 Variables et affectation

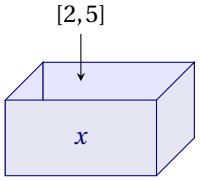
Une variable peut être vue comme une boîte représentant un emplacement en mémoire qui permet de stocker une valeur et à qui on a donné un nom afin de facilement l'identifier (boîte ← valeur) :

```
>> x=1
x =  1
```

```
>> x=[2 5]
x =
   2   5
```

```
>> x='c'
x = c
```

L'affectation `x=[2 5]` crée une association entre le nom `x` et le vecteur $[2,5]$: la boîte de nom `x` contient le vecteur $[2,5]$.



1. Ces instructions calculent la solution du système linéaire $\begin{pmatrix} 2 & 1 & 0 \\ -1 & 2 & 2 \\ 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$. Noter l'usage des points-virgules à la fin de certaines instructions du fichier : ils permettent d'éviter que les résultats de ces instructions soit affiché à l'écran pendant l'exécution du script.

2. Sinon, si ce fichier se trouve dans le répertoire courant d'Octave, pour l'exécuter on peut juste taper son nom **(sans l'extension)** sur la ligne de commande d'Octave : `>> first`
On peut aussi l'exécuter au moyen de la commande `source` qui prend en argument le nom du fichier ou son chemin d'accès (complet ou relatif au répertoire courant). Par exemple : `>> source("Bureau/TP1/first.m")`

© 2022-2023 G. Faccanoni

Il faut bien prendre garde au fait que **l'instruction d'affectation** (=) **n'a pas la même signification que le symbole d'égalité** (=) **en mathématiques** (ceci explique pourquoi l'affectation de `1` à `x`, qu'en Octave s'écrit `x = 1`, en algorithmique se note souvent $x \leftarrow 1$).
Une fois une variable initialisée, on peut modifier sa valeur en utilisant de nouveau l'opérateur d'affectation (=). La valeur actuelle de la variable est remplacée par la nouvelle valeur qu'on lui affecte. Dans l'exemple précédent, on initialise une variable à la valeur 1 et on remplace ensuite sa valeur par le vecteur $[1,2]$.
Il est très important de donner un nom clair et précis aux variables. Par exemple, avec des noms bien choisis, on comprend tout de suite ce que calcule le code suivant :

```
base = 8
hauteur = 3
aire = base * hauteur / 2
```

Octave distingue les majuscules des minuscules. Ainsi `mavariable`, `Mavariable` et `MAVARIABLE` sont des variables différentes.
Les noms de variables peuvent être non seulement des lettres, mais aussi des mots ; ils peuvent contenir des chiffres (à condition toutefois de ne pas commencer par un chiffre), ainsi que certains caractères spéciaux comme le tiret bas «_» (appelé *underscore* en anglais). Cependant, certains mots sont réservés :

`ans`	Nom pour les résultats
`eps`	Le plus petit nombre tel que `1+eps>1`
`inf`	∞
`NaN`	Not a number
`i` ou `j`	i
`pi`	π

```
>> 5/0
warning: division by zero
ans = Inf
>> 0/0
warning: division by zero
ans = NaN
>> 5*NaN   % Most operations with NaN result in NaN
ans = NaN
>> NaN==NaN   % Different NaN's are not equal!
ans = 0
>> eps
ans =    2.2204e-16
```

Si on écrit une instruction sans affectation, le résultat sera affecté à la variable **`ans`**.

```
>> [4,3]
ans =

   4   3
>> 'Ciao'
ans = Ciao
```

Pour effacer la mémoire et désaffecter toutes les variables, utiliser la fonction **`clear all`**.

1.5 Commentaires

Le symbole `%` indique le début d'un **commentaire** : tous les caractères entre `%` et la fin de la ligne sont ignorés par l'interpréteur.

- ⋆ Dans l'éditeur d'Octave, pour commenter plusieurs lignes en même temps, les sélectionner et appuyer sur les touches «Ctrl+R». Pour dé-commenter plusieurs lignes en même temps, les sélectionner et appuyer sur les touches «Ctrl+Maj+R».
- ⋆ Dans l'éditeur de Matlab, pour commenter plusieurs lignes en même temps, les sélectionner et appuyer sur les touches «Ctrl+R». Pour dé-commenter plusieurs lignes en même temps, les sélectionner et appuyer sur les touches «Ctrl+T».

1.6 Affichage

Lors de l'affectation d'une variable, le résultat de l'affectation sera affiché ; le symbole `;` supprime cet affichage.

```
>> a=[1,2]
a =

   1   2

>> a=[4,3];
>> 'Ciao'
ans = Ciao
```

Pour afficher seulement le **contenu** d'une variable utiliser la fonction `disp` (en effet, si on écrit juste le nom de la variable, on affichera aussi le nom de la variable)

```
>> a=[4,3];
>> disp(a)
   4   3
>> a
a =

   4   3
>> disp('Ciao')
Ciao
```

Pour nettoyer la fenêtre de commandes, utiliser la fonction `clc`.

1.7 Opérations arithmétiques

Dans Octave on a les opérations arithmétiques usuelles :

`+`	Addition
`-`	Soustraction
`*`	Multiplication
`/`	Division
`^`	Exponentiation

Quelques exemples :

```
>> a = 100
a =  100
>> b = 17
b =  17
```

```
>> c = a-b
c =  83
>> a/b
ans =  5.8824
```

```
>> a^b
ans =    1.0000e+34
```

Les opérateurs arithmétiques possèdent chacun une priorité qui définit dans quel ordre les opérations sont effectuées. Par exemple, lorsqu'on écrit `1 + 2 * 3`, la multiplication va se faire avant l'addition. Le calcul qui sera effectué est donc `1 + (2 * 3)`. Dans l'ordre, l'opérateur d'exponentiation est le premier exécuté, viennent ensuite les opérateurs *, /, // et %, et enfin les opérateurs + et -.
Lorsqu'une expression contient plusieurs opérations de même priorité, ils sont évalués de gauche à droite. Ainsi, lorsqu'on écrit `1 - 2 - 3`, le calcul qui sera effectué est `(1 - 2) - 3`. En cas de doutes, vous pouvez toujours utiliser des parenthèses pour rendre explicite l'ordre d'évaluation de vos expressions arithmétiques.
Il existe aussi les opérateurs augmentés (seulement dans Octave) :

`a += b`	équivaut à a = a+b
`a -= b`	équivaut à a = a-b
`a *= b`	équivaut à a = a*b
`a /= b`	équivaut à a = a/b
`a ^ = b`	équivaut à a = a^ b

1.8 Division euclidienne

Lorsqu'on divise un nombre entier D (appelé dividende) par un autre nombre entier d (appelé diviseur), on obtient deux résultats : un quotient q et un reste r, tels que $D = qd + r$ (avec $r < d$). La valeur q est le résultat de la division entière et la valeur r celui du reste de cette division. Par exemple, si on divise 17 par 5, on obtient un quotient de 3 et un reste de 2 puisque $17 = 3 \times 5 + 2$. Ces deux opérateurs sont très utilisés dans plusieurs situations précises. Par exemple, pour déterminer si un nombre entier est pair ou impair, il suffit de regarder le reste de la division entière par deux. Le nombre est pair s'il est nul et est impair s'il vaut 1. Une autre situation où ces opérateurs sont utiles concerne les calculs de temps. Si on a un nombre de

© 2022-2023 G. Faccanoni

secondes et qu'on souhaite le décomposer en minutes et secondes, il suffit de faire la division par 60. Le quotient sera le nombre de minutes et le reste le nombre de secondes restant. Par exemple, 175 secondes correspond à `175//60=2` minutes et `175%60=55` secondes.

```
>> q=fix(9/4)
q =  2
>> % Reste de la division euclidienne de 9 par 4
>> r=rem(9,4)
r =  1
>>
>> r=mod(9,4) % 9 modulo 4
r =  1
>> q=fix(175/60)
q =  2
>> r=rem(175,60)
r =  55
```

1.9 Matrices

Pour définir une matrice on doit écrire ses éléments de la première à la dernière ligne, en utilisant le caractère `;` pour séparer les lignes (ou aller à la ligne). Notons que le symbole `;` a deux fonctions : il supprime l'affichage d'un résultat intermédiaire et il sépare les lignes d'une matrice. Par exemple, la commande

```
>> A = [ 1 2 3; 4 5 6]
```

ou la commande

```
>> A = [ 1 2 3
         4 5 6]
```

donnent

```
A =
   1   2   3
   4   5   6
```

c'est-à-dire, une matrice 2×3 dont les éléments sont indiqués ci-dessus.
Un vecteur colonne est une matrice $1 \times n$, un vecteur ligne est une matrice $n \times 1$:

```
>> b = [1 2 3]
b =
   1   2   3

>> b = [1; 2; 3]
b =
   1
   2
   3
```

L'opérateur **transposition** s'obtient par la commande `'` :

```
>> b = [1 2 3]'
b =
   1
   2
   3
```

En Octave, les éléments d'une matrice sont *indexés à partir de* 1. Pour extraire les éléments d'une matrice on utilise la commande `A(i,j)` où `i` et `j` sont la ligne et la colonne respectivement. On peut extraire un sous-vecteur en déclarant l'indice i de **début (inclus)** et l'indice j de **fin (inclus)**, séparés par deux-points `v(i:j)`, ou encore un sous-vecteur en déclarant l'indice i de début (inclus), le pas k et l'indice j de fin (inclus), séparés par des deux-points `v(i:k:j)`. On peut même utiliser un pas négatif. Cette opération est connue sous le nom de *slicing* (en anglais).
On peut combiner ces opérations pour extraire des sous-matrices :

```
A(2,3)      % element  A_{23}
A(:,3)      % vecteur colonne [A_{13};...;A_{n3}]
A(1:4,3)    % [A_{13};...A_{43}] premieres 4 lignes du vecteur colonne [A_{13};...A_{n3}]
A(1,:)      % vecteur ligne [A_{11},...,A_{1n}]
```

```
A(2,3:end)  % [A_{23},...,A_{2n}] vecteur ligne

diag(A)  % vecteur colonne [A_{11};...;A_{nn}] contenant la diagonale de A
```

Voici des exemples :

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

>> A(2,3)   % Element a la ligne 2 colonne 3
ans =  7
>> A(:,2)   % Toutes les lignes, deuxieme colonne
ans =
   1
   5
   9

>> A(2:3,2:3)   % Sous-matrice 2 x 2
ans =
   5   7
   9   2

>> A(3:-1:1,:)  % les lignes de la derniere a la premiere, toutes les colonnes
ans =
   4   9   2
   3   5   7
   8   1   6
```

ATTENTION
Dans Octave les indices commencent à 1, ainsi `A(1,:)` indique la première ligne, `A(2,:)` la deuxième etc.

1.9.1 Concaténation de matrices

Nous pouvons générer une matrice par concaténation de deux ou plusieurs autres matrices :

- ⋆ Concaténation horizontale : `A = [B C]` ou `horzcat(A,B)`
- ⋆ Concaténation verticale : `A = [B ; C]` ou `vertcat(A,B)`

Dans le premier cas, on concatène côte à côte (horizontalement) les matrices $\mathbb{B}$ et $\mathbb{C}$. Dans le second, on concatène verticalement les matrices $\mathbb{B}$ et $\mathbb{C}$. Attention aux dimensions qui doivent être cohérentes : dans le premier cas toutes les matrices doivent avoir le même nombre de lignes, et dans le second cas le même nombre de colonnes.
Voici des exemples :

```
>> B = [1 2 3 ; 4 5 6]
B =

   1   2   3
   4   5   6

>> C = [7 8 9; 10 11 12]
C =

    7    8    9
   10   11   12

>> A = [B C]
A =

    1    2    3    7    8    9
    4    5    6   10   11   12

>> A = [B;C]
A =
```


© 2022-2023 G. FACCANONI

```
    1    2    3
    4    5    6
    7    8    9
   10   11   12
```

1.9.2 Matrices particulières

Construction de matrices particulières :

① La commande **zeros**(m,n) construit la matrice rectangulaire nulle $\mathbb{O}$, *i.e.* celle dont tous les éléments a_{ij} sont nuls pour $i = 1,\dots,m$ et $j = 1,\dots,n$.
La commande **zeros**(n) est un raccourci pour **zeros**(n,n).

② La commande **ones**(m,n) construit une matrice rectangulaire dont les éléments a_{ij} sont égaux à 1 pour $i = 1,\dots,m$ et $j = 1,\dots,n$.
La commande **ones**(n) est un raccourci pour **ones**(n,n).

③ La commande **eye**(m,n) renvoie une matrice rectangulaire dont les éléments valent 0 exceptés ceux de la diagonale principale qui valent 1.
La commande **eye**(n) (qui est un raccourci pour **eye**(n,n)) renvoie une matrice carrée de dimension n appelée matrice identité et notée $\mathbb{I}$.

④ La commande A=[] définit une matrice vide.

⑤ La commande **diag**(v) où **v** est un vecteur de n éléments renvoie une matrice carré de taille n dont les éléments valent 0 exceptés ceux de la diagonale principale qui valent **v**.

⑥ Soit **v** un vecteur de n composantes. La commande **diag**(v) renvoie une matrice diagonale carrée de dimension n qui contient **v** sur la diagonale principale ; la commande **diag**(v,1) renvoie une matrice carrée de dimension $n+1$ qui contient **v** sur la sur-diagonale principale etc.

Notons que **diag**(**ones**(1,4)) équivaut à **eye**(4).

```
>> Z=zeros(2,3)
Z =
   0   0   0
   0   0   0
```

```
>> O=ones(3,2)
O =
   1   1
   1   1
   1   1
```

```
>> E=eye(2,5)
E =
Diagonal Matrix
   1   0   0
       0   0
   0   1   0
       0   0
```

```
>> A=[]
A = [](0x0)
```

```
>> v=[1 2 3]
v =
   1   2   3
```

```
>> F=diag(v)
F =
Diagonal Matrix
   1   0   0
   0   2   0
   0   0   3
```

```
>> G=diag(v,1)
G =
   0   1   0
       0
   0   0   2
       0
   0   0   0
       3
   0   0   0
       0
```

Construction de vecteurs :

① x=[debut:pas:fin]

② x=**linspace**(debut,fin,N) (**x** a N points donc le pas h est $\frac{x_{\text{fin}}-x_{\text{debut}}}{N-1} = \frac{x_N-x_1}{N-1}$)

```
x = [-5 : 0.25 : 1]      % x(k)= -5 + 0.25*(k-1), tant que k<=(fin-debut)/N
y = linspace(-5, 1, 25)  % y(k)= -5 + h*(k-1), k=1,2,...,N avec h=(fin-debut)/(N-1)
```

Notons que la première instruction ne garantit pas que le dernier point soit pris, cela dépend du pas choisi (et des erreurs d'arrondis) :

```
x = [0 : 0.4 : 1]     % output : x=0.00000   0.40000   0.80000
```

Dans la première instructions on peut utiliser un pas négatif :

```
x = [1 : -0.4 : 0]     % output : x=1.00000   0.60000   0.20000
```

Dimensions :

```
A = eye(3,4);
[r,c] = size(A) % r=nb de lignes et c=nb de colonnes de A
x = [0:10];
n = length(x) % n=nb d'elements de x
n = numel(x)  % n=nb d'elements de x
```

1.9.3 Opérations entre matrices

Opérations sur les matrices (lorsque les dimensions sont compatibles) :

- ⋆ Somme $\mathbb{C} = \mathbb{A} + \mathbb{B}$, *i.e.* $\mathbb{C}_{ij} = \mathbb{A}_{ij} + \mathbb{B}_{ij}$: `C=A+B`
- ⋆ Produit $\mathbb{C} = \mathbb{A}\mathbb{B}$, *i.e.* $\mathbb{C}_{ij} = \sum_{k=1} \mathbb{A}_{ik} + \mathbb{B}_{kj}$: `C=A*B` NB il s'agit du **produit matriciel**!
- ⋆ Division à droite $\mathbb{C} = \mathbb{A}\mathbb{B}^{-1}$: `C=A/B`
- ⋆ Division à gauche $\mathbb{C} = \mathbb{A}^{-1}\mathbb{B}$: `C=A\B` (si $\mathbb{B}$ est un vecteur colonne alors $\mathbb{C}$ est un vecteur colonne **solution du système linéaire** $\mathbb{A}\mathbb{C} = \mathbb{B}$)
- ⋆ Élévation à la puissance $\mathbb{C} = \mathbb{A}\mathbb{A}\mathbb{A}$: `C=A^3`
- ⋆ Calcul du déterminant (si la matrice est carrée) : `det(A)`
- ⋆ Calcul de la matrice inverse (si la matrice est inversible) : `inv(A)`

```
>> A=[1 2 3; 4 5 6]
A =
   1   2   3
   4   5   6
```

```
>> B=ones(2,3)
B =
   1   1   1
   1   1   1
```

```
>> C=[1 2; 3 4; 5
    6]
C =
   1   2
   3   4
   5   6
```

```
>> D=eye(3,2)
D =
Diagonal Matrix
   1   0
   0   1
   0   0
```

```
>> E=A(1:2,1:2)
E =
   1   2
   4   5
```

```
>> A+B
ans =
   2   3   4
   5   6   7

>> A*C
ans =
   22   28
   49   64

>> A/B
ans =
   1.00000   1.00000
   2.50000   2.50000

>> b=[28;64]
b =
   28
   64

>> A\b
ans =
   2.0000
   4.0000
   6.0000

>> A\B
ans =
  -5.0000e-01  -5.0000e-01  -5.0000e-01
   8.3267e-17   8.3267e-17   8.3267e-17
   5.0000e-01   5.0000e-01   5.0000e-01

>> E^2
ans =
    9   12
   24   33
```

Quand on tente d'effectuer des opérations entre matrices de dimensions incompatibles on obtient un message d'erreur.

```
>> A+C
error: operator +: nonconformant arguments (op1 is 2x3, op2 is 3x2)
```


© 2022-2023 G. Faccanoni

1.9.4 ♡ Opérations pointées ♡

Quand il s'agit des opérations impliquant des multiplication (donc le produit mais aussi la division et l'élévation à la puissance), la multiplication de deux matrices, avec les notations habituelles, ne signifie pas la multiplication élément par élément mais la multiplication au sens mathématique du produit matriciel. C'est pour cela qu'Octave utilise deux opérateurs distincts pour représenter la multiplication matricielle `*` et la multiplication élément par élément `.*` **Le point placé avant l'opérateur indique que l'opération est effectuée élément par élément.** Les autres opérations de ce type sont la division à droite et l'élévation à la puissance :

- ⋆ Produit $\mathbb{C}_{ij} = \mathbb{A}_{ij}\mathbb{B}_{ij}$: `C=A.*B` NB il s'agit du produit d'Hadamard et non pas du produit matriciel
- ⋆ Division $\mathbb{C}_{ij} = \mathbb{A}_{ij}/\mathbb{B}_{ij}$: `C=A./B`
- ⋆ Élévation à la puissance $\mathbb{C}_{ij} = \mathbb{A}^3_{ij}$: `C=A.^3`

Ces opérations sont à la base de la "**vectorisation**" car elles permettent le **remplacement d'une boucle par une opération matricielle pointée** qui est généralement beaucoup plus performante.

```
>> A=[1 2 3; 4 5 6]
A =
   1   2   3
   4   5   6
>> B=[0 2 0; 0 0 1]
B =
   0   2   0
   0   0   1
>> A.*B
ans =
   0   4   0
   0   0   6
>> A*B
error: operator *: nonconformant arguments (op1 is 2x3, op2 is 2x3)
```

1.9.5 Opérateurs de comparaison et connecteurs logiques

Les opérateurs de comparaison renvoient `1` si la condition est vérifiée, `0` sinon. Ces opérateurs sont

On écrit	<	>	<=	>=	==	~=
Ça signifie	$<$	$>$	$\leq$	$\geq$	$=$	$\neq$

Bien distinguer l'instruction d'affectation = du symbole de comparaison ==.

ATTENTION

Les opérateurs de comparaison agissent élément par élément, ainsi lorsqu'on les applique à une matrice le résultat est une matrice qui contient que des `0` ou `1` (parfois appelée "**masque**"). Par exemple

```
>> A = [1 2 3; 4 -5 6]; B = [7 8 9; 0 1 2];
>> A>B
ans =
   0   0   0
   1   0   1
```

On peut utiliser un masque pour remplacer seulement les éléments qui satisfont une conditions :

```
>> A = [1 -2 3; 4 -5 6];
>> A(A<0)=-100
A =
     1  -100     3
     4  -100     6
```

Les masques sont à la base de la "vectorisation" car permettent le remplacement d'un boucle avec des conditions par une opération matricielle qui est généralement beaucoup plus performante.

EXEMPLE (MASQUE)

Étant donné trois vecteurs :

- ⋆ `hotels` : une liste de noms d'hôtels
- ⋆ `ratings` : leurs notes dans une ville
- ⋆ `cutoff` : la note minimale

on souhaite afficher les noms des hôtels ayant une note supérieure ou égale au seuil.

```
>> hotels =["CityLights";"SeaView";"MarketPlace";"ResortSpa";"Nightingale";"Clubadub";"SkylineView
    ";"MarinaBay";"ComfortFirst";"VillageValley"]; % vecteur colonne
>> ratings = [7.2;8.7;6.5;9.3;4.3;6.9;8.8;5.9;7.4;9.1]; % vecteur colonne
>> cutoff = 8;
>> good = hotels(ratings>=cutoff,:) % NB ":" pour selectionner toute la chaine de caracteres
good =

SeaView
ResortSpa
SkylineView
VillageValley
```

Pour combiner des conditions complexes (par exemple $x > -2$ et $x^2 < 5$), on peut combiner les opérateurs de comparaison avec les connecteurs logiques :

On écrit	Ça signifie
&	et
|	ou
~	non

Par exemple

```
>> (A > B) | (B > 5)
ans =
   1   1   1
   1   0   1
```

1.10 Fonctions

1.10.1 Fonctions prédéfinies

De très nombreuses fonctions sont déjà disponibles dans Octave/Matlab. Voici quelques exemples de fonction mathématique :

```
abs(-5)

sin(pi)
cos(pi)
tan(pi)

factorial(5) % output : 120

r=rem(11,3)

round(3.7) % output : 4
round(3.3) % output : 3
round(-3.7) % output : -4
round(-3.3) % output : -3

fix(3.7) % output : 3
fix(3.3) % output : 3
fix(-3.7) % output : -3
fix(-3.3) % output : -3

floor(3.7) % output : 3
floor(3.3) % output : 3
floor(-3.7) % output : -4
floor(-3.3) % output : -4

ceil(3.7) % output : 4
ceil(3.3) % output : 4
ceil(-3.7) % output : -3
ceil(-3.3) % output : -3
```


© 2022-2023 G. Faccanoni

Remarque (Fonction vectorisée)
Une fonction vectorisée ("universelle" ou «ufunc», abréviation de *universal function*, est le terme exacte) est une fonction qui peut s'appliquer terme à terme aux éléments d'un vecteur ou d'une matrice. Si f est une ufunc et si $a = [a_1, a_2, \ldots, a_n]$ est un tableau, alors $f(a)$ renvoie le tableau $[f(a_1), f(a_2), \ldots, f(a_n)]$. En générale les fonctions prédéfinies sont vectorisées, autrement dit si on applique la fonction à une matrice, elle renvoie une matrice de la même taille en ayant appliqué la fonction à chaque élément.

```
x=[1:5]

sqrt(x)  % output 1.0000    1.4142    1.7321    2.0000    2.2361
exp(x)   % output 2.7183    7.3891    20.0855   54.5982   148.4132
log(x)   % output 0.00000   0.69315   1.09861   1.38629   1.60944
log10(x) % output 0.00000   0.30103   0.47712   0.60206   0.69897
log2(x)  % output 0.00000   1.00000   1.58496   2.00000   2.32193

ismember(2,x)  % output 1
ismember(11,x) % output 0
```

Certaines fonctions sont spécifiques aux matrices :

```
A=[1 2; 3 4];
det(A)
inv(A)
trace(A)
size(A) % dimensions de A
x=[1 2 3];
length(x) % longueur d'un vecteur
numel(x) % nombre d'elements d'un vecteur
```

1.10.2 Définition d'une fonction

On peut définir nos propres fonctions au moyen de la commande `function` et les utiliser dans plusieurs scripts.

```
function [y1,...,yN] = myfunc(x1,...,xM)
   instruction_1
   instruction_2
   ...
   [y1,...,yN] = ...
end
```

La structure type d'une fonction est la suivante :
- ⋆ on utilise la commande `function` dans laquelle on indique les arguments `(x1,...,xM)` et la valeur de retour `[y1,...,yN]`
- ⋆ cette déclaration est suivie du corps de la définition qui est un bloc d'instructions à exécuter et se termine par le mot-clé `end` ou `endfunction`

Quelques conseils :
- ⋆ Lorsque l'on définit une fonction dans un fichier, il est préférable de mettre un `;` à la fin de chaque commande constituant la fonction. Ainsi, on évitera l'affichage de résultats intermédiaires. Attention cependant à ne pas en mettre sur la première ligne.
- ⋆ Lorsque l'on définit une fonction, il est préférable d'utiliser systématiquement les opérateurs terme à terme `.*` `./` et `.^` au lieu de `*` `/` et `^` si l'on veut que cette fonction puisse s'appliquer à des scalaires, mais aussi à des tableaux.

ATTENTION
Pour éviter de surcharger une fonction déjà définie dans Matlab/Octave, prendre l'habitude d'appeler ses fonctions par `my...`

Voir aussi https://fr.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

Fichiers fonctions

Par convention, **chaque définition de fonction est stockée dans un fichier séparé qui porte le nom de la fonction** suivi de l'**extension `.m`** Ces fichiers s'appellent des fichiers de fonction. Notez que c'est la même extension que les fichiers de scripts mais, de plus, **il faut absolument que le fichier s'appelle comme la fonction qu'il contient**.
La structure type d'un fichier de fonction est la suivante :

- ⋆ toute ligne commençant par un # ou un % est considérée comme un commentaire
- ⋆ les premières lignes du fichier sont des commentaires qui décrivent la syntaxe de la fonction. Ces lignes seront affichées si on utilise la commande `help myfunc`
- ⋆ la fonction elle-même.

ATTENTION
Pour éviter toute confusion, utilisez le même nom pour le fichier de fonction et la première fonction du fichier. Matlab/Octave associe votre programme au nom du fichier, pas au nom de la fonction. Les fichiers de script ne peuvent pas avoir le même nom qu'une fonction du fichier.

Remarque (Sous-fonctions)
Un fichier de fonction peut en réalité contenir plusieurs fonctions déclarées au moyen de la commande `function` mais seule la première définition est accessible depuis un script. Les autres définitions concernent des fonctions annexes (on dit parfois des sous-fonctions) qui ne peuvent être utilisées que dans la définition de la fonction principale.

Voici un exemple qui prend en entrée un vecteur de valeurs et renvoie la moyenne et la déviation standard :

Fichier `stat.m`

```
% Cette fonction calcule la moyenne et la deviation
% standard d'un vecteur x
function [m,s] = stat(x)
   n = numel(x);
   m = sum(x)/n;
   s = sqrt(sum((x-m).^2/n));
end
```

Script

```
values = [12.7, 45.4, 98.9, 26.6, 53.1];
[average,stdeviation] = stat(values)
```

À titre d'exemple, écrivons une fonction qui calcule l'aire d'un triangle en fonction des longueurs a, b et c des côtés grâce à la formule de Héron : Aire $= \sqrt{p(p-a)(p-b)(p-c)}$ où $p = (a+b+c)/2$ est le demi-périmètre. On crée pour cela un fichier au format texte appelé `heron.m` contenant les instructions suivantes

```
% Calcule l'aire s d'un triangle par la formule de Heron.
% a, b, c sont les longueurs des aretes.
function s = heron(a, b, c)
   p = (a+b+c)/2;
   s = sqrt(p*(p-a)*(p-b)*(p-c));
endfunction
```

La définition donnée ci-dessus peut être testée directement en chargeant le fichier `heron.m` avec la commande `source` et en invoquant la fonction sur la ligne de commande. Par exemple :

```
>>  source("Bureau/TP1/heron.m")
>>  heron(3,5,4)
ans =  6
```

Fonctions locales dans un script - Matlab VS Octave

Les versions récentes de MATLAB permettent la création de fonctions directement dans un script. C'est très pratique pour de petites fonctions "outils", qui ne nécessitent pas d'être externalisées. En ajoutant des fonctions locales vous pouvez **éviter de créer et de gérer des fichiers de fonctions séparés**. Cependant, la syntaxe est différente entre Matlab et Octave.

Matlab, depuis la version R2016b. Les fonctions locales peuvent apparaître dans n'importe quel ordre mais doivent être placées **à la fin du fichier, après le code du script**. Voici un exemple :

```
x = 1:10;
n = numel(x);
avg = mymean(x,n)
med = mymedian(x,n)

function a = mymean(v,n)
% MYMEAN Local function that calculates mean of array.
    a = sum(v)/n;
end

function m = mymedian(v,n)
% MYMEDIAN Local function that calculates median of array.
```


© 2022-2023 G. FACCANONI

```
    w = sort(v);
    if rem(n,2) == 1
        m = w((n + 1)/2);
    else
        m = (w(n/2) + w(n/2 + 1))/2;
    end
end
```

Octave. Les fonctions locales peuvent apparaître dans n'importe quel ordre mais doivent être placées **avant le code du script et après l'instruction** `1;` Voici un exemple :

```
% Prevent Octave from thinking that this is a function file:
1;

function a = mymean(v,n)
  % MYMEAN Local function that calculates mean of array.
  a = sum(v)/n;
end

function m = mymedian(v,n)
% MYMEDIAN Local function that calculates median of array.
    w = sort(v);
    if rem(n,2) == 1
        m = w((n + 1)/2);
    else
        m = (w(n/2) + w(n/2 + 1))/2;
    end
end

x = 1:10;
n = numel(x);
avg = mymean(x,n)
med = mymedian(x,n)
```

1.10.3 Fonctions anonymes (*lambda functions*)

Les fonctions anonymes permettent de définir une fonction directement dans le script, à condition que la fonction se compose d'une seule instruction. La syntaxe usuelle d'une fonction anonyme est

```
fun = @(arg1, arg2,...,argn) [expr] ; % crochets facultatifs si une seule expression
```

Nous utiliserons les fonctions anonymes surtout pour **écrire directement la fonction dans le fichier de script sans créer un fichier séparé en étant compatibles à la fois avec Matlab et avec Octave.**
Une application courante des fonctions anonymes consiste à définir une expression mathématique.

```
f = @(x) 2*x ; % equivaut a definir f(x)=2x
f(2)           % on evalue f(2) et on obtient 4
```

Cela permet entre autre de calculer rapidement une solution approchée d'une équation :

```
% on veut resoudre x=cos(x)
f = @(x) x.^2-2 ; % on pose f(x)=x^2-2
% fsolve( fct dont on cherche un zero ,  un point pas trop eloigne de la solution )
fsolve( f, 1 )
fsolve( f,-1 )
```

La contrainte d'utiliser une seule instruction n'empêche pas de calculer plusieurs résultats (car on peut renvoyer un vecteur) ni d'écrire des boucles (grâce à l'utilisation des instructions pointées) ni des conditions (grâce à l'utilisation de masques, par exemple pour la définition d'une fonction par morceaux, comme on verra à la page 22).

1.10.4 Arrayfun (listes de compréhension)

Les listes de compréhension sont une fonctionnalité puissante de Python qui permet de créer de nouvelles listes en appliquant une certaine expression à chaque élément d'une séquence (comme une liste, un tuple ou une chaîne de caractères). La syntaxe générale est

```
[ f(x) for x in xx ]
```

où $f(x)$ est l'évaluation d'une fonction f en chaque élément x de la séquence `xx`.
De manière analogue, en Matlab/Octave on peut utiliser `arrayfun` qui permet d'appliquer une fonction à chaque élément d'un tableau. La syntaxe générale est

```
arrayfun( f, xx );
```

Exemples :

Python

```
f = lambda x : x**2
xx = [1,3,10]
[ f(x) for x in xx ]
```

Octave/Matlab

```
f = @(x) x^2;
xx = [1,3,10];
arrayfun( f, xx )
```

Bien-sur dans cet exemple simplifié on peut vectoriser directement f :

```
f = @(x) x.^2;
xx = [1,3,10];
f(xx)
```

1.11 Graphes de fonctions $\mathbb{R} \to \mathbb{R}$

Pour tracer le graphe d'une fonction $f : [a,b] \to \mathbb{R}$, il faut tout d'abord générer une liste de points x_i où évaluer la fonction f, puis la liste des valeurs $f(x_i)$ et enfin, avec la fonction `plot`, Octave reliera entre eux les points $(x_i, f(x_i))$ par des segments. Plus les points sont nombreux, plus le graphe est proche du graphe de la fonction f.
Pour générer les points x_i on peut utiliser

- ⋆ soit l'instruction `linspace(a,b,n)` qui construit la liste de n éléments

$$[a, a+h, a+2h, \ldots, b = a+nh] \quad \text{avec } h = \frac{b-a}{n-1}$$

```
x=linspace(1,5,5)
x =
   1   2   3   4   5
```

- ⋆ soit l'instruction `[a:h:b]` qui construit la liste de $n = E(\frac{b-a}{h}) + 1$ éléments

$$[a, a+h, a+2h, \ldots, a+nh]$$

Dans ce cas, attention au dernier terme : b peut ne pas être pris en compte.

Voici un exemple avec une sinusoïde (en utilisant la fonction prédéfinie sin) :

```
x = linspace(-5,5,101); # x = [-5:0.1:5] with 101 elements
y = sin(x); # operation is broadcasted to all elements of the array
plot(x,y)
```

On obtient une courbe sur laquelle on peut zoomer, modifier les marges et sauvegarder dans différents formats.

Si la fonction n'est pas prédéfinie, il est bonne pratique de la définir pour qu'elle opère composante par composante lorsqu'on lui passe un vecteur ou une matrice. Les opérations `/`, $*$ et `\^` agissant sur elle doivent être remplacées par les opérations point correspondantes `./`, `.*` et `\.^` qui opèrent composante par composante.
Par exemple, on se propose de tracer la fonction

$$\begin{aligned} f : [-2;2] &\to \mathbb{R} \\ x &\mapsto \frac{1}{1+x^2} \end{aligned}$$

Suivant la façon de définir la fonction, on pourra utiliser l'une des trois méthodes suivantes.

Méthode 1. En utilisant une **fonction anonyme**.
Dans un script ou dans la `prompt` on écrit les instructions suivantes :

```
f=@(x)[1./(1+x.^2)] % declaration de la fonction
x=[-2:0.5:2];
y=f(x); % evaluation en plusieurs points
plot(x,y) % affichage des points (x_i,y_i)
```

Méthode 2. En utilisant une **fonction locale**.
Dans un script on écrit les instructions suivantes (attention : syntaxe différente selon qu'on utilise Matlab ou Octave) :

© 2022-2023 G. Faccanoni

Matlab

```
x=[-2:0.5:2];
y=f(x); % evaluation en plusieurs points
plot(x,y) % affichage des points (x_i,y_i)

function y=f(x)
   y=1./(1+x.^2);
end
```

Octave

```
1;

function y=f(x)
   y=1./(1+x.^2);
end

x=[-2:0.5:2];
y=f(x); % evaluation en plusieurs points
plot(x,y) % affichage des points (x_i,y_i)
```

Méthode 3. En utilisant un **fichier fonction**.
On utilise deux fichiers :

3.1. Dans le fichier `f.m` on écrit la fonction informatique suivante

```
function y=f(x)
   y=1./(1+x.^2);
end
```

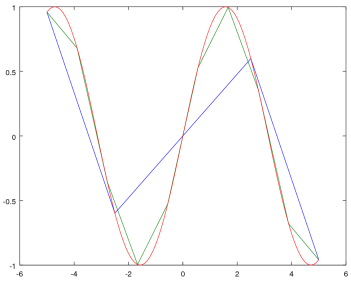
3.2. Dans un script ou dans la `prompt` on écrit

```
x=[-2:0.5:2];
y=f(x); % evaluation en plusieurs points
plot(x,y) % affichage des points (x_i,y_i)
```

1.11.1 Plusieurs courbes sur le même repère

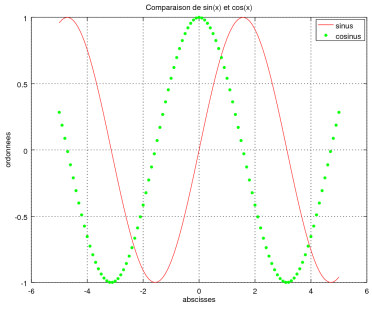
On peut **tracer plusieurs courbes sur le même repère**. Par défaut, MATLAB/Octave efface la figure avant chaque commande de traçage **`plot`**. Vous pouvez tracer plusieurs lignes soit en écrivant plusieurs courbes dans une même instruction **`plot`** ou à l'aide de la commande de mise en attente **`hold on`**. Tant que vous n'utilisez pas de suspension (**`hold off`**) ou que vous ne fermez pas la fenêtre, tous les tracés apparaissent dans la fenêtre de la figure actuelle.
Par exemple, dans la figure suivante, on a tracé la même fonction : la courbe bleu correspond à la grille la plus grossière, la courbe rouge correspond à la grille la plus fine :

```
a = linspace(-5,5,5); % a = [-5,-3,-1,1,3,5]
fa = sin(a);
b = linspace(-5,5,10); % b = [-5,-4,-3,...,5]
fb = sin(b);
c = linspace(-5,5,101); % c = [-5,-4.9,-4.8,...,5]
fc = sin(c);
plot(a,fa,b,fb,c,fc)
% la derniere ligne peut etre remplacee par
% hold on
% plot(a,fa)
% plot(b,fb)
% plot(c,fc)
% hold off
```



On peut spécifier la couleur et le type de trait, changer les étiquettes des axes, donner un titre, ajouter une grille, une légende etc.
Par exemple, dans le code ci-dessous `"r-"` indique que la première courbe est à tracer en rouge (red) avec un trait continu, et `"g."` que la deuxième est à tracer en vert (green) avec des points.

```
x = linspace(-5,5,101); # x = [-5,-4.9,-4.8,...,5] with
    101 elements
y1 = sin(x); # operation is broadcasted to all elements
    of the array
y2 = cos(x);
plot(x,y1,"r-",x,y2,"g.")
legend(['sinus';'cosinus'])
xlabel('abscisses')
ylabel('ordonnees')
title('Comparaison de sin(x) et cos(x)')
grid
```



Voir la table 1.1 et la documentation de `Matlab` pour connaître les autres options.

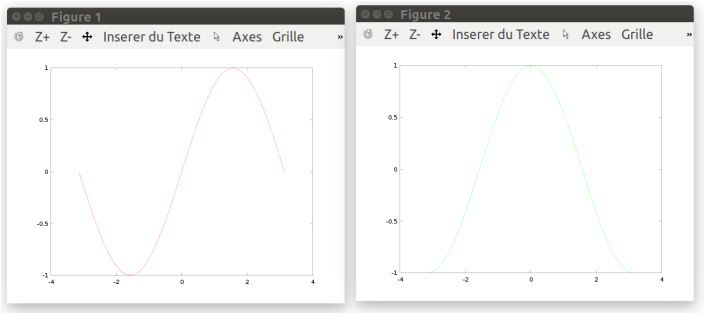
	linestyle=		color=		marker=
-	solid line	r	red	.	points
-	dashed line	g	green	,	pixel
:	dotted line	b	blue	o	filled circles
-.	dash-dot line	c	cyan	v	triangle down
		m	magenta	^	triangle up
		y	yellow	>	triangle right
		w	white	<	triangle left symbols
		k	black	*	star
				+	plus
				s	square
				p	pentagon
				x	x
				X	x filled
				d	thin diamond
				D	diamond

TABLE 1.1 – Quelques options de `plot`

1.11.2 Plusieurs "fenêtres" graphiques

Avec `figure()` on génère une nouvelle fenêtres graphique :

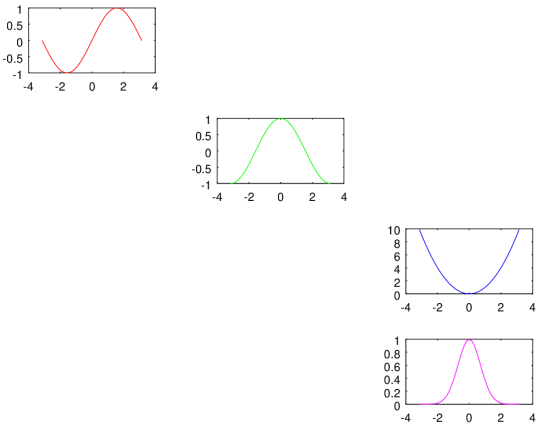
```
x = [-pi:0.05*pi:pi];
figure(1)
plot(x, sin(x), 'r')
figure(2)
plot(x, cos(x), 'g')
```



1.11.3 Plusieurs repères dans la même fenêtre

La fonction `subplot(x,y,z)` subdivise la fenêtre sous forme d'une matrice de x lignes et y colonnes et chaque case est numérotée, z étant le numéro de la case où afficher le graphe. La numérotation se fait de gauche à droite, puis de haut en bas, en commençant par 1.

```
x = [-pi:0.05*pi:pi];

subplot(4,3,1)
plot(x, sin(x), 'r')

subplot(4,3,5)
plot(x, cos(x), 'g')

subplot(4,3,9)
plot(x, x.*x, 'b')

subplot(4,3,12)
plot(x, exp(-x.*x), 'm')
```



EXEMPLE
Dans le code suivant on voit comment tracer plusieurs courbes dans un même graphique (avec légende), plusieurs graphe sur une même figure et plusieurs figures.

© 2022-2023 G. FACCANONI

```
figure(1)
x=[-2:0.5:2];

subplot(2,3,2)
  plot(x,x,'r-',x,exp(x),'b*-')
  legend(['y=x';'y=e^x'])
subplot(2,3,4)
  plot(x,x.^2)
  title('y=x^2')
subplot(2,3,5)
  plot(x,x.^3)
  xlabel('Axe x')
subplot(2,3,6)
  plot(x,sqrt(x))

figure(2)
x=linspace(0.1,exp(2),100);
plot(x,log(x));
```

1.12 Structures itératives

Les structures de répétition se classent en deux catégories : les *répétitions inconditionnelles* pour lesquelles le bloc d'instructions est à répéter un nombre donné de fois et les *répétitions conditionnelles* pour lesquelles le bloc d'instructions est à répéter autant de fois qu'une condition est vérifiée.

1.12.1 Répétition `for`

Lorsque l'on souhaite répéter un bloc d'instructions un nombre déterminé de fois, on peut utiliser un *compteur actif*, c'est-à-dire une variable qui compte le nombre de répétitions et conditionne la sortie de la boucle.
La syntaxe de la commande `for` est schématiquement

```
for var = expression
   instruction_1
   instruction_2
end
```

`expression` peut être un vecteur ou une matrice. Par exemple, le code suivant calcule les 12 premières valeurs de la suite de Fibonacci définie par la relation de récurrence $u_n = u_{n-1} + u_{n-2}$ avec pour valeurs initiales $u_1 = u_2 = 1$:

```
n = 12;
u = ones(1, n); # allocation
for i = 3:n
   u(i) = u(i-1)+u(i-2);
end
disp(u)
```

```
n = 12;
u = [1,1];
for i = 3:n
   u = [ u , u(end)+u(end-1) ]; # concatenation
end
disp(u)
```

Dans les deux cas le résultat affiché est

```
1     1     2     3     5     8    13    21    34    55    89   144
```

Dans l'exemple suivant on calcul la somme des n premiers entiers (et on vérifie qu'on a bien $n(n+1)/2$) :

```
n=100;
s=0;
for i=1:n
   s += i;
end
s
n*(n+1)/2
```

Bien sur, dans ce cas il est préférable d'écrire

```
sum(1:100)
```

Il est possible d'imbriquer des boucles, c'est-à-dire que dans le bloc d'une boucle, on utilise une nouvelle boucle.

```
for x = [10,20,30,40,50] % for x = 10:10:50
   for y=[3,7]
      disp(x+y)
   end
end
```

Dans ce petit programme x vaut d'abord 10, y prend la valeur 3 puis la valeur 7 (le programme affiche donc d'abord 13, puis 17). Ensuite $x = 20$ et y vaut de nouveau 3 puis 7 (le programme affiche donc ensuite 23, puis 27).
Voir aussi https://fr.mathworks.com/help/matlab/ref/for.html

1.12.2 Boucle `while` : répétition conditionnelle

`While` est la traduction de "tant que…". Concrètement, la boucle s'exécutera tant qu'une condition est remplie (donc tant qu'elle renverra la valeur `1`). Le constructeur `while` a la forme générale suivante :

```
i = ...
while condition(i)
   instruction_1
   instruction_2
   i = ...
end
```

où `condition` représente des ensembles d'instructions dont la valeur est `1` ou `0`. Tant que la condition `condition` a la valeur `1`, on exécute les instructions `instruction_i`.

Attention
Si la condition ne devient jamais fausse, le bloc d'instructions est répété indéfiniment et le programme ne se termine pas.

Voici un exemple pour créer la liste $\left[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right]$:

```
nMax = 4;
n = 1;
a = [];
while n<=nMax
   a=[a,1/n]; # Append element to list
   n += 1; % si Matlab: n=n+1
end
disp(a)
```

Dans l'exemple suivant on calcul la somme des n premiers entiers tant que la somme ne dépasse pas 100 :

```
s=0;
n=0;
while s<100
   n += 1; % si Matlab: n=n+1
   s += n; % si Matlab: s=s+n
end
disp(n-1)
disp(s-n)
% En effet avec le dernier n on depasse 100
```

1.12.3 Vectorisation, *i.e.* optimisation des performances

La plupart du temps on manipule des vecteurs et des matrices. Les opérateurs et les fonctions élémentaires sont conçus pour favoriser ce type de manipulation et, de manière plus générale, pour permettre la vectorisation des programmes. Certes, le langage Octave contient des instructions conditionnelles, des boucles et la programmation récursive, mais la vectorisation permet de limiter le recours à ces fonctionnalités qui ne sont jamais très efficaces dans le cas d'un langage interprété. Les surcoûts d'interprétation peuvent être très pénalisants par rapport à ce que ferait un programme C ou FORTRAN compilé lorsque l'on effectue des calculs numériques. Il faut donc veiller à réduire autant que possible le travail d'interprétation en vectorisant les programmes.
Dans les exemples suivant, la fonction `tic` s'utilise avec la fonction `toc` pour mesurer le temps écoulé. La fonction `tic` enregistre l'heure actuelle et la fonction `toc` utilise la valeur enregistrée pour calculer le temps écoulé.

Exemple
Quasiment toutes les fonctions prédéfinies sont vectorisées. Voici un exemple avec la fonction sin.

© 2022-2023 G. Faccanoni

Le code

```
n = 100000;
xx = linspace(0,2*pi,n);
yy = zeros(numel(xx));
tic
for i = 1:n
   yy(i) = sin(xx(i));
end;
toc
```

est significativement plus lent que

```
n = 100000;
xx = linspace(0,2*pi,n);
tic
yy = sin(xx);
toc
```

EXEMPLE
Pour calculer $\sum_{n=1}^{10000} \frac{1}{n^2}$, on peut utiliser les trois codes suivants, le deuxième étant significativement plus rapide :

```
n=1:10^7;

tic
s=0;
for i = n,
   s+=1/i^2; % si Matlab: s=s+1/
       i^2;
end;
s
toc
```

```
n=1:10^7;

tic
s=sum(1./n.^2)
toc
```

```
n=1:10^7;

tic
s=(1./n)*(1./n)'
toc
```

Un exemple de sorties obtenues :
- ⋆ avec le premier script : `Elapsed time is 12.7138 seconds.`
- ⋆ avec le deuxième script : `Elapsed time is 0.116321 seconds.`
- ⋆ avec le troisième script : `Elapsed time is 0.098047 seconds.`

1.13 Structure conditionnelle

Supposons vouloir calculer la valeur $y = f(x)$ d'un nombre x selon la règle suivante :

$$f(x) = \begin{cases} x & \text{si } x \le -5, \\ 100 & \text{si } -5 < x \le 0, \\ x^2 & \text{si } 0 < x < 10, \\ x-2 & \text{sinon.} \end{cases}$$

On a besoin d'une instruction qui opère une disjonction de cas. En Octave il s'agit de l'instruction de choix introduite par le mot-clé `if`. La syntaxe complète est la suivante :

```
if condition_1
   instruction_1.1
   instruction_1.2
   ...
elseif condition_2
   instruction_2.1
   instruction_2.2
   ...
...
else
   instruction_n.1
   instruction_n.2
   ...
end
```

où `condition_1`, `condition_2`... représentent des ensembles d'instructions dont la valeur est `1` ou `0` (on les obtient en général en utilisant les opérateurs de comparaison). La première condition `condition_i` ayant la valeur `1` entraîne l'exécution des instructions `instruction_i.1`, `instruction_i.2`... Si toutes les conditions sont `0`, les instructions `instruction_n.1`, `instruction_n.2`... sont exécutées. Les blocs `elseif` et `else` sont optionnels.

Pour l'exemple donné, la fonction (vectorisée) peut s'écrire comme suit :

```
function y=f1(x)
   n=numel(x);
   for i=1:n
      if x(i)<=-5
         y(i)=x(i);
      elseif x(i)<=0
         y(i)=100;
      elseif x(i)<10
         y(i)=x(i)^2;
      else
         y(i)=x(i)-2;
      end
   end
end
```

La même fonction peut aussi s'écrire comme suit :

```
f2 = @(x) [  x.*(x<=-5) + 100.*(x>-5).*(x<=0) + (x.^2).*(x>0).*(x<10) + (x-2).*(x>=10) ];
```

Pour vérifier qu'on a bien la même fonction on compare le graphe des deux fonctions :

```
xx=[-7:0.1:12];
yy1=f1(xx);
yy2=f2(xx);
plot(xx,yy1,'r',xx,yy2,'b.')
```

Voici un exemple pour établir si un nombre est positif

```
a=-1.5;

if a < 0
   disp('negative')
elseif a > 0
   disp('positive')
else
   disp(sign = 'zero')
end
```

Voici un exemple pour établir si un nombre est compris entre deux valeurs :

```
x = 10;
minVal = 2;
maxVal = 6;

if (x >= minVal) && (x <= maxVal) % equivaut a (x >= minVal) .* (x <= maxVal)
    disp('Value within specified range.')
elseif (x > maxVal)
    disp('Value exceeds maximum value.')
else
    disp('Value is below minimum value.')
end
```

Voir aussi https://fr.mathworks.com/help/matlab/ref/if.html

EXEMPLE

Étant donné deux matrices d'entrée $\mathbb{A}$ et $\mathbb{B}$, vérifier si on peut calculer le produit $\mathbb{AB}$. Si c'est le cas, créez une matrice $\mathbb{C}$ qui contient le produit $\mathbb{AB}$, sinon, $\mathbb{C}$ doit contenir une chaîne de caractère contenant un message d'erreur.

```
function C = in_prod(A,B)
  [rA,cA]=size(A);
  [rB,cB]=size(B);
  if cA==rB
      C = A*B;
  else
      C = "Have you checked the inner dimensions?"
  end
end

# TESTS
C=in_prod([1 2],[2;3])
```


© 2022-2023 G. FACCANONI

```
C=in_prod(-5,100)
C=in_prod([1 2;3 4],[5;6])
C=in_prod([1 2 3; 4 5 6],[2 5;3 6])
```

1.14 Exercices

Pensez à placer la commande `clear all` au début de vos scripts, de manière à nettoyer l'environnement de travail (cela effacera toutes les variables en mémoire). Vous pouvez aussi utiliser la commande `clc` pour nettoyer la fenêtre de commandes.

Pour chaque exercice, on écrira les instructions dans un fichier script nommé par exemple `exo_1_3.m` (sans espaces, ni accents ni points sauf pour l'extension `.m`). On pourra bien-sûr utiliser le mode interactif pour simplement vérifier une commande mais chaque exercice devra in fine être résolu dans un fichier script. Tous ces scripts devront se trouver dans un dossier dont le nom ne contient ni espaces, ni accents, ni points.

Exercice 1.1

Soit les matrices

$$\mathbb{A}=\begin{pmatrix}1&2&3\\-1&0&1\\0&1&0\end{pmatrix} \qquad \mathbb{B}=\begin{pmatrix}2&-1&0\\-1&0&1\end{pmatrix} \qquad \mathbf{u}=\begin{pmatrix}1\\x\\x^2\end{pmatrix} \qquad \mathbf{v}=\begin{pmatrix}1\\0\\1\end{pmatrix}$$

1. Calculer tous les produits possibles à partir de $\mathbb{A}$, $\mathbb{B}$, $\mathbf{u}$ et $\mathbf{v}$.
2. Calculer $(\mathbb{A}-\mathbb{I})^7$ et en extraire le coefficient en position $(2,3)$.
3. Calculer $\mathbb{A}^{-1}$ et la trace de $\mathbb{A}^{-1}$ (*i.e.* la somme des coefficients sur la diagonale).

Correction

Sans utiliser un module spécifique, il n'est pas possible de faire des calculs formels avec MATLAB/Octave, donc on ne peut pas utiliser $\mathbf{u}$ sans donner une valeur numérique à x.

```
A=[1 2 3; -1 0 1; 0 1 0] % 3x3
B=[2 -1 0; -1 0 1]       % 2x3
v=[1;0;1]                % 3x1
% Les produits possibles sont
B*A % (2x3)*(3x3) -> 2x3
A*v % (3x3)*(3x1) -> 3x1
B*v % (2x3)*(3x1) -> 2x1
%
Id=eye(3)
D=(A-Id)^7 % c'est bien ^7 (produit matriciel) et non .^7
D(2,3)     % -> 153
%
invA=A^(-1) % ou inv(A)
sum(diag(invA))
```

Exercice 1.2 (Vectorisation - `if`)

Considérons la fonction $f:\mathbb{R}\to\mathbb{R}$ définie par

$$f(x)=\begin{cases}0 & \text{si } x<0,\\ x & \text{si } 0\le x<1,\\ 2-x & \text{si } 1\le x\le 2,\\ 0 & \text{si } x>2.\end{cases}$$

Écrire cette fonction de deux manières différentes et en afficher le graphe :

1. avec une instruction conditionnelle du type `if ... elseif ... else` (vectorisée) dans une fonction définie avec `function`,
2. avec une instruction conditionnelle vectorisée dans une fonction anonyme.

Correction

Avec function La fonction vectorisée peut s'écrire comme suit :

```
function y=f1(x)
   n=numel(x);
   for i=1:n
```


© 2022-2023 G. Faccanoni

```
      if x(i)<0
            y(i)=0;
      elseif x(i)<1
         y(i)=x(i);
      elseif x(i)<=2
         y(i)=2-x(i);
      else
         y(i)=0;
      end
   end
end
```

Pour la tester, il fat d'abord la **sauvegarder dans un fichier qui porte le même nom** (ici le fichier devra s'appeler `f1.m`) puis on écrira **un autre fichier qui contient le script** (qui se trouve dans le même dossier) **pour la tester** :

```
xx=[-1:0.1:3];
yy1=f1(xx);
plot(xx,yy1,'r')
```

Pour écrire **un script qui contient directement cette fonction** (sans utiliser deux fichiers séparés, l'un contenant la fonction et l'autre le script), **il faut décider s'il sera exécuté avec Matlab ou avec Octave** :

Matlab

```
xx=[-1:0.1:3];
yy1=f1(xx);
plot(xx,yy1,'r')

function y=f1(x)
   n=numel(x);
   for i=1:n
      if x(i)<0
            y(i)=0;
      elseif x(i)<1
         y(i)=x(i);
      elseif x(i)<=2
         y(i)=2-x(i);
      else
         y(i)=0;
      end
   end
end
```

Octave

```
1;

function y=f1(x)
   n=numel(x);
   for i=1:n
      if x(i)<0
            y(i)=0;
      elseif x(i)<1
         y(i)=x(i);
      elseif x(i)<=2
         y(i)=2-x(i);
      else
         y(i)=0;
      end
   end
end

xx=[-1:0.1:3];
yy1=f1(xx);
plot(xx,yy1,'r')
```

Anonyme La fonction anonyme vectorisée peut s'écrire comme suit :

```
f2 = @(x) [  0*(x<0) + x.*(x>=0).*(x<1) + (2-x).*(x>=1).*(x<2) + 0.*(x>2) ];
```

Cette fonction peut être écrite directement avec le script et on a alors **le même script en Octave ou en Matlab** :

```
f2 = @(x) [  0*(x<0) + x.*(x>=0).*(x<1) + (2-x).*(x>=1).*(x<2) + 0.*(x>2) ];

xx=[-1:0.1:3];
yy2=f2(xx);
plot(xx,yy2,'b.')
```

Exercice 1.3

Soit **x** un vecteur de n valeurs. On rappelle les définitions suivantes (pour l'estimation sur une population à partir d'un échantillon) :

Moyenne arithmétique $\overline{m} = \frac{1}{n}\sum_{i=1}^{n} x_i$

Variance $V = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{m})^2$

Écart-type $\sigma = \sqrt{V}$

Médiane C'est la valeur qui divise l'échantillon en deux parties d'effectifs égaux. Soit **y** le vecteur qui contint les composantes de **x** ordonné, alors

$$\text{médiane} = \begin{cases} \dfrac{y_{n/2}+y_{n/2+1}}{2} & \text{si } n \text{ est pair,} \\ y_{n/2+1} & \text{si } n \text{ est impair.} \end{cases}$$

Écrire une fonction qui renvoie la moyenne, l'écart type et la médiane d'un vecteur donné en entré. Vérifier l'implémentation sur un vecteur au choix en comparant avec les valeurs obtenues par les fonctions prédéfinies.

Correction
Dans cette correction on utilise deux fichiers séparés, l'un contenant la fonction et l'autre le script. On peut écrire la fonction et le script dans un même fichier comme indiqué dans l'exercice précédent mais il faudra décider en amont si on exécutera le script avec Matlab ou Octave.

'Fichier stat.m'

```
function [moy,et,med]  =  stat(x)
   n  =  length(x);
   % MOYENNE  =  fonction predefinie mean(x)
   moy  =  sum(x)/n;
   % ECART_TYPE  =  fonction predefinie std(x)
   et  =  sqrt(sum((x-moy).^2)/(n-1));
   % MEDIANE  =  fonction predefinie median(x)
   y = sort(x);
   if (rem(n,2) =  = 0) % si n est un nombre pair
      med  =  (y(n/2)+y((n/2)+1))/2;
   else
      med  =  y((n+1)/2);
   end
end
```

'Script'

```
clear all

x = [0 0 8 1 1 2 2]
[my_moy,my_std,my_mediane]  =  stat(x)
octave_moy = mean(x)
octave_std = std(x)
octave_mediane = median(x)

x = [2 3 3 2]
[my_moy,my_std,my_mediane]  =  stat(x)
octave_moy = mean(x)
octave_std = std(x)
octave_mediane = median(x)
```

Exercice 1.4

Un dispositif fournit un signal $s(t) = A\sin(2\pi t+\varphi)$ avec A et φ inconnus. On mesure le signal à deux instants (en ms) : $s(0.5) = -1.76789123$ et $s(0.6) = -2.469394443$. On posera $\alpha = A\cos(\varphi)$ et $\beta = A\sin(\varphi)$.

1. Écrire et résoudre le système d'inconnues α et β. En déduire A et φ.
2. Tracer le signal et montrer qu'il passe par les points mesurés.

Correction
Rappel : $\sin(a+b) = \sin(a)\cos(b)+\cos(a)\sin(b)$ ainsi

$$s(t) = A\sin(2\pi t+\varphi) = A\big(\sin(2\pi t)\cos(\varphi)+\cos(2\pi t)\sin(\varphi)\big) = \alpha\sin(2\pi t)+\beta\cos(2\pi t).$$

On doit donc résoudre le système linéaire :

$$\begin{cases} \alpha\sin(\pi)+\beta\cos(\pi) = -1.76789123 \\ \alpha\sin\left(\frac{6}{5}\pi\right)+\beta\cos\left(\frac{6}{5}\pi\right) = -2.469394443 \end{cases}$$

qu'on écriture matricielle s'écrit

$$\begin{pmatrix} \sin(\pi) & \cos(\pi) \\ \sin\left(\frac{6}{5}\pi\right) & \cos\left(\frac{6}{5}\pi\right) \end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix} -1.76789123 \\ -2.469394443 \end{pmatrix} \quad \textit{i.e.} \quad \begin{pmatrix} 0 & -1 \\ \sin\left(\frac{6}{5}\pi\right) & \cos\left(\frac{6}{5}\pi\right) \end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix} -1.76789123 \\ -2.469394443 \end{pmatrix}$$

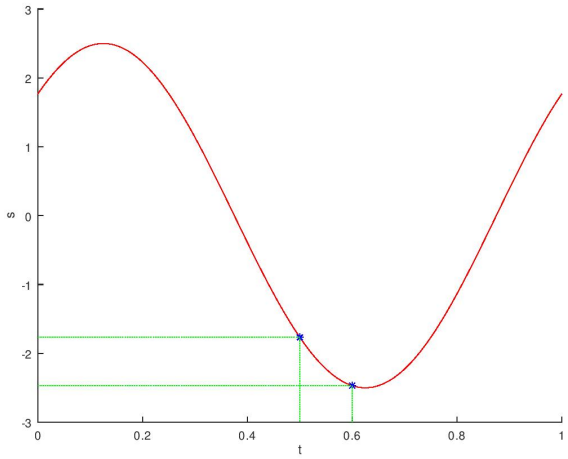
```
xx=[sin(2*pi*0.5) , cos(2*pi*0.5) ;  sin(2*pi*0.6) , cos(2*pi*0.6)]\[-1.76789123;-2.469394443]
alpha=xx(1)
beta=xx(2)
```

Puisqu'on trouve $\alpha = \beta$, alors $\cos(\varphi) = \sin(\varphi)$, *i.e.* $\varphi = \frac{\pi}{4}$ et $A = \sqrt{2}\alpha$ avec $\alpha \approx 1.7679$:

```
s=@(t)[sqrt(2)*alpha*sin(2*pi*t+pi/4)];
tt=[0:0.01:1];
hold on
```


© 2022-2023 G. Faccanoni

```
plot(tt,s(tt),'r-') % la courbe
plot([0.5 0.6],[-1.76789123;-2.469394443],'b*') % les deux points
plot([0 0.5 0.5], [-1.76789123 -1.76789123 -3], 'g:') % trait pointille
plot([0 0.6 0.6], [-2.469394443 -2.469394443 -3], 'g:') % trait pointille
xlabel('t')
ylabel('s')
hold off
print("signal.jpg") % sauvagarde de la figure en .jpg
```



Exercice 1.5 (`fsolve`, `plot`)
Soit la fonction

$$f\colon [-10,10] \to \mathbb{R}$$
$$x \mapsto \frac{x^3\cos(x)+x^2-x+1}{x^4-\sqrt{3}x^2+127}$$

1. Tracer le graphe de la fonction f en utilisant seulement les valeurs de $f(x)$ lorsque la variable x prend successivement les valeurs $-10, -9.2, -8.4, \ldots, 8.4, 9.2, 10$ (*i.e.* avec un pas 0.8).
2. Apparemment, l'équation $f(x) = 0$ a une solution α voisine de 2. En utilisant le zoom, proposer une valeur approchée de α.
3. Tracer de nouveau le graphe de f en faisant varier x avec un pas de 0.05. Ce nouveau graphe amène-t-il à corriger la valeur de α proposée?
4. Demander à Octave d'approcher α (fonction `fsolve`).

Correction
En utilisant un pas de 0.8 il semblerai que $\alpha = 1.89$. En utilisant un pas de 0.05 il semblerai que $\alpha = 1.965$. En utilisant la fonction `fsolve` on trouve $\alpha = 1.9629$.

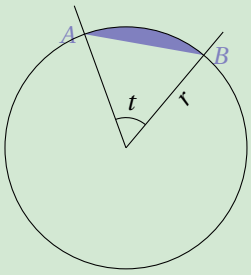
```
clear all; clc;
f  =  @(x) [(x.^3 .*cos(x)+x.^2-x+1) ./ (x.^4-sqrt(3)*x.^2+127)] ;

subplot(1,2,1)
xx = [-10:0.8:10];
plot(xx,f(xx),'r-')
grid()

subplot(1,2,2)
xx = [-10:0.05:10];
plot(xx,f(xx),'r-')
grid()

fsolve(f,1.9)
```

Exercice 1.6 (`fsolve`, `plot`)

On se propose ici d'utiliser Octave pour résoudre graphiquement des équations.

Considérons un cercle de rayon r. Si nous traçons un angle t (mesuré en radians) à partir du centre du cercle, les deux rayons formant cet angle coupent le cercle en A et B. Nous appelons a l'aire délimitée par la corde et l'arc AB (en bleu sur le dessin). Cette aire est donnée par $a = \frac{r^2}{2}(t - \sin(t))$. Pour un cercle donné (c'est à dire un rayon donné), nous choisissons une aire (partie en bleu) a. Quelle valeur de l'angle t permet d'obtenir l'aire choisie? Autrement dit, connaissant a et r, nous voulons déterminer t solution de l'équation

$$\frac{2a}{r^2} = t - \sin(t).$$



1. Résoudre graphiquement l'équation en traçant les courbes correspondant aux membres gauche et droit de l'équation (pour $a = 4$ et $r = 2$). Quelle valeur de t est solution de l'équation?
2. Comment faire pour obtenir une valeur plus précise du résultat?

Correction

```
t=[0:pi/180:2*pi];
rhs=t-sin(t);
a=4;
r=2;
lhs=2*a/(r^2)*ones(size(t));
plot(t,rhs,t,lhs), grid
```

Le graphe nous dit que la solution est entre 2 et 3. On peut calculer une solution approchée comme suit :

```
fsolve( @(x) 2*a/(r^2)-(x-sin(x)) , 2.5)
```

Jusqu'ici nous avons représenté des courbes engendrées par des équations cartésiennes, c'est-à-dire des fonctions de la forme $y = f(x)$. Pour cela, nous générons d'abord un ensemble de valeurs $\{x_i\}_{i=0\ldots N}$ puis l'ensemble de valeurs $\{y_i\}_{i=0\ldots N}$ avec $y_i = f(x_i)$. Il existe d'autres types de courbes comme par exemple les courbes paramétrées (engendrées par des équations paramétriques). Les équations paramétriques de courbes planes sont de la forme

$$\begin{cases} x = u(t), \\ y = v(t), \end{cases}$$

où u et v sont deux fonctions cartésiennes et le couple $(x; y)$ représente les coordonnées d'un point de la courbe paramétrée. La courbe engendrée par l'équation cartésienne $y = f(x)$ est une courbe paramétrée car il suffit de poser $u(t) = t$ et $v(t) = f(t)$. Un type particulier de courbe paramétrique est constitué par les équations polaires de courbes planes qui sont de la forme [3]

$$\begin{cases} x = r(t)\cos(t), \\ y = r(t)\sin(t). \end{cases}$$

Exercice 1.7 (Courbe parametrée)

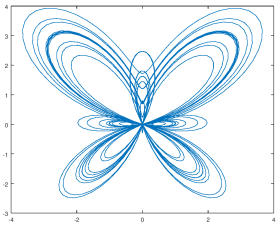
Tracer la courbe papillon ($t \in [0; 100]$) :

$$\begin{cases} x(t) = \sin(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \\ y(t) = \cos(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right) \end{cases}$$

3. r représente la distance de l'origine O du repère $(O; \mathbf{i}, \mathbf{j})$ au point $(x; y)$ de la courbe, et t l'angle avec l'axe des abscisses.

© 2022-2023 G. Faccanoni

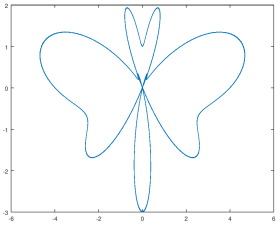
```
x = @(t) [ sin(t).*( exp(cos(t))-2*cos(4*t)-(sin(t/12)).^5 ) ];
y = @(t) [ cos(t).*( exp(cos(t))-2*cos(4*t)-(sin(t/12)).^5 ) ];
tt = [0:0.05:100];
plot(x(tt),y(tt))
```



Exercice 1.8 (Équation polaire)
Tracer la courbe papillon ($t \in [0;100]$) :

$$\begin{cases} x(t) = r(t)\cos(t) \\ y(t) = r(t)\sin(t) \end{cases} \quad \text{avec } r(t) = \sin(7t) - 1 - 3\cos(2t).$$

```
r = @(t) [sin(7*t)-1-3*cos(2*t)];
x = @(t) [ r(t).*cos(t) ];
y = @(t) [ r(t).*sin(t) ];
tt = [0:0.05:100];
plot(x(tt),y(tt))
```



Exercice 1.9 (Fractales)
La répétition de transformations permet de tracer des figures géométriques appelées fractales.
Considérons la suite définie par récurrence suivante :

$$\begin{pmatrix} x \\ y \end{pmatrix}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} x \\ y \end{pmatrix}_{n+1} = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_n + \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}_n$$

Pour chaque cas, tracer l'ensemble de points de coordonnées (x_n, y_n).

Exemple-1 on pose $\mathbb{M} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$ et $\mathbf{q} = \begin{pmatrix} 1 \\ -3 \end{pmatrix}$,

Dragon de Heighway on choisit au hasard et de façon équiprobable $\alpha \in [0;1[$ puis on pose

$$\mathbb{M} = \left(\alpha < \frac{1}{2}\right)\mathbb{M}_1 + \left(\alpha \geq \frac{1}{2}\right)\mathbb{M}_2$$

et

$$\mathbf{q} = \left(\alpha < \frac{1}{2}\right)\mathbf{q}_1 + \left(\alpha \geq \frac{1}{2}\right)\mathbf{q}_2$$

avec $\mathbb{M}_1 = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$, $\mathbb{M}_2 = \frac{1}{2}\begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix}$ et $\mathbf{q}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\mathbf{q}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ avec $n = 0, \ldots, 10000$.

Fougère de Barnsley on choisit au hasard et de façon équiprobable $\alpha \in [0;1[$ puis on pose

$$\mathbb{M} = \left(\alpha < \frac{1}{100}\right)\mathbb{M}_1 + \left(\frac{1}{100} \leq \alpha < \frac{86}{100}\right)\mathbb{M}_2 + \left(\frac{86}{100} \leq \alpha < \frac{93}{100}\right)\mathbb{M}_3 + \left(\alpha \geq \frac{93}{100}\right)\mathbb{M}_4$$

et

$$\mathbf{q} = \left(\alpha < \frac{1}{100}\right)\mathbf{q}_1 + \left(\frac{1}{100} \leq \alpha < \frac{86}{100}\right)\mathbf{q}_2 + \left(\frac{86}{100} \leq \alpha < \frac{93}{100}\right)\mathbf{q}_3 + \left(\alpha \geq \frac{93}{100}\right)\mathbf{q}_4$$

avec $\mathbb{M}_1 = \frac{1}{100}\begin{pmatrix} 0 & 0 \\ 0 & 16 \end{pmatrix}$, $\mathbb{M}_2 = \frac{1}{100}\begin{pmatrix} 85 & 4 \\ -4 & 85 \end{pmatrix}$, $\mathbb{M}_3 = \frac{1}{100}\begin{pmatrix} 20 & -26 \\ 23 & 22 \end{pmatrix}$, $\mathbb{M}_4 = \frac{1}{100}\begin{pmatrix} -15 & 28 \\ 26 & 24 \end{pmatrix}$ et $\mathbf{q}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\mathbf{q}_2 = \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$, $\mathbf{q}_3 = \mathbf{q}_2$, $\mathbf{q}_4 = \begin{pmatrix} 0 \\ 0.44 \end{pmatrix}$ avec $n = 0, \ldots, 10000$.

Arbre on choisit au hasard et de façon équiprobable $\alpha \in [0;1[$ puis on pose

$$\mathbb{M} = \left(\alpha < \frac{1}{3}\right)\mathbb{M}_1 + \left(\frac{1}{3} \leq \alpha < \frac{2}{3}\right)\mathbb{M}_2 + \left(\alpha \geq \frac{2}{3}\right)\mathbb{M}_3$$

et

$$\mathbf{q} = \left(\alpha < \frac{1}{3}\right)\mathbf{q}_1 + \left(\frac{1}{3} \le \alpha < \frac{2}{3}\right)\mathbf{q}_2 + \left(\alpha \ge \frac{2}{3}\right)\mathbf{q}_3$$

avec $\mathbb{M}_1 = \frac{1}{200}\begin{pmatrix}0 & 0\\ 0 & 51\end{pmatrix}$, $\mathbb{M}_2 = \frac{6}{8}\begin{pmatrix}\cos(\vartheta) & -\sin(\vartheta)\\ \sin(\vartheta) & \cos(\vartheta)\end{pmatrix}$, $\mathbb{M}_3 = \frac{1}{8}\begin{pmatrix}5\cos(\psi) & -6\sin(\psi)\\ 5\sin(\psi) & 6\cos(\psi)\end{pmatrix}$ et $\mathbf{q}_1 = \frac{1}{2}\begin{pmatrix}1\\ 0\end{pmatrix}$, $\mathbf{q}_2 = \begin{pmatrix}\frac{1}{2} - \frac{3}{8}\cos(\vartheta)\\ \frac{51}{200} - \frac{3}{8}\sin(\vartheta)\end{pmatrix}$, $\mathbf{q}_3 = \begin{pmatrix}\frac{1}{2} - \frac{5}{16}\cos(\psi)\\ \frac{153}{1000} - \frac{5}{16}\sin(\psi)\end{pmatrix}$ avec $n = 0, \ldots, 100$, $\vartheta = -\frac{\pi}{8}$ et $\psi = \frac{\pi}{5}$.

Source http://www.ac-grenoble.fr/maths/PM/Ressources/568/TP_fractales_Python.pdf

Correction

```
clc;
clear all;
cas=4;

% p et q vecteurs colonne, M matrice carree
transform = @(p,M,q) M*p+q;

% MAIN
n=1000*(cas==1)+10000*(cas==2)+10000*(cas==3)+100*(cas==4);
A = zeros(n,2);
for i=2:n
    if cas==1 % Exemple1
        M=[1,-1;1,1]/2;
        q=[1;-3];
    elseif cas==2 % Dragon
        M1=[1,-1;1,1]/2;
        q1=[0;0];
        M2=[-1,-1;1,-1]/2;
        q2=[1;0];
        alpha=rand();
        M=(alpha<0.5)*M1+(alpha>=0.5)*M2;
        q=(alpha<0.5)*q1+(alpha>=0.5)*q2;
    elseif cas==3 % Fougere
        M1=[0,0;0,0.16];
        q1=[0;0];
        M2=[0.85,0.04;-0.04,0.85];
        q2=[0;1.6];
        M3=[0.2,-0.26;0.23,0.22];
        q3=[0;1.6];
        M4=[-0.15,0.28;0.26,0.24];
        q4=[0;0.44];
        alpha=rand();
        M=(alpha<0.01)*M1+(alpha>=0.01)*(alpha<0.86)*M2+(alpha>=0.86)*(alpha<0.93)*M3+(alpha>=0.93)*M4;
        q=(alpha<0.01)*q1+(alpha>=0.01)*(alpha<0.86)*q2+(alpha>=0.86)*(alpha<0.93)*q3+(alpha>=0.93)*q4;
    else % arbre
        M1=[0,0;0,51/200];
        q1=[0.5;0];
        theta=-pi/8;
        M2=6/8*[cos(theta),-sin(theta);sin(theta),cos(theta)];
        q2=[1/2-3/8*cos(theta);51/200-3/8*sin(theta)];
        psi=pi/5;
        M3=1/8*[5*cos(psi),-6*sin(psi);5*sin(psi),6*cos(psi)];
        q3=[1/2-5/16*cos(psi);153/1000-5/16*sin(psi)];
        alpha=rand();
        M=(alpha<1/3)*M1+(alpha>=1/3)*(alpha<2/3)*M2+(alpha>=2/3)*(alpha<0.93)*M3;
        q=(alpha<1/3)*q1+(alpha>=1/3)*(alpha<2/3)*q2+(alpha>=2/3)*(alpha<0.93)*q3;
    end
    A(i,:)=transform(A(i-1,:)',M,q);
end
plot(A(:,1),A(:,2),'o','MarkerSize',8)
```


© 2022-2023 G. Faccanoni

Chapitre 2

Traitement mathématique des images numériques

Dans cette partie nous allons nous intéresser à la manipulation d'images numériques.

Pour commencer, nous allons considérer une matrice de 0 et 1 comme une image constituée de "carreaux" blancs si la valeur est 0 et noirs si la valeur est 1. Nous allons alors appliquer quelque transformation élémentaire sur la matrice initiale et visualiser sur l'image associée l'effet de chaque transformation.

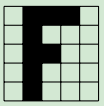
Attention

Pensez à placer la commande `clear all` au début de vos scripts, de manière à nettoyer l'environnement de travail (cela effacera toutes les variables en mémoire). Vous pouvez aussi utiliser la commande `clc` pour nettoyer la fenêtre de commandes.

Pour chaque exemple ou exercice, on écrira les instructions dans un fichier script nommé par exemple `exo_1_3.m` (sans espaces, ni accents ni points sauf pour l'extension `.m`). On pourra bien-sûr utiliser le mode interactif pour simplement vérifier une commande mais chaque exercice devra in fine être résolu dans un fichier script. Tous ces scripts devront se trouver dans un dossier dont le nom ne contient ni espaces, ni accents, ni points.

Exercice 2.1 (Multiplication matricielle appliquée)

On modélise une image en noir et blanc formée de 25 pixels par une matrice de 5 lignes et 5 colonnes, dans laquelle 0 correspond à un pixel blanc et 1 à un pixel noir. L'image à modéliser est la suivante :

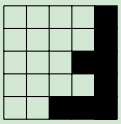


1. Donner la matrice $\mathbb{M}$ associée à l'image.
2. Soient

$$\mathbb{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbb{B} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

   Calculer le produit $\mathbb{AM}$. Quel est l'effet de la matrice $\mathbb{A}$ sur l'image ? Quel est l'effet si on fait le produit $\mathbb{MA}$ sur l'image ? **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle ?**

   Calculer le produit $\mathbb{BM}$. Quel est l'effet de la matrice $\mathbb{B}$ sur l'image ? Quel est l'effet si on fait le produit $\mathbb{MB}$ sur l'image ? **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle ?**
3. Quelle image obtient-on en faisant le produit $\mathbb{AMB}$? **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle ?**
4. Quelle image obtient-on en faisant le produit $\mathbb{AMA}$? **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle ?**
5. Quel produit matriciel peut-on faire pour obtenir la figure suivante ? **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle ?**



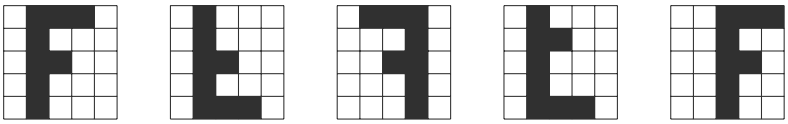
6. Calculer le produit $\mathbb{AM}^T$. **Quelles instructions permettent d'obtenir ces résultats sans utiliser la multiplication matricielle?**

Correction

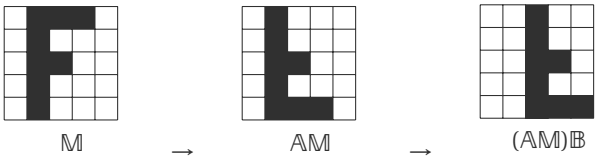
1. $$\mathbb{M} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

2. Le produit $\mathbb{AM}$ correspond à inverser l'ordre des lignes de la matrice $\mathbb{M}$: l'image est alors symétrique par rapport à la troisième ligne.
   Le produit $\mathbb{MA}$ correspond à inverser l'ordre des colonnes de la matrice $\mathbb{M}$: l'image est alors symétrique par rapport à la troisième colonne.
   Le produit $\mathbb{BM}$ correspond à translater les lignes de la matrice $\mathbb{M}$ d'un rang vers le haut (la première ligne passant en cinquième ligne) : l'image est alors translatée d'un rang vers le haut (la première ligne passant en cinquième ligne).
   Le produit $\mathbb{MB}$ correspond à translater les colonnes de la matrice $\mathbb{M}$ d'un rang vers la droite (la cinquième colonne passant en première colonne) : l'image est alors translatée d'un rang vers la droite (la cinquième colonne passant en première colonne).

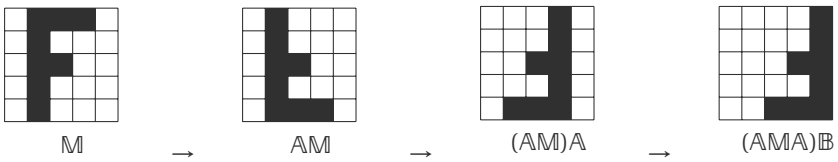
$$\mathbb{AM} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \mathbb{MA} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \mathbb{BM} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \mathbb{MB} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$



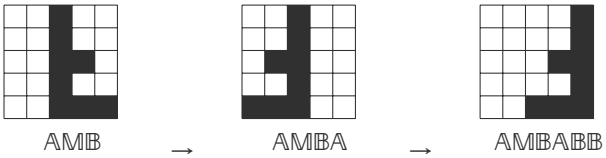
3. Le produit $\mathbb{AMB} = (\mathbb{AM})\mathbb{B}$ correspond par exemple à une symétrie horizontale par rapport à la troisième ligne suivie d'une translation d'un rang vers la droite. On peut aussi l'écrire comme $\mathbb{AMB} = \mathbb{A}(\mathbb{MB})$ qui correspond à une translation d'un rang vers la droite suivie d'une symétrie horizontale par rapport à la troisième ligne. Dans tous les cas on obtient l'image suivante :



4. On peut par exemple calculer $\mathbb{AMAB}$.

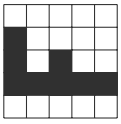


   Ou encore calculer $\mathbb{AMBABB}$.



© 2022-2023 G. Faccanoni

5. Le produit $\mathbb{A}\mathbb{M}^T$ correspond à une rotation antihoraire, elle équivaut à l'instruction `rot90(M)`.



$\mathbb{A}\mathbb{M}^T$

Par multiplications :

```
M=[0 1 1 1 0
   0 1 0 0 0
   0 1 1 0 0
   0 1 0 0 0
   0 1 0 0 0]
```

```
A=eye(5);
A=A(:,5:-1:1)
%
B=diag(ones(4,1),1);
B(5,1)=1;
B
```

```
A*M
M*A
B*M
M*B
A*M*B
A*M*A*B
A*M*B*A*B*B
```

Par transformations :

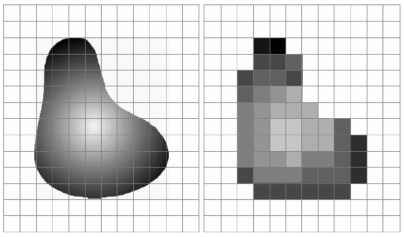
```
M=[0 1 1 1 0
   0 1 0 0 0
   0 1 1 0 0
   0 1 0 0 0
   0 1 0 0 0]
%
A=eye(5);
A=A(:,5:-1:1)
%
B=diag(ones(4,1),1);
B(5,1)=1;
B
```

```
AM = M(end:-1:1,:) % octave flipud(M), matlab flip(
    M)
MA = M(:,end:-1:1) % octave fliplr(M), matlab flip(
    M,2)
BM = [ M(2:end,:) ; M(1,:) ]
MB = [ M(:,end), M(:,1:end-1) ]
AMB = [ AM(:,end), AM(:,1:end-1) ]

AMA = AM(:,end:-1:1);
AMAB = [ AMA(:,end), AMA(:,1:end-1) ]

AMt = M'(end:-1:1,:) % rot90(M)
```

2.1 Introduction : lecture, affichage, sauvegarde

Un peu de nomenclature :

- **Acquisition d'une image (= numérisation)** : dans la figure ci dessous on a à gauche une image réelle et à droite sa version numérisée. La numérisation d'une image réelle comporte une perte d'information due d'une part à l'échantillonnage (procédé de discrétisation spatiale d'une image consistant à projeter sur une grille régulière, *i.e.* en un nombre fini de points, une image analogique continue en lui associant une valeur unique), d'autre part à la quantification (nombre finis de nuances = limitation du nombre de valeurs différentes que peut prendre un point).



- **Les pixels d'une image** : une image numérique en niveaux de gris (*grayscale* image en anglais) est un tableau de valeurs $\mathbb{A}$. Chaque case de ce tableau, qui stocke une valeur a_{ij}, se nomme un pixel (*PICTure ELement*).
  En notant n le nombre de lignes et p le nombre de colonnes du tableau, on manipule ainsi un tableau de $n \times p$ pixels.
  Les valeurs des pixels sont enregistrées dans l'ordinateur ou l'appareil photo numérique sous forme de nombres entiers entre 0 et 255, ce qui fait 256 valeurs possibles pour chaque pixel. La valeur 0 correspond au noir et la valeur 255 correspond au blanc. Les valeurs intermédiaires correspondent à des niveaux de gris allant du noir au blanc.
  On peut définir une fonction comme suit :

$$\begin{aligned} g\colon [\![1:n]\!] \times [\![1:p]\!] &\to [\![0:255]\!] \\ (i,j) &\mapsto g(i,j) = a_{ij} \end{aligned}$$

- **La taille d'une image** : la taille d'une image est le nombre de pixel. Les dimensions d'une image sont la largeur (=nombre de colonnes du tableau) et la hauteur (=nombre de lignes du tableau).
- **La résolution d'une image** : la résolution d'une image est le nombre de pixel par unité de longueur. En générale, on utilise des "pixel par pouce" ou "point par pouce" (`ppp`), on anglais on dit *dot per inch* (`dpi`) : n `dpi` = n pixel pour un puce = n pixel pour 2.54 cm.

> EXEMPLE
> On veut déterminer les dimensions d'une image obtenue à partir d'un scanner pour une page A4 à la résolution de 300 `dpi`.
> - Le format A4 est un rectangle de 21 cm × 29.7 cm = 8.25 inch × 11.7 inch.
> - La largeur de l'image est donc $n = 300$ dpi × 8.25 inch = 2 475 pixel.
> - La hauteur de l'image est donc $p = 300$ dpi × 11.7 inch = 3 510 pixel.
> - La taille de l'image est 2 475 × 3 510 = 8 700 000 pixel.

Matrice → Image

Considérons une matrice carrée avec $n = p = 8$, ce qui représente $2^3 \times 2^3 = 2^6 = 64$ éléments.

```
N = 8;
A = eye(N);
for n = 1:N
   A(:,n) = 20*(n-2:n+N-3);
end
```

```
A =
   -20     0    20    40    60    80   100   120
     0    20    40    60    80   100   120   140
    20    40    60    80   100   120   140   160
    40    60    80   100   120   140   160   180
    60    80   100   120   140   160   180   200
    80   100   120   140   160   180   200   220
   100   120   140   160   180   200   220   240
   120   140   160   180   200   220   240   260
```

Pour voir le tableau `A` comme une image de 64 pixels en niveaux de gris,

© 2022-2023 G. FACCANONI

- ⋆ d'abord on transforme la matrice avec `uint8` (toutes les valeurs inférieurs à 0 sont mises à 0, toutes les valeurs supérieures à 255 sont mises à 255 et les valeurs non entières sont arrondies)
- ⋆ on indique la `colormap` (ici échelle de gris de 0 à 255) pour que l'image s'affiche avec des niveaux de gris (si $M_{ij} = 255$ est affiché blanc et $M_{ij} = 0$ noir, les valeurs intermédiaires en gris), puis on affiche l'image :

```
M = uint8(A)
% si A(i,j)<0 alors M(i,j)=0,
% si A(i,j)>255 alors M(i,j)=255

% methode 1
%colormap(gray(256));
%image(M)

% methode 2
imshow(M,gray(256))

% colorbar % pour voir la table de couleurs
    utilisees (et les valeurs presentes dans l
    image)
```

```
M =
    0    0   20   40   60   80  100  120
    0   20   40   60   80  100  120  140
   20   40   60   80  100  120  140  160
   40   60   80  100  120  140  160  180
   60   80  100  120  140  160  180  200
   80  100  120  140  160  180  200  220
  100  120  140  160  180  200  220  240
  120  140  160  180  200  220  240  255
```

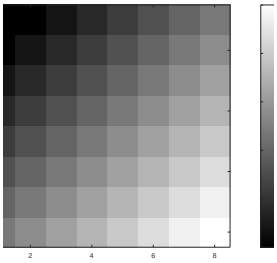


Image → Matrice

Dans ce cas, nous ne construisons pas la matrice à la main, mais nous allons lire un fichier image et l'importer comme une matrice dans Octave. Pour **transformer une image en une matrice** il suffit d'indiquer dans un script : [1]

```
A = imread('lena512.bmp');
```

ATTENTION

L'image doit se trouver dans le même dossier que les fichier script. Ce dossier et le fichier script ne doivent pas contenir ni d'espaces, ni d'accents, ni de points ou autres caractères spéciaux.

Pour connaître la **dimension de la matrice** (*i.e.* de l'image) il suffira d'écrire

```
[row,col] = size(A) % = [hateur,largeur] de l'image
```

On a une matrice carrée de taille $n = p = 512$, ce qui représente $512 \times 512 = 2^{18} = 262\,144$ pixels. [2]

Pour connaître l'**intervalle des valeurs** de la matrice, *i.e.* $\min_{\substack{1\leq i\leq \texttt{row}\\ 1\leq j\leq \texttt{col}}} A_{ij}$ et $\max_{\substack{1\leq i\leq \texttt{row}\\ 1\leq j\leq \texttt{col}}} A_{ij}$, on écrira

```
min(A(:)), max(A(:))
```

Les niveaux de gris de notre image sont compris entre 25 et 245.

Par défaut, avec la fonction **`imread`**, Octave transforme un fichier image en niveaux de gris (ici de type `.bmp`) en une matrice dont chaque coefficient est de type `uint8` (codage sur 8 bits non signés). Cela signifie que ses éléments (qui représentent les niveaux de gris) sont dans l'intervalle d'entiers $[\![0-255]\!]$). Pour pouvoir utiliser toutes les fonctions Octave sur ces données, il est préférable de les convertir en réels (commande `double`) puis, après traitement, les reconvertir en entier codé sur 8 bits non signé avant de sauvegarder en format raster (commande `uint8`). [3]

1. Cette image, "Lena", est une image digitale fétiche des chercheurs en traitement d'images. Elle a été scannée en 1973 dans un exemplaire de Playboy et elle est toujours utilisée pour vérifier la validité des algorithmes de traitement ou de compression d'images. On la trouve sur le site de l'University of Southern California http://sipi.usc.edu/database/.
2. Les appareils photos numériques peuvent enregistrer des images beaucoup plus grandes, avec plusieurs millions de pixels.
3. Octave peut lire des images codées sur 8, 16, 24 ou 32 bits. Mais le stockage et l'affichage de ces données ne peut être fait qu'avec trois types de variables :
   - ⋆ le type `uint8` (entier non signé de 8 bits) de plage $[0;255]$;
   - ⋆ le type `uint16` (entier non signé de 16 bits) de plage $[0;65535]$;
   - ⋆ le type `double` (réel 64 bits) de plage $[0;1]$.

Matrice/Image → affichage et/ou sauvegarde

On peut **voir la matrice comme une image** avec :

```
imshow(A); % imshow(uint8(A)) si on veut s'assurer
    d'avoir des uint8
```

On peut **sauvegarder l'image** avec :

```
imwrite(uint8(A),'monimage.jpg','jpg');
```

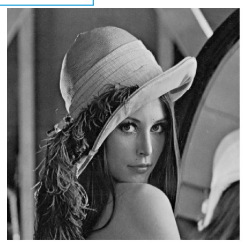


FIGURE 2.1 – Léna (original)
http://www.lenna.org

Exercice 2.2
Afficher les images associées aux matrices suivantes :

$$\mathbb{A} = \begin{pmatrix} 0 & 0 & 0 & 255 & 255 \\ 0 & 0 & 255 & 255 & 255 \\ 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 0 & 0 \\ 255 & 255 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbb{B} = \begin{pmatrix} 0 & 50 & 100 & 150 & 200 & 250 \\ 0 & 50 & 100 & 150 & 200 & 250 \\ 0 & 50 & 100 & 150 & 200 & 250 \end{pmatrix}$$

Correction
Les valeurs 255 sont affichées en blanc, les valeurs 0 en noir, les valeurs intermédiaires en gris.

```
% Dans OCTAVE on peut ecrire une seule instruction
A=255*(eye(5)+diag(ones(1,4),1)+diag(ones(1,4),-1))(:,end
    :-1:1)
% Dans MATLAB il faut la decouper comme suit
% A=255*(eye(5)+diag(ones(1,4),1)+diag(ones(1,4),-1))
% A=A(:,end:-1:1)

B=[0:50:250].*ones(3,1)

subplot(1,2,1)
imshow(A,gray(256));
title("A")
subplot(1,2,2)
% noter la difference avec imshow(B)
imshow(B,gray(256));
title("B")
```

Dans les prochaines sections nous allons considérer deux familles de transformations :

- ⋆ les transformations qui déplacent/éliminent/ajoutent des lignes/colonnes de la matrice ;
- ⋆ les transformations qui modifient la valeur des pixels.

Dans le deuxième cas, il convient de transformer d'abord les valeurs de type `uint` en `double` car certaines opérations ne sont pas définies pour les `uint`. On retransformera en `uint` avant de sauvegarder l'image.

Remarque (`uint8`)
Lorsqu'on effectue des calculs avec des uint8, les valeurs sont toujours des entiers dans $[0;255]$ (toutes les valeurs inférieures à 0 sont mises à 0, toutes les valeurs supérieures à 255 sont mises à 255 et les valeurs non entières sont arrondies). Dans le code ci-dessous, on a $a = 155$ mais a est de type `uint8` donc, lorsqu'on calcule a^2, Octave plafonne à 255 ; lorsqu'on calcule $a/2$, Octave l'arrondi à 78 et lorsqu'on calcule $-a$, Octave plafonne à 0 :

```
A = imread('lena512.bmp');
a = A(10,10);

% On verifie que par default imread donne une matrice de uint8
disp(a)        % output 155
disp(class(a)) % output uint8
disp(a^2)      % output 255 au lieu de 155^2=24025
disp(a/2)      % output 78 au lieu de 155/2=77.5
```


© 2022-2023 G. FACCANONI

```
disp(-a)       % output 0 au lieu de -155

% On transforme en float et on verifie que les resultats ne sont plus tronques ni arrondis
a = double(a);
disp(a)        % output 155
disp(class(a)) % output double
disp(a^2)      % output 155^2=24025
disp(a/2)      % output 155/2=77.5
disp(-a)       % output -155
```

Les fonctions Octave les plus utiles pour gérer les images sont les suivantes :

- ⋆ `imread` : lit une image à partir d'un fichier (formats standards) ;
- ⋆ `image` ou `imagesc` ou `imshow` : affiche une image (objet graphique Image) ;
- ⋆ `imwrite` : écrit une image dans un fichier (formats standards) ;
- ⋆ `imfinfo` : extrait des informations d'un fichier (formats standards) ;
- ⋆ `print` : exporte une image (formats standards).

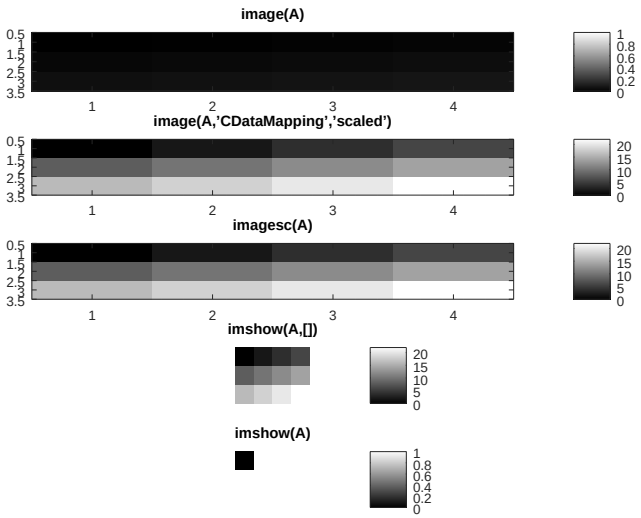
Remarque (`image`, `imagesc` ou `imshow`?)

Soit

$$\mathbb{A} = \begin{pmatrix} 0 & 2 & 4 & 6 \\ 8 & 10 & 12 & 14 \\ 16 & 18 & 20 & 22 \end{pmatrix}$$

Elle contient 3 lignes et 4 colonnes et les valeurs vont de 0 à 22. Analysons le code suivant :

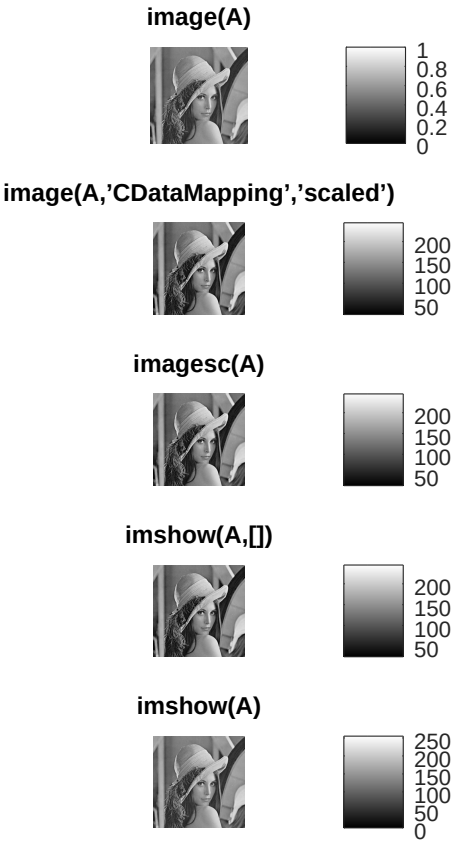
```
clear all
close all

A=[0 2 4 6; 8 10 12 14; 16 18 20 22];
minA=min(A(:))
maxA=max(A(:))

n=5

subplot(n,1,1)
image(A), colormap(gray(256)), colorbar
title("image(A)");

subplot(n,1,2)
image(A,'CDataMapping','scaled'), colormap(gray
    (256)), colorbar
title("image(A,'CDataMapping','scaled')");

subplot(n,1,3)
imagesc(A), colormap(gray(256)), colorbar
title("imagesc(A)");

subplot(n,1,4)
imshow(A,[]), colorbar
title("imshow(A,[])");

subplot(n,1,5)
imshow(A), colorbar
title("imshow(A)");
```



- ⋆ La fonction `image` affiche une image noire : en effet les valeurs de $\mathbb{A}$ vont de 0 à 22 tandis que l'échelle de gris va de 0 à 255. Pour étirer l'échelle il faut le demander explicitement avec les mots clé `'CDataMapping','scaled'`.
- ⋆ La fonction `imagesc` affiche une image en "étirant" l'échelle : elle étale les valeurs pour que la plus petite valeur de $\mathbb{A}$ correspond au 0 de l'échelle de gris et la plus grande valeur de $\mathbb{A}$ au 255 dans l'échelle de gris.
- ⋆ La fonction `image` choisi automatiquement la `colormap` et affiche les cellules comme des carrés mais a le même problème que `image`. Pour qu'elle étale les valeurs pour que la plus petite valeur de $\mathbb{A}$ correspond au 0 de l'échelle de gris et la plus grande valeur de $\mathbb{A}$ au 255 dans l'échelle de gris il faut ajouter `[]`.

Voici un autre exemple avec une image bitmap et des options pour avoir le bon comportement :

```
clear all, close all

A=imread('lena512.bmp');
minA=min(A(:))
maxA=max(A(:))

n=5

subplot(n,1,1)
image(A), colormap(gray(256)), axis image, axis off
    , colorbar
title("image(A)");

subplot(n,1,2)
image(A,'CDataMapping','scaled'), colormap(gray
    (256)), axis image, axis off, colorbar
title("image(A,'CDataMapping','scaled')");

subplot(n,1,3)
imagesc(A), colormap(gray(256)), axis image, axis
    off, colorbar
title("imagesc(A)");

subplot(n,1,4)
imshow(A,[]), colorbar
title("imshow(A,[])");

subplot(n,1,5)
imshow(A), colorbar
title("imshow(A)");
```



© 2022-2023 G. Faccanoni

2.2 Manipulations élémentaires : déplacements des pixels

Exercice 2.3 (Devine le résultat)
Que se passe-t-il lorsqu'on exécute les scripts suivants ?

Script 1

```
clear all

A=imread('lena512.bmp');
colormap(gray(256));

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Originale" );
subplot(1,2,2)
 B=A';
 imshow(uint8(B));
 title ( "Transposee" );
 imwrite(uint8(B),'exotransposee.jpg','jpg');
```

Script 2

```
clear all

A=imread('lena.jpg');
[row,col]=size(A);

for j=1:col
   A=[A(:,2:col) A(:,1)];
   imshow(A,gray(256));
   pause(0.001);
end
```

Correction

1. Dans le premier script la matrice $\mathbb{B}$ est la transposée de la matrice $\mathbb{A}$: l'image obtenue est la symétrique par rapport à la diagonale principale :


Originale


Transposée

2. À chaque étape on enlève la première colonne et on la concatène comme dernière colonne. Le résultat donne un "film" dans lequel l'image "sort" à gauche et "rentre" à droite.

Exercice 2.4 (Flip, H-V-stack, Zoom)
En utilisant une manipulation élémentaire de la matrice (sans faire de boucles et sans utiliser de fonctions prédéfinies) obtenir les images de la figure 2.2. On pourra se baser sur le canevas suivant :

```
clear all
A=imread('lena512.bmp');
B= .... % construire B
subplot(1,2,1)
 imshow(uint8(A));
 title ( "Originale" );
subplot(1,2,2)
 imshow(uint8(B));
 title ( "Transformee" );
```


(a) Original (b) Flip vertical (c) Flip horizontale (d) Flip double (e) Hstack (f) Vstack (g) Zoom

FIGURE 2.2 – Manipulations élémentaires

Correction

1. Flip vertical `B = A(end:-1:1,:);`
   Dans Octave, cela correspond à `B = flipud(A);` (up to down), dans Matlab à `B = flip(A);`

2. Flip horizontale `B = A(:,end:-1:1);`
   Dans Octave, cela correspond à `B = fliplr(A);` (left to right), dans Matlab à `B = flip(A,2);`

3. Flip double `B = A(end:-1:1,end:-1:1);`

4. Hstack `B = [ A(:,end/2:end), A(:,end:-1:end/2)];`

5. Vstack `B = [ A(end/2:end,:); A(end:-1:end/2,:)];`

6. Zoom `B = A(300:400,250:350);`

Exercice 2.5 (Bandes)

On veut représenter un ensemble de 4 raies verticales blanches et 4 raies verticales noires alternées sur une image $M \times N$ ($M = N = 256$).

1. Quelle est la taille en pixels de chaque raie blanche et chaque raie noire ?
2. Une première idée d'algorithme pour construire une telle image consiste d'abord a construire une raie blanche, puis une raie noire puis former l'ensemble de l'image par juxtaposition.
3. L'image définie par $b_{i,j} = \frac{256}{2}\left(1+\cos\left(2\pi\frac{v(j-u)}{N}\right)\right)$ est aussi une image de raies. Comment choisir v et u pour que l'image ainsi formée coïncide à peu prés avec l'image recherchée ?
4. On souhaite malgré tout une image binaire, à savoir une image pour laquelle les pixels n'ont que deux valeurs possibles : 0 ou 255. Modifier l'algorithme précédent en mettant en blanc les pixels pour lesquels $\cos\left(2\pi\frac{v(j-u)}{N}\right) \geq 0$ et en noir ceux pour lesquels $\cos\left(2\pi\frac{v(j-u)}{N}\right) < 0$.
5. Décaler l'image vers la droite d'une demi-raie (*i.e.* à gauche de l'image et tout à droite il y a une demi-raie noire).

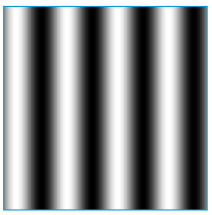
Correction

1. Chaque raie occupe 256×32 pixels. En effet, chaque raie occupe 256 lignes pour $\frac{256}{8} = \frac{2^8}{2^3} = 2^5 = 32$ colonnes.

2.
```
clear all; clc;
N = zeros(256,32);
B = 255-N;
A = [B,N,B,N,B,N,B,N];
imshow(uint8(A));
%imwrite(uint8(A),'exoRaies1.jpg','jpg');
```

3. Il faut que $b_{16,j} = 0$, $b_{48,j} = 1$, etc.

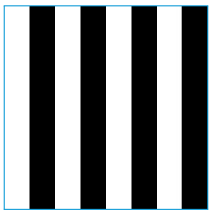
```
clear all; clc;
B = 255*ones(256);
v = 32/8;
u = 32/2;
for j = 1:256
   B(:,j) = 1+cos(2*pi*v*(j-u)/256);
end
% on se ramene a [0;255]
m = min(B(:));
M = max(B(:));
B = 255/(M-m).*(B-m);
imshow(uint8(B));
imwrite(uint8(B),'exoRaies2.jpg','jpg');
```



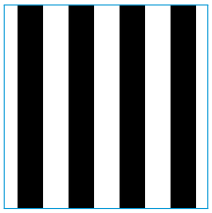
4.

© 2022-2023 G. Faccanoni

```
clear all; clc;
B = 255*ones(256);
v = 32/8;
u = 32/2;
for j = 1:256
   B(:,j) = cos(2*pi*v*(j-u)/256)> = 0;
end
% on se ramene a [0;255]
B = 255*B;
imshow(uint8(B));
imwrite(uint8(B),'exoRaies3.jpg','jpg');
```



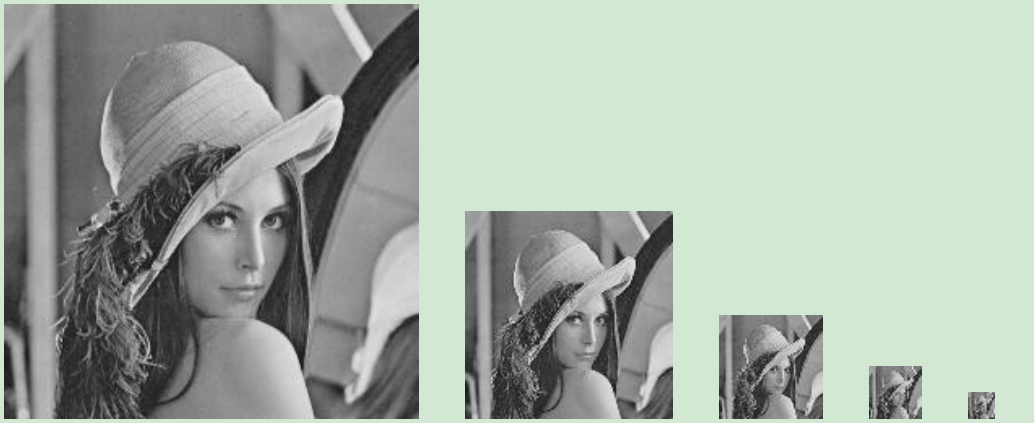
5.
```
clear all; clc;
N = zeros(256,32);
B = 255-N;
A = [B,N,B,N,B,N,B,N];
B = [A(:,16:end),A(:,1:15)];
imshow(uint8(B));
imwrite(uint8(B),'exoRaies4.jpg','jpg');
```



Exercice 2.6 (Réduction de la résolution)

Une matrice de taille $2^9 \times 2^9$ contient 2^{18} entiers, ce qui prend pas mal de place en mémoire. On s'intéresse à des méthodes qui permettent d'être plus économique sans pour cela diminuer la qualité esthétique de l'image. Afin de réduire la place de stockage d'une image, on peut réduire sa résolution, c'est-à-dire diminuer le nombre de pixels. La façon la plus simple d'effectuer cette réduction consiste à **supprimer des lignes et des colonnes dans l'image de départ**.

Les figures suivantes montrent ce que l'on obtient si l'on retient une ligne sur 2^k et une colonne sur 2^k ce qui donne une matrice $2^{9-k} \times 2^{9-k}$. Appliquer cette transformation pour obtenir l'une des images suivantes :



(a) Originale ($k = 1$) (b) $k = 2$ (c) $k = 3$ (d) $k = 4$ (e) $k = 5$

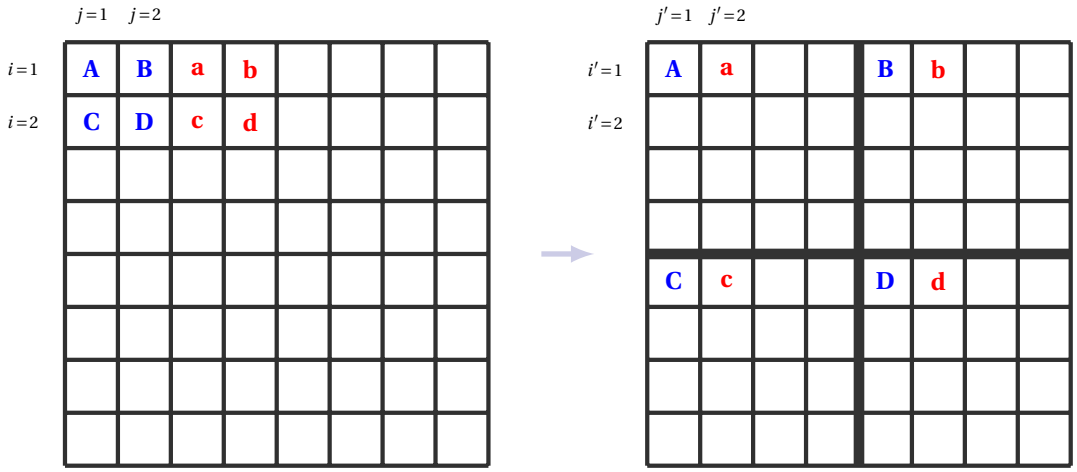
FIGURE 2.3 – Résolution

Correction

```
clear all

A = double(imread('lena512.bmp'));
colormap(gray(256));

[row,col] = size(A)

for k = 1:8
   subplot(1,8,k)
    E = A(1:2^k:row,1:2^k:col);
    imshow(uint8(E));
    title(['k = ' num2str(k)]);
    file = strcat('exo2E', num2str(k) ,'.jpg')
    %imwrite(uint8(E),file,'jpg');
end
```

2.2.1 ☆Transformation du photomaton

On part d'un tableau $n \times n$, avec n pair, chaque élément du tableau représente un pixel. À partir de cette image on calcule une nouvelle image en déplaçant chaque pixel selon une transformation, appelée *transformation du photomaton.*
On découpe l'image de départ selon des petits carrés de taille 2×2. Chaque petit carré est donc composé de quatre pixels. On envoie chacun de ces pixels à quatre endroits différents de la nouvelle image : le pixel en haut à gauche reste dans une zone en haut à gauche, le pixel en haut à droite du petit carré, est envoyé dans une zone en haut à droite de la nouvelle image,...



Par exemple le pixel en position $(1,1)$ (symbolisé par la lettre D) est envoyé en position $(4,4)$.
Explicitons ce principe par des formules. Pour chaque couple (i,j), on calcule son image (i',j') par la transformation du photomaton selon les formules suivantes :

- ⋆ Si l'indice k est impair alors $k' = \frac{k+1}{2}$.
- ⋆ Si l'indice k est pair alors $k' = \frac{k+n}{2}$.

Voici un exemple d'un tableau 4×4 avant (à gauche) et après (à droite) la transformation du photomaton.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	3	2	4
9	11	10	12
5	7	6	8
13	15	14	16

Exercice 2.7 (Photomaton)

1. Écrire une fonction `photomaton(tableau)` qui renvoie le tableau calculé après transformation. Par exemple le tableau de gauche est transformé en le tableau de droite.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	3	2	4
9	11	10	12
5	7	6	8
13	15	14	16

2. Écrire une fonction `photomaton_iterer(tableau,k)` qui renvoie le tableau calculé après k itérations de la transformation du photomaton.
3. Expérimenter pour différentes valeurs de la taille n, afin de voir au bout de combien d'itérations on retrouve l'image de départ.

Correction
Dans le fichier `photomaton.m` on écrit la fonction `photomaton` qui prend une matrice carrée de dimension n pair et renvoi une matrice de même dimension qui a subi la transformation. Pour le calcul des nouveaux indices on écrit une sous-fonction `photomatonIndice` :

```
function B = photomaton(A,n)
   B = zeros(n);
   for i = 1:n
      for j = 1:n
         wi = photomatIndice(i,n);
         wj = photomatIndice(j,n);
         B(wi,wj) = A(i,j);
      end
```


© 2022-2023 G. Faccanoni

```
   end
end


function w = photomatIndice(k,d)
   if rem(k+1,2) =  = 0 %(k+1)/2 =  = floor((k+1)
```

```
       /2)
      w = floor((k+1)/2);
   else
      w = floor((k+d)/2);
   end
end
```

On teste la fonction d'abord sur la matrice 4×4 donnée puis sur une image :

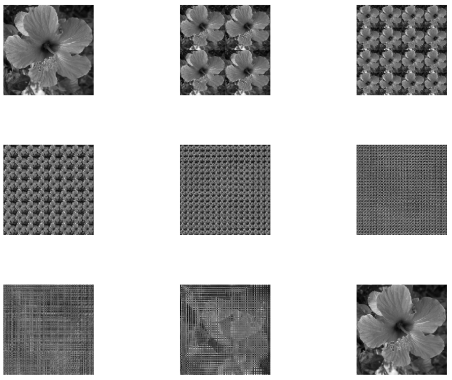
'Script photomatonTEST.m'

```
clear all; clc;

% % TEST
% A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
% B = photomaton(A,4)

A = imread('lena512.bmp');
[r,c] = size(A);
colormap(gray(256));

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Original" );

subplot(1,2,2)
 B = photomaton(A,r);
 imshow(uint8(B));
 title ( "Photomaton" );
 %imwrite(uint8(B),'exoPhot1.jpg','jpg');
```

On boucle :

'Script photomatonTEST2.m'

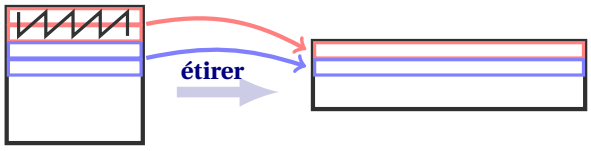
```
clear all; clc;

A = imread('section1-original.png');
[r,c] = size(A);
colormap(gray(256));

subplot(3,3,1)
 imshow(uint8(A));
for k = 2:9
subplot(3,3,k)
 B = photomaton(A,r);
 imshow(uint8(B));
 A = B;
end
```



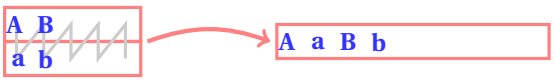
2.2.2 ☆Transformation du boulanger

Cette transformation s'appelle ainsi car elle s'apparente au travail du boulanger qui réalise une pâte feuilletée. L'image est tout d'abord aplatie, coupée en deux puis la partie droite est placée sous la partie gauche en la faisant tourner de 180°.
On part d'un tableau $n \times n$, avec n pair dont chaque élément représente un pixel. On va appliquer deux transformations élémentaires à chaque fois :

- ⋆ *Étirer :* les deux premières lignes (chacune de longueur n) produisent une seule ligne de longueur $2n$ en mixant les valeurs de chaque ligne en alternant un élément du haut, un élément du bas.



Voici comment deux lignes se mixent en une seule :



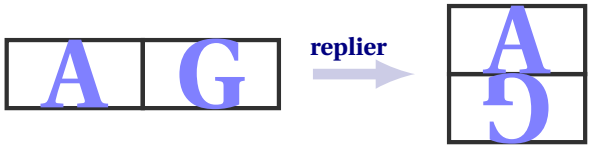
Un élément en position (i, j) du tableau de départ correspond à un élément $(i/2, j/2)$ (si j est pair) ou bien $((i+1)/2, j/2)$ (si j est impair) du tableau d'arrivée.

Exemple. Voici un tableau 4×4 à gauche, et le tableau étiré 2×8 à droite. Les lignes 0 et 1 à gauche donnent la ligne 0 à droite. Les lignes 2 et 3 à gauche donne la ligne 1 à droite.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	2	6	3	7	4	8
9	13	10	14	11	15	12	16

- ⋆ *Replier :* la partie droite d'un tableau étiré est retournée, puis ajoutée sous la partie gauche. Partant d'un tableau $\frac{n}{2} \times 2n$ on obtient un tableau $n \times n$.



Pour $0 \leq i < \frac{n}{2}$ et $0 \leq j < n$ les éléments en position (i, j) du tableau sont conservés. Pour $\frac{n}{2} \leq i < n$ et $0 \leq j < n$ un élément du tableau d'arrivée (i, j), correspond à un élément $\left(\frac{n}{2} - i - 1, 2n - 1 - j\right)$ du tableau de départ.

Exemple. À partir du tableau étiré 2×8 à gauche, on obtient un tableau replié 4×4 à droite.

1	5	2	6	3	7	4	8
9	13	10	14	11	15	12	16

1	5	2	6
9	13	10	14
16	12	15	11
8	4	7	3

La *transformation du boulanger* est la succession d'un étirement et d'un repliement. Partant d'un tableau $n \times n$ on obtient encore un tableau $n \times n$.

Exercice 2.8 (Boulanger)

1. Écrire une fonction `boulanger(tableau)` qui renvoie le tableau calculé après transformation. Par exemple le tableau de gauche est transformé en le tableau de droite.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	2	6
9	13	10	14
16	12	15	11
8	4	7	3

2. Écrire une fonction `boulanger_iterer(tableau,k)` qui renvoie le tableau calculé après k itérations de la transformation du photomaton.

© 2022-2023 G. Faccanoni

3. Vérifier que pour $n = 256$ au bout de 17 itérations on retrouve l'image de départ.

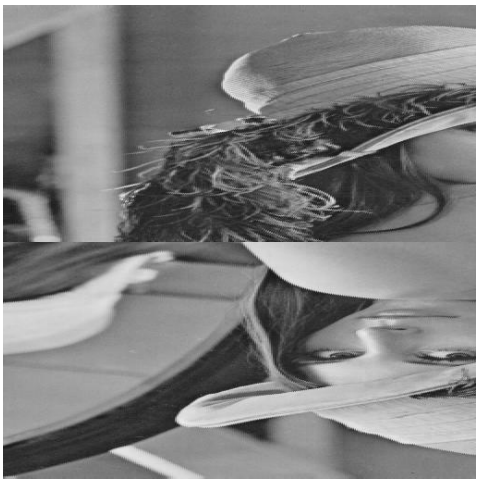
Correction
Dans le fichier `boulanger.m` on écrit la fonction `boulanger` qui prend une matrice carrée de dimension n pair et renvoi une matrice de même dimension qui a subi la transformation.

```
function C=boulanger(A,n)
   B = zeros(fix(n/2),2*n);
   % etirer
   for i = 1:n
      for j = 1:n
         if fix((i+1)/2)==(i+1)/2
            B(fix((i+1)/2),2*j-1) = A(i,j);
         else
            B(fix((i+1)/2),2*j) = A(i,j);
         end
      end
   end
   % couper
   C = [B(:,1:n); B(end:-1:1,end:-1:n+1)];
end
```

On teste nos fonctions d'abord sur la matrice 4×4 donnée puis sur une image :

'Script boulangerTEST.m'

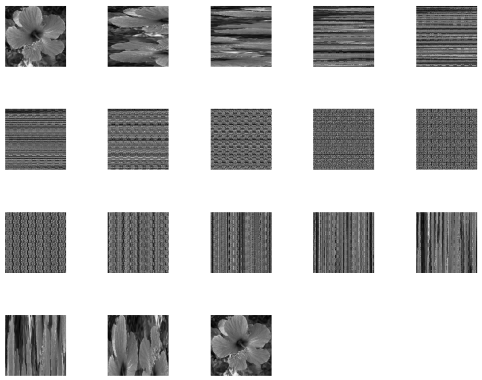
```
clear all; clc;

% TEST
A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
B = boulanger(A,4)

A = imread('lena512.bmp');
[r,c] = size(A);
colormap(gray(256));

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Original" );
subplot(1,2,2)
 B=boulanger(A,r);
 imshow(uint8(B));
 title ( "Boulanger" );
 %imwrite(uint8(B),'exoBou1.jpg','jpg');
```



On boucle :

'Script boulangerTEST2.m'

```
clear all; clc;

A = imread('section1-original.png');
[r,c] = size(A);
colormap(gray(256));

subplot(4,5,1)
 imshow(uint8(A));
for k = 2:18
    disp(k)
    subplot(4,5,k)
    B = boulanger(A,r);
    imshow(uint8(B));
    A = B;
end
```



2.3 Manipulations élémentaires : modification des valeurs des pixels

On s'intéresse aux traitements ponctuels des images numériques qui consistent à faire subir à chaque pixel une correction ne dépendant que de sa valeur.

Exercice 2.9 (Censure, Bords)
En utilisant une manipulation élémentaire de la matrice (sans faire de boucles et sans utiliser de fonctions prédéfinies) obtenir les images de la figure 2.4 (pour la dernière image, regarder la doc de `meshgrid` sinon utiliser deux boucles). On pourra se baser sur le canevas suivant :

```
clear all
A=imread('lena512.bmp');
B= .... % construire B
subplot(1,2,1)
 imshow(uint8(A));
 title ( "Originale" );
subplot(1,2,2)
 imshow(uint8(B));
 title ( "Transformee" );
```

(a) Original (b) Effacer (c) Bord carré (d) Bord cercle

FIGURE 2.4 – Manipulations élémentaires

Correction

1. Effacer
   `B = A; B(250:300,220:375)= 0;`

2. Bord carré
   `B = A; [r,c] = size(A); B([1:20,r-20:r],:)= 0; B(:,[1:20,c-20:c])= 0;`

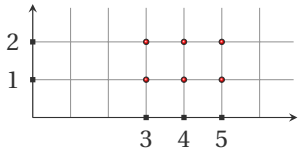
3. Bord cercle
   Avec `meshgrid` (sans boucles) :

```
[l,c] = size(A); r = min(l,c);
[xx,yy] = meshgrid(1:l,1:c);
M = (xx-l/2).^2+(yy-c/2).^2 >= (r/2).^2;
B = A;   B(M) = 255;
```

   Avec boucles :

```
[r,c]=size(A);
B=255*ones(r,c);
for i=1:r
  for j=1:c
    if (i-r/2)^2+(j-c/2)^2 <= (r/2)^2
      B(i,j)=A(i,j);
    end
  end
end
```

   Pour comprendre ce que renvoie la fonction `meshgrid`, considérons les points suivants positionnés sur une grille cartésienne :



© 2022-2023 G. FACCANONI

Il s'agit de l'ensemble de points

$$\{(3,1);(4,1);(5,1);(3,2);(4,2);(5,2)\}.$$

Pour générer deux matrices qui contiennent respectivement les abscisses et les ordonnées correspondantes, on utilise la fonction `meshgrid`.

```
x = 3:5 ;
y = 1:2 ;
[xx,yy] = meshgrid(x,y)

xx =
   3   4   5
   3   4   5

yy =
   1   1   1
   2   2   2
```

Exercice 2.10 (Statistiques, Histogramme)

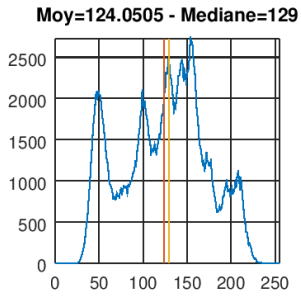
On considère toutes les valeurs des pixels de l'image. Le calcule de la moyenne, de la variance, de l'écart type et de la médiane ainsi que la visualisation de l'histogramme donnent de nombreuses informations sur l'image. L'histogramme h d'une image associe à chaque valeur d'intensité de gris le nombre de pixels prenant cette valeur. La détermination de l'histogramme est donc réalisée en comptant le nombre de pixel pour chaque intensité de l'image. Un histogramme indique, pour chaque valeur entre le noir (0) et le blanc (255), combien il y a de pixels de cette valeur dans l'image ; en abscisse (axe x) le niveau de gris (de 0 à 255) ; en ordonnée (axe y) le nombre de pixels. Les pixels sombres apparaissent à gauche de l'histogramme, les pixels clairs à droite de l'histogramme et les pixels gris au centre de l'histogramme.

1. Calculer la moyenne, l'écart type et la médiane d'une image.
2. Tracer son histogramme.

Correction

1.
```
clear all
clc
A=double(imread('lena512.bmp')); % pour travailler avec des valeurs de type double
moyenne=mean(A(:))
mediane=median(A(:))
ecart_type=std(A(:))
```

2.
```
clear all
A=imread('lena512.bmp');
subplot(2,1,1)
   colormap(gray(256));
   A=double(A);
  imshow(uint8(A));
subplot(2,1,2)
   x=[0:255];
   h=[];
   for i = x
      h=[h, sum(A(:)==i)];
   end
   moy=mean(A(:));
   med=median(A(:));
   plot(x,h,'-',[moy moy],[0,max(h)],[med med
       ],[0,max(h)])
   title([ "Moy=",num2str(moy), " - Mediane=",
       num2str(med)  ])
   axis([0 255 0 max(h)],"square")
   grid()

print('lenahist.png', '-dpng');
```



Remarque : avec Octave, la fonction prédéfinie `imhist` calcule l'histogramme de l'image mais elle nécessite l'installation du package `image`. C'est pourquoi on a préféré ici construire l'histogramme manuellement :

```
clear all
figure(1,"position",get(0,"screensize"))
A=imread('lena512.bmp');

pkg load image
 [counts, nb]=imhist(A,gray(256));
 stem(nb,counts);
 axis([0 255 0 max(counts)])
 grid()

% Rque : h'=counts
```

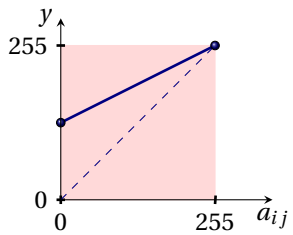
- ⋆ **Une image trop lumineuse aura une moyenne et une médiane supérieures à** 200 **ainsi qu'un histogramme déplacés vers la droite** (on dit que l'image est *brûlée*). **Une image peu lumineuse aura une moyenne et une médiane inférieures à** 100 **ainsi qu'un histogramme déplacés vers la gauche** (on dit que l'image est *bouchée*).
- ⋆ **Une image à faible contraste aura un petit écart type ainsi qu'un histogramme concentré autour de la médiane.** Une image à fort contraste aura un grand écart type ainsi qu'un histogramme étalé.

Suivant les informations données par la moyenne, l'écart type et l'histogramme, on pourra améliorer visuellement l'image en modifiant la valeur de chaque pixel par une fonction f qu'on appliquera à tous les pixels. Pour cela, on pourra se baser sur le canevas suivant :

```
clear all
A=double(imread("lena512.bmp")); % utiliser double pour avoir des calculs precis

f=@(x) .... ; % fonction vectorisee a completer

B=f(A);
subplot(1,3,1)
 plot([0:255], f([0:255]), "b-",[0:255], [0:255], "r--"); % f et identite
 axis([0 255 0 255],"square");
 title("f vs identite")
subplot(1,3,2)
 imshow(uint8(A));
 title ( "Originale" );
subplot(1,3,3)
 imshow(uint8(B));
 title ( "Transformee" );
```

Par exemple, la fonction suivante, appliquée à chaque pixel d'une image, éclaircira l'image :

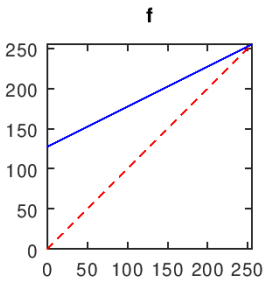
$$\begin{aligned} f\colon [0;255] &\to [0;255] \\ a_{ij} &\mapsto \frac{a_{ij}+255}{2} \end{aligned}$$

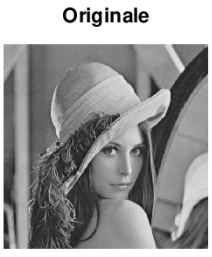


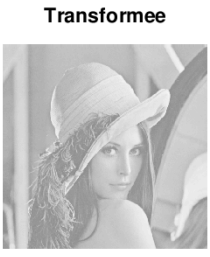
Dans le canevas, il suffit de définir

```
f = @(g) (g+255)/2;
```

et on obtient







Exercice 2.11 (Négatif)

Pour obtenir l'image en négatif de Léna il suffit de prendre le complémentaire par rapport à 255

$$\begin{aligned} f\colon [0;255] &\to [0;255] \\ x &\mapsto 255 - x \end{aligned}$$

© 2022-2023 G. Faccanoni

Appliquer cette transformation pour obtenir l'image 2.5b.

(a) Originale (b) Négatif

FIGURE 2.5 – Négatif

Correction

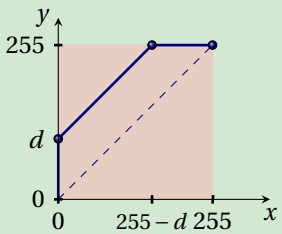
```
f = @(x)255-x;
```

Exercice 2.12 (Luminosité)

1. Pour augmenter la luminosité, il suffit d'ajouter une valeur fixe $d > 0$ à tous les niveaux. Pour diminuer la luminosité il faudra au contraire soustraire une valeur fixe $d > 0$ à tous les niveaux.

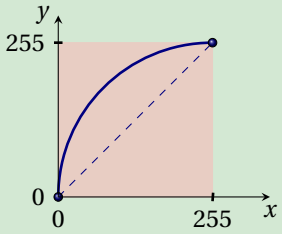
   Appliquer la transformation ci-dessous pour augmenter la luminosité en ajoutant la valeur fixe $d = 50$ à tous les niveaux de gris et obtenir l'image 2.6b.

$$f : [0;255] \to [0;255]$$
$$x \mapsto \begin{cases} x+d & \text{si } x \leq 255-d, \\ 255 & \text{sinon.} \end{cases}$$



2. Il est très mauvais d'augmenter ainsi la luminosité : avec un décalage de d, il n'existera plus aucun point entre 0 et d et les points ayant une valeur supérieure à $255-d$ deviendront des points parfaitement blancs, puisque la valeur maximale possible est 255. La nouvelle image contient des zones dites "brûlées".

   Plutôt que d'utiliser la fonction donnée, **il vaut mieux utiliser une fonction bijective de forte croissance au voisinage de** 0 **et de très faible croissance au voisinage de** 255, comme sur le graphe ci-contre.
   Appliquer cette transformation pour obtenir l'image 2.6c.



(a) Originale (b) Première méthode (c) Deuxième méthode

FIGURE 2.6 – Modification de la luminosité

Correction

1.
```
f = @(g)min(255,g+50);
```

2. On cherche une fonction f continue telle que $f(0) = 0$, $f(255) = 255$ et $f(x) > x$ si $x \in]0;255[$.
   - ⋆ On peut considérer par exemple la fonction $f(x) = \sqrt{255x}$. La fonction f s'écrit `f = @(x)sqrt(255*x);`

- ⋆ On peut considérer le quart de cercle de centre $(255,0)$ et rayon 255. Il a pour équation $(x-255)^2+(y-0)^2=255^2$. Le quart de cercle qui nous intéresse est la partie qui a pour équation $f(x)=\sqrt{255^2-(x-255)^2}$. La fonction f s'écrit `f = @(x)sqrt(255^2-(x-255).^2);`
- ⋆ On peut chercher cette fonction sous la forme d'une parabole, donc d'équation $y=ax^2+bx+c$. L'inégalité se traduit par $a<0$. Les deux conditions $f(0)=0$ et $f(255)=255$ ne donnent que deux équations linéaires en les trois inconnues a,b,c, il faut donc ajouter une autre condition pour fixer une parabole. On peut par exemple choisir la parabole dont le sommet se trouve en $x=255$, *i.e.* la condition $-\frac{b}{2a}=255$. On a alors les trois conditions

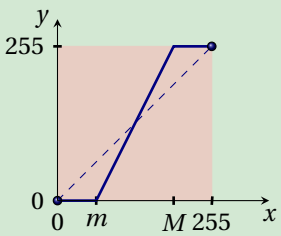
$$\begin{cases} c=0 \\ 255^2a+255b+c=255 \\ 2\times 255a+b=0 \end{cases} \iff \begin{pmatrix} 0 & 0 & 1 \\ 255^2 & 255 & 1 \\ 2\times 255 & 1 & 0 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix}=\begin{pmatrix} 0 \\ 255 \\ 0 \end{pmatrix}$$

On peut résoudre le système à la main et on trouve $c=0$, $a=-\frac{1}{255}$ et $b=2$ ou demander à Matlab/Octave : `[0 0 1;255^2 255 1;2*255 1 0]\[0;255;0]` et finalement la fonction f s'écrit `f = @(x)x.*(255*2-x)/255;`

Exercice 2.13 (Contraste)

1. On se donne une image présentant un histogramme concentré dans l'intervalle $[m;M]$. Les valeurs $m=\min(\mathbb{A})$ et $M=\max(\mathbb{A})$ correspondent aux niveaux de gris extrêmes présents dans cette image. Le recadrage de dynamique consiste à étendre la dynamique de l'image transformée à l'étendue totale $[0;255]$ (*Histogram Stretching*). La transformation de recadrage est donc une application affine qui s'écrit :

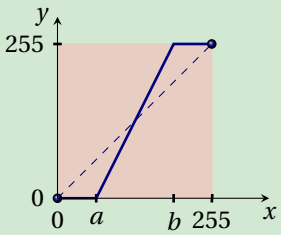
$$f:[0;255]\to[0;255]$$
$$x\mapsto\begin{cases} 0 & \text{si } x\le m \\ \frac{255-0}{M-m}(x-m)+0 & \text{si } m<x<M \\ 255 & \text{si } x\ge M \end{cases}$$



Il s'agit de la droite qui passe par les points $(m,0)$ et $(M,255)$ sur $[m,M]$.
Appliquer cette transformation pour obtenir l'image 2.7b.

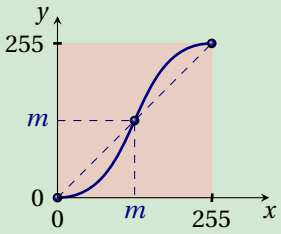
2. On peut augmenter le contraste (*Contrast Stretching*) en "mappant" par une fonction linéaire par morceaux.

$$f:[0;255]\to[0;255]$$
$$x\mapsto\begin{cases} 0 & \text{si } x\le a \\ \frac{255-0}{b-a}(x-a)+0 & \text{si } a<x<b \\ 255 & \text{si } x\ge b \end{cases}$$



Appliquer cette transformation avec $a=64$ et $b=192$ pour obtenir l'image 2.7c.
Un cas particulier s'obtient lorsque $a=b$ (*Contrast Thresholding* ou *seuillage*). Appliquer cette transformation pour obtenir l'image 2.7d

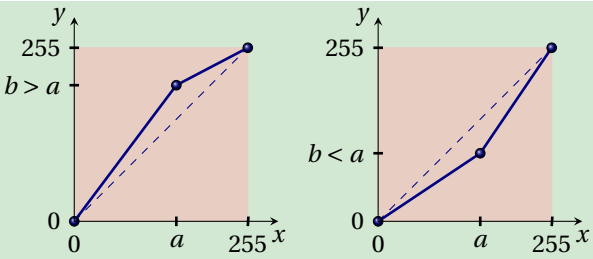
Avec cette transformation, les points les plus blancs auront une valeur égale à b et il n'existera plus aucun point entre b et 255. De même, les points ayant une valeur comprise entre 0 et a deviendront noirs, puisque la valeur minimale est 0. Il y aura donc là perte d'informations. Ainsi il est plus judicieux d'adoucir la courbe comme dans l'image ci-contre :



3. On peut rehausser le contraste en "mappant" par une fonction linéaire par morceaux

© 2022-2023 G. Faccanoni

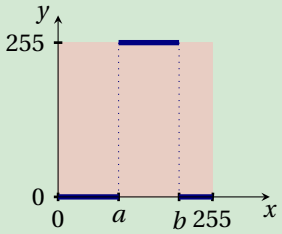
$$f\colon [0;255] \to [0;255]$$
$$x \mapsto \begin{cases} \dfrac{b}{a}x & \text{si } x \le a \\ \dfrac{(255-b)x + 255(b-a)}{255-a} & \text{si } x \ge a \end{cases}$$



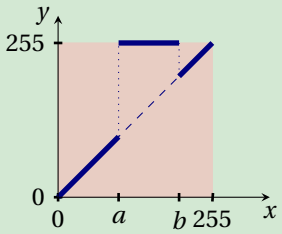
Appliquer cette transformation avec $a = 100$ et $b = 200$ (dilatation de la dynamique des zones claires) et comparer avec $a = 100$ et $b = 50$ (dilatation de la dynamique des zones sombres). Appliquer ces transformations pour obtenir les images 2.7e et 2.7f.

4. On pourrait vouloir mettre en avant certains niveaux de gris dans un certain intervalle. Cette approche peut prendre deux formes : le *gray level slicing without background* et le *gray level slicing with background*. Dans la première approche, une grande valeur est donnée aux pixels dont le niveau de gris appartient à la plage choisie et les autres sont remplacés par 0. Dans la deuxième approche on ne modifie que les pixels à mettre en valeur. Cela correspond aux deux fonctions suivantes :

$$f\colon [0;255] \to [0;255]$$
$$x \mapsto \begin{cases} 0 & \text{si } x \le a \text{ ou } x \ge b \\ 255 & \text{si } a \le x \le b \end{cases}$$



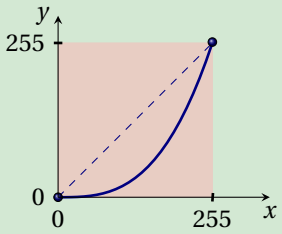
$$f\colon [0;255] \to [0;255]$$
$$x \mapsto \begin{cases} x & \text{si } x \le a \text{ ou } x \ge b \\ 255 & \text{si } a \le x \le b \end{cases}$$



Appliquer ces deux transformations avec $a = 100$ et $b = 155$ pour obtenir les images 2.7g et 2.7h .

5. On peut modifier le contraste en “mappant” par une fonction croissante plus rapidement ou plus lentement que la fonction identité $i\colon [0;255] \to [0;255]$, $i(x) = x$. Par exemple, la fonction suivante modifie la caractéristique de gamma (plus clair si $0 < a < 1$, plus foncé si $a > 1$).

$$f\colon [0;255] \to [0;255]$$
$$x \mapsto 255\left(\frac{x}{255}\right)^a$$



Appliquer cette transformation avec $a = 3$ et $a = 1/3$ pour obtenir les images 2.7i et 2.7j.

(a) Originale (b) Histogram Stretching (c) Contrast Stretching (d) Contrast Thresholding (e) Dilatation de la dynamique des zones claires

(f) Dilatation de la dynamique des zones sombres (g) Gray level slicing without background (h) Gray level slicing with background (i) Non linear (j) Non linear

FIGURE 2.7 – Modification du contraste

Correction

1. *Histogram Stretching :*
   Tout d'abord il faut calculer m et M. On écrit[4]
   soit `M=max(A(:)); m=min(A(:));`
   soit `M=max(max(A)); m=min(min(A));`
   Ensuite il faut définir la fonction f :
   ```
   f = @(x)(x<=m)*0 + (x>=M)*255 + (x>m).*(x<M).*( 255/(M-m)*(x-m));
   ```
   Cependant, on peut juste définir f comme
   ```
   f = @(x)255*(x-m)/(M-m);
   ```
   En effet, on obtiendra $f(x) < 0$ si $x < m$ et $f(g) > 255$ si $x > 255$ et comme on applique la fonction `uint8` à B, les valeurs négatifs seront considérés égales à 0 et les valeurs supérieurs à 255 seront considérés égales à 255.

2. *Contrast Stretching* :
   ```
   a=64; b=192; f = @(x)(x<=a)*0 + (x>=b)*255 + (x>a).*(x<b).*( 255/(b-a)*(x-a));
   ```
   ou, pour la même raison que le point précédent,
   ```
   a=64; b=192; f = @(x)255/(b-a)*(x-a);
   ```
   Si $a = m$ et $b = M$ on retrouve l'*Histogram Stretching*.

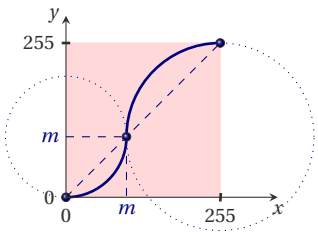
   Contrast Thresholding :
   ```
   a=128; f = @(x)(x<=a)*0 + (x>=a)*255;
   ```

   Amélioration :
   On cherche une fonction f continue telle que $f(0) = 0$, $f(m) = m$, $f(255) = 255$, $f(x) < x$ si $x < m$ et $f(x) > x$ si $x > m$.

   ⋆ On peut considérer les deux quarts de cercle suivants. Le premier est le cercle de centre $(0, m)$ et rayon m qui a pour équation $(x-0)^2 + (y-m)^2 = m^2$ et la partie qui nous intéresse a pour équation $f(x) = m - \sqrt{m^2 - x^2}$. Le deuxième est le cercle de centre $(255, m)$ et rayon $255 - m$ qui a pour équation $(x-255)^2 + (y-m)^2 = (255-m)^2$ et la partie qui nous intéresse a pour équation $f(x) = m + \sqrt{(255-m)^2 - (x-255)^2}$.
   La fonction f s'écrit `f = @(x)cb(x).*(x<m)+ch(x).*(x>=m);` avec `cb = @(x)m-sqrt(m^2-x.^2);` et `ch = @(x)m+sqrt((255-m)^2-(x-m).^2);`

4. Pour comprendre voir cette exemple :
```
A=[1 2;3 4]
A(:)
min(A)
```


© 2022-2023 G. FACCANONI



⋆ On peut choisir de définir f par morceaux avec deux paraboles :

$$f(x) = \begin{cases} p_1(x) & \text{si } x \le m, \\ p_2(x) & \text{si } x \ge m. \end{cases}$$

On a $p_1(0) = 0$, $p_1(m) = m$, p_1 est convexe mais il faut ajouter une condition pour fixer cette parabole, par exemple on demande à ce que le sommet se trouve en 0. On obtient ainsi $p_1(x) = x^2/m$:

```
p1=@(x)(x.^2/m);
```

On a $p_2(255) = 255$, $p_2(m) = m$, p_2 est concave mais il faut ajouter une condition pour fixer cette parabole, par exemple on demande à ce que le sommet se trouve en 255. Si on écrit la parabole sous la forme $p_2(x) = ax^2+bx+c$, on doit résoudre le système linéaire

$$\begin{cases} 255^2 a + 255b + c = 255 \\ m^2 a + mb + c = m \\ 2 \times 255 a + b = 0 \end{cases} \iff \begin{pmatrix} 255^2 & 255 & 1 \\ m^2 & m & 1 \\ 2\times 255 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 255 \\ m \\ 0 \end{pmatrix}$$

On peut bien sûr résoudre ce système à la main, mais on peut demander à Matlab/Octave de le faire pour nous :

```
sol=[255^2 255 1; m^2 m 1; 2*255 1 0]\[255;m;0];
a=sol(1);
b=sol(2);
c=sol(3);
p2=@(x)(a*x.^2+b*x+c);
```

et enfin on a la fonction

```
f=@(x)p1(x).*(x<=m)+ p2(x).*(x>m)
```

3. Dilatation de la dynamique des zones claires :

```
a=100; b=200; f = @(x)(x<=a)*(b/a).*x + (x>a).*( ((255-b)*x+255*(b-a))/(255-a));
```

Dilatation de la dynamique des zones sombres :

```
a=100; b=50; f = @(x)(x<=a)*(b/a).*x + (x>a).*( ((255-b)*x+255*(b-a))/(255-a));
```

4. *Gray level slicing without background :*

```
a=85; b=170; f = @(x)(x<=a)*0 + (x>=b)*0 + (x>a).*(x<b)*255 ;
```

ou, plus simplement

```
a=85; b=170; f = @(x)(x>a).*(x<b)*255 ;
```

Gray level slicing with background :

```
a=85; b=170; f = @(x)(x<=a).*x + (x>=b).*x + (x>a).*(x<b)*255 ;
```

Attention à ne pas écrire `f = @(x)(x<=a).*(x>=b).*x + (x>a).*(x<b)*255 ;` car le produit `(x<=a).*(x>=b)` correspond à $x \le a$ ET $x \ge b$ alors que nous voulons $x \le a$ OU $x \ge b$.

5. *Non linear :*

```
a=3; f = @(x)255*(x/255).^a;
a=1/3; f = @(x)255*(x/255).^a;
```

Exercice 2.14 (Quantification)

Une autre façon de réduire la place mémoire nécessaire pour le stockage consiste à utiliser moins de nombres entiers pour chaque valeur. On peut par exemple utiliser uniquement des nombres entiers entre 0 et 3, ce qui donnera une image avec uniquement 4 niveaux de gris. Une telle opération se nomme quantification.

On peut effectuer une conversion de l'image d'origine vers une image avec 2^{8-k} niveaux de valeurs en effectuant les remplacements suivant : tous les valeurs entre 0 et 2^k sont remplacées par la valeur 0, puis tous les valeurs entre 2^k et

2^{k+1} sont remplacées par la valeur 2^k etc. Appliquer cette transformation pour obtenir les images suivantes :

(a) 16 niveaux de gris ($k=4$) (b) 8 niveaux de gris ($k=5$) (c) 4 niveaux de gris ($k=6$) (d) 2 niveaux de gris ($k=7$)

FIGURE 2.8 – Quantification

Correction

```
clear all

A = imread('lena512.bmp');
colormap(gray(256));
A = double(A);
[row,col] = size(A)

for k = [4,5,6,7]
    figure(k)
    subplot(2,2,1)
    imshow(uint8(A));
    title ( "Original" );
    subplot(2,2,3)
    hist(A(:),0:255);
    subplot(2,2,2)
    B = fix(A/2^k)*2^k; %idivide(A,2^k)*2^k;
    imshow(uint8(B));
    title ( strcat("Quantifiee, k = ",num2str(k)) )
        ;
    subplot(2,2,4)
    hist(B(:),0:255);
    %imwrite(uint8(B),strcat("exo3",num2str(k-3),".
        jpg"),'jpg');
end
```


© 2022-2023 G. FACCANONI

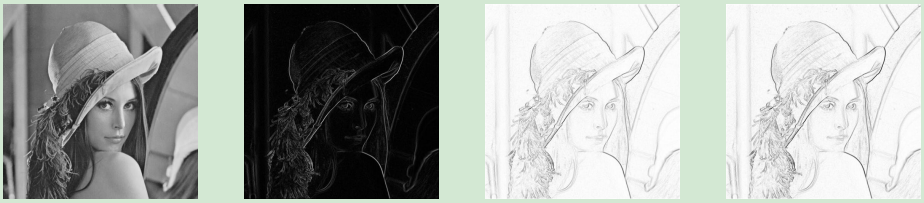
2.4 Détection des bords

Jusqu'à maintenant, le traitement de chaque pixel d'une image était indépendant des pixels voisins. Il s'agit maintenant d'aborder des algorithmes dont le résultat est lié aux pixels voisins.
Afin de localiser des objets dans les images, il est nécessaire de détecter les bords de ces objets. Ces bords correspondent à des zones de l'image où les valeurs des pixels changent rapidement. C'est le cas par exemple lorsque l'on passe du chapeau (qui est clair, donc avec des valeurs grandes) à l'arrière plan (qui est sombre, donc avec des valeurs petites). Afin de savoir si un pixel avec une valeur est le long d'un bord d'un objet, on prend en compte les valeurs de ses quatre voisins (deux horizontalement et deux verticalement) en calculant une valeur $b_{i,j}$ suivant la formule

$$b_{i,j} = \sqrt{(a_{i+1,j} - a_{i-1,j})^2 + (a_{i,j+1} - a_{i,j-1})^2}$$

On peut remarquer que si $b_{i,j} = 0$, alors on a $a_{i-1,j} = a_{i+1,j}$ et $a_{i,j-1} = a_{i,j+1}$. Au contraire, si $b_{i,j}$ est grand, ceci signifie que les pixels voisins ont des valeurs très différentes, le pixel considéré est donc probablement sur le bord d'un objet. Notons que l'on utilise ici seulement 4 voisins, ce qui est différent du calcul de moyennes où l'on utilisait 8 voisins. Ceci est important afin de détecter aussi précisément que possible les bords des objets.

Exercice 2.15 (Détection des bords)
Appliquer cette transformation pour obtenir les images 2.9b et 2.9c. Pour améliorer le rendu, ramener les niveaux de gris à l'intervalle [0;255] par une transformation affine et obtenir l'image 2.9d.



(a) Original (b) Contour (c) Contour Négatif (d) Contour Négatif (normalisé)

FIGURE 2.9 – Masques

Correction

```
clear all
```

```
A  =  double(imread('lena512.bmp'));
colormap(gray(256));

[row,col]  =  size(A);
B  =  zeros(row,col);

I  =  2:row-1;
J  =  2:col-1;
B(I,J)  =  sqrt( (A(I+1,J)-A(I-1,J)).^2+(A(I,J+1)-A(I,J-1)).^2 );

% equivaut aux boucles suivantes :
%for i = 2:row-1
%  for j = 2:col-1
%    B(i,j) = sqrt( (A(i+1,j)-A(i-1,j))^2+(A(i,j+1)-A(i,j-1))^2  );
%  end
%end

subplot(1,4,1)
 imshow(uint8(A));
  title ( "Original" );
  %imwrite(uint8(A),'exo6orig.jpg','jpg');
 subplot(1,4,2)
  imshow(uint8(B));
  title ( "Contour" );
  %imwrite(uint8(B),'exo6grad.jpg','jpg');
 subplot(1,4,3)
```

```
 imshow(uint8(255-B));
 title ( "Negatif du contour" );
 %imwrite(uint8(255-B),'exo6gradNeg.jpg','jpg');
subplot(1,4,4)
% on se ramene a [0;255]
 m = min(min(B(I,J)))
 M = max(max(B(I,J)))
 B = 255/(M-m).*(B-m);
 imshow(uint8(255-B));
 title ( "Negatif du contour (normalise)" );
 %imwrite(uint8(255-B),'.jpg','jpg');
```

2.5 ⭐Masques (filtrage par convolution)

Exercice 2.16 (Enlever du bruit par moyennes locales)

Les images sont parfois de mauvaise qualité. Un exemple typique de défaut est le bruit qui apparaît quand une photo est sous-exposée, c'est-à-dire qu'il n'y a pas assez de luminosité. Ce bruit se manifeste par de petites fluctuation aléatoires des niveaux de gris. La figure 2.10b montre une image bruitée obtenue par modification de l'image 2.10a par les instructions

```
sigma=50;
B=A+randn(size(A))*sigma;
```

Afin d'enlever le bruit dans les images, il convient de faire subir une modification aux valeurs de pixels. L'opération la plus simple consiste à remplacer la valeur a_{ij} de chaque pixel par la moyenne de a_{ij} et des 8 valeurs $a_{i-1,j-1}$, $a_{i-1,j}$, $a_{i-1,j+1}$, $a_{i,j-1}$, $a_{i,j+1}$, $a_{i+1,j-1}$, $a_{i+1,j}$, $a_{i+1,j+1}$ des 8 pixels voisins de a_{ij}.
En effectuant cette opération pour chaque pixel, on supprime une partie du bruit, car ce bruit est constitué de fluctuations aléatoires, qui sont diminuées par un calcul de moyennes. Appliquer cette transformation pour obtenir l'image 2.10c Cependant, tout le bruit n'a pas été enlevé par cette opération. Afin d'enlever plus de bruit, on peut moyenner plus de valeurs autour de chaque pixel. Le moyennage des pixels est très efficace pour enlever le bruit dans les images, malheureusement il détruit également une grande partie de l'information de l'image. on peut en effet s'apercevoir que les images obtenues par moyennage sont floues (c'est justement l'effet recherché lors de la pixellisation). Ceci est en particulier visible près des contours, qui ne sont pas nets.
Afin de réduire ce flou, on peut remplacer la moyenne par la médiane. Appliquer cette transformation pour obtenir l'image 2.10d

(a) Originale (b) Image bruitée (c) Image débruitée par moyenne (d) Image débruitée par médiane

FIGURE 2.10 – Dé bruitage

Correction

```
clear all, close all

A=double(imread('lena512.bmp'));
[r,c]=size(A);

sigma=50;
B=A+randn([r,c])*sigma;
C=B;
D=B;
```

```
for i=2:r-2
   for j=2:c-2
     sousB=B(i-1:i+1,j-1:j+1);
     C(i,j)=mean(sousB(:));
     D(i,j)=median(sousB(:));
   end
end

subplot(1,4,1)
imshow(uint8(A),[])
title("Image originale");
```


© 2022-2023 G. FACCANONI

```
imwrite(uint8(A),strcat("exoAA.jpg"),'jpg');

subplot(1,4,2)
imshow(uint8(B),[])
title("Image bruitee");
imwrite(uint8(B),strcat("exoBB.jpg"),'jpg');

subplot(1,4,3)
```

```
imshow(uint8(C),[])
title("Image debruitee par moyenne");
imwrite(uint8(C),strcat("exoCC.jpg"),'jpg');

subplot(1,4,4)
imshow(uint8(D),[])
title("Image debruitee par mediane");
imwrite(uint8(D),strcat("exoDD.jpg"),'jpg');
```

Exercice 2.17 (Pixellisation)

La pixellisation est souvent utilisée pour flouter une image, en partie ou dans sa totalité. La zone pixelisée fait apparaître des blocs carrés contenant $n \times n$ pixels de couleur uniforme. On observe la même chose en agrandissant une image jusqu'au niveau du pixel, sauf que là, chaque carré représente 1 pixel.
Lorsqu'on diminue la résolution, les images sont de plus en plus petites (dans l'exercice à chaque étape on divisait par deux le nombre de lignes et de colonnes de la matrice). Ici nous voulons obtenir des images qui ont toujours la même dimension mais avec cet effet de pixellisation. Pour arriver à ce résultat, le principe de l'algorithme est de diviser l'image en blocs carrés de largeur n pixels et remplacer tous les pixels du bloc par leur moyenne. On parcourt cette image non pas pixel par pixel mais par blocs de $n \times n$ pixels.
Appliquer cette transformation avec $n = 16$ pour obtenir l'image 2.11b

(a) Originale (b) $n = 16$

FIGURE 2.11 – Pixellisation

Correction

```
clear all

A = double(imread('lena512.bmp'));
colormap(gray(256));
[r,c] = size(A);
n = 16;
B = A;
for i=1:n:r-n+1
   for j=1:n:c-n+1
     % octave
      B(i:i+n-1,j:j+n-1) = mean(A(i:i+n-1,j:j+n-1)
          (:));
     %%%%%%% matlab
```

```
     %mask = A(i:i+n-1,j:j+n-1);
      %B(i:i+n-1,j:j+n-1) = mean(mask(:));
   end
end

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Original" );
subplot(1,2,2)
 imshow(uint8(B));
 title (strcat([ "Pixellisation n=" num2str(n)]) );
 %imwrite(uint8(B),'pixellisation.jpg','jpg');
```

★Convolution

Le principe du filtrage est de modifier la valeur des pixels d'une image, généralement dans le but d'améliorer son aspect. En pratique, il s'agit de créer une nouvelle image en se servant des valeurs des pixels de l'image d'origine.
Un filtre est une transformation mathématique (appelée produit de convolution) permettant de modifier la valeur d'un pixel en fonction des valeurs des pixels avoisinants, affectées de coefficients. Le filtre est représenté par un tableau (une matrice), caractérisé par ses dimensions et ses coefficients, dont le centre correspond au pixel concerné. La somme des coefficients doit faire 1.
Voici un exemple. Soit $\mathbb{A}$ une matrice et considérons le pixel $a_{i,j}$ et les 9 pixels voisins. On peut construire une matrice $\mathbb{B}$ où le pixel $b_{i,j}$ est une combinaison linéaire (*i.e.* une moyenne pondérée) de tous ces pixels : [5]

$$b_{i,j} = c_1 a_{i-1,j-1} + c_2 a_{i-1,j} + c_3 a_{i-1,j+1} + c_4 a_{i,j-1} + c_5 a_{i,j} + c_6 a_{i,j+1} + c_7 a_{i+1,j-1} + c_8 a_{i+1,j} + c_9 a_{i+1,j+1}$$

On peut visualiser cela comme une masque : on considère $\mathbb{A}_{\substack{i-1,\dots,i+1 \\ j-1,\dots,j+1}}$ la sous-matrice carrée d'ordre 3 et le masque $\mathbb{M}$

5. La relation ci-dessus se généralise aux masques de convolution 5×5, 7×7, etc.

suivantes :

$$\mathbb{A}_{\substack{i-1,\dots,i+1\\ j-1,\dots,j+1}} = \begin{array}{|ccc|}\hline a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1}\\ a_{i,j-1} & a_{i,j} & a_{i,j+1}\\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1}\\ \hline\end{array} \qquad \mathbb{M} = \begin{array}{|ccc|}\hline c_1 & c_2 & c_3\\ c_4 & c_5 & c_6\\ c_7 & c_8 & c_9\\ \hline\end{array}$$

Rappelons que les valeurs des composantes des pixels sont des nombres entiers compris entre 0 et 255. Si les nouvelles valeurs ne sont plus des entiers, il faudra les arrondir. Il peut même arriver que la nouvelle valeur ne soit plus comprise entre 0 et 255, il faudra alors choisir si les écrêter ou appliquer une transformation affine.
De plus, on voit que la relation ne peut s'appliquer sur les bords de l'image. Il faut donc prévoir un traitement spécial pour ces rangées. La solution la plus simple, consiste à remplir ces rangées avec un niveau constant (par exemple noir). Une autre solution est de répliquer sur ces rangées les rangées voisines. Dans ces deux cas, l'information sur les bords est perdue. On peut aussi décider de ne pas modifier ces rangées, ce que nous allons faire. En tout cas, on évite de réduire la taille de l'image.

Exercice 2.18 (Devine le résultat)

Appliquer une fois le masque suivant à la matrice $\mathbb{A} = \begin{pmatrix} 0&0&0\\ 0&1&0\\ 0&0&0\end{pmatrix}$:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Correction

Seul le pixel en position $(2,2)$ sera modifié car on a décidé de ne pas modifier les pixels de bord :

$$b_{2,2} = \frac{1}{9}\sum_{i,j=1}^{3} a_{ij} = \frac{1}{9}a_{2,2} = \frac{1}{9}$$

ainsi

$$\mathbb{B} = \begin{pmatrix} 0 & 0 & 0\\ 0 & 1/9 & 0\\ 0 & 0 & 0\end{pmatrix}$$

Exercice 2.19 (Lissage, floutage, accentuation)

Écrire une fonction qui prend en entrée $\mathbb{A}$ (l'image) et $\mathbb{M}$ le masque (matrice d'ordre 3) et renvoie $\mathbb{B}$ l'image modifiée. Avant d'afficher $\mathbb{A}$ ou $\mathbb{B}$ appliquer une transformation affine pour que les valeurs soient comprises entre 0 et 255.

1. *Lissage (filtre passe-bas).* On veut lisser les endroits à fort gradient pour enlever du bruit ou flouter une image. Le principe de la plupart des méthodes de débruitage est simple : il consiste à remplacer la valeur d'un pixel par la valeur moyenne des pixels voisins. La réduction du bruit s'accompagne d'une réduction de la netteté. Appliquer 50 fois le masque suivant pour obtenir l'image 2.12b.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

2. *Floutage d'une partie.* Appliquer 50 fois le masque précédent à une sous-matrice bien choisie pour obtenir l'image 2.12c.
3. *Accentuation (filtre passe-haut).* Appliquer 1 fois le masque suivant pour obtenir l'image 2.12d.

0	−1/2	0
−1/2	3	−1/2
0	−1/2	0

4. *Rehaussement.* L'objectif de rehaussement de contours est de rendre plus clairs et nets les contours en réduisant la transition de l'intensité. Donc un opérateur de rehaussement vise à remplacer le pixel central par la somme des différences avec ses voisins. Dans un premier temps, appliquer simplement le masque suivant à l'image de départ ce qui permet de détecter les contours. Puis retrancher ce résultat multiplié par un coefficient $\alpha = 0.1$ à l'image de départ pour obtenir l'image 2.12e. Cette dernière opération permet dans le cas d'un échelon moue (contour pas

© 2022-2023 G. Faccanoni

très distingué) de rehausser le contour si α est positif.

1/8	1/8	1/8
1/8	−1	1/8
1/8	1/8	1/8

5. *Contours.* Ci dessous on a indiqué deux masques. Appliquer le premier masque à l'image de départ pour obtenir une image qu'on appellera $\mathbb{X}$. Appliquer le deuxième masque à l'image de départ pour obtenir une image qu'on appellera $\mathbb{Y}$. Afficher l'image associée à la matrice $\mathbb{G}$ définie par $\mathbb{G}_{ij} = \sqrt{\mathbb{X}_{ij}^2 + \mathbb{Y}_{ij}^2}$ pour obtenir l'image 2.12f.

1/8	0	−1/8
1/4	0	−1/4
1/8	0	−1/8

1/8	1/4	1/8
0	0	0
−1/8	−1/4	−1/8

Même exercice avec les deux masques :

0	−1	0
0	0	0
0	1	0

0	0	0
−1	0	1
0	0	0


(a) Original


(b) Floutage


(c) Floutage sous-matrice


(d) Filtre passe-haut


(e) Rehaussement


(f) Contours

FIGURE 2.12 – Masques

Correction
D'abord on définit la fonction :

```
function Bnew=myflou(A,K,n)
   [r,c]=size(A);
   Bold=A;
   Bnew=Bold;
   for t=1:n
      Bnew(2:r-1,2:c-1) =K(1)*Bold(1:r-2,1:c-2)+K(2)*Bold(1:r-2,2:c-1)+K(3)*Bold(1:r-2,3:c);
      Bnew(2:r-1,2:c-1)+=K(4)*Bold(2:r-1,1:c-2)+K(5)*Bold(2:r-1,2:c-1)+K(6)*Bold(2:r-1,3:c);
      Bnew(2:r-1,2:c-1)+=K(7)*Bold(3:r,1:c-2)+K(8)*Bold(3:r,2:c-1)+K(9)*Bold(3:r,3:c);
      % on se ramene a [0;255], inutile de diviser avant
      m=min(Bnew(:));
      M=max(Bnew(:));
      Bnew=255/(M-m).*(Bnew-m);
      Bold=Bnew;
   end
end
```

Puis on fait appelle à cette fonction en changeant le masque et le nombre d'itérations :

1.
```
clear all

A = double(imread('lena512.bmp'));
% on se ramene a [0;255]
m = min(A(:));
M = max(A(:));
A = 255/(M-m).*(A-m);
colormap(gray(256));
```

```
B = myflou(A
    ,[1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9],50);

subplot(1,3,1)
 imshow(uint8(A));
 title ( "Original" );
 %imwrite(uint8(A),'exo5orig.jpg','jpg');

subplot(1,3,2)
```

```
imshow(uint8(B));
title ( "Moyenne 9" );
%imwrite(uint8(B),'exo5flout.jpg','jpg');
```

```
subplot(1,3,3)
 imshow(uint8(abs(B-A)));
 title ( "|moy(A)-A|" );
 %imwrite(uint8(abs(B-A)),'exo5err.jpg','jpg');
```

2.
```
clear all

A = double(imread('lena512.bmp'));
% on se ramene a [0;255]
m = min(A(:));
M = max(A(:));
A = 255/(M-m).*(A-m);
colormap(gray(256));

%B = myflou(A
    ,[-1/9,-1/9,-1/9,-1/9,1,-1/9,-1/9,-1/9,-1/9],1)
    ;
B = myflou(A,[0,-1/2,0,-1/2,3,-1/2,0,-1/2,0],1);
```

```
subplot(1,3,1)
 imshow(uint8(A));
 title ( "Original" );
 %imwrite(uint8(A),'exo52orig.jpg','jpg');

subplot(1,3,2)
 imshow(uint8(B));
 title ( "Filtre rehausseur de contraste" );
 %imwrite(uint8(B),'exo52flout.jpg','jpg');

subplot(1,3,3)
 imshow(uint8(abs(B-A)));
 title ( "|reh(A)-A|" );
 %imwrite(uint8(abs(B-A)),'exo52err.jpg','jpg');
```

3.
```
clear all

A = double(imread('lena512.bmp'));
% on se ramene a [0;255]
m = min(A(:));
M = max(A(:));
A = 255/(M-m).*(A-m);
colormap(gray(256));

B = myflou(A,[-1,-1,-1,-1,8,-1,-1,-1,-1],1);

subplot(1,3,1)
 imshow(uint8(A));
```

```
 title ( "Original" );
 %imwrite(uint8(A),'exo53orig.jpg','jpg');

subplot(1,3,2)
 imshow(uint8(B));
 title ( "Detection des contours" );
 %imwrite(uint8(B),'exo53flout.jpg','jpg');

subplot(1,3,3)
 imshow(uint8(abs(B-A)));
 title ( "|dc(A)-A|" );
 %imwrite(uint8(abs(B-A)),'exo53err.jpg','jpg');
```

4.
```
clear all

A = double(imread('lena512.bmp'));
% on se ramene a [0;255]
m = min(A(:));
M = max(A(:));
A = 255/(M-m).*(A-m);
colormap(gray(256));

B = myflou(A
    ,[1/8,1/8,1/8,1/8,-1,1/8,1/8,1/8,1/8],1);
```

```

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Original" );
 %imwrite(uint8(A),'5rehaussement.jpg','jpg');

subplot(1,2,2)
 N = A-0.1*B;
 imshow(uint8(N));
 title ( "Rheaussement" );
 %imwrite(uint8(N),'5rehaussement.jpg','jpg');
```

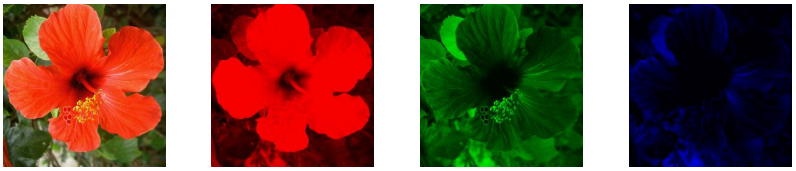
5.
```
clear all

A = double(imread('lena512.bmp'));
% on se ramene a [0;255]
m = min(A(:));
M = max(A(:));
A = 255/(M-m).*(A-m);
colormap(gray(256));

B1 = myflou(A
    ,[1/8,0,-1/8,1/4,0,-1/4,1/8,0,-1/8],1);
B2 = myflou(A
```

```
    ,[1/8,1/4,1/8,0,0,0,-1/8,-1/4,-1/8],1);
G = sqrt(B1.^2+B2.^2);

subplot(1,2,1)
 imshow(uint8(A));
 title ( "Original" );

subplot(1,2,2)
 imshow(uint8(G));
 title ( "Contours" );
 %imwrite(uint8(G),'6cont.jpg','jpg');
```


© 2022-2023 G. Faccanoni

2.6 Images à couleurs

Une image couleur est en réalité composée de trois images, afin de représenter le rouge, le vert, et le bleu. Chacune de ces trois images s'appelle un canal. Cette représentation en rouge, vert et bleu mime le fonctionnement du système visuel humain.
La figure suivante montre la décomposition d'une image couleur en ses trois canaux constitutifs.



(a) Originale (b) Canal rouge (c) Canal Vert (d) Canal Bleu

FIGURE 2.13 – Canaux RGB

Chaque pixel de l'image couleur contient ainsi trois nombres (r, g, b), chacun étant un nombre entier entre 0 et 255. Un tel nombre est représentable sur 8 bits et s'appelle un octet. Il y a donc $256^3 = 2^{24} = 16777216$ couleurs possibles.
Si le pixel est égal à $(r, g, b) = (255, 0, 0)$, il ne contient que de l'information rouge, et est affiché comme du rouge. De façon similaire, les pixels valant $(0, 255, 0)$ et $(0, 0, 255)$ sont respectivement affichés vert et bleu.

On peut afficher à l'écran une image couleur à partir de ses trois canaux (r, g, b) en utilisant les règles de la synthèse additive des couleurs : le triplet (0, 0, 0) correspond à un pixel noir alors qu'un pixel blanc est donné par (255, 255, 255). La figure suivante montre les règles de composition cette synthèse additive des couleurs. Un pixel avec les valeurs $(r, g, b) = (255, 0, 255)$ est un mélange de rouge et de vert, il est ainsi affiché comme jaune.

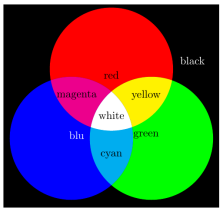
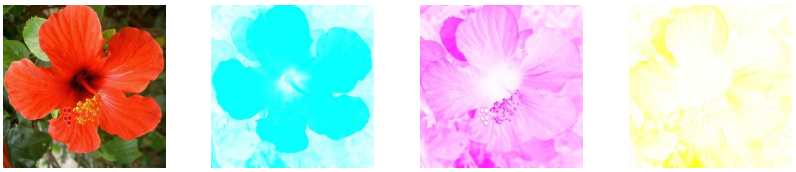


FIGURE 2.14 – Synthèse additive des couleurs

Une autre représentation courante pour les images couleurs utilise comme couleurs de base le cyan, le magenta et le jaune. On calcule les trois nombres (c, m, j) correspondant à chacun de ces trois canaux à partir des canaux rouge, vert et bleu (r, v, b) comme suit

$$\begin{cases} c = 255 - r, \\ m = 255 - v, \\ j = 255 - b. \end{cases}$$

Par exemple, un pixel de bleu pur $(r, v, b) = (0, 0, 255)$ va devenir $(c, m, j) = (255, 255, 0)$. La figure suivante montre les trois canaux (c, m, j) d'une image couleur.



(a) Originale (b) Canal cyan (c) Canal magenta (d) Canal jaune

FIGURE 2.15 – Canaux CMY

Afin d'afficher une image couleur à l'écran à partir des trois canaux (c, m, j), on doit utiliser la synthèse soustractive des couleurs. La figure suivante montre les règles de composition cette synthèse soustractive. Notons que ces règles sont celles que l'on utilise en peinture, lorsque l'on mélange des pigments colorés. Le cyan, le magenta et le jaune sont appelés couleurs primaires.

FIGURE 2.16 – Synthèse soustractive des couleurs

On peut donc stocker sur un disque dur une image couleur en stockant les trois canaux, correspondant aux valeurs (r,g,b) ou (c,m,j).
On peut modifier les images couleur tout comme les images en niveaux de gris. La façon la plus simple de procéder consiste à appliquer la modification à chacun des canaux.

Exercice 2.20 (Devine le résultat)
Devine le résultat :

```
clear all
mat = zeros(2,2,3); % images en couleurs
mat(1,1,1) = 255;   % mat(i,j,1)  = > R(i,j)
mat(2,1,2) = 255;   % mat(i,j,2)  = > G(i,j)
mat(1,2,3) = 255;   % mat(i,j,3)  = > B(i,j)
%mat(2,2,:) = 255;
%mat(2,2,[1,2]) = 255;
%mat(2,2,[1,3]) = 255;
%mat(2,2,[2,3]) = 255;
imshow(uint8(mat));
```

Correction
Il s'agit des trois matrices $\mathbb{R}$, $\mathbb{G}$, $\mathbb{B}$ suivantes (correspondantes aux trois canaux RGB) :

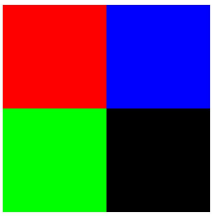
$$\mathbb{R}=\begin{pmatrix}255 & 0\\ 0 & 0\end{pmatrix} \qquad \mathbb{G}=\begin{pmatrix}0 & 0\\ 255 & 0\end{pmatrix} \qquad \mathbb{B}=\begin{pmatrix}0 & 255\\ 0 & 0\end{pmatrix}$$

ainsi

$255r+0g+0b$	$0r+0g+255b$
$0r+255g+0b$	$0r+0g+0b$

$=$

red	blue
green	black



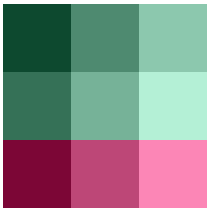
Exercice 2.21 (Affichage image couleur)
Soit les trois matrices

$$\mathbb{R}=\begin{pmatrix}0 & 64 & 128\\ 64 & 128 & 192\\ 128 & 192 & 255\end{pmatrix} \qquad \mathbb{G}=\begin{pmatrix}64 & 128 & 192\\ 128 & 192 & 255\\ 0 & 64 & 128\end{pmatrix} \qquad \mathbb{B}=\begin{pmatrix}128 & 192 & 255\\ 0 & 64 & 128\\ 64 & 128 & 192\end{pmatrix}$$

Après avoir défini l'hypermatrice $\mathbb{A}$ par l'instruction `A=cat(3, R,G,B );` afficher l'image associée.

Correction

```
clear all

R = [0 64 128; 64 128 192; 128 192 255];
G = [64 128 192; 128 192 255; 0 64 128];
B = [128 192 255; 0 64 128; 64 128 192];
A = cat(3, R,G,B );
imshow(uint8(A));
%imwrite(uint8(A),'exo1Color.jpg','jpg');
```



Exercice 2.22
On considère l'image 2.17a. Obtenir les autres images par permutations des canaux.

© 2022-2023 G. Faccanoni


(a) RGB


(b) BRG


(c) GRB


(d) RBG


(e) BGR


(f) GBR

FIGURE 2.17 – Permutations

Correction

```
clear all

A = double(imread('hibiscus.png'));
% size(A)
R = A(:,:,1);
G = A(:,:,2);
B = A(:,:,3);

subplot(2,3,1)
 imshow(uint8(A));
 title ( "RGB" );
 %imwrite(uint8(A),'exo2ColorRGB.jpg','jpg');

subplot(2,3,2)
 N = cat(3, B,R,G );
 % ou, en modifiant A
 % A(:,:,[1,2,3]) = A(:,:,[3,1,2]);
 imshow(uint8(N));
 title ( "BRG" );
 %imwrite(uint8(N),'exo2ColorBRG.jpg','jpg');

subplot(2,3,3)
 N = cat(3, G,R,B );
 imshow(uint8(N));
 title ( "GRB" );
 %imwrite(uint8(N),'exo2ColorGRB.jpg','jpg');

subplot(2,3,4)
 N = cat(3, R,B,G );
 imshow(uint8(N));
 title ( "RBG" );
 %imwrite(uint8(N),'exo2ColorRBG.jpg','jpg');

subplot(2,3,5)
 N = cat(3, B,G,R );
 imshow(uint8(N));
 title ( "BGR" );
 %imwrite(uint8(N),'exo2ColorBGR.jpg','jpg');

subplot(2,3,6)
 N = cat(3, G,B,R );
 imshow(uint8(N));
 title ( "GBR" );
 %imwrite(uint8(N),'exo2ColorGBR.jpg','jpg');
```

Exercice 2.23 (Négatif)

On considère l'image 2.18a. Obtenir l'image 2.18b en considérant le négatif pour tous les canaux.


(a) Original


(b) Canaux négatifs

FIGURE 2.18 – Négatif

Correction

```
clear all

A = double(imread('hibiscus.png'));
size(A)

subplot(1,2,1)
 imshow(uint8(A));
 title ( "RGB" );
 %imwrite(uint8(A),'exo5ColorRGB.jpg','jpg');

subplot(1,2,2)
 N = 255-A;
 imshow(uint8(N));
 title ( "Negatif" );
 %imwrite(uint8(N),'exo5ColorRGBNeg.jpg','jpg');
```

Exercice 2.24 (Modification d'un seul canal)

On considère l'image 2.19a. Obtenir l'image 2.19b en considérant le négatif pour le canal rouge.

(a) Original (b) Canal rouge négatif

FIGURE 2.19 – Modification du canal de rouge

Correction

```
clear all

A = double(imread('hibiscus.png'));
size(A)

subplot(1,2,1)
 imshow(uint8(A));
 title ( "RGB" );
```

```
%imwrite(uint8(A),'exo3ColorRGB.jpg','jpg');

subplot(1,2,2)
 M = (255-A(:,:,1));
 N = cat(3, M,A(:,:,2),A(:,:,3));
 imshow(uint8(N));
 title ( "Modification canal rouge" );
%imwrite(uint8(N),'exo3Color0GB.jpg','jpg');
```

On peut calculer une image en niveaux de gris à partir d'une image couleur en moyennant les trois canaux :

$$b_{ij} = \frac{r_{ij} + g_{ij} + b_{ij}}{3}$$

qui s'appelle la luminance de la couleur. La figure ci-contre montre l'image de luminance associée à une image couleur.

(a) Originale (b) Luminance

FIGURE 2.20 – Luminance

Exercice 2.25 (RGB → niveau de gris)

On considère l'image RGB 2.21a.

① Obtenir l'image en niveaux de gris 2.21b en considérant, pour chaque pixel, la moyenne arithmétique des trois canaux (luminance).

② L'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités de couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris de chaque pixel en fonction des trois composantes est

$$\text{gray} = E(0.21 \cdot \text{r} + 0.72 \cdot \text{g} + 0.07 \cdot \text{b})$$

où $E(\cdot)$ désigne la partie entière. Obtenir l'image en niveaux de gris 2.21c selon cette moyenne.

③ Une autre moyenne consiste à prendre, **pour chaque pixel**, la moyenne du canal le plus présent et celui le moins présent :

$$\text{gray} = E\left(\frac{\max\{r,g,b\} + \min\{r,g,b\}}{2}\right).$$

Obtenir l'image en niveaux de gris 2.21d selon cette moyenne.

© 2022-2023 G. FACCANONI

(a) Original (b) Moyenne arithmétique (c) Moyenne pondérée (d) Moyenne max-min

FIGURE 2.21 – RGB → niveau de gris

Correction

```
clear all
```

```
A = double(imread('hibiscus.png'));
[row,col,couches] = size(A)

 R = A(:,:,1);
 G = A(:,:,2);
 B = A(:,:,3);

subplot(1,4,1)
 imshow(uint8(A));
 title ( "RGB" );
 %imwrite(uint8(A),'exo6Color.jpg','jpg');

subplot(1,4,2)
 N = floor(R+G+B)/3;
 imshow(uint8(N));
 title ( "Aritmetique" );
 %imwrite(uint8(N),'exo6Color1.jpg','jpg');

subplot(1,4,3)
 N = floor(0.21*R+0.72*G+0.07*B);
 imshow(uint8(N));
 title ( "Ponderee" );
 %imwrite(uint8(N),'exo6Color2.jpg','jpg');

subplot(1,4,4)
 N = ( max(max(R,G),B) + min(min(R,G),B) ) /2 ;
 imshow(uint8(N));
 title ( "Max-Min" );
 %imwrite(uint8(N),'exo6Color3.jpg','jpg');
```

Exercice 2.26 (Fusion de deux images)

Considérons les deux images 2.22a et 2.22e (qui ont la même dimension). Superposer les deux images. Créer un film qui montre le passage de l'une à l'autre.

(a) Première image (b) Intermédiaire (c) Intermédiaire (d) Intermédiaire (e) Deuxième image

FIGURE 2.22 – Morphing

Correction

```
clear all; clc;

[fig1,map1] = imread("s1.png");
im1 = ind2rgb(fig1,map1);
[fig2,map2] = imread("s2.png");
im2 = ind2rgb(fig2,map2);
N = 16;
for k = 1:N
   %subplot(4,4,k)
   morphing = (k/N)*im1+(1-k/N)*im2; % NE PAS
       APPELER LE FICHIER COMME LA MATRICE!
   imshow(morphing);
   pause(0.5);
end
%imwrite(0.9*im1+0.1*im2,'s12a.png','png')
%imwrite(0.5*im1+0.5*im2,'s12.png','png')
%imwrite(0.1*im1+0.9*im2,'s12c.png','png')
```

2.7 ⭐Décomposition en valeurs singulières et compression JPG

Tout comme pour la réduction du nombre de pixels, la réduction du nombre de niveaux de gris influe beaucoup sur la qualité de l'image. Afin de réduire au maximum la taille d'une image sans modifier sa qualité, on utilise des méthodes plus complexes de compression d'image. Une des première méthode s'appelle JPEG, basée sur la décomposition en valeurs singulières. La méthode la plus efficace s'appelle JPEG-2000. Elle utilise la théorie des ondelettes. Pour en savoir plus à ce sujet, vous pouvez consulter cet article d'Erwan Le Pennec.
Soit $\mathbb{A} \in \mathbb{R}^{n\times p}$ une matrice rectangulaire. Un théorème démontré officiellement en 1936 par C. Eckart et G. Young affirme que toute matrice rectangulaire $\mathbb{A}$ se décompose sous la forme

$$\mathbb{A} = \mathbb{U}\mathbb{S}\mathbb{V}^T$$

avec $\mathbb{U} \in \mathbb{R}^{n\times n}$ et $\mathbb{V} \in \mathbb{R}^{p\times p}$ des matrices orthogonales (*i.e.* $\mathbb{U}^{-1} = \mathbb{U}^T$ et $\mathbb{V}^{-1} = \mathbb{V}^T$) et $\mathbb{S} \in \mathbb{R}^{n\times p}$ une matrice diagonale qui contient les r valeurs singulières de $\mathbb{A}$, $r = \min\{n, p\}$, $\sigma_1 \geq \sigma_2 \geq \sigma_r \geq 0$. Ce qui est remarquable, c'est que n'importe quelle matrice admet une telle décomposition alors que la décomposition en valeurs propres (la diagonalisation d'une matrice) n'est pas toujours possible.
Notons $\mathbf{u}_i$ et $\mathbf{v}_i$ les vecteurs colonne des matrices $\mathbb{U}$ et $\mathbb{V}$. La décomposition s'écrit alors

$$\mathbb{A} = \mathbb{U}\mathbb{S}\mathbb{V}^T = \underbrace{\begin{pmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_r & \mathbf{u}_{r+1} & \dots & \mathbf{u}_n\end{pmatrix}}_{n\times n} \underbrace{\begin{pmatrix}\sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0\end{pmatrix}}_{n\times p} \underbrace{\begin{pmatrix}\mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_r^T \\ \mathbf{v}_{r+1}^T \\ \vdots \\ \mathbf{v}_p^T\end{pmatrix}}_{p\times p}$$

$$= \underbrace{\begin{pmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_r\end{pmatrix}}_{n\times r} \underbrace{\begin{pmatrix}\sigma_1 & & \\ & \ddots & \\ & & \sigma_r\end{pmatrix}}_{r\times r} \underbrace{\begin{pmatrix}\mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_r^T\end{pmatrix}}_{r\times p} = \sum_{i=1}^{r} \sigma_i \underbrace{\mathbf{u}_i \times \mathbf{v}_i^T}_{r\times r}$$

Pour calculer ces trois matrices on remarque que $\mathbb{A} = \mathbb{U}\mathbb{S}\mathbb{V}^T = \mathbb{U}\mathbb{S}\mathbb{V}^{-1}$ et $\mathbb{A}^T = \mathbb{V}\mathbb{S}\mathbb{U}^T = \mathbb{V}\mathbb{S}\mathbb{U}^{-1}$ ainsi, pour $i = 1,\dots,r$, en multipliant par $\mathbb{A}$ à gauche $\mathbb{A}^T\mathbf{u}_i = \sigma_i\mathbf{v}_i$ et en multipliant par $\mathbb{A}$ à droite $\mathbb{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ on obtient

$$\mathbb{A}\mathbb{A}^T\mathbf{u}_i = \sigma_i\mathbb{A}\mathbf{v}_i = \sigma_i^2\mathbf{u}_i, \qquad \text{pour } i = 1,\dots,r$$
$$\mathbb{A}^T\mathbb{A}\mathbf{v}_i = \sigma_i\mathbb{A}^T\mathbf{u}_i = \sigma_i^2\mathbf{v}_i, \qquad \text{pour } i = 1,\dots,r$$

ainsi les σ_i^2 sont les valeurs propres de la matrice $\mathbb{A}\mathbb{A}^T$ et les $\mathbf{u}_i$ les vecteurs propres associés mais aussi les σ_i^2 sont les valeurs propres de la matrice $\mathbb{A}^T\mathbb{A}$ et les $\mathbf{v}_i$ les vecteurs propres associés (attention, étant des valeurs propres, ils ne sont pas définis de façon unique).
On peut exploiter cette décomposition pour faire des économies de mémoire.

- ⋆ Pour stocker la matrice $\mathbb{A}$ nous avons besoin de $n \times p$ valeurs.
- ⋆ Pour stocker la décomposition SVD nous avons besoin de $n\times r + r + r\times p = (n+p+1)r > (n+p+1)r$ valeurs donc à priori on ne fait pas d'économies de stockage. Cependant, s'il existe $s < r$ tel que $\sigma_s = \sigma_{s+1} = \dots = \sigma_r = 0$, alors nous n'avons plus besoin que de $n\times s + s + s\times p = (n+p+1)s$ valeurs. Si $s < np/(n+p+1)$ on fait des économies de stockage.

Idée de la compression : si nous approchons $\mathbb{A}$ en ne gardant que les premiers s termes de la somme (sachant que les derniers termes sont multipliés par des σ_i plus petits, voire nuls)

$$\widetilde{\mathbb{A}} = \sum_{i=1}^{s} \sigma_i \underbrace{\mathbf{u}_i \times \mathbf{v}_i^T}_{\in\mathbb{R}^{n\times p}}, \qquad \text{où } s < r$$

- ⋆ pour stocker la matrice $\widetilde{\mathbb{A}}$ nous avons toujours besoin de $n \times p$ valeurs,
- ⋆ pour stocker la décomposition SVD nous avons besoin de $n\times s + s + s\times p = (n+p+1)s$ valeurs. Si $s < np/(n+p+1)$ on fait des économies de stockage.

© 2022-2023 G. Faccanoni

Exercice 2.27 (Valeurs singulières)
Calculer analytiquement et vérifier numériquement sa décomposition SVD de la matrice

$$\mathbb{A} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \end{pmatrix}$$

Correction
$\mathbb{A} \in \mathbb{R}^{n\times p}$ avec $n=2$ et $p=3$ donc $r=2$.
Pour calculer la décomposition SVD nous allons calculer les valeurs et vecteurs propres des matrices $\mathbb{A}\mathbb{A}^T$ et $\mathbb{A}^T\mathbb{A}$.

	Valeurs propres	Vecteurs propres unitaires
$\mathbb{A}\mathbb{A}^T = \begin{pmatrix} 5 & 4 \\ 4 & 5 \end{pmatrix}$	$\lambda_1 = 9 > \lambda_2 = 1$	$\mathbb{U} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
$\mathbb{A}^T\mathbb{A} = \begin{pmatrix} 5 & 4 & 0 \\ 4 & 5 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\lambda_1 = 2 > \lambda_2 = 1 > \lambda_3 = 0$	$\mathbb{V} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix}$

Donc

$$\mathbb{A} = \mathbb{U}\mathbb{S}\mathbb{V}^T = \underbrace{\begin{pmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_r & \mathbf{u}_{r+1} & \dots & \mathbf{u}_n \end{pmatrix}}_{\in\mathbb{R}^{n\times n}} \underbrace{\begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix}}_{\in\mathbb{R}^{n\times p}} \underbrace{\begin{pmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_r^T \\ \mathbf{v}_{r+1}^T \\ \vdots \\ \mathbf{v}_p^T \end{pmatrix}}_{\in\mathbb{R}^{p\times p}}$$

$$= \underbrace{\begin{pmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_r \end{pmatrix}}_{\in\mathbb{R}^{n\times r}} \underbrace{\begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix}}_{\in\mathbb{R}^{r\times r}} \underbrace{\begin{pmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_r^T \end{pmatrix}}_{\in\mathbb{R}^{r\times p}} = \sum_{i=1}^{r} \sigma_i \underbrace{\mathbf{u}_i \times \mathbf{v}_i^T}_{\in\mathbb{R}^{r\times r}}$$

devient

$$\mathbb{A} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\left(\begin{array}{cc|c} 3 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right)\frac{1}{\sqrt{2}}\left(\begin{array}{ccc} 1 & 1 & 0 \\ 1 & -1 & 0 \\ \hline 0 & 0 & \sqrt{2} \end{array}\right)$$

$$\overset{r=2}{=} \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \end{pmatrix}$$

$$= \frac{3}{2}\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \end{pmatrix}$$

Notons que la décomposition n'est pas unique, par exemple avec Octave on trouve

Vecteurs propres unitaires :

$$\mathbb{U} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \qquad \mathbb{V} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix}$$

ce qui donne le même résultat (heureusement!)

```
A=[1 2 0; 2 1 0]
[n,p]=size(A)
r=min(n,p)

AAT=A*A'
[VecAAT,ValAAT]=eig(AAT)  % unsorted list of all eigenvalues
% To produce a sorted vector with the eigenvalues, and re-order the eigenvectors accordingly:
```

```
[ee,perm] = sort(diag(abs(ValAAT)),"descend");
ValAAT=diag(ee)
VecAAT=VecAAT(:,perm)

ATA=A'*A
[VecATA,ValATA]=eig(ATA)
[ee,perm] = sort(diag(abs(ValATA)),"descend");
ValATA=diag(ee)
VecATA=VecATA(:,perm)

myS=diag(sqrt(diag(ValATA)),n,p)
myU=VecAAT
myV=VecATA

[UU,SS,VV]=svd(A)

dS=diag(SS)

AA=zeros(5,4);
for i=1:numel(dS)
   temp=dS(i)*UU(:,i)*VV(i,:)
   AA+=temp
end
```

Exercice 2.28 (Compression par SVD)

1. Tester la compression avec $s = 10$ et $s = 100$ pour obtenir les images 2.23 ainsi que la carte des erreurs.
2. Calculer $\max\left\{ s \in [0;r] \;\middle|\; s < \frac{np}{n+p+1} \right\}$ qui est la limite en dessous de laquelle le stockage de la décomposition SVD permet des économies de mémoire par rapport au stockage de l'image initiale.
3. Supposons que la précision d'Octave soit de l'ordre de 3 chiffres significatifs. Alors les valeurs singulières significatives doivent avoir un rapport de moins de 10^{-3} avec la valeur maximale σ_1, les autres étant considérées comme «erronées». Calculer le nombre de valeurs singulières «significatives», *i.e.* la plus grande valeur de s telle que $\frac{\sigma_i}{\sigma_1} < 10^{-3}$ pour $i = s+1,\dots,r$.

Correction

```
clear all

A = double(imread('lena512.bmp'));
[row,col] = size(A)

colormap(gray(256));

subplot(5,2,1)
imshow(uint8(A));
s = min(row,col);
title ( strcat("s = ",num2str(s)) );
imwrite(uint8(A),strcat(['exo6-' num2str(s) '.jpg'
    ]),'jpg');

[U,S,V] = svd(A);

% on fait des economies de stockage si "s" est < a
    "economie" :
economie = row*col/(row+col+1)
vsSignif = sum(sum( (S./S(1,1))>1.e-3 ))
```

```
n = 2;
for s = [vsSignif,floor(economie),100,10]
   subplot(5,2,2*n-1)
    X = U(:,1:s)*S(1:s,1:s)*(V(:,1:s))';
    imshow(uint8(X));
    title ( strcat("s = ",num2str(s)) );
    %imwrite(uint8(X),strcat(['exo6-' num2str(s) '.
        jpg']),'jpg');
   subplot(5,2,2*n)
    erreur = abs(X-A);
    somerr = sum(erreur(:));
    m = min(erreur(:));
    M = max(erreur(:));
    erreur = 255-255/(M-m)*(erreur-m);
    imshow(uint8(erreur));
    %imwrite(uint8(erreur),strcat(['exo6-' num2str(
        s) 'E.jpg']),'jpg');
   n+ = 1;
end
```

On a $\max\left\{ s \in [0;r] \;\middle|\; s < \frac{np}{n+p+1} \right\} = 255$: on fait des économies de stockage tant qu'on garde au plus les premières 255 valeurs singulières.
La photo de Lena, de taille 512×512, possède 275 valeurs singulières «significatives».

© 2022-2023 G. Faccanoni

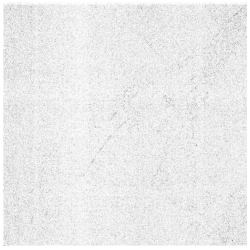
(a) Original $s = n = p = 512$

(b) $\min_{275}\left\{ s \in [0;r] \;\middle|\; \frac{\sigma_s}{\sigma_1} < 10^{-5} \right\} =$

(c) $\max_{255}\left\{ s \in [0;r] \;\middle|\; s < \frac{np}{n+p+1} \right\} =$

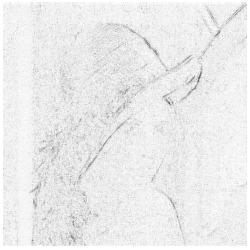
(d) $s = 100$

(e) $s = 10$



(f) $\min_{275}\left\{ s \in [0;r] \;\middle|\; \frac{\sigma_s}{\sigma_1} < 10^{-5} \right\} =$

(g) $\max_{255}\left\{ s \in [0;r] \;\middle|\; s < \frac{np}{n+p+1} \right\} =$



(h) $s = 100$

(i) $s = 10$

FIGURE 2.23 – SVD - exercice 2.28