INF8725 - Traitement de signaux et d'images

TP0 - Rappel Matlab
Automne 2017

Professeur : Fantin Girard
Chargés de laboratoire : Clément Playout, Gabriel Lepetit-Aimon

---

Objectifs :
Ce laboratoire a pour objectif de faire les rappels sur les fonctions de Matlab utiles aux laboratoires du cours INF8725.

Découverte de l'interface de cours

L'interface de cours permet de tester les effets des algorithmes vus en cours sur des signaux 1D et sur des images. Elle peut non seulement vous permettre d'avoir une meilleure compréhension du cours mais aussi de vérifier la validité des algorithmes que vous aurez à implémenter dans les laboratoires.

1. Téléchargez et extrayez l'archive nommée `Interface Matlab Traitement de signal` disponible sur Moodle. Cette archive contient les scripts nécessaires au fonctionnement de l'interface (les fichiers : `.m`, `.fig` et `.mat`) ainsi que les dossiers `filtres`, `signaux` et `images` contenant des ressources sur lesquelles expérimenter.

2. Ouvrez dans matlab le script `signal1D.m`, exécutez-le et vérifiez que l'interface est opérationnelle. Ce script permet de manipuler les algorithmes de filtrages 1D vus en cours (Chapitre 1, 2 et 3) .

3. Exécutez le script `signal2D.m`. Ce script présente les fonctionnalités de filtrage spatial (Chapitre 4), de restauration d'image (Chapitre 6) et de segmentation d'images (Chapitre 7). Les échantillons d'images du dossier `images` vous seront particulièrement utiles comme sujet de test.

4. Enfin exécutez le script `Sinus2D_FFT.m` qui vous sera utile pour visualiser la transformée de Fourier en 2D.

> **?** Pour exécuter un script Matlab entier, on peut taper son nom dans la console (il suffira alors d'effectuer le raccourci **`Ctrl+C`** pour l'interrompre). On peut aussi, une fois le script ouvert dans l'éditeur, cliquer sur l'icone **`Executer`** ou appuyer sur la touche **`F5`**.
> On utilise plutôt **`Ctrl + Entrée`** pour n'exécuter que la section courante ou alors **`F9`** pour n'exécuter que les lignes sélectionnées.

Exercice I : Variables, Scripts et Courbes

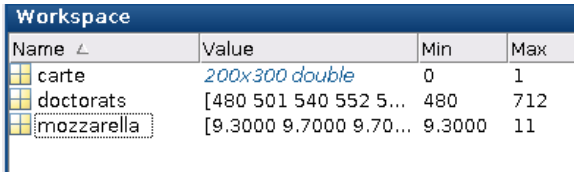
Pour cet exercice et pour la suite du laboratoire vous aurez besoin de télécharger et de décompresser l'archive **`Laboratoire0.zip`** disponible sur Moodle.

Cet exercice n'est qu'un rappel des fonctions de base de Matlab pour afficher des graphiques (tracer des courbes, des points, superposer de graphes, ajouter titre et légende...). Si ces fonctionnalités vous sont familières, vous pouvez passer directement à l'Exercice II.

1. Via l'explorateur de fichier de Matlab déplacez vous dans le dossier où vous avez décompressé l'archive du laboratoire. Puis dans la console tapez la commande :

```
>> load lab0_data.mat
```

De nouvelles variables devraient avoir été ajoutées au *workspace* :



Le vecteur **`doctorats`** contient le nombre de doctorants annuel en génie civil aux États-Unis entre 2000 et 2009, et le vecteur **`mozzarella`** contient la consommation annuelle de mozzarelle par habitant du même pays sur la même période (en livre). On va tracer ces deux variables.

2. Affichez les variables **`doctorats`** et **`mozzarella`** en tapant leur nom dans la console ou en cliquant sur elles dans le workspace.

Comme vous pouvez le constater **`doctorats`** est un vecteur ligne alors que **`mozzarella`** est un vecteur colonne. Par défaut on utilise plutôt des vecteurs ligne dans matlab, on va donc transposer **`mozzarella`** grâce à `'`. Tapez donc dans la commande :

```
>> mozzarella = mozzarella'
```

? Lorsqu'on exécute une commande Matlab son résultat est affiché dans la console (ici la nouvelle valeur de **`mozzarella`**) sauf si la commande se termine par un `;`. Pour ne pas saturer la console lors de l'exécution d'un script on ajoute donc des `;` à la fin de chaque ligne.

3. Pour tracer les graphiques on a besoin du vecteur contenant les années de chaque point de **`mozzarella`**, c'est-à-dire le vecteur des entiers entre 2000 et 2009. On pourrait déclarer ce vecteur par la commande :

```
>> annee = [2000 2001 2002 2003 2004 2005 2006 2007 2008 2009];
```

On peut aussi utiliser l'opérateur d'énumération qui permet d'énumérer des nombres en choisissant une borne de départ, un pas et une borne d'arrivée : **`2000:1:2009`**. Dans le cas où le pas est égal à 1 on peut simplement écrire :

```
>> annee = 2000:2009;
```

? Pour déclarer un vecteur ou une matrice dans Matlab il suffit d'écrire son contenu entre `[ ]` en séparant les colonnes par des `,` ou des espaces et les lignes par des `;` ou des retours à la lignes.
Matlab possède plusieurs fonctions pour créer rapidement certaines matrices : pour une matrice nulle on utilisera `zeros(n,m)`, pour une matrice unitaire : `ones(n, m)`, pour une matrice identité : `eyes(n)`.

4. Tracez la courbe de la consommation de mozzarella entre 2000 et 2009 avec la commande **plot** :

```
>>  plot( annee, mozzarella);
```

Sur le graphique qui s'affiche on peut constater que la valeur pour l'année 2001 est erronée : au lieu de 0 elle devrait valoir 9.7 livres.

5. Rectifiez la seconde valeur de **mozzarella** avec la commande qui suit et retracez le graphe (vous pouvez utiliser la flèche du haut sur le clavier pour remonter dans l'historique des commandes).

```
>> mozzarella(2) = 9.7;
```

? Que ce soit pour lire ou écrire les valeurs d'un vecteur ou d'une matrice, on utilise les **( )**. Ainsi **M(1,1)** accède à la case situé sur la première ligne et sur la première colonne de **M**. On peut aussi accéder au dernier élément par l'indice **end**. L'accès à la première ligne, dernière colonne de **M** s'écrit donc **M(1,end)**.
Enfin, l'opérateur d'énumération **:** permet d'accéder à plusieurs cases : **v(1:3)** accèdera aux 3 premiers éléments de **v**, **v(5:end)** retournera toutes les cases de **v** à partir de la 5e et **M(:, 1)** retournera la première colonne de **M**.

! Dans Matlab les indices des tableaux commence à **1** et non pas à **0** comme dans la plupart des autres langages en informatique.

Pour le moment, retracer le graphe ne nécessite qu'une ligne de commande mais une fois que nous aurons paramétré les axes, le titre et la légende ce ne sera plus aussi simple. Pour éviter de re-exécuter manuellement ces commandes à chaque fois et pour pouvoir sauvegarder notre travail, on préfère généralement utiliser des scripts que la console Matlab.

6. Créez un script Matlab (fichier texte avec pour extension : **.m**) dans le même dossier et copiez-y les commandes suivantes :

```
%% Nettoyage du workspace
clear all;
close all;
clc;

%% Exercice I
% Initialisation des variables
load lab0_data.mat;
mozzarella = mozzarella';
mozzarella(2) = 9.7;
annee = 2000:2009;
```

Expliquons rapidement ces quelques lignes :

— Les `%%` permettent de déclarer une nouvelle section : la portion de code comprise entre ces `%%` et les suivants peut être exécutée seule via **`Ctrl+Entrée`**.

— Le `%` seul permet d'écrire un commentaire : ce qui le suit ne sera pas lu par matlab.

— **`clear all; close all; clc;`** permettent respectivement de supprimer toutes les variables du workspace, fermer toutes les fenêtres de matlab et nettoyer la console. Les scripts commencent généralement par ces trois commandes pour avoir une session propre.

7. Ajoutez au script l'instruction pour tracer **`mozzarella`**. Puis donnez un titre aux axes avec les commandes **`xlabel`** et **`ylabel`**.

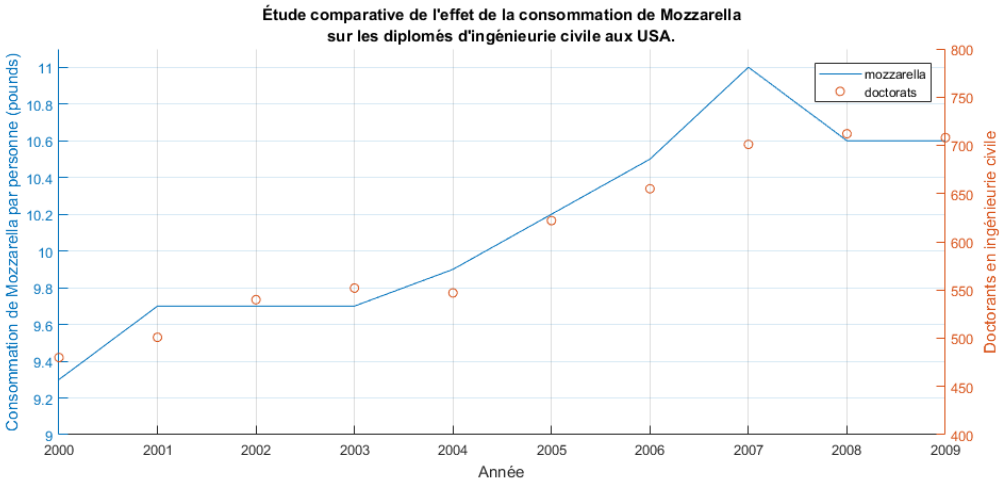
> **?** La documentation Matlab est très complète et contient beaucoup d'exemples. Lorsque vous n'êtes pas certains de l'utilisation d'une commande, tapez simplement son nom et appuyer sur **`F1`** pour faire apparaitre l'aide contextuelle. Vous pouvez même ne taper que le début de la commande et l'auto-compléter en appuyant sur **`TAB`**.

8. Toujours dans le script, tracez **`doctorats`** sous forme de points via la commande **`scatter`** (qui s'utilise comme **`plot`**), et ajoutez un titre aux axes.

> **?** Par défaut Matlab écrase l'ancien graphique par le nouveau. Pour afficher plusieurs graphes dans des fenêtres différentes il faut précéder le tracer d'un graphe par la commande **`figure`** qui ouvrira une nouvelle fenêtre.

9. Savoir exploiter des données revient bien souvent à savoir les présenter. Superposez les deux graphes en ajoutant le mot clé **`hold on`** avant le premier graphe. En précédent la commande **`plot`** de **`yyaxis left;`** et la commande **`scatter`** de **`yyaxis right;`** vous pourrez paramétrer séparément les axes d'ordonnées des deux graphes. Fixez ainsi les limites du premier entre 9 et 11.1 livres de mozzarella et le deuxième entre 400 et 800 doctorants avec **`ylim`**. Vous pouvez aussi fixer les bornes d'abscisses avec **`xlim`**. Ajoutez enfin une grille avec **`grid on;`**, une légende avec **`legend`** et un titre avec **`title`** (quelques choses comme «Étude comparative de l'effet de la consommation de Mozzarella sur les diplômes d'ingénierie civile aux USA.» semble de rigueur).

Vous devriez alors obtenir un graphe similaire à :



Les données utilisées pour cet exercice sont issues du site Spurious Correlations qui recense des corrélation insensées et purement statistiques.

Exercice II : Générer et tracer des signaux

1. Déclarez à l'aide d'une fonction anonyme le signal : $s_0(x) = \sin(x)$. Et affichez ses valeurs pour 0, $\frac{\pi}{4}$ et $\frac{3\pi}{4}$.

> **?** Les fonctions anonymes de matlab permettent de déclarer simplement des fonctions au sens mathématiques sous la forme :
> `nomFonction = @(x1, x2, ...) expression de x;`
> Ainsi $f(x = x^2)$ se déclare `f = @(x) x^2;`, et s'utilise comme un fonction standard : `f(3.14)`.

2. Déclarez de la même manière : $s_1(x) = \dfrac{\sin(3x)}{3}$ et $s_2(x) = \dfrac{\sin(5x)}{5}$.

3. Affichez sur le même graphique : s_0, s_1 et s_2 pour $x \in [0;4\pi]$ avec une résolution de 500 points (vous pouvez utiliser `linspace` pour générer le vecteur x). N'oubliez pas le titre et la légende !

4. Superposez à ces signaux leur somme $s_0 + s_1 + s_2$. Pour rendre le diagramme plus lisible vous pouvez tracer les 3 signaux en pointillés en ajoutant `'--'` en dernier argument de `plot`. Vous pouvez aussi tracer la somme en gras (`plot(x, ..., 'LineWidth', 2)`).

On peut observer que la somme des signaux reste périodique mais ressemble de moins en moins aux sinusoïdes dont elle est composée. Essayons d'augmenter encore le nombre de signaux sommés.

5. Calculez sur le même intervalle x, la somme :

$$\sum_{i=0}^{50} \frac{\sin((2i+1)x)}{2i+1}$$

(c'est-à-dire la somme des $\frac{\sin(kx)}{k}$ avec k impair de 1 à 101).

Affichez ce signal dans une nouvelle figure.

Dans matlab les boucles for s'expriment :

```
for i=V
    ...
end
```

où `i` prendra successivement toutes les valeurs du vecteur `V`. Ce vecteur sera remplacé dans la plupart des cas par une énumération : `for i=0:50` ou `for k=1:2:101`.

6. Réessayez pour i allant de 0 à 500 (k allant de 1 à 1001). Vous venez d'approximer un signal carré par une somme de sinusoïdales !

En traitement du signal, on nomme *décomposition en série de Fourier* le procédé qui permet de décomposer un signal périodique en une somme infinie de sinusoïdes de fréquences supérieures. Sa généralisation à tout signal complexe continu par morceaux s'appelle *la transformé de Fourier* que vous découvrirez au chapitre 2 de ce cours.

Exercice III : Images, fonctions et histogrammes

1. À l'aide de **`imread`** chargez l'image **`chat.png`** et affichez la avec **`imshow`**.

2. Affichez le minimum et le maximum des valeurs de l'image.

> **?** Les méthodes matricielles de matlab (**`sum`**, **`min`**, **`max`**...) ne s'appliquent par défaut qu'à la première dimension de la matrice et ne renvoient donc pas un scalaire. Pour appliquer ces méthodes à la totalité de la matrice il suffit d'ajouter **`(:)`** après la matrice : **`min(M(:))`**.

> **i** Pour limiter la taille globale de l'image, les pixels sont codés sur un octet (valeurs comprises entre 0 et 255) par canal. Le format de variable associé est nommé **`uint8`** (entier non signé sur 8 bits). Par défaut, matlab utilise plutôt le format **`double`** (flottant signé sur 64 bits). Pour appliquer des filtres à une image, il est parfois nécessaire de la convertir en **`double`** : **`double(img)`**, puis de la reconvertir en **`uint8`** pour l'afficher : **`uint8(img)`**

3. Affichez les dimensions de l'image avec **`size`** et identifiez la hauteur, la largeur et le nombre de canaux de l'image.

> **i** Pour stocker et manipuler les images couleurs, on encode la couleur de chaque pixel dans 3 canaux. Quand cette image est destinée à être affichée sur un écran, on utilise le plus souvent le format **`rgb`** où chaque canal encode l'intensité du rouge (1er canal), vert (2e canal), bleu (3e canal).

4. Affichez le canal bleu de l'image avec **`imshow`** et vérifiez que les deux smileys apparaissent noirs (ils sont parfaitement vert ou rouge et ne contiennent pas de bleu).

> **?** On peut extraire un canal particulier d'une matrice en replaçant par **`:`** les indices des autres dimensions.

5. Affichez côte à côte dans la même fenêtre tous les canaux de l'image.

> **?** **`subplot(n, m, i);`** permet de subdiviser une fenêtre pour y afficher plusieurs graphiques. **`n`** et **`m`** correspondent respectivement au nombre de lignes et au nombre de colonnes divisant la fenêtre. **`i`** permet de choisir la subdivision dans laquelle le prochain graphe sera tracé.

Comme vous avez pu le constater, lorsqu'on affiche une matrice à une dimension avec **`imshow`** celle-ci apparait en noir et blanc. Pour améliorer la lisibilité de notre image on va afficher chaque canal dans leur couleur originale. Pour se faire, l'image transmise doit avoir 3 canaux dont 2 nuls.

6. Créez une nouvelle fonction (**`New` → `Function`**) qui prend en argument une image couleur et l'indice d'un canal à sélectionner et qui renvoie l'image modifiée où les canaux non-désirés ont été mis à 0.

> **?** Pour factoriser du code (ici l'extraction d'un canal dans un image), on peut écrire sa propre fonction. Le nom de la fonction doit-être aussi le nom du fichier qui contient sa définition (on ne peut donc avoir qu'une fonction par fichier). La première ligne du fichier contient la signature de la fonction sous la forme : **`function [out] = nomDeLaFonction( inputArgs )`** où **`out`** est la sortie de la fonction et **`inputArg`** la liste des paramètres de ses paramètres (séparés par des **`,`**). Avant la fin de la fonction (**`end`**), une valeur doit avoir été assignée à **`out`** .

7. Modifiez l'affichage des canaux pour qu'il utilise votre fonction et affiche les canaux en couleur. N'oubliez pas d'ajouter un titre aux images !

8. À l'aide de **`imhist`** calculez l'histogramme du canal bleu et affichez le avec **`bar`**.

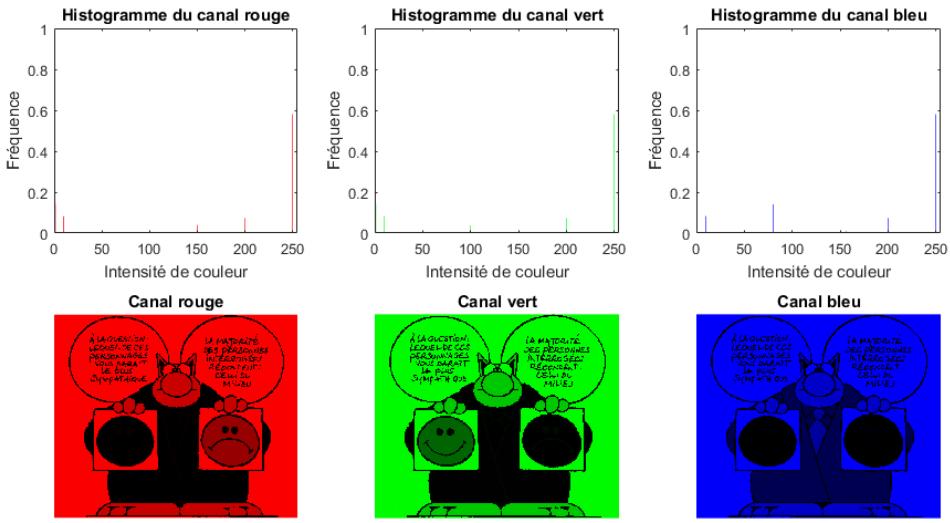
> **i** L'histogramme peut être vu comme une signature d'une image. En pratique il représente la fréquence d'apparition de chacune des 256 intensités de couleur (entre 0 et 255). Ainsi un fort pic dans les intensités de couleurs élevées signifie que l'image contient une grande zone claire.

9. Créez une nouvelle fonction qui prend en argument une image et l'indice d'un canal et n'a pas de valeur de sortie. Cette fonction calculera et affichera l'histogramme *normalisé* de ce canal. La valeur des axes sera forcée entre 0 et 255 pour les abscisses et 0 et 1 pour les ordonnées. L'histogramme devra être tracé dans la couleur du canal.

> **i** L'histogramme *normalisé* est semblable à l'histogramme standard, mais ses valeurs sont comprises entre 0 et 1 et leur somme est égale à 1. On l'obtient avec : **`histogramme = histogramme / sum(histogramme);`**

> **?** Pour choisir la couleur du graphe vous devrez utiliser un code de la forme :
>
> ```
> if (condition1)
>     ...
> elseif (condition2)
>     ...
> else
>     ...
> end
> ```
>
> Où **`condition1`** et **`condition2`** sont des expressions booléennes pouvant utiliser : **`>`**, **`<`**, **`>=`** (supérieur ou égal), **`<=`** (inférieur ou égal), **`==`** (égal), **`~=`** (différent).

10. Modifiez l'affichage des canaux pour y intégrer les histogrammes au dessus de l'image des canaux, vous devriez obtenir un graphe semblable à :



Exercice IV : Images binaires, calcul matriciel, convolution et jeu de la vie

Le jeu de la vie est un automate cellulaire qui modélise l'activité cellulaire. Ses règles, imaginées en 1970 par John Horton Conway, sont très simples :

- Le «plateau du jeu» est une grille à deux dimensions dont chaque case (ou cellule) peut-être soit vivante soit morte.
- Une cellule morte ayant 3 voisines vivantes devient vivante ou «nait».
- Une cellule vivante ayant 2 ou 3 voisines vivantes reste vivante.
- Une cellule vivante ayant 4 voisines ou plus meurt de surpopulation.
- Le jeu est itératif : une itération consiste à recalculer l'état de chaque cellule.

Reformulons la règle de décision permettant de choisir si une cellule sera vivante ou non à la prochaine itération. Si on nomme `e` l'état d'une cellule dont on veut déterminer l'état à l'itération suivante (vivante : `1`, morte : `0`) et `n` le nombre de cellules actuellement vivantes dans un carré de 3 cases de cotés centré sur cette cellule, alors elle sera vivante que si :

`n == 3 ou (e==1 et n==4).`

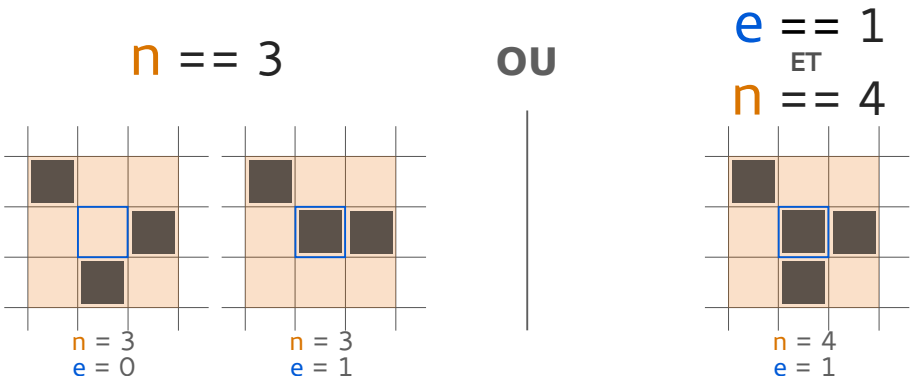


FIGURE 1 – Naissance ou survie de la cellule centrale : en noir, les cellules vivantes.

1. Chargez le fichier **`lab0_data.mat`** et affichez la matrice **`carte`** avec **`imagesc`** (vous pouvez afficher l'échelle de couleur grâce à **`colorbar`**). Cette matrice est l'image binaire que nous allons faire évoluer avec l'algorithme du jeu de la vie.

2. On nomme **`E`** l'image binaire (où les pixels vivants valent **`1`** et les pixels morts valent **`0`**) et **`N`** la carte contenant pour chaque pixel la valeur **`n`** telle que définie précédemment (nombre de cellules vivantes dans un carré de 3x3 autour du pixel). Traduisez, en fonction des matrices globales **`N`** et **`E`**, l'équation locale : **`n == 3 ou (e==1 et n==4)`**.

> **?** Matlab supporte les opérateurs de comparaisons sur les matrices : **`M==2`** renverra une matrice binaire de la même taille que **`M`** et qui vaudra 1 pour les cases de **`M`** égale à **`2`** et 0 sinon.
> Matlab est aussi capable de multiplier élément par élément deux matrices de même dimensions via l'opérateur : **`M1 .* M2`**.

> **i** En algèbre booléenne où **`VRAI`** est représenté par **`1`** et **`FAUX`** par **`0`**, on peut remplacer l'opérateur **`OU`** par **`+`** et l'opérateur **`ET`** par **`*`**.

3. Définissez la fonction anonyme **`N(carte)`** comme ceci :

```
N = @(map) conv2(map, ones(3,3), 'same');
```

Vérifiez que la fonction **`N(map)`** renvoie bien la matrice **`N`** telle que définie précédemment en affichant dans la console ou avec **`imagesc`** les matrices **`carteG`** et **`N(carteG)`** .

> **`conv2`** est l'implémentation d'une opération de traitement d'image appelée *la convolution*. Cette opération somme, pour chaque pixel, son intensité et celles des ces voisins pondérées par un masque (ici un masque unitaire 3x3). Vous la découvrirez plus en détail dans le chapitre 2 et 4.

4. Dans une boucle **`while`** et à l'aide de la fonction **`N`** et de l'expression matricielle que vous avez déterminée plus haut, implémentez l'algorithme du jeu de la vie pour qu'à chaque itération la carte soit mise-à-jour puis affichée.

Pour laisser le temps à Matlab de faire le tracé, pausez (avec **`pause(0.02)`**) l'exécution de la boucle après qu'il ait dessiné la carte. Pour sortir de la boucle lorsque vous fermerez la fenêtre, utilisez **`ishghandle`** comme condition d'arrêt :

```
fenetreJeu = figure('Name', 'Jeu de la vie');

while( ishghandle(fenetreJeu) )
   ...

   pause(0.2);
end
```

Généralités

POLYTECHNIQUE MONTRÉAL

Raccourcis Clavier (Windows)

Executer le script	**F5**
Executer la sélection	**F9**
Executer la section courrante	**Ctrl + Enter**
Interompt un script dans la console	**Ctrl + C**
Auto-complétion	**TAB**
Commenter une ligne	**Ctrl + R**
Décommenter une ligne	**Ctrl + T**
Afficher la documentation contextuelle	**F1**

Vecteurs

Vecteur ligne	**[** *1*, *2*, *3***]** ou **[** *1 2 3***]**
Vecteur colonne	**[** *1* ; *2* ; *3* **]**
Vecteur ligne: [j , j+i , ... , k]	**j : i : k**
Vecteur ligne: *N* éléments de j à k	**linspace(** *j* , *k* , *N* **)**
Accès aux éléments de *i* à *j*	**V (***i* **:** *j***)**
Taille d'un vecteur	**length(** *V* **)**
Trier un vecteur (ascendant)	**sort(** *V* **)**
Transposé d'un vecteur	*V* '
Multiplication éléments par éléments	*U* **.*** *V*
Division éléments par éléments	*U* **./** *V*
Concaténation Horizontale / Verticale	**[***U* , *V***]** / **[***U* ; *V***]**
Indexes où **V** est différent de **0**	**find(** *V* **~=** *0* **)**

Graphiques

Créer une nouvelle fenêtre	**figure**
Subdiviser la fenêtre	**subplot(** *nColonne*, *nLigne*, *i***)**
Superposer les graphiques	**hold on**
Tracer une courbe	**plot(** *x*, *y* **)**
Tracer un nuage de point	**scatter(** *x*, *y* **)**
Tracer un diagramme en barre	**bar(** *x*, *y*)
Ajouter un titre au graphique	**title(** "*titre*" **)**
Ajouter une légende	**legend(** "*courbe1*", ...**)**
Ajouter un titre aux axes	**xlabel(** "*x*" **); ylabel(** "*y*" **)**
Modifier les bornes des axes	**xlim([***0*, *1***]); ylim([***0*, *1***])**
Afficher la grille	**grid on**

Commandes générales

Nettoyer le terminal et les variables	**clc ; clear all**
Fermer toutes les fenêtres	**close all**
Mettre en pause l'execution	**pause**
Débuter une nouvelle section	**%%**
Commentaire	**%**
Sauvegarder toutes les variables	**save** "*fichier.mat*"
Restaurer des variables	**load** "*fichier.mat*"
Afficher un fichier texte	**type**

Matrices

Matrice	**[** *1*, *2*; *3*, *4***]**
Accès à la colonne *j*	**M (:,** *j***)**
Taille de la matrice	**size(** *M* **)**
Nombre d'élément de la matrice	**numel(** *M* **)**
Matrice inverse	**inv(***M***)**
inv(A) * B	*A* **** *B*
Créer une matrice: Identité n x n	**eye(***n***)**
Créer une matrice: Nulle n x m	**zeros(** *n* , *m* **)**
Créer une matrice: Unitaire n x m	**ones(** *n* , *m* **)**

Programmation

Déclarer une fonction	**function [***y***] =** *nom***(** x, ... **)** ... **end**
Conditions	**if(** ... **)** ... **elseif(** ... **)** ... **else** ... **end**
Boucle: FOR	**for** *i* **=** *1* **:** *n* ... **end**
Boucle: Pour tout ***v*** dans ***V***	**for** *v* **=** *V* ... **end**
Boucle: WHILE	**while(** *critère* **)** ... **end**
Déclarer une fonction inline	*f* **= @(***x***)** *a*x + b*
Convertir nombre vers string	**num2str(** *x* **)**

Statistiques

Moyenne	**mean(** *x* **)**
Variance	**var(** *x* **)**
Écart type	**std(** *x* **)**
n nombres aléatoires, dist. uniforme	**rand(** *n* **)**
n nombres aléatoires, dist. normale	**randn(** *n* **)**

MATLAB CHEATSHEET

Traitement du Signal

POLYTECHNIQUE MONTRÉAL

Filtrage 1D

Calculer la convolution de **u** et **v**	**conv(** *u* , *v* **)**
Générer un filtre FIR	**fir1(** *ordre*, *f*, ...**)**
Appliquer un filtre à un signal **x**	**filter(** ..., *x* **)**
Tracer un filtre	**freqz(** ... **)**
Transformée de Fourier Rapide	**fft(***x***)**
Transformée de Fourier Inverse	**ifft(***X***)**
Centrer la distribution de Fourier	**fftshift(***X***)**
Calculer l'histogramme d'un signal	**hist(***x***)**

Filtrage 2D

Convoluer un canal et un masque	**conv2(** *img* , *masque* **)**
Convoluer une image et un masque	**convn(** *img* , *masque* **)**
Générer le noyau d'un filtre 2D	**fspecial(** "*type*" **)**
Appliquer le noyau d'un filtre 2D	**imfilter(** *img*, *filtre* **)**
Transformée de Fourier Rapide 2D	**fft2(***img***)**
Transformée de Fourier Inverse 2D	**ifft2(***X***)**
Appliquer le filtre de Wiener	**deconvwnr(***img*, *PSF*, *NSR***)**
Calculer l'histogramme d'une image	**imhist(***img***)**

Manipulation d'Audio

Charger un son	**audioread(** "*son.wav*" **)**
Créer un lecteur de son	**audioplayer(** *son*, *Fe***)**
Jouer un son	**play(** *player***)**
Arrêter la lecture d'un son	**pause(** *player***)**

Manipulation d'Images

Charger une image	**imread(** "*fichier.jpg*" **)**
Afficher une image	**imshow(** *image* **)**
Afficher une carte d'intensité	**imagesc(** *carte* **)**
Afficher l'échelle d'intensité	**colorbar**
Effectuer une rotation	**imrotate(** *img*, *angle* **)**

Filtrage Morphologique

Générer une forme	**strel(** *"forme"* , ... **)**
Effectuer une érosion	**imerode(** *img* , *forme* **)**
Effectuer une dilatation	**imdilate(** *img* , *forme* **)**
Effectuer une ouverture	**imopen(** *img* , *forme* **)**
Effectuer une fermeture	**imclose(** *img* , *forme* **)**
Identifier les objets connexes	**bwlabel(** *img* , *taille* **)**

Conversion de couleur

Convertir une image en entier 0 - 255	**uint8(** *img* **)**
Convertir une image en flottant 0 - 1	**double(** *img* **)**
Convertir une image en niveau de gris	**rgb2grey(** *img* **)**
Appliquer une table de conversion	**intlut(** *img*, *table* **)**

Pour plus de détails sur les commandes de traitement de signaux, consulter la documentation de Matlab ou les sections Filtering and convolution et Fourier Transform de > **help datafun.**