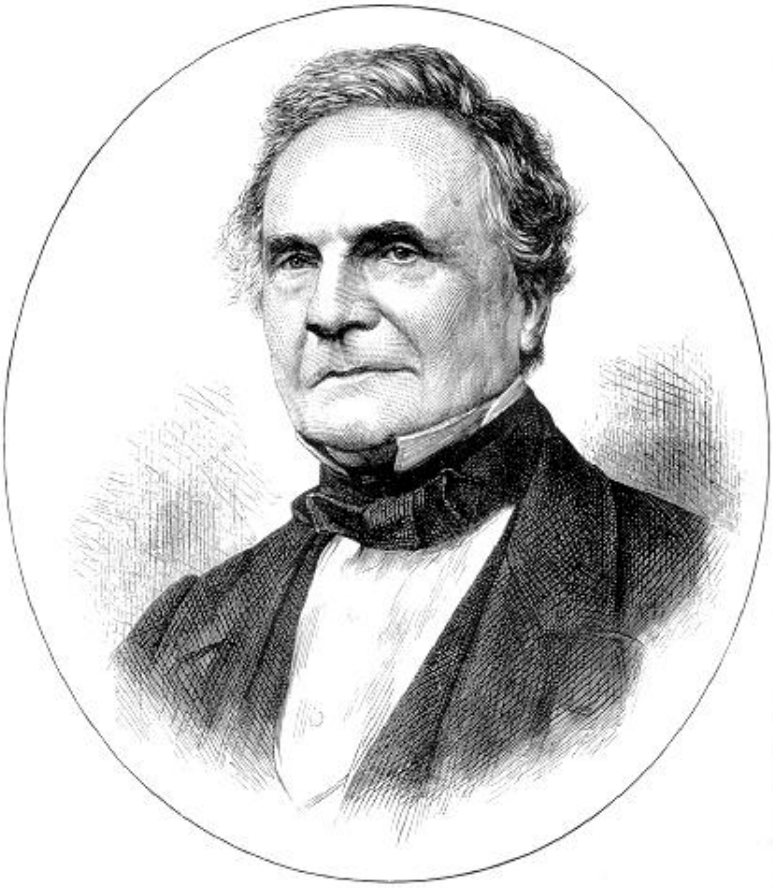

Charles Babbage, l'un des pères fondateurs de l'informatique.
`http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Babbage.html`

Ada Byron, Lady Lovelace. Le premier programmeur de l'histoire était une programmeuse.
http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Lovelace.html

Fragment 0

Initiation à Matlab

Ce premier chapitre est consacré à une brève initiation Matlab. Tout ce qui va suivre concerne Matlab version 6[1], mais reste en grande partie valable pour Matlab 5 et les logiciels Octave 2.1 et Scilab 2.6. De façon générale, chaque bout de code « Matlab » figurant dans ce document fonctionne avec Matlab et au moins l'un de ces logiciels, parfois avec tous [2]. Matlab est un logiciel commercial. Nous vous encourageons à vous procurer les logiciels libres Octave et Scilab. Pour en savoir plus sur les logiciels libres, vous pouvez consulter la page web :

`http://www.fsf.org/philosophy/free-sw.fr.html`

Les logiciels libres sont en parfaite adéquation avec l'idée que l'on peut se faire de l'école publique et de l'éducation nationale. Outre leur gratuité, toute amélioration due à une tierce partie pourra être diffusée de la même manière, ce qui garantit que tout le monde en profitera librement.

Matlab est un logiciel commercial de calcul matriciel développé par la société MathWorks[3]. Son nom est la contraction de « Matrix Laboratory ». Il consiste essentiellement en un *interpréteur de commandes*, écrites dans un langage de programmation spécifique appelé langage Matlab. Les commandes Matlab sont saisies et interprétées ligne à ligne dans une fenêtre en mode texte. Comme nous le verrons plus loin, elles peuvent également être regroupées dans un fichier dont le nom se termine par `.m`. Les versions récentes de Matlab comportent un éditeur de tels fichiers, qui permet d'exécuter les commandes pas à pas (débogage).

Les variables Matlab sont toutes des *tableaux*[4], définis au moment de leur affectation. Il n'y a donc pas besoin de les déclarer. Les vecteurs, matrices et tenseurs sont des tableaux particuliers. Un nombre entier, réel ou complexe est un tableau de taille 1. Le langage Matlab a été conçu pour faciliter les opérations sur les vecteurs, les matrices.

Le langage Matlab permet de manipuler des données de différents types dont certains sont imbriqués : nombres entiers, réels ou complexes, caractères, booléens. Ces données peuvent êtres assemblées en tableaux de différents types imbriqués : vecteurs, matrices, tenseurs (matrices à plus de deux dimensions), tableaux cellulaires, tableaux structurés. Les chaînes de caractères sont des vecteurs de caractères.

Les tableaux cellulaires et structurés ne sont pas abordés dans cette introduction. Le lecteur curieux pourra taper `help lists`, `help cell` et `help struct` pour obtenir de l'aide sur ces aspects.

[1] Testé avec Matlab 6 sur la machine Irix `ondine.cict.fr` du Centre Interuniversitaire de Calcul de Toulouse, `http://www.cict.fr/`.

[2] Pour la compatibilité avec Octave, on pourra consulter les documentations de Scilab et Octave aux adresses `http://www.scilab.org/`, `http://www.octave.org/` et `http://octave.sourceforge.org/`.

[3] `http://www.mathworks.com/`

[4] La documentation en langue anlaise de Matlab parle d'« array ».

Ce chapitre a été écrit dans le soucis d'être à la fois intuitif et peu rébarbatif. Il peut donc manquer de rigueur pour un lecteur familier avec les présentations rationnalisées des langages de programmation.

0.1 M-a-Ma... Matlab

En Matlab, les nombres réels sont représentés en virgule flottante avec 52 chiffres significatifs en base 2, soit un peu moins de 16 en base 10, cf. `eps`, `realmax` et `realmin`. Les opérations élémentaires sur les nombres réels sont :

`+` `-` `*` `/` `\` `^`

qui représentent respectivement l'addition, la soustraction, la multiplication, la division à gauche et la division à droite, et enfin l'élévation à une puissance. Ces opérations s'appliquent également aux matrices, et dans ce cas, \ et / sont différentes.

Comme nous allons le voir, Matlab différentie minuscules et majuscules dans les noms des variables. L'affectation est notée =. La table 0.1 donne quelques variables spéciales de Matlab, qui sont toujours définies.

`[]`	matrice vide
`ans`	dernière évaluation effectuée par Matlab
`computer`	type de machine (cf. aussi `ispc` et `isunix`)
`cputime`	temps CPU depuis le lancement de Matlab
`eps`	précision des flottants
`i` ou `j`	$\exp(i\pi/2) = \sqrt{-1}$
`Inf`	infini positif
`NaN`	pas-un-nombre (Not a Number)
`pi`	valeur de π
`realmax`	plus grand nombre réel flottant positif
`realmin`	plus petit nombre réel flottant positif

TAB. 1 – Quelques variables spéciales de Matlab.

Matlab effectue ses calculs dans l'ensemble des tableaux à plusieurs dimensions, donc en particulier les matrices carrées ou non et les vecteurs ligne et colonne. Les opérations sur ces tableaux ne sont faites que lorsqu'elles ont un sens bien entendu. En général, Matlab donne un sens assez intuitif aux opérations entre matrices. Cela dit, son langage contient de nombreuses spécificités que nous allons apprendre à utiliser.

Voici un exemple de code Matlab, à saisir ligne par ligne dans la fenêtre de commandes. Vous pouvez vous dispenser de saisir les lignes précédées du caractère % car ce sont des commentaires. Elles sont donc ignorées par l'interpréteur de Matlab.

```
% Cette ligne est un commentaire car elle est préfixée par le caractère %
% On crée une variable réelle nommée a, initialisée à 4
% Matlab affiche sa valeur une fois que l'on a appuyé sur la touche entrée
a=4
% En suffixant par un point-virgule, on évite l'affichage de la valeur de a
a=4;
% Pour connaître le contenu d'une variable, il suffit d'invoquer son nom
a
% Matlab différencie majuscules et minuscules.
```



```
% Ainsi, on peut créer la variables A, différente de a
A=1
% Vous pouvez rappeler les commandes précédement executées au moyen
% des touches fléchées de votre clavier (haut et bas).
%
% Voici un calcul compliqué à base des variables a et A précédentes.
A*a+cos(a)/(1+sqrt(1+A^2)) % sqrt est la racine carrée (« square root »)
% La variable spéciale ans contient la dernière réponse de Matlab
% qui n'a pas été affectée à une variable par le symbole =
ans
% On peut lister les variables actuellement définies avec la commande whos
% On voit que pour Matlab, les variables a et A sont des matrices 1 x 1
whos
% Pour une liste plus succincte, on peut utiliser la commande who
who
% On peut détruire une variable au moyen de la commande clear
clear a          % destruction de la variable a
% Vérifions que la variable a n'existe plus
who
% On peut aussi détruire toutes les variables comme suit
clear
% Vérification
who
% Pour obtenir de l'aide (en anglais !) sur une commande, on peut
% utiliser la commande help
help who
% La commande helpwin permet d'obtenir une liste
% des commandes Matlab classée par thême.
helpwin
% Valeurs particulières que l'on est amené à rencontrer
1/0   % Infini positif Inf
-1/0  % Infini négatif -Inf
0/0   % Not a Number (NaN)
0*Inf % Not a Number (NaN)
1/Inf % Donne bien zéro
pi      % Donne bien le nombre pi
help pi % Donne sa définition pour Matlab
i       % Racine carrée (complexe) de -1.
help i  % Sa définition pour Matlab.
        % On peut quand même utiliser i comme variable...
```

Remarque 0.1.1 *(À l'aide !).* La commande `help` permet d'obtenir de l'aide sur les commandes Matlab, `help help` donne de l'aide sur l'aide ! Enfin, la commande `lookfor` permet de lister les commandes Matlab par mots clés.

Remarque 0.1.2 *(Affichages).* Si `X` est un tableau :

- la commande `disp(X)` affiche son contenu sans afficher son nom. S'il est vide, rien n'est affiché. La variable `X` vide est symbolisée par `[]`, on peut tester si `X` est vide grâce à la commande `isempty(X)`.
- la commande `display(X)` affichera le nom de la variable (`X` ici) ainsi que son contenu, même s'il est vide. Cette commande est utilisée automatiquement par Matlab lorsque qu'une expression ne se termine pas par un point-virgule.

```
% Une matrice à deux lignes et trois colonnes en spécifiant ses entrées
```

```
% Les lignes sont séparées par des points virgules
% Les entrées d'une ligne sont séparées par des virgules
A=[2,4,8;3,9,7]
% On peut accéder à une entrée particulière de la matrice A comme suit
A(2,3)
% b est un vecteur ligne contenant la deuxième ligne de A
b=A(2,:)
% c est un vecteur colonne contenant la troisième colonne de A
c=A(:,3)
% B est la sous-matrice 2x2 principale de A
B=A(1:2,1:2)
% Pour afficher à nouveau le contenu de la matrice A
A
% U est un vecteur ligne contenant les entiers de 1 à 20
U=[1:20]
% V est un vecteur ligne contenant les entiers de 1 à 20 par incrément de 2
V=[1:2:20]
% On doit obtenir un vecteur ligne nul
V-U(1:2:20)
% W est un vecteur contenant les réels de Pi/2 à Pi par incrément de 0.1
W=[pi/2:.1:pi]
```

Les commandes comme `sqrt` qui prennent un ou plusieurs arguments (ou paramètres) entre parenthèses constituent des *fonctions*. Nous verrons plus loin comment en créer de nouvelles.

Exercice 0.1.3. À quoi peuvent servir les fonctions `fix`, `round`, `floor`, `ceil`, `mod` et `rem` ? Idem avec les commandes `imag`, `conj`, `angle` et `abs` ?

0.2 Calculs matriciels élémentaires

```
A=[1,2,0;4,0,6;0,8,9] % Création à la main d'une matrice 3x3
B=(A>0)      % B = matrice tq Bij=1 si Aij>0 et 0 sinon
V=[5;4;1]    % Vecteur colonne de dimension 3, s'écrit aussi [5,4,1]'
length(V) % Renvoie la longueur du vecteur V
size(A)   % La fonction size renvoie la taille
%
%% Opérations matricielles classiques
%
B=A' % B = transposée de A
A*B  % Multiplication matricielle de A et de B
A+B  % Addition matricielle de A et de B
p=2;
A^p % Puissance matricielle
expm(A)  % Exponentielle matricielle
inv(A)   % Matrice inverse
sqrtm(A) % Racine carrée matricielle cf. help sqrtm
logm(A)  % Logarithme matriciel cf. help logm
A*V  % Image du vecteur colonne V par la matrice carrée A
V'*A  % Devinez...
A\V  % Solution du système linéaire AX=V (par pivot, et pas par inv(A))
help slash  % Aide explicative sur la division matricielle précédente
%
%% Opérations entrée par entrée
```

```
%
C=A.*B     % C(i,j)=A(i,j)*B(i,j)
C=A./B     % C(i,j)=A(i,j)/B(i,j)
C=A.^3     % C(i,j)=A(i,j)^3
C=3+A      % C(i,j)=3+A(i,j)
C=3*A      % C(i,j)=3*A(i,j)
C=A./3     % C(i,j)=A(i,j)/3
C=cos(A)   % C(i,j)=cos(A(i,j))
C=log(A)   % C(i,j)=log(A(i,j)).  Ne pas confondre avec logm(A) !
C=sqrt(A)  % C(i,j)=sqrt(A(i,j)). Ne pas confondre avec sqrtm(A) !
C=exp(A)   % C(i,j)=exp(A(i,j)).  Ne pas confondre avec expm(A) !
C=abs(A)   % C(i,j)=|A(i,j)|
% Notez qu'à chaque fois, la variable C est redéfinie,
% sans que cela pose problème.
% Listons les variables actuellement définies
whos
% Consultons l'aide sur les opérations élémentaires
help ops
% et sur les opérations élémentaires matricielles
help elmat
%
% Pour finir...
%
n=4;
A=ones(n)  % Matrice carrée de dimension 4 dont tous les éléments valent 1
B=zeros(n) % Matrice carrée nulle de dimension 4
C=eye(n)   % Matrice identité d'ordre 4. Que donne eye(2,5) ?
J=[A,B,C]    % J = juxtaposition horizontale des matrices A, B et C
K=[A;B;C]    % K = juxtaposition verticale des matrices A, B et C
N=0*J        % Astuce pour obtenir une matrice N nulle de même taille que J
             % sans connaitre la taille de J.
% Création d'une matrice M 4x3 « creuse » à partir d'un vecteur de données
% [0.5,0.1,0.2], et de vecteurs de position des données [1,2,4] et [2,1,3]
% Donc M(i,j)=0 sauf M(1,2)=0.5, M(2,1)=0.1 et M(4,3)=0.2
M=sparse([1,2,4],[2,1,3],[0.5,0.1,0.2],4,3)
```

On a donc les opérations supplémentaires suivantes sur les matrices :

```
.*    ./    .\    .^
```

qui opèrent entrée par entrée. Vous pouvez faire `help arith` pour de l'aide sur les opérations arithmétiques et `help @` pour de l'aide sur les opérateurs en général et les caractères spéciaux. Comme on vient de le voir, les sous matrices s'obtiennent en spécifiant des intervalles d'indices. Si i, j, k sont des nombres réels, alors

1. `i:j` est identique à $i, i+1, i+2, \ldots, j$
2. `i:j:k` est identique à $i, i+j, i+2j, \ldots, k$

L'intervalle vide est représenté par la matrice vide `[]`. Une ligne ou une colonne entière peut être obtenue en utilisant le caractère `:` seul. Enfin, une matrice d'entiers `E` peut servir à spécifier les indices d'une matrice `A`, en écrivant `A(E)`. Il ne faut pas confondre les expressions de la forme `[...]`, qui permettent de fabriquer des matrices, avec celles de la forme `M(...)`, qui permettent de considérer une sous-matrice de la matrice `M`.

Exercice 0.2.1. À quoi peuvent servir les fonctions `fliplr`, `flipud` et `sort` ? Dites-vous bien que `lr` est l'abréviation de « left,right » et que `ud` est l'abréviation de « up,down ». Vous connaissez à présent `ones`, `zeros` et `eye`, à quoi correspondent `vander`, `toeplitz`, `hankel`, `pascal`, `magic`, `kron`, `hadamard` et `hilbert` ?

Exercice 0.2.2. Construire la matrice `A` carrée symétrique d'ordre `n` dont le terme général est donné, pour $i \geqslant j$, par $\mathtt{a}_{ij} = b^{i-j}$ où b est un nombre réel donné.

0.3 Calculs matriciels plus élaborés

```
n=4
M=pascal(n)
V=[1:10]
det(M)    % Déterminant de M
rank(M)   % Rang de la matrice
trace(M)  % Trace de M
D=eig(M)  % Renvoie le vecteur colonne des valeurs propres de M
[P,D]=eig(M) % Diagonalise M = P*D*P^-1 avec D diagonale
null(M) % Renvoie une BON du noyau (« null space » en anglais)
tril(M)     % Partie triangulaire inférieure (lower) de M
tril(M,-1) % Idem sans la diagonale, cf. help tril
triu(M)     % Partie triangulaire supérieure (upper) de M
triu(M,+1) % Idem sans la diagonale, cf. help triu
diag(M)     % Diagonale de M, dans un vecteur colonne
diag(V)     % Matrice diagonale dont la diagonale est le vecteur V
blkdiag(eye(2),M) % Matrice block diagonale de blocks eye(2) et M
%
sum(V)    % Somme des éléments du vecteur V
sum(M)    % Somme des colonnes de la matrice M (renvoie un vecteur ligne)
sum(M,2) % Somme les éléments de la matrice M selon la dimension 2
sum(sum(M)) % Somme totale des éléments de la matrice M
cumsum(V)    % Sommes cumulatives des entrées du vecteur V
cumsum(M)    % Matrice des sommes cumulatives des colonnes de M
cumprod(V)   % Produits cumulatifs des entrées du vecteur V
cumprod(M)   % Matrice des produits cumulatifs des colonnes de M
max(M)       % Renvoie un vect. ligne = au max sur chaque col. de M
max(max(M)) % Maximum des entrée de la matrice M. Que donne min ?
range(V)     % Renvoie l'étendue de V, c'est-à-dire max(V)-min(V)
%% Réplication de matrice par blocs
repmat(M,3,4) % Matrice par block 3x4 où chaque block = M
              % M n'a pas besoin d'être une matrice carrée.
%% Redimentionnement de matrice, colonne par colonne
reshape([1,2,3,4],2,2) % Donne la matrice 2x2 [1,3;2,4]
reshape([1,2,3;4,5,6],3,2) % Donne la matrice 3x2 [1,5;4,3;2,6]
%% Aide sur les opérations matricielles élémentaires
%% et sur les fonctions matricielles
helpwin elmat
helpwin matfun
```

Exercice 0.3.1. À quoi peuvent servir les fonctions `svd`, `norm` et `chol` ?

Remarque 0.3.2. La fonction Matlab `funm` permet d'évaluer une fonction sur une matrice. Vous pouvez l'ignorer en première lecture (et même en seconde). Elle nécessite la connaissance de la notion de des-

cripteur de fonction (*function handle* en anglais). Cf. `help functions`, `help function_handle`, `help str2func` et `help func2str`.

Exercice 0.3.3. À quoi peuvent servir les fonctions `expm1`, `expm2`, `expm3` ?

0.4 Premiers pas en statistiques

```
rand  % Renvoie un nombre pseudo-aléatoire en 0 et 1 selon la loi uniforme
randn % Renvoie un nombre pseudo-aléatoire tiré selon la loi normale
% Une matrice pseudo-aléatoire 4x4 dont les entrées sont
% des réalisations pseudo-indépendantes de loi uniforme sur [0,1]
rand(4)
% Idem mais avec la loi normale
randn(4)
% n=1000 réalisations pseudo-uniformes et pseudo-indépendantes
n=1000;X=rand(n,1); % X est un vecteur colonne ici.
m=mean(X)     % Moyenne empirique de X
Me=median(X)  % Médiane de X
v=var(X)      % Variance empirique de X avec une division par n-1
sigma=std(X)      % Écart-type empirique de X avec une division par n-1
var(X,1);std(X,1) % Mêmes quantitées avec division par n
plot(X)           % Tracé du vecteur X. Abscisses ? Ordonnées ?
plot(X,'r--')    % Autre tracé avec couleur et type de ligne
Y=[1:2:2*length(X)];
plot(X,Y,'r-') % Vous comprenez...
V=(X-m)/sigma;  % Normalisation du vecteur X
mean(V); std(V) % Vérification
Z=randn(n,1);
hist(Z)           % Histogramme (10 classes par défaut, cf. help hist)
[Effectifs,Classes]=hist(Z,50);  % Histogramme à 50 classes de Matlab
                                 % Classes = centres des 50 classes
                                 % L'histogramme n'est pas tracé.
[Effectifs,Classes]=histo(Z,50); % Histogramme à 50 classes de Stixbox
                                 % Classes = 51 extrémités des classes
                                 % L'histogramme est tracé.
%
%
figure                           % Création d'une nouvelle fenêtre graphique.
% On trace les moyennes empiriques successives de Z.
% Il y a convergence vers 0 d'après la LGN.
plot(cumsum(Z)'./[1:length(Z)],'b-')
title('Loi des grands nombres')                % Titre de la figure
xlabel('Nombre de réalisations')               % Titre des abscisses
ylabel('Moyennes empiriques')                  % Titre des ordonnées
hold on                                        % Garde la fenêtre graphique
plot(0*ones(n,1),'r--')
legend('Empirique','Theorique')                % Légende
hold off                                       % Sort de la fenêtre graphique
%
%
n=20;A=eye(n)+(hankel(1:n)>=n)+rand(n);
figure
imagesc(A);              % Trace la matrice, couleurs = valeurs relatives,
```

```
title('Ma matrice')      % Titre de la figure
xlabel('Dimension 1')    % Titre des abscisses
ylabel('Dimension 2')    % Titre des ordonnées
hold off
figure
contour(A);              % Trace les lignes de niveau
```

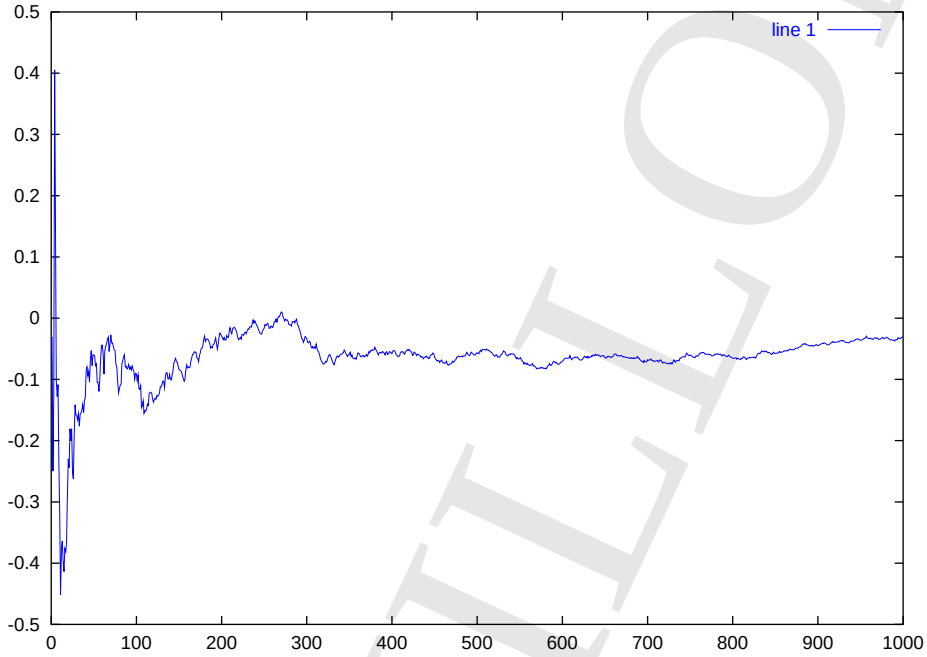

FIG. 1 – Ce que donne `plot(cumsum(randn(1,1000))./[1:1000],'b-')`.

Exercice 0.4.1. À quoi peuvent servir les fonctions `mesh`, `bar` et `stairs` ?

Exercice 0.4.2 *(Sous-graphiques).* Apprenez à vous servir de la commande `subplot` qui permet de tracer des sous-graphiques.

Exercice 0.4.3. Étudiez tous les arguments des fonctions `histo` et `hist`.

Exercice 0.4.4 *(Tracés de graphiques faciles ?).* À quoi peuvent servir les fonctions `ezplot`, `ezcontour`, `ezcontourf`, `ezmesh` `ezmshrc`, `ezplot3`, `ezpolar`, `ezsurf`, et `ezsurfc` ?

Exercice 0.4.5. Les fonctions `sprand` et `sprandn` permettent de générer des matrices creuses aléatoires selon la loi uniforme ou normale. Elles sont en quelque sorte obtenues par croisement de `sparse`, `rand` et `randn`. À quoi peuvent servir les fonctions `speye`, `spones` ?

La bibliothèque Stixbox fournit quelques fonctions supplémentaires utiles pour faire de la simulation. Nous les utiliserons au fur et à mesure de nos avancées et vous en trouverez la liste page 151. Cependant, nous n'utiliserons pas la boite à outils de statistique de Matlab, qui est pourtant bien plus complète, car elle n'est pas disponible le jour de l'épreuve d'agrégation. Vous en trouverez un descriptif page 152.

0.5 Chaînes de caractères

En Matlab, les caractères constituent un type de données, au même titre que les nombres réels. Ainsi, `'1'` représente le caractère « 1 », qui n'est pas de même nature que le nombre 1. Une chaîne de caractères est un vecteur dont les entrées sont des caractères. Une présentation de la gestion des chaînes de caractère sous Matlab est donnée en faisant `help strings`. Le code Matlab :

```
S='Ceci est une chaîne'
```

définit une variable `S` de type chaîne de caractères, qui n'est rien d'autre qu'un vecteur de caractères. En tant que vecteur, `S(1)` vaut `'C'` et `S(4)` vaut `'i'`, tandis que `length(S)` vaut 19. La commande `ischar` permet de tester si une expression est du type chaîne de caractères. Ainsi, `ischar('1.32')` renvoie `1`, alors que `ischar(1.32)` renvoie `0`. Les chaînes de caractères étant des vecteurs, elles sont concaténables, ainsi, le code Matlab

```
['Ceci est ','une chaîne']
```

donnera

```
'Ceci est une chaîne'
```

On peut également définir sans difficulté des matrices de caractères.

Il est possible de convertir un réel en une chaîne de caractères qui représente son écriture en base 10. Ainsi `num2str(1.32)` renvoie la chaîne de caractères `'1.32'`. L'opération inverse peut être obtenue par la commande `str2num`. La commande `dec2base` permet d'obtenir la chaîne de caractères qui représente l'écriture dans une base de numération d'un entier. Ainsi `dec2base(11,3)` renvoie la chaîne `'102'`, qui est l'écriture en base 3 du nombre 11. Les commandes `dec2bin` et `dec2hex` sont des raccourcis pour les bases 2 et 16 respectivement. La commande `base2dec` permet d'effectuer l'opération inverse : `base2dec('102',3)` renvoie `11`, qui n'est pas une chaîne mais un entier donc un nombre.

La commande `char` permet d'obtenir une chaîne de caractères à partir d'un vecteur de codes ASCII. Ainsi, `char([65,66,67])` renvoie `'ABC'`. Les codes supérieurs ou égaux à 127 donneront des caractères qui dépendent de la page de code du système.

Les commandes `lower` et `upper` permettent de convertir la chaîne de caractères passée en argument en minuscules et en majuscules respectivement.

La commande `sprintf` permet de construire des chaînes de caractères à partir de variables de différents types, cf. `help sprintf`. Réciproquement, la commande `sscanf` permet de décomposer une chaîne de caractères en plusieurs variables de différents type, cf. `help sscanf`. Ces deux commandes généralisent les commandes `num2str` et `str2num` respectivement. Elles sont très pratiques. Voici un exemple explicatif :

```
%% Illustration de l'utilisation de sprintf.
clear;
r=1.20;
n=5;
s='blabla';
chaine=sprintf('r vaut %f, n vaut %d et s vaut %s',r,n,s);
disp(chaine)
%% Illustration de l'utilisation de sscanf.
clear
chaine=['1.2 65 3.8 44','BLAbla '];
X=sscanf(chaine,'%f %s %f %d %s')
```

0.6 Entrées et sorties

La commande `input` permet de demander à l'utilisateur de saisir des valeurs de variables à utiliser. La commande `pause` permet de stopper l'exécution de Matlab pendant un temps déterminé. On peut préciser le nombre de secondes de pose ou revenir à Matlab en appuyant sur n'importe quelle touche. La forme spéciale `pause off` désactive les pauses tandis que `pause on` les réactive.

La commande `save` permet de sauvegarder le contenu de certaines variables dont vous souhaitez garder une trace ainsi que leur nom dans un fichier dont le nom par défaut est matlab.mat. Ce fichier peut être appelé par la commande `load` qui restaure donc toutes les variables que vous avez sauvegardées.

```
% La commande input permet d'afficher une question et de récupérer la
% réponse dans une variable
n=input('Donnez la valeur de n : ')
%
% La commande sprintf permet de faire un affichage formaté de chaînes
% de caractères comprenant des variables. Faire help sprintf.
%
entier=1;
reel=pi;
sprintf('Voici un entier %i et un réel %12.8f\n',entier,reel)
%
%% Sauvegarde et lecture des variables dans des fichiers
%
% La commande save permet de sauvegarder une variable dans un fichier
A=rand(1000,10);
save 'mata' A; % La matrice A est sauvegardée dans le fichier mata.dat
% La commande load permet de lire un fichier de données
clear A        % On détruit A
load 'mata'    % On lit le fichier mata.dat
whos           % A apparaît de nouveau
% Pour sauvegarder A sous forme de fichier texte
% (le nom de la variable est perdu)
save -ASCII 'mata.txt' A
% Pour lire la matrice dans un fichier texte et l'affecter à une variable
B=load('mata.txt')
% Pour sauvegarder toutes les variables en cours
save   % Sauvegarde toutes les variables dans le fichier matlab.mat
load   % Pour restaurer les variables sauvegardées
```

Les commandes `fprint` et `fscanf` permettent de faire des lectures et écritures formatées vers des fichiers de données :

```
%%% Exemple d'entrées sorties sur un fichier
clear;

% Une matrice
position=rand(1,10);
vitesse=rand(1,10);

% Ouverture du fichier en mode réécriture (w+)
% Le contenu précédent est détruit
id=fopen('donnees.txt','w+');
% Ecriture des variables dedans
fprintf(id,'Position = %f, vitesse = %f\n',position,vitesse);
```

```
% Fermeture du fichier
fclose(id);

% Jetez un coup d'oeil au fichier donnees.txt !

% Ouverture du fichier en mode lecture
id=fopen('donnees.txt','r');
% Lecture de 10x10 éléments
PV=fscanf(id,'Position = %f, vitesse = %f\n',[10 10]);
% Fermeture
fclose(id);

% Comparer PV et [position',vitesse']
```

0.7 Opérations logiques, boucles et exécutions conditionnelles

Pour Matlab, tout nombre peut être considéré comme une valeur logique, en identifiant les nombres non nuls à *vrai* (noté `1`) et le nombre zéro à *faux* (noté `0`). La commande `boolean` permet d'obtenir le booléen associé à un nombre vu comme une valeur logique. Les principales opérations sur les valeurs logiques sont :

- la négation logique `~`
- le « ou » logique `|`
- le « et » logique `&`

On peut également utiliser les fonctions `or`, `not` et `and`. Il y en a d'autre, faites donc un `help ops`. On peut créer des valeurs logiques à partir de nombres avec les opérations de comparaisons, qui activent automatiquement la nature logique de leurs arguments :

- « `a` égal à `b` » s'écrit : `a == b`
- « `a` différent de `b` » s'écrit : `a ~= b`
- « `a` supérieur strictement à `b` » s'écrit : `a > b`
- « `a` supérieur ou égal à `b` » s'écrit : `a >= b`
- « `a` inférieur strictement à `b` » s'écrit : `a < b`
- « `a` inférieur ou égal à `b` » s'écrit : `a <= b`

On peut auusi utiliser les fonctions `eq`, `ne`, `gt`, `ge`, `lt` et `le`. Le code Matlab `R=randn(5,5); M=(R>2.1)` crée une matrice `M` de même taille que `R` qui contient des `0` là où `R` est inférieure à 2.1 et des `1` là ou `R` est supérieure ou égale à 2.1. C'est donc une matrice « booléenne ». La commande `any(V)` renvoie `1` si au moins l'un des éléments du vecteur `V` est non nul, et `0` sinon. La fonction `all(V)` renvoie `1` si tous les éléments du vecteur `V` sont non nuls, et `0` sinon. Enfin la fonction `find(V)` renvoie les indices correspondants aux éléments non nul du vecteur `V`. Ainsi, `V(find(V>2))` renvoie les valeurs supérieures à 2 du vecteur `V`.

Les commandes Matlab `isreal`, `ischar`, `issparse`, `isnan`, `isinf`, `isfinite` permettent de tester la nature des variables. Ainsi, `isreal(a)` renverra `1` si la variable `a` est un réel et `0` sinon.

```
%%% Exemples de calculs logiques
V=[1,0,-3,3.4];     % Un vecteur de nombres
boolean(V)          % Doit donner [1,0,1,1]
~V                  % Doit donner [0,1,0,0]
ischar(V)           % Doir donner 0
isreal(V)           % Doit donner 1
isfinite(V)         % Doit donner [1,1,1,1]
isinf(1./V)         % Doit donner [0,1,0,0]
```

```
isnan(cos(1./V))    % Doit donner [0,1,0,0]
V>0.5               % Doit donner [1,0,0,1]
(V>-10 & V<=3)      % Doit donner [1,1,1,0]
(V>-10 | V<=3)      % Doit donner [1,1,1,1]
find(V>0.5)         % Doit donner [1,4]
V(find(V>0.5)).^2   % Doit donner [1.0,11.56]
any(V)              % Doit donner 1
all(V)              % Doit donner 0
% Aide sur les opérations élémentaires
help ops
```

Exercice 0.7.1. Que donne `find(M)` pour une matrice `M` ? Même question pour les commandes `[I,J]=find(M)` et `[I,J,K]=find(M)`.

Le langage Matlab comprend les boucles du type `for` et `while` ainsi que les structures d'exécutions conditionnelles du type `if`. L'argument d'un `if` et d'un `while` est de type logique, ce qui n'est pas le cas de celui d'un `for`, qui est de type matriciel. La commande `break` permet de sortir de la boucle `while` où `for` la plus interne.

```
% Boucles FOR
N=10;
for n=1:N,
 for m=1:N,
  A(n,m)=1/(n+m+1);
 end
end

% Boucle WHILE (TANT QUE en français)
n=10;
r=-Inf;
while (A(n,n)>0),
 r=A(n,n);
 n=n+1;
end

% Exécution conditionnelle IF
if n==m
 A(n,m)=1;
elseif abs(n-m)==1
 A(n,m)=-1;
else
 A(n,m)=0;
end

% Exécution ESSAI/ERREUR/INTERCEPTION
% Tente de déterminer l'inverse de M
% Affiche NaN si ça n'est pas possible
% Aucune erreur n'est provoquée.
% À tester avec M=eye(2) et M=ones(1,2)
try, inv(M), catch, disp(NaN); end
```


0.8 Fonctions et fichiers .m

Les *commandes* Matlab peuvent toutes être considérées comme des *fonctions*, c'est-à-dire des *entitées nommées* qui prennent des paramètres éventuels (arguments) et qui renvoient des résultats (valeurs de retour). Nous allons voir qu'il est très facile de créer de nouvelles fonctions, qui nous serviront de sous-programmes.

Beaucoup de fonctions Matlab, comme par exemple `mean`, sont en réalité déjà écrites en Matlab et le code Matlab correspondant est stocké dans un fichier dont le nom se termine par .m. Pour `mean`, il s'agit de mean.m. Ajouter de nouvelles fonctions à Matlab revient donc à écrire de nouveaux fichiers de ce type. Il est d'usage d'appeler une fonction par le même nom que le fichier Matlab correspondant, au suffixe .m près.

Voici un code qui définit une fonction `stat`, qui prend comme paramètre un vecteur `x` et qui renvoie sa moyenne et son écart type. Ce code devra être stocké dans le fichier nommé stat.m pour que Matlab fasse le lien avec la fonction `stat`. Le commentaire de la ligne n°2 constitue l'aide qui est affichée lorsque l'utilisateur tape `help stat`.

```
function [moyenne,ecartype] = stat(x)
%STAT Revoie la moyenne et l'écart type du vecteur passé en argument.
n = length(x);
moyenne = sum(x) / n;
ecartype = sqrt(sum((x - moyenne).^2)/n);
return % En fait, inutile en fin de fonction...
```

Le fichier stat.m devra être placé dans un répertoire que Matlab scrutera. En général, Matlab cherche automatiquement dans le répertoire en cours. Pour afficher le répertoire en cours, utilisez la fonction `pwd`, pour lister son contenu, utilisez la fonction `ls` et pour changer de répertoire courant, utilisez la fonction `cd`.

Remarque 0.8.1 *(Sous-fonctions).* On peut définir des fonctions dans le corps d'une fonction. Elles ne seront visibles que pour la fonction principale, et l'on parle alors de sous-fonctions, cf. `help function`. *Vous pouvez vous en passer en première lecture.*

0.8.1 Nombre variable d'arguments et de valeurs de retour

Il est parfois commode d'écrire des fonctions qui prennent un nombre variable d'arguments. Ceci peut être fait en Matlab grâce à la commande `nargin` qui renvoie le nombre d'arguments passés à la fonction lors de son appel. Voici un exemple :

```
function fd = firstdigit(x,b,convert)
%fd = firstdigit(x,b)
% Renvoie le premier chiffre non nul dans l'écriture en base b
% de x, en partant de la gauche.
% * x doit être un vecteur de réels >0 pas trop grands.
% * b doit être un entier entre 2 et 36
%     36 = 10 + 26, histoire de pouvoir écrire avec l'alphabet...
%     ce paramètre est optionnel et vaut par défaut 10.
% * convert est un paramètre optionnel valant par défaut 0.
%   si convert n'est pas nul, la sortie fd est un entier correspondant
%   à la valeur en base 10 du premier chiffre, alors que si convert est
%   nul (par défaut), la sortie fd est un caractère alphanumérique.

%%% base 10 par défaut
if (nargin==1), b=10; end
```

```
%%% si dépassement, on sort avec une erreur
if ~(ceil(b)==b & b>1 & b<37)
 error('b n''est pas un entier entre 2 et 36');
end
%%% si x n'est pas positif, on prend sa valeur absolue
if ~(x>0)
 warning('x n''est pas positif, il est remplace par -x');
 x=-x;
end
x=reshape(x,[1,length(x)]);
%%% décalage de x en base b tel que y>=1 en base b
y=x.*b.^ceil(-log(x)/log(b));
%%% si dépassement, on sort avec une erreur
if isinf(y) | isnan(y) | ~isreal(y),
 error('x trop petit ou trop grand par rapport à b');
end
c=floor(y);                    % c = premier chiffre en base 10
s=dec2base(c,b);               % conversion en base b
if nargin==3,
  if convert,
       fd=base2dec(s(:,1),b);  % valeur en base 10 du premier caractère
  end
else
  fd=s(:,1);                   % Premier caractère en base b.
end
return;
```

De la même manière, on peut définir grâce à `nargout` des fonctions qui renvoient un nombre variable de valeurs de retour, un peu comme le fait la fonction `eig` que nous avons déjà vue. Voici un exemple de fonction Matlabqui renvoie une approximation constante par morceaux de la fonction de répartition empirique d'un échantillon aléatoire, en utilisant la fonction Matlab `hist` :

```
function  [ecdf,mi,ma,dt,rg] = empcdf(e,n)
%EMPCDF renvoie une approximation constante par morceaux de la
%fonction de répartition empirique d'un échantillon.
%[ecdf,min,max] = empicdf(e,n)
% * e est un échantillon (vecteur de réalisations i.i.d.)
% * n est un entier optionnel valant par défaut 10.
% * L'intervalle [min(e),max(e)] est subdivisé en n sous-intervalles
%   de même longueur. ecdf est un vecteur de longueur n.
%   ecdf(i) représente la valeur de la fonction de répartition
%   empirique à l'extrémité droite du i-ème sous-intervalle (classe).
% * Les valeurs de retour mi,ma,rg et dt sont optionnelles
%   et correspondent respectivement à l'étalement de l'échantillon et au
%   pas de discrétisation associé à n.
% * Les sorties peuvent être exploitées pour un tracé d'une approximation
%   affine par morceaux de la fonction de répartition empirique avec
%   plot([mi:dt:ma],ecdf)
%EMPCDF fait appel à la fonction HIST.
if (nargin==1), n=10; end;
if (nargout >= 2), mi=min(e); end
if (nargout >= 3), ma=max(e); end
if (nargout >= 4), dt=(ma-mi)/n; end
if (nargout >= 5), rg=ma-mi; end
```

```
[frequences,classes]=hist(e,[mi:dt:ma]);
ecdf=cumsum(frequences)/length(e);
return;
```

0.8.2 Laïus sur les fichiers .m et Matlab

La commande Matlab `type` permet de lister le contenu du fichier .m d'une fonction. Ainsi, `type rbinom` va vous montrer le code source Matlab de la fonction Stixbox `rbinom`. Un certain nombre de fonctions Matlab ne correspondent à aucun fichier .m, elles sont « internes » à Matlab pour une plus grande efficacité et l'on parle de fonctions « built-in » en anglais. C'est par exemple le cas de la fonction `type` elle-même ! Cette séparation entre fonctions internes et externes se retrouve dans la plupart des interpréteurs de langages. La commande `exist` permet de connaître le type d'une variable ou d'une fonction. Vous pouvez essayer par exemple `exist('mean')`.

Sur la plupart des systèmes informatiques, les fichiers sont organisés en une structure arborescente de répertoires qui contiennent des fichiers et des sous-répertoires. La commande Matlab `which` donne le chemin du fichier .m associé à une fonction Matlab, lorsqu'il existe. Par exemple, `which mean -ALL` affichera tous les fichiers mean.m avec leurs chemins. Seul le premier sera utilisé par Matlab ! La commande `what` liste les fichiers .m du répertoire en cours. Faites `help what` et `help which` pour plus d'information.

Matlab a besoin de savoir où chercher les fichiers .m dans l'arborescence des fichiers du système. Il recherchera automatiquement dans une liste de réperoires appelée « PATH ». La commande `path` vous liste les répertoires du « PATH ». Les commandes `addpath` et `rmpath` permettent d'ajouter et d'enlever des répertoires de la liste « PATH ». Ces modifications ne sont pas permanentes et sont perdues lorsque vous quittez Matlab.

Par défaut, la liste « PATH » contient les répertoires de Matlab qui contiennent eux-même les fichiers .m constituant Matlab. Si Stixbox a été correctement installée, le répertoire qui contient les fichiers .m qui la constituent a été ajouté au « PATH » par l'administrateur système. Afin d'accélérer la lecture des fichiers .m, Matlab maintient en permanence une base de données des fichiers .m avec le répertoire associé qui les contient. La commande `rehash` permet de mettre à jour cette base de données en relisant la liste « PATH ».

Il faut enfin savoir que Matlab cherche d'abord les fichiers .m des fonctions dans le répertoire courant (« working directory »). Répétons-le, la commande `pwd` affiche le répertoire courant (« print working directory »), la commande `cd` change de répertoire courant (« change directory ») et les commandes `ls` ou `dir` listent le contenu du répertoire courant.

En général, vous n'aurez pas à modifier la liste « PATH ». Sa modificaton permanente dépend du système utilisé (Unix, MS-Windows,...).

0.8.3 Récursivité

Le langage Matlab autorise la récursivité, comme le montre l'exemple classique suivant :

```
function fn=fact(n);
%FACT calcule de façon récursive et inefficace la factorielle de n
%fn=fact(n);
if (n<=0), fn=1; else fn=n*fact(n-1); end;
return;
```

0.8.4 Fonctions en ligne

Pour des fonctions très légères, on peut procéder autrement que par la création d'un fichier .m : la directive `inline` permet de créer des fonction nouvelles à la volée. Voici un exemple :

```
% On définit une nouvelle fonction trisin qui prend 3 paramètres a,b et c
trisin=inline('sin(a)*sin(b)*sin(c)');
% On teste notre nouvelle fonction
trisin(pi/2,pi/4,pi/2)
```

Matlab détermine automatiquement le nombre de paramètres de la fonction. Bien entendu, une fois que vous quitterez Matlab, cette fonction sera perdue car elle n'est pas sauvegardée dans un fichier.

Exercice 0.8.2. Apprenez à utiliser la fonction `fplot` pour tracer les graphes de fonctions définies avec `inline`.

0.9 Débogage & optimisation

Diverses commandes permettent de déboguer les programmes Matlab. Citons par exemple `dbstack`, `dbstep`, `dbstop`, `dbcont`, `dbclear`, `dbtype`, `dbup`, `dbdown`, `dbstatus` et `dbquit`. Leur aide vous renseignera sur leur usage.

La commande `keyboard` permet à l'utilisateur de reprendre le contrôle en interrompant momentanément l'interpreteur Matlab, afin par exemple d'examiner le contenu des variables, cf. `help keyboard`.

Dans une fonction, la commande `error` permet d'afficher un message d'erreur et provoque la sortie immédiate de la fonction, tandis que la commande `warning` permet d'afficher un message d'avertissement, sans pour autant interrompre l'exécution de la fonction. Les messages d'avertissement peuvent être désactivés, cf. `help warning`. Les commandes `lasterr` et `lastwarn` permettent d'afficher respectivement le dernier message d'erreur et le dernier message d'avertissement. La commande `mfilename` permet de connaître le fichier .m qui contient la fonction en cours d'exécution.

Les commandes `tic` et `toc` permettent de déterminer le temps d'execution d'une commande, ce qui est très utile pour l'optimisation et la comparaison lors du test de différentes méthodes. Voici un exemple dont la sortie graphique est donnée par la figure 2 :

```
% Comparaison de la rapidité de plusieurs méthodes
% de calcul de la factorielle grace a tic & toc.

clear; nmin=2; nmax=150; N=[nmin:nmax];
for n=N,
 m=n-nmin+1;
 tic; fact(n); T(m,1)=toc;
 tic; f=nmin; for i=nmin+1:n; f=f*i; end; T(m,2)=toc;
 tic; cumprod(nmin:n); T(m,3)=toc;
 tic; gamma(n-1); T(m,4)=toc;
end
clf; hold on; title('Calcul de la factorielle');
plot(N,T(:,1),N,T(:,2),N,T(:,3),N,T(:,4));
xlabel('n'); ylabel('Temps');
legend('Recursif','Boucle','cumprod','gamma built-in',2);
```

La commande `cputime` permet d'obtenir le *temps machine* utilisé par Matlab depuis son lancement, cf. `help cputime`.

Exercice 0.9.1. Pourquoi les tracés sont linéaires dans l'exemple précédent ?

Exercice 0.9.2. Utiliser `cputime` en lieu et place de `tic` et `toc` dans l'exemple précédent.

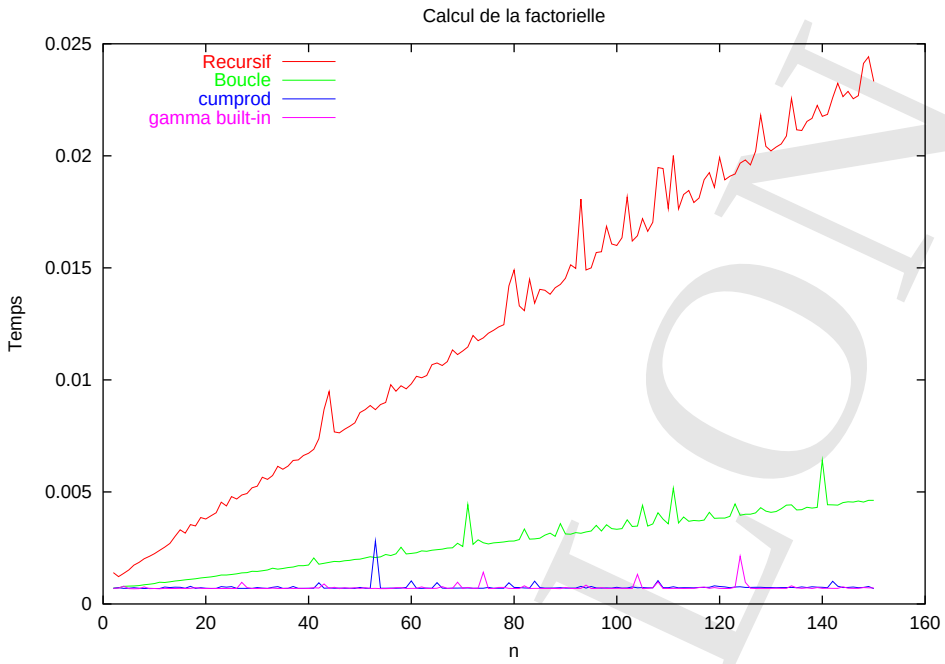

FIG. 2 – Comparaison des temps de calcul de la fonction factorielle (Octave 2.1).

0.10 Derniers conseils

Nous verrons, tout au long de ce livre, plusieurs autres fonctions Matlab dont il n'a pas été question dans ce chapitre. Voici quelques petits conseils avant de vous lancer seul :

1. Expérimentez vous-même en vous servant de l'aide au moyen des fonctions `help` et `lookfor`.
2. La combinaison de touches `Ctrl-c` (appuyer sur la touche `Ctrl` et sur la touche `c` en même temps) permet d'interrompre l'exécution de la commande en cours, ce qui peut être très utile pour sortir d'une longue boucle... La touche `tab` permet de compléter la commande que l'on a commencé à saisir.
3. N'oubliez pas de désactiver l'affichage d'une commande en la suffixant par un point-virgule, surtout dans une boucle ou pour une grosse matrice.
4. Sauvegardez régulièrement votre code Matlab dans un fichier pour éviter de le perdre en cas de problème.
5. Dites-vous bien que vous vous tromperez bien plus souvent que Matlab !
6. Sous Matlab, il est souvent plus efficace d'adopter une formulation matricielle en lieu et place de boucles. Il faut apprendre à « penser matriciel » ce qui vous apportera un gain de temps substantiel. Néanmois, si vous débutez en Matlab, les boucles sont parfois plus lisibles, à vous de voir.
7. Si vous disposez d'un micro-ordinateur PC chez vous, nous vous encourageons à installer les logiciels libres Scilab ou Octave, qui ressemblent beaucoup à Matlab.
8. Exercez-vous tout au long de l'année pour vous familiariser et réflechir à des développements, n'attendez pas le lendemain des écrits pour vous y mettre !

9. Dites-vous bien que le jury n'attend pas des « petits génies » de l'informatique, mais plutôt des candidats qui l'épatent avec des mathématiques appliquées...
10. Ne prenez pas cette liste pour les dix commandements.

Un dernier petit exercice avant de clore ce chapitre introductif :

Exercice 0.10.1. Comprenez-vous le code suivant ?

```
clear;
n=100; a=1; b=2;
A=repmat([-n:n].^2,2*n+1,1);
B=(sqrt(a*A+b*A')<n-1)+(sqrt(a*A+b*A')>n+1);
figure(1); clf; imagesc(B);
```


