INTRODUCTION A MATLAB

MATLAB, abréviation de MATrix LABoratory, est un langage très performant pour le calcul numérique.
Les utilisations classiques sont :

- Calculs mathématiques
- Développement d'algorithmes
- Modélisation et simulation
- Analyse de données et visualisation
- Développement d'applications telles que les interfaces utilisateur

1 Commandes d'aide

Deux instructions seront très utiles par la suite. Ce sont :

- `lookfor ‘‘mot_clé’’` qui permet de trouver une commande à l'aide de mots clés. Exemple: `lookfor filter`
- `help ‘‘commande’’` donne une aide sur la commande. Exemple : `help sum`. Ne pas hésiter à employer largement cette instruction dès que l'on a une hésitation sur l'utilisation d'une commande.
- `help ‘‘toolbox’’` donne toutes les commandes de la toolbox par catégories. Exemple : `help comm.`

2 Matrices

2.1 Construction

MATLAB travaille essentiellement sur des matrices, et toutes les variables sont représentées par des matrices. Un vecteur ou un scalaire étant une matrice avec une ligne et/ou une colonne.
La saisie d'une matrice peut s'effectuer de différentes façons :

- soit en la rentrant comme une liste d'éléments ;
- soit en la générant dynamiquement à l'aide de fonctions ;
- soit en la chargeant depuis un fichier.

Par exemple :

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

crée une matrice 3×3 et l'affecte à la variable `A`. MATLAB affiche alors

```
A =  1   2   3
     4   5   6
     7   8   9
```

Remarque : d'une manière générale, pour ne pas afficher la valeur d'une variable ou le résultat d'une fonction ou d'un calcul, on écrit un ";" en fin de ligne. Ainsi,

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

affecte la matrice 3×3 et l'affecte à la variable `A`, mais ne l'affiche pas.

Les éléments d'une ligne sont séparés par des espaces ou des virgules et les lignes sont séparées par des point-virgules.
MATLAB traite aussi les nombres complexes :

$$\begin{aligned} \texttt{A} &= [1\ 2;\ 3\ 4] + \texttt{i} * [5\ 6;\ 7\ 8] \\ \texttt{A} &= [1 + 5 * \texttt{i}\ \ 2 + 6 * \texttt{i};\ \ 3 + 7 * \texttt{i}\ \ 4 + 8 * \texttt{i}] \end{aligned}$$

`i` ou j peuvent être utilisés indifféremment pour la notation imaginaire. Dans le cas où ils seraient utilisés comme variables, on peut les recréer à l'aide de :

$$\texttt{ii} = \texttt{sqrt}(-1)$$

Le nombre π est donné par `pi`.
Certaines fonctions permettent aussi de créer des matrices. La commande `rand(n)`, par exemple, crée une matrice $n \times n$ dont les éléments sont uniformément distribués entre 0 et 1 ; `rand(m,n)` crée une matrice de taille $m \times n$ suivant la même distribution.
Les éléments d'une matrice sont référencés de la façon suivante :
`A(2,3)` est l'élément de la deuxième ligne et troisième colonne de la matrice A. Si l'on définit un vecteur B, alors `B(4)` est le quatrième élément de ce vecteur, indifféremment qu'il soit ligne ou colonne. Les coordonnées des éléments doivent être des entiers strictement positifs.
On peut affecter une valeur scalaire à un élément d'une matrice comme suit :

$$\texttt{A}(2,4) = 8$$

qui va affecter la valeur 8 à l'élément de la deuxième ligne et quatrième colonne de `A`.

2.2 Transformations de matrices

L'utilisation de vecteurs et de matrices permet une réduction de la complexité des calculs et permet l'économie de boucles par rapport à la plupart des langages de programmation. Ce paragraphe va permettre d'évaluer la puissance de MATLAB pour ce type de calcul, en comparant ces instructions par rapport à ce qui devrait être programmé en Pascal ou C pour arriver au même résultat. Dans ce but, MATLAB utilise la notation "`:`".
Par exemple :

`1:5` est le vecteur `[1 2 3 4 5]`.

Bien sûr, les incréments peuvent ne pas être entiers :

`0.2:0.2:1.2` est le vecteur `[0.2 0.4 0.6 0.8 1.0 1.2]`,
et `5:-1:1` est `[5 4 3 2 1]`.

L'exemple suivant crée une table de sinus :

```
x = [0.0:0.1:2.0];
y = sin(x);
[x y]
```

La notation "`:`" permet aussi d'accéder aux sous-matrices d'une matrice. Par exemple, `A(1:4,3)` est le vecteur colonne composé des 4 premiers éléments de la $3^{\text{ème}}$ colonne de la matrice `A`.
`A(:,3)` est la troisième colonne de la matrice `A` et `A(1:4,:)` est composée des quatre premières lignes de la matrice `A`.
Cette notation permet aussi de remplacer des morceaux de matrices :
`A(:,[2 4 5]) = B(:,1:3)` remplace les colonnes 2, 4 et 5 de A par les trois premières colonnes de B.
La fonction `reshape(A,m,n)` reconstruit la matrice `A` avec m lignes et n colonnes, en considérant les éléments colonne par colonne. Ainsi, si

$$\texttt{A} = \begin{matrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{matrix},$$

`B=reshape(A,2,3)` donne

$$\texttt{B} = \begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix}$$

2.3 Opérations sur les matrices

Les opérations suivantes sont disponibles dans MATLAB:

+	addition
−	soustraction
*	multiplication
^	puissance
'	conjugué transposé si la matrice est complexe
	ou transposé si elle est réelle
.'	transposé
\	division à gauche
/	division à droite
`inv`	inversion d'une matrice carrée

Ces commandes s'appliquent aussi aux scalaires. Si les tailles des matrices sont incompatibles, alors un message d'erreur apparait, sauf dans le cas d'une opération entre un scalaire et une matrice où l'opérateur va s'appliquer au scalaire et aux éléments de la matrice.
La division matricielle est assez spéciale. Soit A une matrice inversible, et b un vecteur respectivement colonne ou ligne, alors:

`x=A\b` est la solution de $A * x = b$, et
`x=A/b` est la solution de $x * A = b$.

Les divisions à droite et à gauche sont liées par `b/A=(A'\b')'`.
Remarque 1 : pour résoudre le système $A*x = b$, on peut aussi écrire `x=inv(A)*b`, mais ceci n'est pas équivalent (dans son fonctionnement) à `x=A\b`. En effet, `x=inv(A)*b` fait le calcul explicite de l'inverse de A, alors que `x=A\b` résoud le système sans inverser la matrice, ce qui est plus rapide.
Remarque 2 : on peut aussi résoudre le système $A * x = b$ à l'aide de `x=A\b` dans le cas où la matrice est de taille $m \times n$ avec $m \geq n$.

2.4 Opérations sur les éléments d'une matrice

Les opérations précédentes sont les opérations matricielles classiques. Les opérations *, ^, \, / peuvent aussi s'appliquer élément par élément sur une matrice ou un vecteur en les faisant précéder d'un point. Par exemple:

`[1 2 3 4].*[1 2 3 4]` ou `[1 2 3 4].^2`
donnent `[1 4 9 16]`.

2.5 Exercices

1. A l'aide de la commande `reshape`, créer la matrice $A(i,j)$ telle que $A(i,j) = 2(j-1)+i$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, c'est-à-dire :

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \end{bmatrix}.$$

   Créer alors la matrice $B(i,j)$ telle que $B(i,j) = 16 - (2(j-1)+i)$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, c'est-à-dire :

$$B = \begin{bmatrix} 15 & 13 & 11 & 9 & 7 & 5 & 3 & 1 \\ 14 & 12 & 10 & 8 & 6 & 4 & 2 & 0 \end{bmatrix}.$$

   On pourra de nouveau utiliser `reshape`, ou bien utiliser la soustraction matricielle.

   A partir des matrices A et B, créer alors la matrice $C(i,j)$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, telle que :

$$C(i,j) = \frac{(16 - (2(j-1)+i))^{1/2}}{\sin(2(j-1)+i)}$$

2. Créer le vecteur $D = \left[\cos(0)\ \cos(\frac{\pi}{3})\ \cos(\frac{2\pi}{3})\ \cos(\pi)\right]$. Créer la matrice

$$M = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}.$$

Résoudre alors le système :

$$\begin{bmatrix} 9 & 4 & 1 & 0 \\ 4 & 9 & 4 & 1 \\ 1 & 4 & 9 & 4 \\ 0 & 1 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{\cos(0)} \\ \frac{1}{\cos(\frac{\pi}{3})} \\ \frac{1}{\cos(\frac{2\pi}{3})} \\ \frac{1}{\cos(\pi)} \end{bmatrix}$$

3 Fonctions matricielles intégrées

Les fonctions matricielles les plus couramment utilisées sont:

`zeros`	matrice de zéros
`ones`	matrice de uns
`eye`	matrice identité
`linspace`	vecteur en progression arithmétique
`logspace`	vecteur en progression logarithmique
`diag`	création ou extraction de la diagonale d'une matrice

Par exemple, `zeros(m,n)` crée une matrice $m \times n$ remplie de zéros, et `zeros(n)` crée une matrice remplie de zéros et de taille $n \times n$.
Si x est un vecteur, `diag(x)` est la matrice diagonale avec x sur la diagonale, les autres éléments étant nuls. Si A est une matrice carrée, `diag(A)` est un vecteur composé de la diagonale de A.
Les matrices peuvent être construites par blocs. Soit A une matrice 3×3:

$$\texttt{B} = [\texttt{A},\ \texttt{zeros}(3,2);\ \texttt{zeros}(2,3),\ \texttt{eye}(2)]$$

va construire une certaine matrice 5×5.

3.1 Fonctions élémentaires

Les fonctions élémentaires disponibles sous Matlab, s'appliquent aussi bien à des matrices qu'à des scalaires. Les plus courantes sont:

`sin`, `cos`, `tan`, `asin`, `acos`, `atan`
`sinh`, `cosh`, `tanh`, `asinh`, `acosh`, `atanh`
`exp`, `log`, `log10`
`rem`, `abs`, `angle`, `real`, `imag`, `conj`, `sqrt`, `sign`
`round`, `floor`, `ceil`, `fix, nchoosek`

3.2 Fonctions sur des vecteurs

D'autres fonctions de MATLAB ne s'appliquent que sur des vecteurs et renvoient un scalaire. Dans le cas où on les appliquent sur des matrices, elles renvoient un vecteur. Ce sont :

`max`, `min`,
`sort`,
`sum`, `prod`, `median`, `mean`, `std`
`any`, `all`

Par exemple, l'élément maximal d'une matrice A est donné par `max(max(A))`, et non pas par `max(A)` qui renvoie un vecteur.

3.3 Fonctions sur des matrices

Les plus courantes sont :

`eig`	valeurs propres
`svd`	décomposition en valeurs singulières
`poly`	polynome caractéristique
`det`	déterminant
`inv`	inverse
`expm`	matrice exponentielle
`sqrtm`	matrice racine carrée
`size`	taille d'une matrice

3.4 Exercices :

1. A l'aide des commandes `ones` et `eye`, créer la matrice 10×10

$$A = \begin{bmatrix} 6 & 4 & \dots & 4 \\ 4 & 6 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 4 \\ 4 & \dots & 4 & 6 \end{bmatrix}.$$

   Remplacer la $2^{ème}$ ligne par la ligne [2 3 2 2 ... 2]. Vérifier alors que les valeurs propres de A^{-1} sont les inverses des valeurs propres de A.

2. On souhaite créer un vecteur $x = [x_1, x_2, \dots, x_{10}]$ tel que

$$x_i = \frac{1}{10} \sum_{j=1}^{10} \exp\left(\frac{1}{\log\left[10(j-1)+i\right]+1}\right),\ 1 \leq i \leq 10.$$

   On commence par créer la matrice A de taille 10×10 telle que $A(i,j) = 10(j-1)+i$, c'est-à-dire

$$A = \begin{bmatrix} 1 & 11 & \dots & 91 \\ 2 & 12 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 99 \\ 10 & \dots & 90 & 100 \end{bmatrix}.$$

   On pourra utiliser pour cela la commande `reshape`, comme dans l'exercice . On crée alors à partir de A la matrice B telle que $B(i,j) = \exp\left(\frac{1}{\log(10(j-1)+i)+1}\right)$, en utilisant les opérations élément-par-élément. On obtient alors le vecteur x en calculant la moyenne de B (commande `mean`).

4 Traitements, expressions et variables

MATLAB est un langage qui utilise des expressions qui sont ensuite interprétées et évaluées. Les traitements ont généralement la forme suivante :

variable = (*expression)* ou simplement

expression

Les expressions sont généralement composées d'opérateurs, de fonctions et de noms de variables. L'évaluation d'une expression produit une matrice qui est ensuite affichée et assignée à la variable. Si le nom de la variable est oubliée, alors le résultat est automatiquement assigné à la variable `ans`. Si le dernier caractère d'un traitement est un point virgule, alors l'affichage est supprimé.

Les fonctions suivantes permettent d'avoir une vue globale de l'espace de travail :

`who`	liste les variables
`whos`	liste les variables, leur taille et leur type
`what`	liste les fichiers d'extension .m et .mat

Les variables peuvent être supprimées à l'aide de la commande

`clear nom_variable`

Attention à la commande `clear` qui utilisée seule efface toutes les variables.

5 Sauvegarde d'une session

Lorsque l'on quitte MATLAB, toutes les variables sont perdues. On peut les sauver avec la commande `save nom_sauvegarde`, qui stocke les variables dans un fichier d'extension .mat (c'est-à-dire `nom_sauvegarde.mat`). Lorsque l'on relance MATLAB, on peut les recharger avec `load nom_sauvegarde`.

6 Boucles, tests et relations

Dans MATLAB, ce type de fonctions opérent quasiment de la même façon que pour les langages de type Pascal ou C.

6.1 Boucles

L'exécution répétitive d'une suite d'instruction peut-être réalisée à l'aide de `for` ou `while`.

6.1.1 *For*

La syntaxe de la boucle `for` est la suivante :

```
for variable
                instructions
    end
```

Par exemple, soit n donné :

```
x = [   ];  %crée une matrice vide
for
                i = 1:n
              x = [x, i.^2]
end
```

6.1.2 *While*

La syntaxe de la boucle `while` est la suivante :

```
while expression
                    traitement
     end
```

Le traitement va être répété tant que l'expression est vraie. Les groupes d'instructions qui vont être exécutées ou non selon la condition, peuvent être délimitées par les instruction `case` et `otherwise`.

6.2 Test : instruction if

Cette instruction permet d'exécuter une suite d'instructions conditionnelles. Sa syntaxe est la suivante :

```
if condition
                instructions
    end
```

Le traitement va être exécuté uniquement si la condition est vraie. Des branchements multiples sont aussi possibles :

```
if condition1
                instructions1
elseif condition2
                instructions
else
                instructions
end
```

Remarque : Remplacer une boucle par un traitement matriciel, lorsque cela est possible, conduit à un gain de temps d'exécution très important (cf. partie 1.7.4).

6.3 Relations

Les principaux opérateurs relationnels sont les suivants :

<	plus petit
>	plus grand
<=	plus petit ou égal
>=	plus grand ou égal
==	égal (= correspond à une affectation)
~=	différent

Ils peuvent être combinés avec des opérateurs logiques suivants :

&	et
|	ou
~	non

Lorsque une relation est appliquée à des scalaires, le résultat est alors égal à 1 ou 0 selon qu'elle soit vraie ou fausse. Sur des matrices ou des vecteurs de même taille, l'opération est effectuée élément par élément. Le résultat est une matrice (ou un vecteur) dont les éléments prendront les valeurs 1 ou 0.
Une relation entre matrices est interprétée par `while` et `if` comme vraie si elle est vérifiée par tous les éléments de la matrice. Par conséquent, si l'on veut exécuter un traitement quand des matrices A et B sont égales, on peut faire :

```
if A == B
                traitement
end
```

tandis que si on veut l'exécuter lorsqu'elles ne sont pas égales, on peut faire:

```
if any(any(A~= B))
                traitement
end
```

ou tout simplement

```
if A == B else
                traitement
end
```

Il faut noter que :

```
if A~ = B
        traitement
end
```

ne va pas donner le résultat escompté, car le traitement va être exécuté seulement si deux éléments au moins de A et B sont différents.
Les fonctions `any` et `all` permettent de réduire les relations entre matrices à des vecteurs ou des scalaires. Consulter l'aide à leur sujet.
La fonction `find` est souvent très pratique : elle renvoie l'indice des éléments obéïssant à une condition. Par exemple, soit y un vecteur, alors `find(y>2)` va renvoyer le rang de tous les éléments de y qui sont strictement supérieurs à 2.

6.4 Exercices

1. Génération de bits :

   On veut générer des bits (0 ou 1) avec les probabilités $P[0] = 1-p$ et $P[1] = p$, avec $p \in [0;1]$. Pour cela, on utilise la fonction `rand` : `rand(M,N)` renvoie une matrice de taille $M \times N$ de nombres aléatoires tirés indépendamment de manière uniforme sur $[0;1]$. Utiliser cette fonction, ainsi qu'un opérateur relationnel approprié pour générer (en une seule commande) un vecteur x de 100 bits avec une probabilité $p = 0.2$. Générer de même un deuxième vecteur y. A l'aide de la commande `find`, trouver les indices i pour lesquels on a $x_i = y_i$.

2. Calcul de probabilités :

   On souhaite estimer la probabilité de 0 et de 1 à partir du vecteur x. Essayer les méthodes suivantes :

   - en utilisant la commande `mean` ;
   - en utilisant la commande `sum` ;
   - en utilisant la commande `find` et la commande `length` (qui donne la longueur d'un vecteur).

7 Pause et break

Lors de l'exécution d'un programme, on peut, à l'aide de l'instruction `pause`, suspendre son exécution et la relancer si l'utilisateur appuie sur une touche.
L'instruction `break` arrête l'exécution d'une boucle `while` ou `for`.

8 Fichiers .m

MATLAB peut exécuter un ensemble d'instructions contenues dans un programme. Ces programmes ont une extension `.m`. Il y a deux types de programmes `.m`:

- les scripts
- les fonctions

Les corps des programmes de MATLAB peuvent être observés avec la commande `type`:

```
type sum.m
```

8.1 Scripts

Un fichier script est composé d'un ensemble d'instructions MATLAB. Si le fichier a le nom `prog.m`, alors la commande `prog` va l'exécuter. Les variables d'un programme script sont globales, et son exécution va changer leurs valeurs dans l'espace de travail.

8.2 Fonctions

Les variables d'une fonction sont locales, mais peuvent aussi être déclarées comme globales.
Soit la fonction:

```
function y = moyenne(x)
             %MOYENNE valeur moyenne.
         y = sum(x)/length(x);
```

% permet d'insérer des commentaires.
Cette fonction va calculer la moyenne des éléments d'un vecteur. Dans la ligne de commande, on peut alors écrire :

```
vect = [1 2 3 4 5 6];
 moy = moyenne(vect)
```

qui va renvoyer la valeur moyenne des éléments du vecteur *vect*.
La première ligne du script d'une fonction doit obligatoirement être de la forme :

```
function [resultat1, resultat2, ...] = nom_fonction(argument1,argument2,...)
```

Cette fonction doit être sauvegardé dans un fichier qui a pour nom `nom_fonction.m` (ce fichier doit être accessible depuis MATLAB : pour cela, se placer grâce à la commande `cd` dans le répertoire où la fonction a été sauvegardée).

8.2.1 Exercice :

A l'aide de la commande `sum`, écrire une fonction `moments(x)` qui renvoie les moment d'ordre 1, 2, et 3 d'un vecteur d'entrée x. Tester cette fonction avec le vecteur x généré précédemment.

8.3 Textes, entrées, messages d'erreurs

Les textes doivent être encadrés par des quotes :

```
s = 'texte'
```

Ils peuvent être affichés :

```
disp('affichage')
```

Les messages d'erreurs peuvent être affichés à l'aide de la commande `error` qui arrête aussitôt l'exécution du programme :

```
error('desole!')
```

On peut aussi les saisir lors de l'exécution :

```
iter = input('nombre d iterations? ')
```

8.4 Vectorisation et pré-allocation

Dans le but d'accélérer l'exécution de programmes, il est important de vectoriser les algorithmes dans les fichiers `.m`. En effet, là où d'autres langages utilisent des boucles, MATLAB peut utiliser des opérations vectorielles ou matricielles. Considérons l'exemple suivant :

```
i = 0;
for t = 0 : 0.01 : 10,
                    i = i + 1;
                    y(i) = sin(t)
end;
```

Si on vectorise ce programme, il suffit d'écrire:

```
t = 0:0.01:10;
y = sin(t);
```

Dans le cas où il n'est pas possible de vectoriser, il est possible de rendre les boucles `for` plus rapides en pré-allouant des vecteurs ou des matrices dans lesquels les résultats seront stockés. Par exemple :

```
y = zeros(1,100);
for i = 1:100

    y(i) = det(A.^i);
end
```

En effet, si on ne pré-alloue pas un vecteur, MATLAB va devoir changer la taille du vecteur y à chaque boucle d'itération, ce qui ralentit l'exécution du programme.

9 Gestion des fichiers

L'ouverture de l'éditeur de programmes se fait à l'aide de :

`edit` ou

`edit nom_du_prog`

Les commandes DOS classiques sont valables dans MATLAB :

`pwd`	indique le chemin du répertoire courant
`cd`	change de répertoire
`dir`	liste tous les éléments d'un répertoire
`what`	liste les fichiers .m, .mat et .mex d'un répertoire

D'autres commandes DOS peuvent être exécutées si elles sont précédées de "!" :

`!del nom_fichier`

La commande `path` permet d'obtenir ou d'initialiser un chemin. Pour plus d'information utiliser le help.

10 Graphiques

MATLAB permet de réaliser des graphiques 2-D et 3-D.
Pour avoir un aperçu des capacités de MATLAB en matière de graphismes, utiliser la commande `demo`.
Dans les travaux pratiques, nous utiliserons essentiellement des tracés 2-D. Soit deux vecteurs x et y de même taille. La commande `plot(x,y)` va tracer y en fonction de x. Soit à tracer la fonction sinus de 0 à 4 :

```
x = 0:0.01:4;
y = sin(x);plot(x,y);
```

Si l'on souhaite tracer un deuxième graphique sur une figure différente, taper :

```
figure(2)
```

tandis que si l'on souhaite tracer un deuxième graphique sur la même figure, taper :

```
hold on
```

L'effet de cette commande s'annule avec `hold off`.
L'écriture de texte sur les axes d'un graphe ou sur le graphe s'effectue à l'aide des commandes :

`title`	titre du graphe
`xlabel`	titre de l'axe des abcisses
`ylabel`	titre de l'axe des ordonnées
`gtext`	placement d'un texte avec la souris
`text`	positionnement d'un texte spécifié par des coordonnées

La commande `grid` place une grille sur un graphe.
Les commandes précédentes doivent être saisies après la commande `plot`.
Par défaut les courbes sont tracées par des lignes continues, mais il y d'autres choix:

`- -`, `:`, `- .`, `.`, `+`, `*`, `o` et `x`

De même, on peut préciser les couleurs avec:

`y`, `m`, `c`, `r`, `g`, `b`, `w` et `k`

Par exemple, `plot(x,y,'r*')` va tracer en rouge et avec des * la courbe de y en fonction de x.
La commande `subplot` permet de partitionner un graphe en plusieurs graphes:

$$\texttt{subplot}(\texttt{m}, \texttt{n}, \texttt{p})$$

divise la fenêtre de la figure en une matrice $m \times n$ de petits sous-graphes et sélectionne le $p^{ème}$ sous-graphe pour la figure courante.
Les commandes `semilogx` et `semilogy` à la place de `plot` permettent d'avoir respectivement l'axe des abscisses ou des ordonnées en échelle logarithmique.

10.1 Exercice :

Pour N variant de 100 à 20000 avec un pas de 100, générer un vecteur aléatoire x de longueur N avec une probabilité $p = 0.5$ (de la même façon que précédemment). Pour chacun de ces vecteurs, estimer la probabilité de 0, et sauvegarder chacune de ces probabilités dans un vecteur *proba*. Afficher ensuite la courbe $proba = f(N)$. Recommencer avec $p = 0.2$, puis $p = 0.7$. Superposer les résultats obtenus avec le précédent.