

Introduction à MATLAB

EMMANUEL ZENOU

zenou@isae.fr

<http://www.zzenou.net>

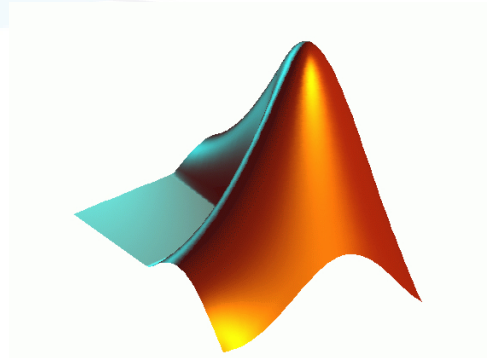
Institut Supérieur de l'Aéronautique et de l'Espace

Formation SUPAERO

<http://www.isae.fr>

Cette initiation à MATLAB a pour objectif de se familiariser à un outil très utilisé à SUPAERO. Il a également pour objectif d'initier (pour ceux qui n'y ont jamais touché) à la programmation et à l'algorithmique, ce qui est indispensable à tout bon ingénieur aujourd'hui. En effet, beaucoup de notions introduites ici ne sont pas propres à MATLAB mais à tout langage structuré comme le C/C++, le Java, *etc.*

Pourquoi MATLAB? Le succès actuel de MATLAB vient de sa simplicité de prise en main et d'utilisation. De plus, il existe des boîtes à outils (*toolbox*) optionnelles mais très utiles dans certains domaines comme l'optimisation, le traitement du signal et de l'image, l'apprentissage (réseaux de neurones...), l'automatique (Simulink), *etc.* Ce logiciel est de plus très utilisé tant dans le monde industriel que dans le monde universitaire.



Il existe un certain nombre de concurrents à MATLAB comme Mapple ou SciLab¹. Ce dernier, développé par l'INRIA (Institut National de Recherche en Informatique et en Automatique) et l'ENPC, a pour avantage d'être *open source*, c'est-à-dire non seulement gratuit mais dont le code source est accessible et réutilisable -sauf en général pour des activités commerciales. Il est malheureusement encore aujourd'hui moins avancé et surtout moins généraliste que MATLAB.

La première partie (page 4) présente les principales instructions MATLAB utilisées presque tous les jours. La seconde partie (page 14) introduit les principales notions de programmation à connaître sous MATLAB. La troisième partie (page 20) propose quelques applications simples où MATLAB s'avère efficace et simple à mettre en œuvre.

Vous trouverez ici et là sur le net ou dans les bibliothèques bon nombre de documents sur MATLAB. Vous avez toute initiative pour rechercher par vous même les informations dont vous avez besoin. Vous y trouverez à la fin un index qui regroupe l'ensemble des commandes les plus courantes.

Enfin, il est indispensable de savoir que ce document est disponible aussi bien d'Oulan-Bator que du siège de l'ONU à New-York, par internet (<http://pro.zzenou.net/supaero/matlab>). Vous trouverez toutes les sources nécessaires à ce projet.

Bon travail !

¹<http://www.scilab.org>

Table des matières

1	Généralités	4
1.1	L'environnement MATLAB	4
1.2	Premiers pas	4
1.3	Typage	5
1.4	Macros	5
1.5	Affichage d'une donnée	6
1.6	Vecteurs	6
1.7	Matrices	7
1.8	Figures	9
1.9	Initialisation de variables	10
1.10	Variables aléatoires	11
1.11	Opérateurs logiques	12
1.12	Polynômes	12
1.13	Entrées - Sorties	13
1.14	Attention, danger !	13
2	Programmation	14
2.1	Instructions classiques	14
2.2	Fonctions	14
2.3	Structure de données	15
2.4	Paramètres dynamiques	16
2.5	Récurrence	17
2.6	Récursivité	17
2.7	Déboguer un programme	18
3	Applications	20
3.1	Régression polynomiale et Interpolation	20
3.2	Modélisation de données SCAO	20
3.2.1	Régression linéaire	21
3.2.2	Régression à l'ordre n	21
3.3	Reconstitution de signal	22
3.3.1	Le signal acoustique	22
3.3.2	Enregistrement du signal accoustique	22
3.3.3	Interpolation par fenêtre	23
3.3.4	Fenêtre glissante	24
A	Annexe A - Principales instructions MATLAB	25
A.1	Généralités	25
A.2	Valeurs spéciales	25
A.3	Matrices particulières	26
A.4	Opérateurs matriciels	26
A.5	Programmation	27
A.6	Figures	27
A.7	Fonctions trigonométriques	28
A.8	Polynômes	28
A.9	Instructions diverses	29

B	Index des fonctions les plus courantes	31
B.1	Généralités	31
B.2	Affichage	31
B.3	Vecteurs, Matrices	31
B.4	Aléatoire	31
B.5	Programmation	31



ISAE

Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

1 Généralités

Vous trouverez dans ce document

- une liste classée des principales **instructions** ou **commandes** que vous trouverez sous MATLAB, annexe A page 25,
- et un **index** en dernière page.

Le symbole \leftrightarrow signifie qu'il faille appuyer sur la touche entrée (ou return).

1.1 L'environnement MATLAB

Pour lancer MATLAB, sous UNIX, ouvrir une fenêtre de commande et taper `matlab&`, le `'&'` permettant de garder la main dans la fenêtre de commande. Sous Windows® ou Mac, double-cliquer sur l'icône correspondant.

Un environnement s'affiche à l'écran sous vos yeux émerveillés. Il est composé en général des cinq fenêtres suivantes :

- *Current directory*, qui vous indique le répertoire courant,
- *Command History*, qui regroupe dans une pile les commandes passées,
- *Workspace*, très utile, qui vous donne les variables en mémoire, leur type et leur taille,
- *Launch Pad*, où vous avez accès en outre aux boîtes à outils présentes,
- et enfin *Command Window*, la fenêtre de commande.

► Créer un répertoire `InitiationMatLab` et se mettre dans ce répertoire.

L'aide sous MATLAB est en général très bien faite. Pour y accéder, cliquer sur l'icône représentant un point d'interrogation bleu. Sur la gauche, cinq onglets sont accessibles :

- *Contents*, où sont rangées les fonctions par thème et boîte à outils,
- *Index*, qui permet de rechercher une fonction à partir du nom,
- *Search*, qui permet de chercher une fonction à partir de mots-clés,
- *Demos*, à découvrir par vous-même,
- et enfin *Favorite*, qui permet de stocker les pages utiles sans avoir à les rechercher à chaque fois.

1.2 Premiers pas

Dans un premier temps, tapez simplement

`a=7` \leftrightarrow

puis dans un deuxième temps

`b=3 ;` \leftrightarrow

(avec le point-virgule)

► **Question 1** *Que constatez-vous ? Quel est l'intérêt du point-virgule ?*

Remarque : Si vous oubliez le point-virgule, ou si l'exécution d'un programme est trop long, il arrive parfois que MATLAB se bloque dans son exécution (on voit alors *"busy"* en bas à gauche de la fenêtre). Pour arrêter l'exécution, sélectionner la fenêtre de commande et faire `Control C`.

► **Question 2** *Effectuer les différentes opérations : $a+b$, $a*b$, a/b .*

Remarque : il est possible d'utiliser des commandes UNIX telles que `ls`, `cd`, `pwd`, etc. dans la fenêtre de commandes MATLAB.

1.3 Typage

Il existe deux techniques pour afficher les variables en mémoire : soit dans l'onglet *Workspace*, soit par les commandes `who` et `whos`.

► **Question 3** *Taper `who` et `whos`. Quelle différence y a-t-il entre ces deux instructions ?*

Dans la fenêtre *Workspace*, on voit apparaître un certain nombre d'informations liées à chacune des variables. L'une de ces informations est le **typage** de la donnée. Dans la plupart des langages de programmation, les variables doivent être préalablement définies, à partir d'une commande liée à son type. Sous MATLAB, il n'est pas nécessaire de déclarer la variable en amont, et le logiciel adapte le typage nécessaire, ce qui peut s'avérer dangereux (Voir section 1.14)

Le typage permet d'associer à toute variable un *type* de variable, c'est-à-dire un protocole de codage de la variable en langage machine. Il est associé également un espace mémoire nécessaire à coder cette variable. Par exemple, si une variable est de type `logical`, elle ne comprend que deux valeurs possibles : 1 (TRUE) et 0 (FALSE). En théorie, seul 1 bit est nécessaire pour coder cette variable.

Il en existe plusieurs dans les différents langages de programmation. Sous MATLAB, les types les plus courants sont **double** (pour double précision -64 bits), **logical** (binaire), **char** (caractère -8 bits), **uint8** (entier non signé - 8 bits), mais il en existe plein d'autres (voir l'aide).

1.4 Macros

Il est en général bien plus commode d'écrire l'ensemble des fonctions successivement utilisées dans un fichier, afin de garder une trace des opérations successives. Dans la fenêtre principale, ouvrir `File -> New -> M-file` ou bien cliquer sur le rectangle blanc. Une fenêtre s'ouvre (que l'on peut intégrer à l'espace de travail principal en faisant `View -> Dock Untitled1`) dans laquelle il est possible d'inscrire autant d'instructions que nécessaire.

La plupart du temps il est préférable de commencer une macro par une remise à zéro complète. Écrire dans le fichier :

```
clear all, close all, clc.
```

► **Question 4** *Que signifient ces commandes ?*

Pour sauvegarder la macro, le plus rapide est d'appuyer simultanément sur les touches `Ctrl + S` ([pomme] + `S` sous Mac, `Ctrl + x` puis `Ctrl + S` sous UNIX), ou de cliquer sur la disquette. Sauvegarder votre macro sous le nom 'Essai.m' dans un répertoire approprié.

► **Question 5** *Écrire dans votre macro les commandes suivantes (nous verrons la signification plus loin) :*

```
clear all, close all, clc;
N=100;
p=.8;
A = rand(1,N);
prob = sum(A<p)/N
erreur = (p-prob)/p
```

Afin de lancer l'exécution d'une macro, la méthode la plus rapide consiste à cliquer sur l'icône représentant une feuille blanche avec une flèche bleue orientée vers le bas ; une deuxième méthode consiste à écrire le nom de la macro dans la fenêtre de commande.

► **Question 6** Exécuter cette macro pour plusieurs valeurs de N et p .

1.5 Affichage d'une donnée

En tapant simplement dans la fenêtre de commande $a=9$, il s'affiche sur deux lignes le résultat de cette opération. Pour lire de nouveau la valeur de a , il suffit de taper a et de valider.

Il est possible d'afficher des données proprement avec la commande `disp` :

```
disp(a) ←
```

Afin de mettre en forme l'affichage, il est possible d'écrire :

```
disp(['la valeur de a est ' num2str(a)]) ←
```

► **Question 7** Quelle est l'utilité de la fonction `num2str` ?

Remarque : Il est possible d'afficher une apostrophe ' en la doublant dans la chaîne de caractères :

```
disp('L''éléphant d''Ukraine est d''enfer !') ←
```

1.6 Vecteurs

Pour créer un vecteur, taper

```
V1 = [1 2 3 4 5 6] ←
```

Le vecteur s'affiche dans la fenêtre de commande.

Taper ensuite :

```
V2 = V1' ←
```

► **Question 8** Quelle est la fonction de l'opérateur ' ?

Dans une fenêtre de commande, taper successivement :

```
V3 = 1 :11 ←
```

```
V4 = 1 :2 :11 ←
```

```
V5 = 1 :.1 :3 ←
```

► **Question 9** Que constatez-vous ?

► **Question 10** En déduire la construction d'un vecteur X de $-\pi$ à 2π , de pas 0.01 radian. Quelle est la longueur de ce vecteur ?

Pour obtenir la valeur de la composante i , il suffit de taper

```
V5(i) ←
```

► **Question 11** Quelle est la troisième composante du vecteur $V5$?

En tapant `V1(end)`,

► **Question 12** Quelle est l'utilité de la variable `end` ?

Une autre façon d'obtenir ce résultat est connaître la taille (longueur) du vecteur.

► **Question 13** *Quelle commande permet-elle d'obtenir la longueur du vecteur ?*

► **Question 14** *Donner deux formulations différentes pour obtenir le dernier élément d'un vecteur.*

Les données dans MATLAB sont toujours représentées sous forme de vecteur ou de matrice. Une des fonctions essentielles est la détermination du minimum et du maximum des éléments d'un vecteur. Dans une fenêtre de commande, taper successivement :

$V = \text{rand}(6, 1) \leftarrow$

$a = \text{min}(V) \leftarrow$

$[a, b] = \text{min}(V) \leftarrow$

La fonction `rand` permet de tirer au hasard une valeur entre 0 et 1 (voir plus loin).

► **Question 15** *Quels sont les arguments de sortie des fonctions `min` et `max` ?*

1.7 Matrices

Les premières versions de MATLAB il y a une vingtaine d'années étaient des bibliothèques d'inversion de matrices (l'acronyme MATLAB signifiant "Matrix Laboratory"). Le type fondamental des variables de MATLAB est le type matrice. Une matrice est un tableau à deux dimensions d'expressions de même type.

Soit M la matrice de nombres aléatoires suivante : $M = \text{rand}(4, 6) \leftarrow$

► **Question 16** *Quelle est la taille de la matrice M ? Afficher proprement (i.e. avec une belle phrase pleine de mots poétiques) les informations relatives à cette variable.*

► **Question 17** *Utiliser les fonctions `min` et `max` sur M : que constatez-vous ? Comment obtenir les valeur `min` et `max` sur tous les éléments de la matrice ?*

On constate que les termes d'une même ligne sont séparés dans la déclaration par des virgules ou par des espaces. Les lignes sont séparées entre elles par des points-virgules. Raisononnons sur des matrices plus simples.

Tapez successivement les commandes suivantes :

$A = [1\ 2\ 3\ 4; 5\ 6\ 7\ 8; 9\ 10\ 11\ 12; 13\ 14\ 15\ 16] \leftarrow$

$B = A(2, 2) \leftarrow$

$C = A(:, 2) \leftarrow$

$D = A(1 : 3, 2 : 3) \leftarrow$

$E = A' \leftarrow$

► **Question 18** *Interpréter dans ce cas les opérateurs `:` et `'`*

► **Question 19** *En déduire les matrices F et G formées respectivement des deux dernières lignes et des deux dernières colonnes de A .*

Les opérations matricielles classiques (multiplication, puissance, division) sont directement accessibles par leur symbole normal. Dans la fenêtre de commande, taper

$F * G \leftarrow$

puis

$F .* G' \leftarrow$

► **Question 20** *Que constatez-vous ? Quelle différence y'a-t-il entre les opérateurs * et .* ?*

De même,

$A^2 \leftarrow$

puis

$A.^2 \leftarrow$

► **Question 21** *En déduire l'utilité d'un point avant un opérateur quelconque.*

Le plus intéressant et le plus inhabituel dans MATLAB est la surcharge de type des opérations. Généralement, en dehors des opérateurs classiques, MATLAB interprète toute opération dont les termes sont des matrices comme une opération à faire terme à terme.

À l'aide du vecteur X précédemment construit (question 10), taper et exécuter la macro `vecteurx.m` suivante :

```
k = 2;
```

```
ksin2x = k*sin(2*X);
```

```
ksin2xpositif = ksin2x > 0;
```

```
disp(['Taille du vecteur ksin2x : ' num2str(length(ksin2x)) ...
```

```
' et du vecteur ksin2xpositif : ' num2str(length(ksin2xpositif))])
```

On constate que la taille des nouvelles variables construites se sont adaptées à la taille initiale de X . Ceci est un avantage *et* un inconvénient. En effet, cela permet une souplesse et une simplicité d'écriture et de programmation, mais rend difficile la recherche de certaines erreurs lors de l'exécution du programme.

Remarque : il est important de constater que les fonctions sont construites à partir de variables *discrétisées*, c'est-à-dire d'une succession de valeurs séparées par un intervalle donné appelé *pas de discrétisation*.

► **Question 22** *Quelle est l'utilité des trois points ... en fin de ligne ?*

En posant

```
A = [-1.9, -0.2, 3.4; 5.6, 7, 2.4; -0.5 -2.2 3.3 ]
```

► **Question 23** *Quelle est l'utilité des quatre fonctions `round`, `floor`, `ceil` et `fix` ?*

► **Question 24** *À l'aide de la fonction `sum`, déterminer la moyenne approchée de la valeur absolue d'une sinusoïde. Vérifiez avec un calcul théorique.*

Il est facile, sous MATLAB, de rechercher les valeurs propres (*eigenvalues*) et vecteurs propres (*eigenvectors*) d'une matrice carrée avec la commande `eig`.

► **Question 25** *Quelles sont les valeurs propres et vecteurs propres de `ceil(A)` ?*

Il existe plusieurs instructions permettant de construire des matrices particulières.

► **Question 26** *Que signifient les instructions suivantes : `eye`, `zeros`, `ones`, `diag`, `triu`, `tril`, `magic`, `toeplitz` ?*

Il existe également plusieurs instructions relatives à des fonctions d'analyse linéaire.

► **Question 27** *Que signifient les instructions suivantes : det, rank, inv, size, norm, eig?*

► **Question 28** *Déterminer les vecteurs propres et valeurs propres de la matrice suivante :*

$$M = \begin{bmatrix} -1 & 0 & 3 \\ 5 & 7 & 2 \\ 0 & -2 & 3 \end{bmatrix}$$

1.8 Figures

La représentation de fonctions se fait essentiellement sous MATLAB par la commande `plot`. Avant cela, la commande `figure` : permet d'ouvrir une nouvelle fenêtre. Dans la macro précédente, taper

```
figure ; plot (X,ksin2x) ;
```

puis

```
figure, plot (X,ksin2x,X,ksin2xpositif) ;
```

► **Question 29** *Quelles sont les options de cette fonction plot ?*

► **Question 30** *Représenter la fonction ksin2x en rouge avec des petits cercles et ksin2xpositif en bleu avec des pointillés.*

► **Question 31** *Utiliser les commandes legend et title pour agrémenter la figure.*

Pour sauvegarder une figure, dans la fenêtre aller dans File -> Export (ou parfois File -> Save as) pour enregistrer dans le format désiré.

► **Question 32** *Sauvegarder la figure en JPG et en BMP.*

Si l'on désire réactiver la figure *i*, il faut l'appeler en tapant `figure(i)`.

► **Question 33** *Activer la première figure et dessiner la fonction ksin3x = k*sin(3*X). Que s'est-il passé sur la première courbe ?*

Pour résoudre le problème, il faut utiliser la commande `hold`.

► **Question 34** *Superposer les courbes des fonctions ksin2x et ksin3x.*

Une autre possibilité consiste à utiliser la commande `subplot`.

► **Question 35** *Représenter les courbes des fonctions ksin2x et ksin3x à l'aide de la fonction subplot.*

Nous allons maintenant essayer de visualiser une figure 3D, représentant par une surface les variations d'une fonction de deux variables. Prenons comme objet d'étude la fonction de deux variables suivante (appelée *Gaussienne*) :

$$z = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-1)^2+y^2}{2\sigma^2}}$$

sur le pavé $[-5, 5] \times [-5, 5]$. Le principe est, comme dans `plot`, de représenter graphiquement des points définis par 3 tableaux de points, le premier pour les abscisses, le second pour les ordonnées et le troisième pour les côtes. A partir du maillage des abscisses et des ordonnées défini par

```
x = -5 :0.2 :5; y = -5 :0.2 :5; ←
```

il faut créer deux tableaux bidimensionnels `xx` et `yy` qui affecteront à chacun des points de la grille respectivement son abscisse et son ordonnée. La fonction `meshgrid` réalise ce travail :

```
[xx,yy] = meshgrid(x,y)
```

Il reste à calculer la côte correspondante en utilisant la vectorisation de la fonction `exp` et des opérations arithmétiques sans oublier le "."

```
z = 1/(sigma*sqrt(2*pi))*exp(-((xx-1).^2+yy.^2)/(2*sigma^2));
```

On prendra $\sigma = 1$. On peut maintenant obtenir un tracé 3D par l'instruction `surf` (ou `surfc`) :

```
figure; surf(xx,yy,z);
```

ou un tracé de lignes de niveaux par l'instruction `contour3` :

```
figure; contour3(z,30)
```

Sauvegarder ces belles figures, que vous pourrez afficher fièrement sur la porte de votre chambre. Constatez qu'il est possible de les faire tourner pour une meilleure visualisation (voir figure 1).

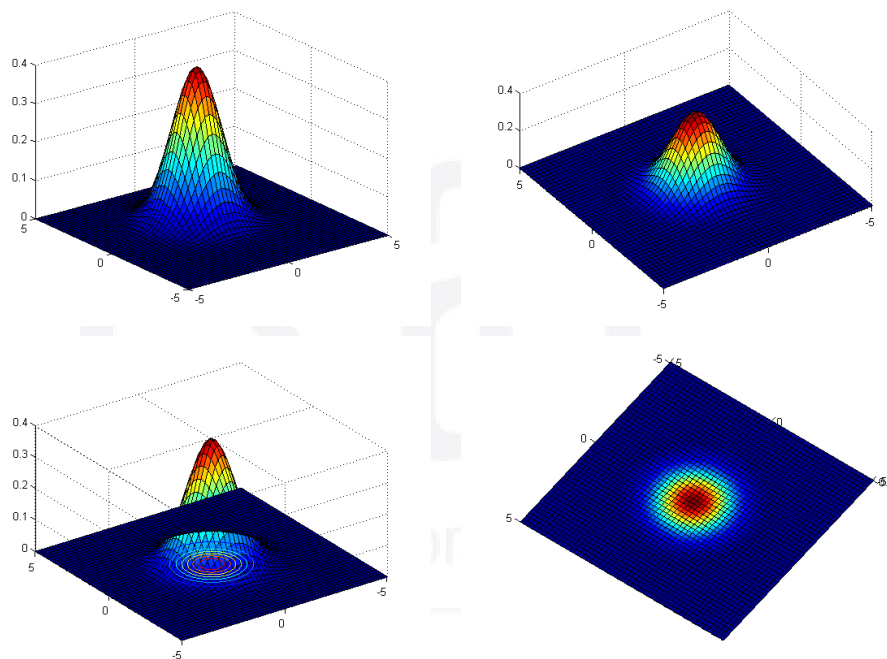


FIG. 1 – Gaussienne 2D de différents points de vue.

► Question 36 Représenter la fonction

$$z = \sin(x^2 + y^2), x \in [-\pi, \pi], y \in [-\pi, \pi]$$

1.9 Initialisation de variables

MATLAB est un outil très souple qui permet d'anticiper les aspirations de son utilisateur. Ainsi, en tapant

```
V6(5) = 7 ←
```

On constate que le vecteur s'est initialisé tout seul...

D'une manière générale, il n'est pas nécessaire, sous MATLAB, de déclarer ses variables. MATLAB utilise l'allocation dynamique de mémoire qui permet de modifier la taille des variables en permanence. Cette facilité ne permet pas de détecter des erreurs de programmation dans la gestion des matrices. Il est conseillé au programmeur d'établir des contrôles pour éviter d'être induit en erreur par l'allocation dynamique de mémoire.

Il est ainsi possible d'initialiser une variable avec la commande

```
V6 = [] ←
```

Ou bien en lui donnant la valeur 0

```
V6 = zeros(1,5) ←
```

Ou une matrice 3×5 :

```
M1 = zeros(3,5) ←
```

D'autres initialisations sont possibles : taper

```
M2 = ones(3,5) ←
```

```
M3 = ones(5) ←
```

Ou bien :

```
M4 = eye(3,5) ←
```

► **Question 37** *Que fait la fonction `eye` ?*

1.10 Variables aléatoires

Il existe différentes techniques pour initialiser aléatoirement un vecteur. Par exemple, taper

```
randperm(7) ←
```

► **Question 38** *Dans une macro `loto.m`, écrire un mini-programme (2 lignes suffisent !) qui permet de tirer 6 numéros au hasard sur 49.*

Les commandes `rand` et `randn` permettent de tirer aléatoirement des nombres en fonction d'une densité de probabilité donnée. Ainsi, taper dans la fenêtre de commande :

```
rand(5) ←
```

puis

```
randn(5) ←
```

► **Question 39** *Quelle différence existe-t-il entre ces deux fonctions ? Rechercher sur le net des informations relatives aux différentes lois de probabilité (notamment la loi uniforme et la loi normale ou gaussienne).*

Afin de visualiser ces distributions, taper dans une macro `tirage.m` :

```
N1 = 100 ; N2 = 10000 ;
```

```
% Tirages :
```

```
X1 = rand(N1,1) ;
```

```
X2 = rand(N2,1) ;
```

```
Y1 = randn(N1,1) ;
```

```
Y2 = randn(N2,1) ;
```

```
% Affichages :
```

```
figure, hist(X1,200) ;
```

```
figure, hist(X2,200);
figure, hist(Y1,200);
figure, hist(Y2,200);
```

► **Question 40** *Interpréter ce que vous observez.*

► **Question 41** *Afficher dans une même fenêtre les histogrammes représentatifs respectivement des trois gaussiennes $\mathcal{G}(\mu, \sigma)$ suivantes :*

- avec $\mu = 0, \sigma = 1,$
- avec $\mu = -3, \sigma = 3,$
- avec $\mu = 2, \sigma = 7,$

Dans une seconde macro `tirage2.m`, taper et exécuter les commandes suivantes :

```
tic; N = 100000; A = randn(N,1); AA = (A<-1); NN = sum(AA); P = NN / N; toc; disp(['P = ' num2str(P)])
```

► **Question 42** *Que signifie cette suite d'instructions ?*

► **Question 43** *Que signifient les commandes `tic` et `toc` ?*

1.11 Opérateurs logiques

Lorsque qu'une comparaison est faite, le résultat est de type `logical array` (voir variable `AA`). Il est ensuite possible de définir des opérations logiques sur ce type de données. Taper :

```
M1 = floor(10*rand(5)); MM1 = M1 > 3;
M2 = floor(10*rand(5)); MM2 = M2 > 7;
N1 = ~MM1;
N2 = MM1 & MM2;
N3 = MM1 | MM2;
```

► **Question 44** *Quelles sont les fonctions de ces opérateurs ?*

1.12 Polynômes

Les polynômes sont gérés, sous MATLAB, par des vecteurs de coefficients dans l'ordre décroissant. Aussi le polynôme

$$x^5 + 2x^4 - x^2 - x + 1$$

est-il représenté sous MATLAB par le vecteur `[1 2 0 -1 -1 1]`. Prenons la variable x :

```
x = -5 : 0.01 : 5;
puis taper
polyval(p,x)
polyder(p)
polyder(p,x)
```

► **Question 45** *Quelle est l'utilité des fonctions `polyval` et `polyder` ?*

► **Question 46** *Représenter le polynôme ci-dessus.*

1.13 Entrées - Sorties

En général, il n'est pas nécessaire de sauvegarder des données et/ou l'espace complet de travail : l'algorithme stocké dans une macro suffit. Cependant, dans certains cas (comme lorsqu'un algorithme est très long à tourner -parfois jusqu'à plusieurs jours !-), il est utile de sauvegarder tous ces éléments.

Pour sauvegarder une ou plusieurs variables il faut taper

```
save [NomFichier] var1 var2 [...] varn
```

ou pour sauvegarder l'espace de travail au complet, cliquer sur File -> Save Workspace As...

Pour récupérer ces données, il suffit de taper

```
load [NomFichier]
```

Il est possible également de sauvegarder des informations dans un fichier sous forme de texte formaté. Pour cela, la première chose à faire est d'utiliser la fonction `fopen` pour ouvrir un fichier :

```
f = fopen(' [NomFichier] ', 'w') en écriture,
```

```
f = fopen(' [NomFichier] ', 'r') en lecture.
```

Ensuite, utiliser la fonction `fprintf` pour écrire dans un fichier (avec le format désiré), et la fonction `fscanf` pour lire un fichier ASCII. Se reporter à l'aide MATLAB pour les options. Pour fermer le fichier, il suffit d'utiliser `fclose`.

1.14 Attention, danger !

MATLAB est un langage facile à prendre en main, et surtout très souple. Probablement TROP souple. En effet, si l'on tape :

```
a = [ 1 2 ; 3 4]
```

Puis

```
a(3,4) = 2
```

Que constatez-vous ?

De même, si on tape :

```
x = -pi :pi :2*pi ;
```

```
y = sin(x) ;
```

```
figure, plot(x,y) ;
```

Que constatez-vous ?

Ainsi MATLAB anticipe les besoins du programmeur, ce qui peut s'avérer extrêmement dangereux dans des contextes délicats. L'opérateur perd un peu la main sur les variables qu'il manipule, notamment (ici) la taille des matrices, l'interpolation de points, etc.

Restez *aware* !

2 Programmation

2.1 Instructions classiques

Nous voyons dans cette section quelques instructions très classiques en programmation : l'instruction de test `if`, les boucles `for` et `while`. On va réécrire le programme précédent à l'aide de ces instructions.

Écrire dans une macro le programme suivant :

```
tic; cpt = 0; N = 100000;
for (n=1 :N)
    a = randn;
    if (a < -1)
        cpt = cpt + 1;
    end;
end;
P = cpt / N;
toc;
disp(['P = ' num2str(P)]);
```

► **Question 47** Comparer le temps d'exécution avec précédemment. Conclure.

Remarque : ne jamais afficher des données dans des boucles, cela ralentit considérablement l'exécution du programme. Pour vous en convaincre, enlever le ; de la ligne `a = randn` et relancer la macro...

Attention : pour valider une égalité, il faut doubler le signe `=`. Pour illustrer ceci, taper dans une fenêtre de commande :

```
a = 2; b = 5;
b == a
b = a
b == a
```

► **Question 48** Réécrire le programme précédent avec l'instruction `while`.

Conclusion : En Matlab, il importe de rendre vectorielles, dans la mesure du possible, toutes les opérations à effectuer, et de réduire au strict minimum les boucles de calcul en utilisant les surcharges de type. A cette condition, les programmes Matlab sont comparables en temps d'exécution, pour des problèmes de taille raisonnable, à des programmes écrits en langage plus performants (C ou C++) et sont beaucoup, beaucoup plus faciles à développer et à tester.

2.2 Fonctions

Lorsqu'on fait appel à un même algorithme plusieurs fois en ne changeant que les variables d'entrée/sortie, il est possible de créer des *fonctions*.

Ouvrir un nouveau fichier et écrire

```
function P = proba(LoiProba, Val, NbIter)
```

Puis enregistrer. Par défaut, il vous propose `proba.m`. Valider et garder ce nom : **le nom du fichier et le nom de la fonction doivent toujours être identique**. Conséquent, vous aurez

toujours un seul fichier `.m` par fonction.

Attention : les noms de fonctions ne doivent comporter ni espace ni lettres accentuées.

Ce nouveau fichier n'est plus une macro mais une fonction. Il est important d'avoir le même nom pour cette fonction que pour le fichier correspondant. Écrire ensuite :

```
% Cette fonction retourne une probabilité empirique P en fonction
% d'une loi de probabilité, d'une valeur et du nombre d'itérations
% si LoiProba = 1, alors la loi de probabilité est une loi uniforme
% sinon, la loi de probabilité est une loi normale
Pu = sum(rand(NbIter,1) < Val) / NbIter;
Pn = sum(randn(NbIter,1) < Val) / NbIter;
if (LoiProba == 1)
    P = Pu;
else
    P = Pn;
end
```

Puis taper dans la fenêtre de commande :

```
MaProba = proba(0, -1, 10000)
```

► **Question 49** taper `Pu` ou `Pn` dans la fenêtre de commande : que constatez-vous ?

Les variables `Pu` et `Pn` sont des variables **locales**, c'est-à-dire qu'elles n'existent que lors de l'appel de fonction. Si une variable de même nom existe déjà en mémoire, en tant que variable *globale*, il n'y aura pas d'interférence entre ces deux données. De la même façon, les variables locales sont inaccessibles à partir de l'espace principal. Ceci est valable pour le nom des variables d'entrée/sortie. Ceci est une grande différence avec les macros, où les variables restent dans l'espace principal.

Il est cependant possible de déclarer dans une fonction des **variables globales**, avec l'instruction `global`, ce qui peut être très utile pour éviter de passer en paramètre toutes les variables de votre code.

Enfin, bien entendu, il est possible d'*optimiser* le code :

```
P = (LoiProba == 1) (sum(rand(NbIter,1) < Val))
+ (LoiProba ~= 1) (sum(randn(NbIter,1) < Val)) ;
```

Mais est-ce bien utile ? Un code doit avant tout être clair et lisible par un autre individu bien moins intelligent et brillant que vous... voire par vous-même après un certain temps, et les 10.000 neurones que vous perdez chaque jour n'expliquent pas toujours tout !

2.3 Structure de données

Si l'on désire sortir plusieurs données d'une fonction, le formalisme est le suivant :

```
function [s1, s2, ..., sn] = NomFonction(e1, e2, ..., ep)
```

► **Question 50** Écrire une fonction qui prend en entrée une valeur `val` et un nombre d'itérations `NbIter`, et qui délivre les deux probabilités empiriques relatives aux deux lois uniforme et nor-

male :

```
function [Puniv,Pnorm] = probas(Val,NbIter)
```

Faire tourner votre fonction. Si vous l'appellez simplement par `MaProba = probas(-1,10000)`, que constatez-vous ?

Le principal inconvénient d'une telle écriture est la rigidité de manipulation : si, en cours de programmation, on désire faire sortir une autre variable non prévue initialement mais utile dans la fonction, il faut non seulement changer la fonction elle-même, mais également tous les appels à cette fonction.

Il est ainsi pratique d'attacher au même nom de variable une série de variables auxiliaires inhomogènes qui en sont des attributs. On crée alors une structure de données dont les composantes sont les *champs*. Ces structures sont très utilisées en C/C++ ou dans des bases de données. Prenons un exemple simple. Taper dans le fenêtre de commande :

```
P.Univ = proba(1,0.7,10000) ←
```

```
P.Norm = proba(0,0.7,10000) ←
```

► **Question 51** *Que constatez-vous ? Quelle est la structure de P ?*

Il est ainsi possible de donner à toute variable des champs de manière dynamique.

► **Question 52** *Réécrire la fonction `probas.m` avec une seule sortie P.*

2.4 Paramètres dynamiques

Il est parfois nécessaire de passer en paramètre de fonction non plus des variables mais d'autres fonctions. Prenons un exemple simple de calcul intégral (numérique par approximation) d'une fonction f , à l'aide de la fonction MATLAB `quad`.

► **Question 53** *Créer une fonction MATLAB (appelée par exemple `mafct`) permettant de construire un vecteur z en fonction de x de la forme $z = e^{-x^2}$.*

Ensuite, pour calculer la surface dans l'intervalle $[-1, 1]$, il suffit de taper :

```
somme = quad(@mafct,-1,1)
```

le caractère '@' permet de faire passer l'adresse de la fonction, et non pas sa valeur.

► **Question 54** *Calculer la moyenne de la fonction $e^{\cos(x)}$.*

Dans le cas d'une fonction monovariante, il n'y a pas d'ambiguïté sur la variable d'intégration. Avec une fonction de plusieurs variables, il faut alors préciser la variable d'intégration et fixer la valeur des autres variables. Par exemple :

```
x = 3;
```

```
somme = quad(@(y)mafct(x,y),-1,1)
```

► **Question 55** *Calculer l'intégrale suivante pour différentes valeurs de x :*

$$\int_{-1}^1 e^{(x-y)^2} dy, \quad x = \{-1, 0, 1\}$$

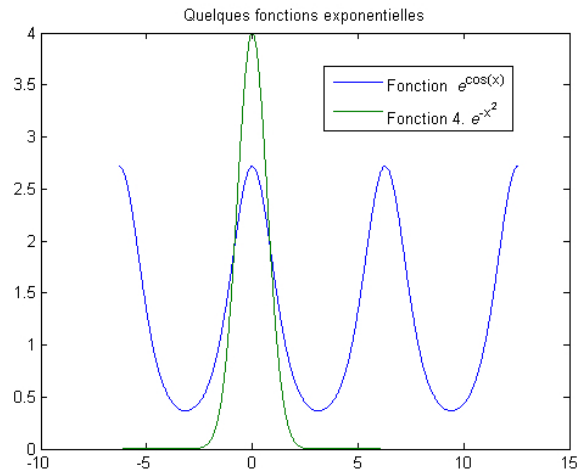


FIG. 2 – Fonctions exponentielles.

2.5 Récurrence

Les fonctions sont très utiles dès que l'on veut introduire de la récurrence dans le programme. Prenons le cas de la suite de Fibonacci, régie par la récurrence suivante :

$$f_n = f_{n_1} + f_{n_2}$$

Elle se construit avec le programme suivant :

```
function f = fibonacci(n)
f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3 :n
    f(k) = f(k-1) + f(k-2);
end
```

► **Question 56** Etablir une courbe montrant le temps de calcul de la suite de Fibonacci sous MATLAB en fonction de la profondeur : $t = f(n)$.

2.6 Récursivité

Une manière élégante et synthétique (mais souvent peu lisible...) d'écrire un programme de récurrence est d'introduire une récursivité, c'est-à-dire lorsque qu'une fonction fait appel à elle-même. Ainsi pour la suite de Fibonacci :

```
function f = fibo_rec(n)
if n <= 1
    f=1;
else
    f=fibo_rec(n-1)+fibo_rec(n-2);
end
```

► **Question 57** Comparer le temps d'exécution avec précédemment.

2.7 Déboguer un programme

Lorsqu'un programme est un peu long ou complexe, il est possible de suivre son évolution "pas à pas" pour comprendre son fonctionnement ou corriger des *erreurs* (i.e. qu'il ne donne pas les résultats attendus *a priori*). On utilise pour cela le mode *debug*.

Écrire les deux programmes suivants :

```
function premiers=NombresPremiers(max)

% Cette fonction permet de trouver tous les nombres
% premiers jusqu'à la valeur 'max'
% Exemple d'utilisation :
% >> NombresPremiers(50)

premiers=[2];
s=size(premiers);
for n=3:max
    reste=99;
    i=1;
    while (reste~=0)&(i<=s(1))&(premiers(i)<=n/2)
        reste=rem(n,premiers(i));
        i=i+1;
    end
    if reste~=0
        [premiers,s] = empile(premiers,n);
    end
end

avec

function [pile_out,taille]=empile(pile_in,x)

% Cette fonction empile un élément dans une pile
pile_out = [pile_in ; x] ;
taille = size(pile_out) ;
```

Il faut maintenant ajouter un point d'arrêt (*breakpoint*, comme disait Shakespeare quand il programmait en MATLAB) sur une ligne du programme soit en appuyant sur l'icône représentant une page blanche avec un point rouge sur le côté gauche, soit dans le menu Debug -> Set/Clear breakpoint de l'environnement. Lorsque le programme est lancé, il s'arrête sur la ligne en question.

► **Question 58** *Taper help NombresPremiers. Que se passe-t-il ?*

Ensuite, plusieurs solutions (dans l'ordre des icônes consacrées) :

- soit on continue le programme pas à pas (*step*),
- soit on continue dans la *function* appelée (*step in*),
- soit on ressort dans la *function* appelée (*step out*),
- soit on continue l'exécution du programme jusqu'au prochain point d'arrêt (*Continue*).

► **Question 59** utiliser ces fonctionnalités pour comprendre pas à pas le programme NombresPremiers.

► **Question 60** utiliser ces fonctionnalités pour comprendre pas à pas le programme fibo_rec.



ISAE

Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

3 Applications

Nous allons voir dans cette partie les deux applications simples suivantes :

1. la régression de points, appliquée à la modélisation polynomiale de données SCAO ;
2. l'interpolation de points, appliquée à la reconstitution de signal.

3.1 Régression polynomiale et Interpolation

Tout système d'équations linéaires de N équations à P inconnues

$$S : \begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1P}x_P = y_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2P}x_P = y_2 \\ \dots \\ a_{N1}x_1 + a_{N2}x_2 + \dots + a_{NP}x_P = y_N \end{cases} \quad (1)$$

peut s'écrire sous forme matricielle

$$S : \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1P} \\ a_{21} & a_{22} & \dots & a_{2P} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NP} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix} \quad (2)$$

ou

$$A.X = Y \quad (3)$$

Partant de cette équation matricielle, trois cas sont possibles :

1. dans le cas où il y a plus d'équations indépendantes que d'inconnues, la matrice A n'est pas inversible mais les matrices $A^t A$ et AA^t le sont. Dans ce cas

$$X = A^\dagger.Y \quad (4)$$

avec ici

$$A^\dagger = (A^t A)^{-1} A^t \quad (5)$$

A^\dagger représente la **pseudo-inverse** de A (voir Annexe ??). Une application typique est la régression polynomiale de données.

2. dans le cas où il y a moins d'équations indépendantes que d'inconnues (cas surjectif), alors certaines variables peuvent être fixées arbitrairement ;
3. enfin, dans le cas où il y a autant d'équations indépendantes que d'inconnues (cas bijectif), alors la matrice A est inversible et

$$X = A^{-1}.Y \quad (6)$$

Une application typique est l'interpolation de données.

3.2 Modélisation de données SCAO

Le fichier `data.txt`² est un fichier ASCII contenant des données du système de contrôle d'attitude et d'orbite d'un satellite d'observation en phase d'acquisition. Les données se présentent sous forme d'un tableau de 4 colonnes :

²qui n'a pas changé de place depuis le début de ce BE : <http://www.zzenou.net/supaero/matlab>

- colonne 1 : vecteur d'index ;
- colonne 2 : temps de mesure ;
- colonne 3 : mesure de roulis ;
- colonne 4 : mesure de tangage ;

► **Question 61** Charger le fichier `data.txt` avec la commande `load`.

► **Question 62** Visualiser dans un même graphe les mesures de roulis et de tangage en fonction du temps.

3.2.1 Régression linéaire

Soit $t = (t_0 \ t_1 \ t_2 \ \dots \ t_N)$ le temps et $f = (f(t_0) \ f(t_1) \ f(t_2) \ \dots \ f(t_N))$ une fonction à approximer. Le but est de trouver l'équation d'une droite qui minimise l'erreur quadratique. Supposons dans un premier temps une droite $\mathcal{D}_0 : y = a \times t$ passant par l'origine. On cherche ainsi à minimiser l'erreur quadratique du terme :

$$\|f - a \times t\|_2 \quad (7)$$

Le a optimal s'obtient grâce à la pseudo-inverse :

$$a = f.t^\dagger \quad (8)$$

► **Question 63** Approximer le roulis et le tangage avec une droite passant par l'origine.

Pour approximer avec une droite quelconque $\mathcal{D} : y = a_1 \times t + a_0$, il suffit de modifier le vecteur temps en une matrice incluant des constantes :

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & \dots & t_N \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

La matrice $A = (a_1 \ a_0)$ des coefficients de cette droite est trouvée par la formule suivante :

$$A = f.T^\dagger$$

Une fois cette matrice trouvée, il est nécessaire d'utiliser la fonction `polyval` pour construire une fonction à partir des coefficients d'un polynôme :

`g = polyval(A, t)` Attention à l'ordre des coefficients ! La commande `perms` pourra éventuellement vous être utile.

► **Question 64** Approximer le roulis et le tangage avec une droite quelconque.

3.2.2 Régression à l'ordre n

On se propose de construire maintenant le polynôme de degré n passant au mieux par les N points de mesure en roulis et en tangage. La nouvelle matrice de temps aura donc la forme :

$$T = \begin{pmatrix} t_0^n & t_1^n & t_2^n & \dots & t_N^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ t_0^2 & t_1^2 & t_2^2 & \dots & t_N^2 \\ t_0 & t_1 & t_2 & \dots & t_N \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

et pour la matrice A :

$$A = (a_n \dots a_2 \ a_1 \ a_0)$$

► **Question 65** Écrire la fonction `Regression` qui calcule les coefficients du polynôme de degré n pour des mesures prises aux temps t .

► **Question 66** Représenter pour le roulis et le tangage un graphique représentant les trois approximations précédentes (pour la troisième, on prendra $n = 5$).

3.3 Reconstitution de signal

Dans cette partie, nous allons aborder les notions importantes de discrétisation et d'interpolation, au travers d'une étude hautement scientifique : le langage de la baleine de Mongolie.

3.3.1 Le signal acoustique

De très grands chercheurs renommés et reconnus par leurs pairs³ des plus grandes universités mondiales se sont penchés sur le chant de la baleine de Mongolie lors de débats pendant une période électorale. Ces gens très intelligents ont modélisé ces chants puis en ont déduit un algorithme stochastique, décrit dans `Chant.m`, qui vous permet de simuler différents noms d'oiseaux échangés entre les cétacés.

3.3.2 Enregistrement du signal accoustique

Sur le terrain, le signal est enregistré électroniquement avec une période d'échantillonnage T_e . Le signal échantillonné est donc une mesure partielle du son réel. L'échelle de temps est la ms : tous les vecteurs temps sont indicés par un temps en ms . Ceci est illustré figure 3.

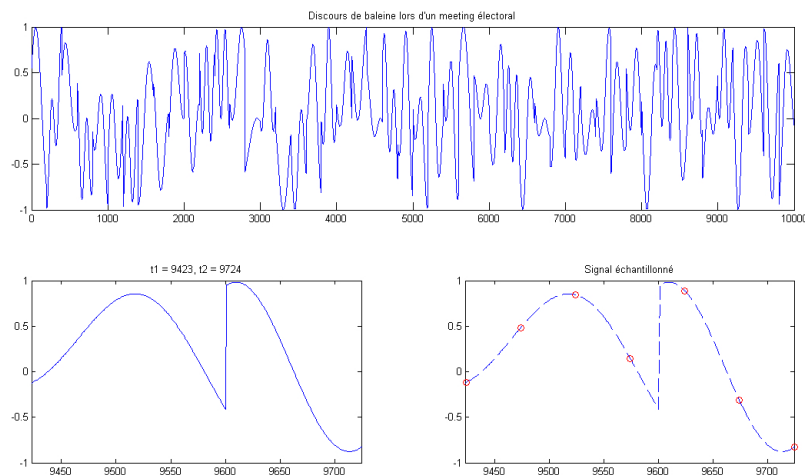


FIG. 3 – Signal initial, fenêtre temporelle et échantillonnage.

³accessoirement, mais pour certains d'entre eux seulement, par leur père aussi.

Ainsi est enregistrée, électroniquement, la valeur de $s(t)$ toutes les T_e ms. Nous allons simuler ceci, en ne prenant qu'une valeur de $s(t)$ sur 50 de manière régulière. Nous allons faire cette étude sur une fenêtre temporelle de taille $T = 300$ ms.

► **Question 67** En étudiant la fonction `Chant.m`, créer un petit programme (macro)

1. qui lance une simulation, avec un pas minimal de 1ms, 50 phases minimales et un index des fréquences maxi de 30 ;
2. qui extrait une fenêtre temporelle au hasard de taille $T = 300$,
3. qui extrait une valeur sur 50 du signal initial, simulant ainsi un échantillonnage du signal,
4. et qui affiche l'ensemble des données avec plein de titres et de légendes.

Deux niveaux de numérisation du signal sont mises en œuvre :

1. En enregistrant, pour chaque échantillon, la valeur du signal,
2. et en enregistrant, toujours pour chaque échantillon, la valeur du signal et de sa dérivée.

Nous allons voir ici deux méthodes de reconstitution du signal temporel à partir des échantillons enregistrés : la première méthode consiste à faire une interpolation entre les échantillons sur une fenêtre entière ; la seconde méthode consiste à utiliser une fenêtre glissante. Chaque méthode se décline selon deux approches : la première approche consiste à n'utiliser que la valeur des échantillons (premier ordre) ; la seconde approche consiste à utiliser la valeur des échantillons et leur dérivée (deuxième ordre).

3.3.3 Interpolation par fenêtre

* Premier ordre

Supposons, dans la fenêtre que vous venez de sélectionner, l'ensemble des N échantillons enregistrés. Pour reconstituer le signal avec la première méthode de numérisation, il suffit de trouver un polynôme de degré $n = N - 1$ passant par ces N points.

► **Question 68** Poser le système d'équation correspondant sous forme matricielle, puis proposer une méthode de résolution simple par pseudo-inverse.

► **Question 69** Implémenter une fonction permettant de répondre au problème. La mettre en œuvre sur la fenêtre T .

► **Question 70** Proposer un outil permettant de mesurer la différence entre le signal réel et le signal approximé (en %).

* Deuxième ordre

Ensuite, avec la seconde méthode de numérisation, il est possible de tenir compte de la dérivée de chaque échantillon. Le polynôme minimum pour obtenir un tel signal est de degré $n = 2 \times N - 1$.

► **Question 71** Poser le système d'équation correspondant sous forme matricielle, puis proposer une méthode de résolution simple par pseudo-inverse. On pourra mettre ce système sous la forme suivante :

$$(a_{2 \times N - 1} \ a_{2 \times N - 2} \ \dots \ a_2 \ a_1 \ a_0) \times \begin{pmatrix} X \\ \end{pmatrix} = \begin{pmatrix} Y \\ \dots \\ dY \end{pmatrix}$$

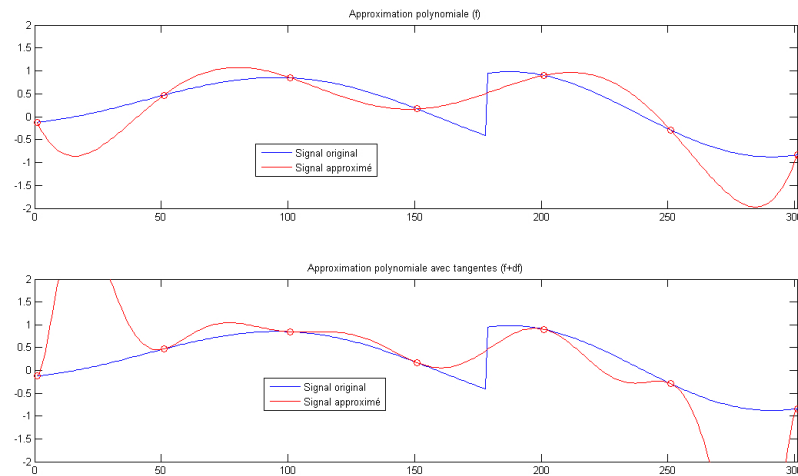


FIG. 4 – Interpolation.

► **Question 72** Implémenter une fonction permettant de répondre au problème. La mettre en œuvre sur la fenêtre T .

► **Question 73** Comparer le résultat avec précédemment. Commenter.

3.3.4 Fenêtre glissante

Les fonctions B-splines sont très utilisées dans tous les problèmes d'interpolation et d'approximation aussi bien dans le plan que dans l'espace. Elles sont présentes à l'heure actuelle dans tous les systèmes de Conception ou de Dessin Assisté par Ordinateur.

Supposons deux échantillons aux instants t et $t + T_e$ du signal : $s(t)$ et $s(t + T_e)$. Il est possible de relier ces deux points, en fonction de la stratégie de numérisation choisie, avec une fonction

- linéaire pour respecter les valeurs du signal en chaque extrémité,
- de degré 3 pour respecter les valeurs et les dérivées.

► **Question 74** Poser les équations permettant de relier linéairement deux échantillons consécutifs et proposer une méthode de résolution simple par pseudo-inverse.

► **Question 75** Poser les équations permettant de relier, en tenant compte de la dérivée de chacun d'entre eux, deux échantillons consécutifs et proposer une méthode de résolution simple par pseudo-inverse.

► **Question 76** Implémenter et mettre en œuvre ces méthodes sur la fenêtre T .

► **Question 77** Comparer le résultat avec précédemment.

► **Question 78** Comparer les différentes méthodes sur le signal complet. Conclure.

A Annexe A - Principales instructions MATLAB

Les tableaux suivants résument les principales instructions MATLAB.

A.1 Généralités

Nom	Description	Exemple(s)
help	décrit une instruction	help eig
who	liste les variables en mémoire	
whos	liste et décrit les variables en mémoire	
what	donne l'ensemble des fichier matlab dans le répertoire	
clear	efface une variable	clear M
disp	affiche une chaîne de caractères	disp('tralalalère.');
clear all	efface toutes les variables	clear all
clc	efface l'espace de commandes	
[Ctrl]+C	arrête une routine	
exit	arrête MATLAB	
quit	idem	
!	exécute une commande de l'OS	!ls -la sous Linux/UNIX !cmd sous Windows
%	permet de commenter le reste de la ligne	% Oh ! La belle % instruction!
;	n'affiche pas le résultat	x=3;
...	permet de continuer l'instruction... à la ligne	x=atan(pi ^ ... 3)+2;
date	affiche la date (codifiée)	now datestr(now)
home	revient au répertoire 'home' de MATLAB	

A.2 Valeurs spéciales

Nom	Description	Exemple(s)
ans	retourne la dernière valeur calculée	cos(ans)
pi	retourne π	cos(pi)
exp(1)	retourne e (exponentielle)	exp(pi) > pi ^ (exp(1))
i, j	désigne le nombre imaginaire ($\sqrt{-1}$)	1+i
inf	désigne l'infini (∞)	1/inf
NaN	désigne un nombre indéfini (Not a Number)	

A.3 Matrices particulières

Nom	Description	Exemple(s)
<code>zeros</code>	renvoie une matrice nulle	<code>zeros(2)</code>
<code>ones</code>	renvoie une matrice de 1	<code>ones(2,3)</code>
<code>eye</code>	retourne la matrice carrée identité	<code>eye(2)</code>
<code>compan</code>	retourne la matrice compagnon	<code>compan([1 0 -7 6])</code>
<code>hilb</code>	retourne la matrice de Hilbert	
<code>magic</code>	retourne une matrice correspondante au carré magique	
<code>pascal</code>	affiche la matrice correspondante au triangle de Pascal	<code>pascal(5)</code>
<code>vander</code>	Retourne la matrice de vandermonde	

A.4 Opérateurs matriciels

Nom	Description	Exemple(s)
<code>size</code>	taille d'une matrice	<code>size(M)</code> <code>[NbL, NbC] = size(M)</code> <code>size(M, 2)</code>
<code>length</code>	retourne la taille d'un vecteur	<code>length(V)</code>
<code>end</code>	renvoie le dernier élément d'un vecteur	<code>V(end)</code>
	équivalent à	<code>V(length(V))</code>
<code>+</code>	additionne deux matrices	<code>M1+M2</code>
<code>-</code>	soustrait deux matrices	<code>M1-M2</code>
<code>*</code>	multiplie deux matrices	<code>M1*M2</code>
<code>/</code>	divise A par B (B inversible)	<code>M1/M2</code>
<code>^</code>	met à la puissance	<code>M^2</code>
<code>.[op]</code>	exécute l'opération terme à terme	<code>M1.^3</code> <code>M1./M2</code>
<code>'</code>	conjugue et transpose	<code>M1'</code>
<code>.'</code>	transpose	<code>M1.'</code>
<code>>, >=, <, <=, ==</code>	compare terme à terme	<code>M3=M1<=M2</code>
<code>~</code>	Non logique	<code>M2=~M1</code>
<code>~</code>	Non-égal	<code>M2~=M1</code> <code>M3=(M1~=M2)</code>
<code>&</code>	ET logique	<code>M1&M2</code>
<code> </code>	OU logique	<code>M1 M2</code>
<code>xor</code>	OU exclusif	<code>C=xor(A,B)</code>
<code>find</code>	donne les indices des termes non nuls	<code>find(M1&M2)</code>
<code>rot90</code>	« tourne » une matrice	<code>rot90(eye(2));</code>
<code>eig(A)</code>	retourne les valeurs propres et les vecteurs propres d'une matrice	<code>[valp, vecp]=eig(A);</code>

A.5 Programmation

Nom	Description	Exemple(s)
if	si...	if (det(M)~=0)
then	alors...	N=inv(M) ;
else	sinon...	else disp('Matrice M non inversible') ;
elseif	sinon si...	elseif (det(M)>1)
for	pour...	for i=1 :n
while	tant que...	while(i<=n)
break	sort de la boucle	if (det(M) ==0) break ;
pause	attend que l'opérateur appuie sur une touche pour continuer	
error	permet d'afficher un message d'erreur	error('La matrice n'est pas inversible') ;

A.6 Figures

Nom	Description	Exemple(s)
plot(x)	affiche des composantes du vecteur x	plot(rand(100,1)) ;
plot(x,y)	affiche les points (x_i, y_i)	
subplot	permet d'afficher plusieurs figures dans une fenêtre	
loglog	affiche en échelle logarithmique	
semilogx	affiche en échelle log. pour l'axe x	
semilogy	affiche en échelle log. pour l'axe y	
polar	affiche en coordonnées polaires	
figure	ouvre une nouvelle fenêtre	figure ;
figure(n)	sélectionne la figure n^{e}	figure(2) ;
title	affiche un titre	title('Courbes de Bézier') ;
legend	affiche la legend	legend('f(x)', 'g(x)') ;
xlabel	étiquette l'axe x	xlabel('temps') ; xlabel('temps {\it t}') ;
ylabel	étiquette l'axe y	ylabel('Energie') ; ylabel('{\it e} ^{-\alpha t}') ;
axis	permet de définir l'échelle des axes	
grid	dessine une grille	
hold	maintient des courbes sur une figure	

A.7 Fonctions trigonométriques

Nom	Description	Exemple(s)
sin	sinus	sin(pi/2)
asin	arcsin	asin(1)
sinh	sinus hyperbolique	
asinh	arcsinus hyperbolique	
cos	cosinus	cos(pi)
acos	arccosinus	acos(-1)
cosh	cosinus hyperbolique	
acosh	arccosinus hyperbolique	
tan	tangente	tan(pi/4)
atan	arctangente	atan(1)
atan2(y, x)	arctangente sur $[0..2\pi]$	atan2(sqrt(3)/2, -1/2)
tanh	tangente hyperbolique	
atanh	arctangente hyperbolique	

A.8 Polynômes

Nom	Description	Exemple(s)
$[a_n \dots a_1 a_0]$	Polynôme $a_n x^n + \dots + a_1 x + a_0$	
roots	calcule les racines d'un polynôme	roots([1 -1 -1])
polyval	évalue un polynôme	polyval(P, x);
polyder	dérive un polynôme	polyder(P);
polyder	évalue la dérivée un polynôme	polyder(P, x);
poly	renvoie le polynôme caractéristique	

Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

A.9 Instructions diverses

Nom	Description	Exemple(s)
rand	renvoie un nombre avec une probabilité uniforme	rand rand(3,5)
randn	renvoie un nombre au hasard avec une probabilité de loi normale	x=randn(2,5); y=moy+sqrt(var)*randn;
mean	calcule la moyenne le long des colonnes	mean(rand(100,1))
var	calcule la variance le long des colonnes	var(rand(100,1))
std	calcule l'écart-type (<i>deviation standart</i>) le long des colonnes	std(rand(100,1))
abs	calcule le module	abs(-6) abs(1+i)
median	retourne la valeur médiane	
min	retourne le minimum par colonne	
max	retourne le maximum par colonne	
sum	somme les éléments d'une colonne	
angle	calcule la phase	angle(1+i)
real	renvoie la partie réelle	
imag	renvoie la partie imaginaire	
conj	renvoie le conjugué	
round	renvoie l'entier le plus proche	
floor	renvoie l'entier inférieur	
ceil	renvoie l'entier supérieur	
sign	renvoie la partie entière du nombre	
exp	renvoie l'exponentielle	exp(3)
log	renvoie le logarithme	
log10	renvoie le logarithme décimal	
num2str	Convertit un nombre en chaîne de caractères	disp(['Nhasard = ' ... num2str(rand)]);
quad	permet de calculer une intégrale	

Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

Index

min, max, 6
Tic et Toc, 12

Affichage, 5
 disp, 5
 num2str, 5
 whos, 5
 who, 5

Aléatoire, 11
 randn, 11
 randperm, 11
 rand, 11

Débuguer un programme, 17

Entrées - Sorties, 12
 fclose, 13
 fopen, 12
 fprintf, 13
 load, 12
 save, 12
 scanf, 13

Figures, 9
 hold, 9
 legend, 9
 meshgrid, 9
 plot, 9
 subplot, 9
 surf, 9
 title, 9

Fonctions, 15
 function, 15
 variables globales, 15
 variables locales, 15

Initialisation, 10
 eye, 11
 ones, 11
 zeros, 10
 initialisation automatique, 10

Instructions, 14, 24

Macro, 5
 clc, 5
 clear all, 5
 close all, 5

Matrices, 7
 :, 7
 ceil, 8
 eig, 8
 fix, 8
 floor, 8
 round, 8
 size, 7
 sum, 8
 Opération terme à terme., 7
 Transposée :, 7

Opérateurs logiques, 12
 ET &, 12
 NON ~, 12
 OU , 12

Paramètres dynamiques, 16

Polynômes, 12
 polyder, 12
 polyval, 12

Programmation
 boucle for, 14
 boucle while, 14
 test if, 14

Pseudo-inverse, 29

Structures de données, 16

Vecteurs, 6
 ', 6
 :, 6
 end, 6
 length, 6
 sum, 8

B Index des fonctions les plus courantes

Ici sont regroupées les fonctions MATLAB les plus couramment utilisées dans votre école favorite.

B.1 Généralités

`clear, close, clc, save, load, who, whos, tic, toc`

B.2 Affichage

`disp, figure, display, legend, title`

B.3 Vecteurs, Matrices

`size, length, zeros, ones, eye, hist, sum, eig`

B.4 Aléatoire

`randperm, rand, randn`

B.5 Programmation

`if, then, else, end, for, while`

ISAE

Institut Supérieur de l'Aéronautique et de l'Espace

SUPAERO

Index

min, max, 6
Tic et Toc, 12

Affichage, 5
 disp, 5
 num2str, 5
 whos, 5
 who, 5

Aléatoire, 11
 randn, 11
 randperm, 11
 rand, 11

Débuguer un programme, 17

Entrées - Sorties, 12
 fclose, 13
 fopen, 12
 fprintf, 13
 load, 12
 save, 12
 scanf, 13

Figures, 9
 hold, 9
 legend, 9
 meshgrid, 9
 plot, 9
 subplot, 9
 surf, 9
 title, 9

Fonctions, 15
 function, 15
 variables globales, 15
 variables locales, 15

Initialisation, 10
 eye, 11
 ones, 11
 zeros, 10
 initialisation automatique, 10

Instructions, 14, 24

Macro, 5
 clc, 5
 clear all, 5
 close all, 5

Matrices, 7
 :, 7
 ceil, 8
 eig, 8
 fix, 8
 floor, 8
 round, 8
 size, 7
 sum, 8
 Opération terme à terme., 7
 Transposée :, 7

Opérateurs logiques, 12
 ET &, 12
 NON ~, 12
 OU , 12

Paramètres dynamiques, 16

Polynômes, 12
 polyder, 12
 polyval, 12

Programmation
 boucle for, 14
 boucle while, 14
 test if, 14

Pseudo-inverse, 29

Structures de données, 16

Vecteurs, 6
 ', 6
 :, 6
 end, 6
 length, 6
 sum, 8