UPS, M1 EEA, UE Signaux et systèmes

TP1 : Numérisation des signaux

- Vous devez avoir lu entièrement le sujet de TP avant la séance.
- Les questions dont les numéros sont suivis du signe ‘ † ’ sont des questions théoriques ou algorithmiques, qui doivent être résolues avant la séance de TP.
- La préparation de TP, et en particulier les réponses aux questions théoriques seront vérifiées et notées en début de séance.
- A la fin de séance, il faut rendre un compte-rendu qui contient les réponses aux questions posées dans le sujet, les explications et les justifications des résultats des manipulations, et le listing des programmes.
- Le TP est noté à partir de la préparation, du travail fait pendant la séance, et du compte-rendu.

But de la manipulation

L’objectif de cette manipulation est d’illustrer les effets de la numérisation des signaux (échantillonnage et quantification).
Avant d’arriver en séance, vous devez impérativement avoir assimilé les notions théoriques concernant l’échantillonnage et la quantification et en particulier le théorème de Shannon.

I. Echantillonnage des signaux audio

Un grand nombre de signaux varient continûment au cours du temps et leur stockage sur un support numérique nécessite de ne prendre en compte que des valeurs prises par le signal à certains instants : il s’agit de l’échantillonnage. On ne s’intéressera ici qu’au cas de l’échantillonnage régulier, c’est-à-dire que les instants où l’on mesure la valeur du signal $x(t)$ sont régulièrement espacés dans le temps : $x[n] = x(nTe)$, où T_e est la période d’échantillonnage.

1.† Rappeler l’effet en fréquence de l’échantillonnage temporel d’un signal et la condition de Shannon pour éviter le repliement de spectre et pouvoir reconstruire le signal original.

2.† Pour un signal $x[n]$ échantillonné à la fréquence F_e, quelle est la fréquence d’échantillonnage du signal sous-échantillonné d’un facteur S : $y[n] = x[S \cdot n]$? Quel est l’effet en fréquence d’un tel sous-échantillonnage ?

On va illustrer les effets de l’échantillonnage sur des signaux sonores. De tels signaux sont disponibles dans des fichiers au format *wave* (*.wav) et peuvent être chargés dans Matlab à l’aide de la fonction Matlab wavread. La fonction PlaySignal permet d’écouter un signal sonore contenu dans un vecteur, en effectuant éventuellement un sous-échantillonnage.

3. Charger le signal contenu dans le fichier signal0.wav. Quelle est la fréquence d’échantillonnage de ce signal et sur combien de bits a-t-il été codé ? Ecouter le signal.

4. Ecouter le signal sous-échantillonné d’un facteur S pour différentes valeurs de S. Décrire brièvement de façon qualitative le signal restitué suivant les valeurs prises par S. A partir de quelle valeur de S entend-on que le signal a subi des modifications ?

5. Tracer la représentation fréquentielle du signal en utilisant la fonction tfsc2. D'après le théorème de Shannon, à quelle fréquence ce signal peut-il être échantillonné sans perte d'information ? Cela vous paraît-il cohérent avec vos réponses à la question précédente ?

II. Echantillonnage et reconstruction d'un signal simple

Après avoir entendu les effets de la numérisation sur un signal sonore, on va travailler sur un signal d'école afin d'illustrer l'échantillonnage, la reconstruction, et l'application pratique du théorème de Shannon. Le signal est présent dans le fichier SignalReconst.mat (*x* avec pour temps associé *t*) et peut être récupéré à l'aide de la commande Matlab load SignalReconst.mat. Ce signal a été échantillonné à une fréquence très élevée (on travaillera par la suite en fréquence normalisée, c'est-à-dire que l'on considère *Fe* = 1) et jouera le rôle de signal analogique.

1. Tracer les représentations temporelle et fréquentielle du signal *x*.

2. Simuler l'échantillonnage d'un signal continu en ne conservant qu'un échantillon sur *S* de *x* (sous-échantillonnage d'un facteur *S*) et en mettant les autres échantillons à zéro. Tracer les représentations temporelles et fréquentielle des signaux résultants. Commenter les résultats suivant les valeurs de *S*.

3. Quelle est la fréquence minimale avec laquelle on peut échantillonner le signal ? A quel facteur de sous-échantillonnage correspond-elle ? Qu'observe-t-on si l'on prend un facteur de sous-échantillonnage trop grand ? Que doit-on faire en pratique, lorsque la fréquence d'échantillonnage est fixée, pour éviter un tel phénomène ?

Le théorème de Shannon nous donne des conditions sur la fréquence d'échantillonnage afin de ne pas perdre d'information lors de l'échantillonnage et assurant une reconstruction exacte du signal continu. On va donc maintenant essayer de reconstruire le signal que l'on a sous-échantillonné.

4. Pour un sous-échantillonnage d'un facteur *S*, quelle est la réponse en fréquence du filtre idéal permettant de reconstruire le signal continu ?

5. Construire une telle réponse en fréquence et effectuer le filtrage dans le domaine fréquentiel. La fonction tfsc2_inv permet ensuite de calculer l'inverse de la transformée de Fourier pour retrouver le signal temporel reconstruit. Comparer le signal reconstruit au signal original. Commenter pour différentes valeurs de *S*.

III. Quantification

En pratique, un signal peut avoir des valeurs continues et son stockage sur un support numérique ne permet de prendre en compte qu'un nombre fini de valeurs discrètes. La quantification permet alors de convertir un signal à valeurs continues en un signal à valeurs discrètes.

1.† En utilisant la fonction round de Matlab, écrire une fonction x_q =*quantif*(*x*, *N*) permettant de simuler la quantification d'un signal *x* sur *N* bits. Avant de quantifier, cette fonction doit normaliser le signal *x* en utilisant deux paramètres *a* et *b* tels que y=(*x*-*b*)/*a* soit

dans [0, 1]. Après la quantification, elle doit dénormaliser le signal quantifié en utilisant la formule $x_q = a y_q + b$ pour revenir à l'échelle de départ. Tester votre fonction sur le signal x=[-2 : 0.01 : 1]. Tracer x et x_q sur une même figure pour différentes valeurs de N.

2. Pour différentes valeurs de N, écouter le résultat de la quantification du signal contenu dans le fichier SignalQuantif.mat (variables x – signal – et F_e – fréquence d'échantillonnage – que l'on peut récupérer avec la commande load SignalQuantif.mat). Décrire de façon qualitative la qualité de restitution.

3. Pour différentes valeurs de N, calculer le rapport signal sur bruit de quantification en dB, défini par $RSB = 10 \log_{10} \frac{P_S}{P_B}$, où P_S et P_B représentent respectivement la puissance du signal et la puissance de l'erreur de quantification. On notera que la puissance d'un signal x peut être calculée en Matlab avec mean(x.^2). Tracer RSB en fonction de N. Conclusion ?

4. Pour différentes valeurs de N, tracer l'histogramme de l'erreur de quantification sur 100 niveaux. Commenter le résultat.

Annexe A : Quelques fonctions standard de Matlab utilisées dans ce TP

wavread

y = wavread(filename) loads a WAVE file specified by the string *filename*, returning the sampled data in *y*.
[*y*, *Fs*] = wavread(*filename*) returns the sample rate (*Fs*) in Hertz used to encode the data in the file.
[*y*, *Fs*, *nbits*] = wavread(*filename*) returns the number of bits per sample (*nbits*).

load

S = load(filename) loads the variables from a MAT-file into a structure array, or data from an ASCII file into a double-precision array.

round

Y = round(X) rounds the elements of X to the nearest integers.

abs

abs(X) returns an array Y such that each element of Y is the absolute value of the corresponding element of X.
If X is complex, abs(X) returns the complex modulus (magnitude).

mean

M = mean(A) returns the mean values of the elements along different dimensions of an array. If A is a vector, mean(A) returns the mean value of A.

hist

A histogram shows the distribution of data values.

n = hist(Y) bins the elements in vector Y into 10 equally spaced containers and returns the number of elements in each container as a row vector. If Y is an m-by-p matrix, hist treats the columns of Y as vectors and returns a 10-by-p matrix n. Each column of n contains the results for the corresponding column of Y. No elements of Y can be complex or of type integer.

n = hist(Y,nbins) where nbins is a scalar, uses nbins number of bins.

hist(...) without output arguments produces a histogram plot of the output described above.

Annexe B : Fonctions Matlab qui vous seront fournies en séance de TP

PlaySignal

PlaySignal(x,Fs) envoie le signal contenu dans le vecteur x à la carte son avec une fréquence d'échantillonnage Fs.

PlaySignal(x,Fs,S) envoie le signal sous-échantillonné d'un facteur S.

tfsc2

[Xhat,f] = tfsc2(x,fe) calcule la transformée de Fourier discrète du signal x échantillonné avec une fréquence d'échantillonnage fe, et renvoie le résultat (Xhat) ainsi que le vecteur des fréquences (f).

tfsc2_inv

x = tfsc2_inv(Xhat) calcule la transformée de Fourier inverse.

TP2 : Détection de cibles par corrélation

Les questions dont les numéros sont suivis du signe † sont des questions théoriques, qui doivent impérativement être résolues avant la séance de TP.

1 Présentation du TP

L'objectif de ce TP est d'illustrer le principe de détection de cibles par corrélation. Cette technique, utilisée dans les systèmes radar et sonar, en échographie biomédicale, en sismique et en contrôle non-destructif, peut être résumée de la manière suivante : un émetteur (antenne ou sonde) envoie via des ondes électromagnétiques ou ultrasonores un motif de signal. Ce signal est réfléchi par une ou plusieurs cibles et l'émetteur reçoit en retour ce signal réfléchi, qui est généralement noyé dans le bruit. Connaissant le motif envoyé, on peut, par corrélation, détecter la présence de ce motif dans le signal reçu et estimer la distance des cibles [1].

Le traitement consistera donc en l'estimation de la corrélation entre le signal reçu et le motif émis suivi d'une étape de détection de l'instant auquel le signal est revenu.

2 Corrélation

D'un point de vue théorique, l'autocorrélation d'un signal aléatoire stationnaire $x[n]$ est définie par $R_x[m] = E\{x[n+m]x[n]\}$, et l'intercorrélation entre deux signaux $y[n]$ et $x[n]$ conjointement stationnaires est définie par

$$R_{yx}[m] = E\{y[n+m]x[n]\}$$

.

1† Rappeler l'expression de la fonction d'autocorrélation d'un bruit blanc.

2† Calculer la fonction d'autocorrélation d'une sinusoïde à phase aléatoire.

Le calcul théorique des fonctions d'autocorrélation et d'intercorrélation nécessite la connaissance des densités de probabilité des signaux. En pratique, on ne dispose souvent que d'une seule réalisation de longueur N et l'on doit donc *estimer* les fonctions d'autocorrélation et

1. On connaît la vitesse de propagation des ondes et le signal a parcouru deux fois la distance émetteur–cible.

d'intercorrélation à partir de cette réalisation [2]. On connaît deux estimateurs empiriques de l'intercorrélation, le premier est théoriquement non-biaisé :

$$R_{yx}^{nb}[m] = \frac{1}{N-m} \sum_{n=1}^{N-m} y[n+m]x[n],$$

le second est théoriquement biaisé mais a une variance plus faible :

$$R_{yx}^{b}[m] = \frac{1}{N} \sum_{n=1}^{N-m} y[n+m]x[n].$$

Les estimateurs de la fonction d'autocorrélation peuvent être obtenus avec les mêmes formules en remplaçant y par x.

La fonction Matlab *xcorr* permet d'obtenir ces estimations de la fonction d'intercorrélation pour l'indice m variant de $-N+1$ à $N-1$:
- RyxNB=xcorr(y,x,'unbiased') : Estimateur non-biaisé,
- RyxB=xcorr(y,x,'biased') : Estimateur biaisé.

On va maintenant utiliser ces deux estimateurs pour estimer l'autocorrélation de différents motifs, de longueur fixée $N = 50$:
– un bruit blanc (par exemple `x = randn(1,50);`)
– une sinusoïde (par exemple `x = sin(2*pi*(1:50)*.1);`)
– une sinusoïde modulée par une gaussienne (par exemple : `x = sin(2*pi*(1:50)*.1).*exp(-(-24:25).^2/100);`)
– un chirp (sinus dont la fréquence varie linéairement dans le temps) soit par exemple : `x = sin(2*pi*(1:50).^2/100);`.

3. Pour chacun des motifs ci-dessus tracer les fonctions d'autocorrélation (avec $x = y$ dans la fonction Matlab *xcorr*) estimées avec les estimateurs biaisé et non-biaisé. Pour les deux premiers motifs, comparer les résultats obtenus avec les valeurs théoriques.

3 Etude de corrélation pour la détection de cible

Soient $x[n]$ le signal envoyé par l'émetteur et $y[n] = ax[n - n_0]$ le signal réfléchi par la cible et reçu par le récepteur (supposé sans bruit pour l'instant). Dans cette formule, a et n_0 représentent respectivement l'atténuation et le retard du signal reçu par rapport au signal émis.

1† Calculer l'intercorrélation entre le signal reçu et le signal envoyé $R_{yx}[m] = E\{y[n+m]x[n]\}$, et l'exprimer en fonction de l'autocorrélation de x, $R_x[m]$.

2† Que devient cette relation quand le signal envoyé est un bruit blanc ? Proposer alors une méthode pour estimer la distance de la cible à partir de cette relation.

2. Cela suppose également que les signaux sont ergodiques.

3† Que devient cette relation quand le signal envoyé est une sinusoïde ? Peut-on encore estimer la distance de la cible ?

4. Parmi les 4 motifs présentés ci-dessus, quels sont ceux qui vous semblent les plus appropriés pour l'utilisation en détection de cibles ? Justifier votre réponse.
5. Parmi les estimateurs biaisé et non biaisé, quel est celui qui vous semble le plus approprié pour l'utilisation en détection de cibles ? Justifier votre réponse.

4 Détection des cibles dans le cas non-bruité

Les motifs envoyés pourront être l'un quelconque des signaux précédents. On considère que le radar envoie ce motif et reçoit en retour plusieurs répliques de ce motif décalées dans le temps en fonction de la distance des différentes cibles.

1. Choisir l'un des 4 motifs ci-dessus, puis simuler un tel signal reçu pour un faible nombre de cibles (prendre par exemple un signal de longueur 500, avec 5 répliques du motif positionnées en 15, 120, 136, 379 et 400). On fera attention aux superpositions éventuelles des répliques du motif renvoyé.
2. Calculer et afficher l'intercorrélation (pour les indices positifs uniquement) entre le signal reçu et le motif envoyé (on prend soin d'ajouter des zéros afin de leur donner la même longueur). Qu'observe t-on ?
3. Recommencer pour les différents motifs utilisés. Quel motif se prête le mieux à la détection ?
4. Retrouver la position des cibles de façon automatique en effectuant un seuillage (ici on se contentera de retrouver les positions du motif dans le signal reçu).

5 Détection des cibles dans le cas bruité

En pratique, le signal reçu est bruité et peut être écrit comme $y[n] = ax[n - n_0] + b[n]$, où b est un bruit centré et non-corrélé avec le signal utile x.

1† Calculer l'intercorrélation entre le signal reçu et le signal envoyé $R_{yx}[m] = E\{y[n+m]x[n]\}$. La puissance de bruit doit-elle, théoriquement, modifier le résultat ?

2. Répéter l'expérience de Section 4 pour un niveau de bruit plus ou moins élevé (bruit d'écart-type 0.25, 0.5 et 1). Conclusion ?

6 Stationnarité

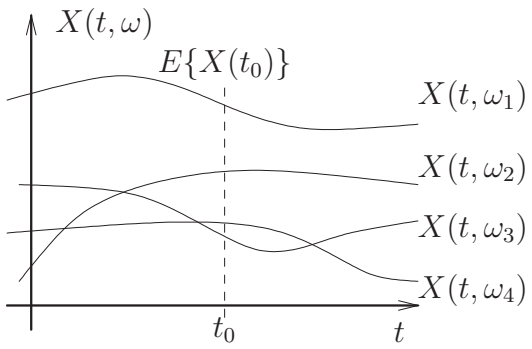
Dans ce TP, on a supposé que les signaux traités étaient stationnaires. Dans la suite, on s'intéresse aux propriétés de stationnarité de la moyenne et de la variance d'un signal aléatoire.

4† Rappeler la définition d'un signal stationnaire au sens large.

5† Est-ce qu'on peut dire que la variance d'un signal stationnaire au sens large est constante ? Pourquoi ?

Pour vérifier la stationnarité de la moyenne ou de la variance d'un signal aléatoire à partir de K de ses réalisations, on peut estimer, à différents instants t, la moyenne $m_X(t) = E\{X(t)\}$ grâce à une simple moyenne empirique sur les réalisations :

$$\widetilde{m}_X(t) = \frac{1}{K}\sum_{k=1}^{K} X(t,\omega_k),$$



et la variance $\sigma_X^2(t) = E\{(X(t) - m_X(t))^2\}$ par une variance empirique :

$$\widetilde{\sigma}_X^2(t) = \frac{1}{K}\sum_{k=1}^{K}(X(t,\omega_k) - \widetilde{m}_X(t))^2,$$

et vérifier qu'elles ne varient pas trop avec le temps (des variations résiduelles étant dues au fait que l'on observe des moyennes et des variances estimées et non des moyennes et variances théoriques).

Nous allons maintenant illustrer cela sur une sinusoïde à phase aléatoire. Pour des raisons évidentes, nous travaillerons sur des signaux échantillonnés (à la fréquence $F_e = 1$), mais cela ne modifie en rien les conclusions à tirer sur les signaux aléatoires.

Soit une sinusoïde $X[n] = \cos(2\pi f_0 n + \phi)$ de fréquence $f_0 = 0,1$ Hz et de phase à l'origine ϕ aléatoire, uniformément distribuée sur l'intervalle $[0, 2\pi[$.

6. Calculer la valeur moyenne $m_X[n]$ et la variance $\sigma_X^2[n]$ théoriques d'un tel signal. Ce signal est-il stationnaire pour sa moyenne et sa variance ?
7. Avec Matlab, tirer un certain nombre de réalisations d'un tel signal. A l'aide des fonctions *mean* et *var* de Matlab, estimer la moyenne et la variance empiriques à partir de ces réalisations et les superposer à la moyenne et la variance théoriques. Que peut-on conclure en pratique sur la stationnarité de la moyenne et de la variance de ce signal ?

Eviter au maximum les boucles, *e.g.*,

```
>> for k=1 : N, x(k)=k; end
```

est équivalent à

```
>> x=1 : N;
```

Autre exemple :

```
>> r=1 : 10; A=[]; % init. de A à vide
>> for k=1 : 5, A=[A; r]; end
```

peut être écrit

```
>> r=1 : 10; A=ones(5,1)*r;
```

et même

```
>> r=1 : 10; A=r([1 1 1 1 1], :);
```

ou

```
>> r=1 : 10; A= r(ones(1,5), :);
```

Attention à ne pas utiliser `i` ou `j` comme indice de boucle car `i` et `j` prédéfinis à $\sqrt{-1}$.

6 Signal et image

6.1 Matrices particulières

Matrices de Tœplitz :

```
>> c=[ 1 2 3 ]; l=[ 1.5 2.5 3.5 4.5 5.5 ];
>> toeplitz(c,l)
   ans = 1.0 2.5 3.5 4.5 5.5
         2.0 1.0 2.5 3.5 4.5
         3.0 2.0 1.0 2.5 3.5
```

Matrices de Hankel, Hadamard, Hilbert, *etc...*

6.2 Filtrage et convolution

Fonction `filter` (2D : `filter2`) : filtrage rationnel par un filtre de coefficients dans `A` et `B` :

```
>> Y = filter(B,A,X);
```

Convolution `conv` (2D : `conv2`) :

```
>> Z = conv(X,Y);
```

La fonction `roots` : racines (zéros, pôles). Calcul des racines d'un polynôme donné par ses coefficients (ici $P(z) = z^3 - 6z^2 - 72z - 27$) :

```
>> p = [1 -6 -72 -27]; r=roots(p)
```

La fonction `polyval` calcule la valeur d'un polynôme en certains points :

```
>> polyval(p,[ 1, exp(j*pi/4) ])
   ans = 1.0e+002 *
   -1.0400 -0.7862-0.5620i
```

La fonction `poly` inverse la fonction `roots`.

```
>> p=poly(r)
   p = 1.000 -6.000 -72.000 -27.000
```

6.3 Transformation de Fourier

`fft`, `ifft`. Calcul de la FFT du signal `x`, sur 1024 points régulièrement distribués dans $[0, 1]$:

```
>> TF=fft(x,1024);
```

Pour $[-0.5, +0.5]$ utiliser `fftshift`. En dimension 2 : `fft2` et `ifft2`.

6.4 Un peu d'aléa

`rand` : génération pseudo-alétoire uniforme sur $[0, 1]$
`randn` : génération gaussienne $\mathcal{N}(0, 1)$
`mean` : moyenne empirique
`std` : écart-type empirique
`xcorr` : corrélation empirique
`cov` : covariance empirique

6.5 Fonctions mathématiques spéciales

`besselj`, `bessely`, `gamma`, `erf`, `erfinv`, *etc...*

6.6 Exemple de fichier de commande

Problème : On possède N mesures (x_n, y_n) bruitées censées correspondre à des points d'une droite. On va estimer la pente a et la valeur à l'origine b de cette droite par moindres carrés, c'est-à-dire minimisant :

$$J(a,b) = \sum_{k=1}^{N} (y_k - a x_k - b)^2$$

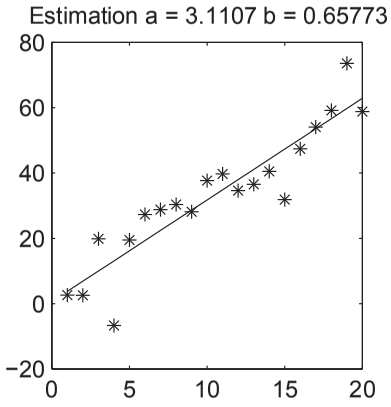
Solution : En annulant les dérivées partielles par rapport à a et b on trouve (un seul minimum) :

$$a = \frac{\sum_{k=1}^N x_k y_k - \frac{1}{N}(\sum_{k=1}^N x_k)(\sum_{k=1}^N y_k)}{\sum_{k=1}^N x_k^2 - \frac{1}{N}(\sum_{k=1}^N x_k)^2}$$

$$b = \frac{1}{N}(\sum_{k=1}^{N} y_k - a \sum_{k=1}^{N} x_k)$$

Script Matlab :

```
% Simulation
  a = pi; b = 1/3;
% Tirage d'un ensemble de mesures
  N = 20; x = 1 :N;
  bruit = 10*randn(1,N); y = a*x+b+bruit;
% Estimation
  aest = (sum(x.*y) - sum(x)*sum(y)/N)/...
       (sum(x.^2) - sum(x)^2/N);
  best = (sum(y) - aest*sum(x))/N;
% Affichage
  plot(x,y,'*',x,aest*x+best)
  title(['Estimation a=',num2str(aest),...
       ' b=',num2str(best)])
```



— Matlab en bref —

Table des matières

1 Généralités 1
1.1 Introduction . . . 1
1.2 Démarrer . . . 1
1.3 Aide en ligne . . . 1
1.4 Langage interprété . . . 1
1.5 Variables . . . 1
1.6 Variables complexes . . . 1
1.7 Vecteurs, matrices et manipulations . . . 1
1.8 L'opérateur d'énumération « : » . . . 2
1.9 Chaînes de caractères . . . 2

2 Opérations matricielles 2

3 Affichages 2
3.1 Affichage alpha-numérique . . . 2
3.2 Graphiques 1D : `plot` . . . 2
3.3 Graphiques 2D : `mesh` . . . 3
3.4 Affichage de plusieurs courbes . . . 3

4 Fichiers, scripts, fonctions 3
4.1 Répertoire de travail . . . 3
4.2 Sauvegarde et chargement . . . 3
4.3 Scripts . . . 3
4.4 Fonctions . . . 3

5 Boucles et contrôles 3
5.1 Test « `if` » : . . . 3
5.2 Boucle « `for` » : . . . 3

6 Signal et image 4
6.1 Matrices particulières . . . 4
6.2 Filtrage et convolution . . . 4
6.3 Transformation de Fourier . . . 4
6.4 Un peu d'aléa . . . 4
6.5 Fonctions mathématiques spéciales . . . 4
6.6 Exemple de fichier de commande . . . 4

1 Généralités

1.1 Introduction

Matlab pour « Matrix Laboratory » : **tout est matrice**. Environnement de calcul matriciel, adapté à l'automatique et au traitement du signal et de l'image : facilité d'emploi, possibilités graphiques, boites à outils : signal processing, image processing, optimization, *etc...*

1.2 Démarrer

Cliquez sur l'icône Matlab ou tapez `matlab` dans un shell. Tapez les commandes Matlab ou le nom d'un fichier de commandes à la suite des chevrons `>>` .

1.3 Aide en ligne

Commande `help` : `help NomFct`. Également `lookfor`. Avant d'écrire une fonction, assurez-vous qu'une fonction similaire n'existe pas déjà. De même, avant d'utiliser une fonction, vérifiez qu'elle réalise bien l'opération souhaitée. Vérifiez également sa syntaxe. En cas d'erreur, lisez le message d'erreur que Matlab a la gentillesse de vous indiquer.

1.4 Langage interprété

Pas de compilation, ni de déclaration de variable, ni de réservation mémoire. Résultat affiché et stocké dans la variable `ans`.

```
>> 2+2
   ans = 4
>> 2*sin(pi/4)
   ans = 1.4142
```

Fonctions usuelles : `sin`, `cos`, `exp`, `log`, *etc...*

1.5 Variables

Les noms de variable commencent par une lettre. Attention, Matlab fait la différence entre `X` et `x`.

```
>> x = pi/4
   x = 0.7854
```

Point virgule « ; » évite l'affichage du résultat. Plusieurs commandes par ligne : séparées par « ; » ou « , » :

```
>> x = pi/2; y = sin(x);
```

1.6 Variables complexes

Matlab gère réels et complexes. `i` et `j` initialisés à $\sqrt{-1}$.

```
>> z = 3 + 2*i
   z = 3.0000 + 2.0000i
```

Fonctions prédéfinies : `real`, `imag`, `abs`, `angle`, *etc...*

```
>> r = abs(z);
>> theta = angle(z);
>> y = r*exp(i*theta);
```

1.7 Vecteurs, matrices et manipulations

Matrice sous forme d'un tableau :

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

Expressions à évaluer possible :

```
>> x = [-1.3 sqrt(3) (1+2+3)*4/5]
   x = -1.3000 1.7321 4.8000
```

Éléments référencés par leurs indices :

```
>> x(2)
   ans = 1.7321
>> x(5) = abs(x(1))
   x = -1.300 1.732 4.800 0.000 1.300
```

Ajout de lignes à une matrice :

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

Ajout de colonnes : « , » à la place de « ; »

Fonctions `zeros`, `ones`, et `eye` :

```
>> eye(2) % eye comme I : Identity
   ans = 1 0
         0 1
>> x = ones(1,5)
   ans = 1 1 1 1 1
```

Taille d'une matrice : `size`, `length` :

```
>> size(x)
   ans = 1 5
```

Transposée, conjuguée, retournement, rotation :

```
>> A' % Transposée conjuguée de A
>> A.' % Transposée de A
>> flipud(A) % Up - Down
>> fliplr(A) % Left - Right
>> rot90(A) % Rotation de 90°
```

1.8 L'opérateur d'énumération « : »

Pas à pas, incrément :

```
>> x = 0.5 : 0.1 : 0.85
   x = 0.5000 0.6000 0.7000 0.8000
```

Incrémentation par défaut : 1 (voir aussi `linspace`) :

```
>> x = 1 : 5
   x = 1 2 3 4 5
```

Sélection d'éléments :

```
>> A(1,3) ; A(1,1 : 3) ; A( : ,3) ;
```

Si on a :

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

alors, extraction lignes 1 à 2 et toutes les colonnes :

```
>> A(1 : 2, : )
   ans = 1 2 3
         4 5 6
```

1.9 Chaînes de caractères

Variables contenant des chaînes de caractères :

```
>> mess = 'Bienvenue sur Matlab' ;
```

Manipulations de même type que pour les vecteurs :

```
>> mess = [mess ' v 5']
   mess = Bienvenue sur Matlab v 5
```

Conversion de nombres en chaînes de caractères : `num2str`, `int2str`, `sprintf` :

```
>> mess = ['pi vaut ', num2str(pi)]
   mess = pi vaut 3.142
```

Évaluation d'une chaîne : `eval` et `feval` :

```
>> nomvar = 'x' ;
>> eval([nomvar '1 = sqrt(-1)'])
   x1 = 0.0000 + 1.0000i
>> NomFct = 'sin' ;
>> feval(NomFct,pi)
   ans = 1.2246e-16 % sin(pi) 0 !
```

2 Opérations matricielles

Opérations usuelles définies de façon naturelle :

```
>> 2*A
>> A*B % Produit matriciel
>> A^p
>> inv(A)
```

Résolution de systèmes linéaires :

```
>> X = A\B % solution de A*X = B
>> X = B/A % solution de X*A = B
```

Attention :

```
>> A.*B      % Produit terme à terme
>> X = A./B  % Division terme à terme
>> A.^p      % Puissance terme à terme
```

Autres fonctions utiles sur les matrices :

```
>> poly(A)        % Polyn. caract. de A
>> det(A)         % Déterminant de A
>> trace(A)       % Trace de A
>> [V,D] = eig(A) % Val. et vect. propres
```

Matrices creuses : gain en mémoire et en coût de calcul

```
>> A = eye(2) ; B = sparse(A) ;
>> A*[1 ;1] ; % 4 multiplications
>> B*[1 ;1] ; % 2 multiplications
```

Pour certaines fonctions, matrice = tableau de valeurs :

```
>> exp(A) ; log(A) ; sqrt(A) ;
>> round(A) ; sign(A) ; rem(A,2) ;
```

Pour d'autres, matrice = suite de vecteurs colonnes

```
>> min(A) ; % Minima des colonnes
>> max(A) ; % Maximum
>> sum(A) ; % Somme
>> prod(A) ; % Somme
>> cumsum(A) ; % Sommes cumulés
>> cumprod(A) ; % Produits cumulés
>> sort(A) ; % Tri des colonnes
>> median(A) ; % Val. médiane des col
```

Décomposition de matrices :

```
>> [U,S,V] = svd(X,0) ; % Valeurs singulières
>> R = chol(X) ; % Cholesky
>> [Q,R] = qr(X) ; % QR
>> [L,U] = lu(X) ; % LU (Lower, Upper)
```

3 Affichages

3.1 Affichage alpha-numérique

Façon Matlab ou façon C

```
>> disp(mess)
   pi vaut 3.142
>> fprintf('pi vaut %1.3e \n',pi)
   pi vaut 3.141e+000
```

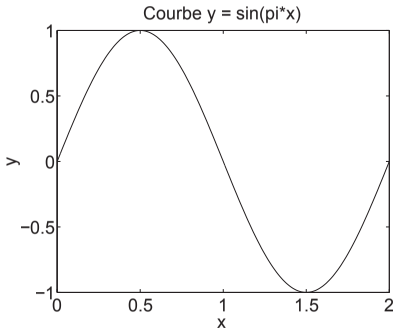
Saisie d'une valeur :

```
>> rep=input('Valeur de lambda : ') ;
```

3.2 Graphiques 1D : `plot`

Tracé d'une fonction avec axes et titre :

```
>> x = (0 : 0.1 : 2) ; y = sin(x*pi) ;
>> plot(x,y) ; % abscisse, ordonnée
>> title('Courbe y = sinus(pi*x)')
>> xlabel('x') ; ylabel('y')
```

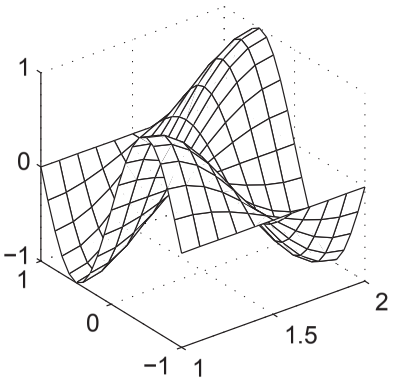



3.3 Graphiques 2D : `mesh`

Pour une fonction de deux variables :

```
>> x = 1 : 0.1 : 2 ; y = -1 : 0.1 : 1 ;
>> [X,Y] = meshgrid(x,y) ;
>> mesh(X,Y,cos(pi*X).*sin(pi*Y))
```

Mais aussi `meshc`, `contour`, `image`, `surf`, *etc...*



3.4 Affichage de plusieurs courbes

`subplot` divise la fenêtre graphique :

```
>> subplot(2,1,1)
>> plot(x,y)
>> subplot(2,1,2)
>> plot(x,y.^2)
```

1	2
3	4

La commande `figure` crée une nouvelle fenêtre graphique. `figure(2)` permet d'adresser la figure n° 2.

4 Fichiers, scripts, fonctions

4.1 Répertoire de travail

Commandes `cd` et `dir` (*idem* DOS) ; `cd` et `ls` (*idem* Unix).

4.2 Sauvegarde et chargement

Sauvegarde :

```
>> save NomFichier NomsVariables
```

Exemple :

```
>> save test.mat A x y
```

Par défaut sauvegarde dans `matlab.mat`.
Chargement :

```
>> load NomFichier
```

Voir aussi les commandes `who`, `whos`, `pack` et `clear`.

4.3 Scripts

Script : suite de commandes dans un fichier. Commentaires par : « % ». Nom de fichier avec extension « .m » (`nomfich.m`). Exécution sous Matlab : nom du fichier, sans « .m ». Variables de l'espace de travail visbibles dans le script et réciproquement.

4.4 Fonctions

Définition :

```
function y = sinuscardinal(x)
   z = sin(x) ; % Variable de stockage
   y = z./x ; % Variable de sortie
```

Appel identique aux fonctions Matlab :

```
>> sincpi = sinuscardinal(pi) ;
```

Autre exemple :

```
function [min, max] = minetmax(x)
   min = min(x) ; max = max(x) ;
```

Appel :

```
>> [miny, maxy] = minetmax(y) ;
```

Passage par valeur seulement. Nombres d'arguments en entrée et en sortie acessibles dans `nargin` et `nargout`. Variables de l'espace de travail invisible dans les fonctions et réciproquement. Outils de mise au point (déboggage) : plus simple : `keyboard` ; plus évolués : déboggeur intégré...

5 Boucles et contrôles

5.1 Test « `if` » :

Structure générale :

```
if (expression logique)
   suite d'instructions 1 ;
else
   suite d'instructions 2 ;
end
```

Opérateurs logiques : et (`&`), ou (`|`, `xor`), égal (`==`), différent (`~=`), supérieur (`>`, `>=`), inférieur (`<`, `<=`), non (`~`). Fonctions logiques : `exist`, `any`, `find`, `isinf`, `isnan`, *etc...* Expression considérée fausse si `0`, et vraie sinon.

5.2 Boucle « `for` » :

Structure générale :

```
for k=1 : 10
   suite d'instructions;
end
```