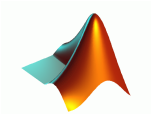


TRAVAUX PRATIQUES DE GÉNIE INFORMATIQUE

1 • But de la séance.

Ces deuxième et troisième séances ont pour but de vous faire avancer dans la programmation sous Matlab. Vous y découvrez les fonctions, les sous-programmes et la manipulation d'objets. Dans ces exercices **il est très important de bien contrôler où vous sauvegardez vos fichiers**. N'oubliez pas, de plus, les règles de bonne programmation vous imposant de créer un dossier de version chaque fois que vous modifiez quelque chose d'important dans votre programme. Vous devez créer **au moins** un dossier de version par séances, mais vous pouvez en faire plus. Essayer d'aller le plus loin possible.

2 • Programmes, sous programmes et fonctions sous Matlab.

2.1 • Programme.

Pour Matlab, un programme est simplement un fichier texte contenant une liste d'ordres Matlab. Ce fichier est considéré comme un exécutable Matlab a partir du moment où il possède l'extension .m.

-> • Travail : mettez vous dans un nouveau répertoire de travail, éditez un nouveau programme Matlab à l'aide de la commande `edit` :

```
>> edit MonPremierProgramme.m
```

Dans ce programme, nous allons réaliser un filtre passe-bas à réponse impusionnelle exponentielle. Si x_n est l'entrée du filtre et y_n est la sortie du filtre, l'équation récursive de ce filtre est simplement : $y_n = \alpha x_n + (1-\alpha) y_{n-1}$ avec $\alpha \in [0, 1]$ *(c'est peut-être l'occasion de réviser vos cours sur le filtrage)*.

Pour lancer le programme sous Matlab, il suffit de donner son nom dans la ligne de commande :

```
>> MonPremierProgramme
```

Vous pouvez aussi le lancer à partir de l'éditeur.

Vous allez générer un signal d'entrée qui sera une somme de trois sinusoïdes de fréquences différentes et d'un bruit gaussien de variance égale 0.3. Par exemple :

```
>> delta_t = 0.01 ;
>> nombre_echantillons = 1000 ;
>> temps = ( 0:(nombre_echantillons-1) ) * delta_t ;
>> signal = 1.9*sin(5*temps) + 1.5*sin(11*temps) + 1.7*sin(2.3*temps)
>> signal = signal + sqrt(0.3)*randn(size(temps)) ;
```

Vous mettrez ensuite à en œuvre le filtrage de x en créant une boucle itérative. Affichez les signaux de sortie, regardez l'influence du paramètre α sur le signal de sortie. Par exemple si $\alpha = 1$ que se passe-t-il ?

-> • Conseil : créez des variables pour le facteur du filtre α, les période de vos sinusoïdes, le nombre de données de vos signaux numériques, ...
De façon générale, même si Matlab ne vous oblige pas à ça, créez les variables que vous utilisez en début de programme.
Essayez d'utiliser le debugger, fixez des points d'arrêt.

2.2 • Sous-programme.

Les sous-programmes permettent d'isoler une partie d'un programme et d'ainsi clarifier la lecture du programme principal. Dans cet exercice, vous allez copier la boucle de filtrage dans un autre fichier que vous appellerez `filtrage.m` et la partie affichage dans un programme que vous appellerez `affichage.m`. Votre programme `MonPremierProgramme.m` se résume alors à définir des variables et appeler des sous-programmes.

2.3 • Fonction.

Lorsqu'on utilise des sous-programmes il faut que les variables soient toujours appelées avec le même nom. Par exemple, si votre sous-programme de filtrage a été défini pour filtrer une variable nommée x et son résultat se nomme y, il vous faudra toujours appeler ces variables sous ce nom. Essayez par exemple de définir plusieurs signaux avec des noms différents et de les filtrer avec votre sous-programme. Vous devez (normalement) vous rendre compte que ce n'est pas très pratique à cause de la dépendance des différents sous-programmes.
Pour résoudre ce problème, on peut définir une fonction.
Une fonction est définie dans un fichier à part comme le sous-programme mais le fichier comporte en en-tête la déclaration du fait que c'est une fonction. **Le nom de la fonction et du fichier dans lequel elle est définie doivent être le même.**
Ainsi une fonction réalisant le filtrage ci-dessus devrait être définie dans un fichier du nom de `FiltreExponentiel.m` et avoir le prototype suivant :

```
function [sortie] = FiltreExponentiel(entree, alpha)
% ici vous mettez votre calcul qui permet
% de definir la sortie a partir de l'entree.
% Par exemple
 NombreEchantillon = length(entree) ;
 sortie = alpha * ones(1,n) ;
```

Ce n'est bien sûr pas le bon code. Pour appeler cette fonction dans votre programme principal il suffit d'écrire par exemple dans ce programme principal :

```
y=FiltreExponentiel(x, 0.4) ;
```

Explorer les possibilités offertes par les fonctions. Voyez entre autre que les variables des fonction sont locales (par exemple `NombreEchantillon` n'existe pas dans le programme principal a moins qu'il ait été défini par ailleurs).

2.4 • Fonctions à nombre d'arguments variables.

Il est possible de créer des fonctions dont le nombre d'arguments est variable. Dans ce cas il faut utiliser les fonctions `nargin` et `nargout` qui vous donnent le nombre d'entrées et de sorties requise pour votre fonction.

Utilisez le help de Matlab, modifiez votre fonction `FiltreExponentiel` de façon à ce que la valeur par défaut de 0.4 soit attribuée à alpha lorsque la fonction n'est appelée qu'avec un argument.

2.5 • Modification de votre fonction.

(si vous travailler lentement, ne faites pas cette question).
Vous devez faire une modification de votre fonction de filtrage exponentiel. Ce filtrage est causal et provoque donc un déphasage. Considérez cette modification comme une révision majeure de votre programme, donc créez un nouveau dossier de version.
La nouvelle fonction que vous devez créer réalise un filtrage non-causal séparable (c'est à dire que l'algorithme est composé d'un filtrage causal et d'un filtrage non-causal). La récursion causale est identique à la récursion causale précédente à savoir :

$y_n^C = \alpha x_n + (1-\alpha) y_{n-1}^C$ (y_n^C est la partie causale du filtre),

$y_n^A = \alpha x_n + (1-\alpha) y_{n+1}^A$ (y_n^A est la partie anti-causale du filtre), toujours

$\alpha \in [0, 1]$, et enfin $y_n = \frac{1}{2-\alpha}(y_n^C + y_n^A - \alpha x_n)$ où y_n est la sortie du filtre. Programmez proprement de façon à minimiser le nombre de calculs réalisés par la machine. Regardez la différence de comportement de votre ancien filtrage (causal) et de ce nouveau filtrage (non-causal).

2.6 • Fonctions de fonctions.

Cette situation arrive lorsqu'on souhaite trouver les solutions d'une équations, ou des minima d'une fonction, ... Dans ce cas, il faut passer à ces fonction, une autre fonction en argument (ce qui reviendrait en C à passer un pointeur de fonction). Sous Matlab, ce passage de pointeur est obtenu facilement en utilisant le symbole `@`.
Par exemple créez la fonction suivante :

$$f(x) = 0{,}8 - \left(e^{\frac{-(x-3)^2-5}{100}} - \sin\left(\pi\left(\frac{x}{30}+1\right)\right)\right)$$

en la nommant (par exemple) `MaFonction`. Créez la et tracez la.
Vous chercherez le minimum de cette fonction entre -20 et +20 grâce à la fonction `fminbnd`. Vous chercherez la valeur annulant cette fonction qui est la plus proche de 20 grâce à la fonction `fzero`. Répétez cette opération pour chercher la valeur annulant cette fonction qui est la plus proche de 22.

```
>> fzero(@Mafonction,20) ;
```

Regardez les différentes options de ces fonctions.
Vous pouvez connaître le type d'une fonction en tapant :

```
>> functions(@MaFonction)
```

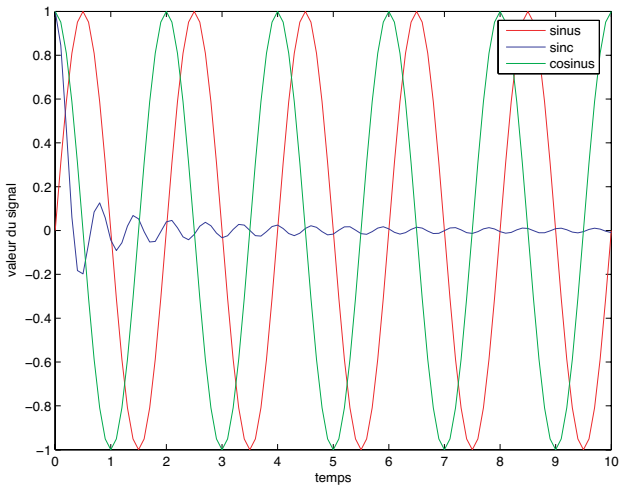
3 • Un peu plus loin avec l'affichage.

3.1 • Manipulation des axes.

La commande `axis` permet de contrôler les abscisses et ordonnées des axes que vous manipulez. Regardez l'aide de cette fonction. Créez une variable aléatoire x et affichez la sur un graphique qui soit compris entre 0 et 1.
La commande `axes` permet de contrôler la taille des graphiques dans une fenêtre. Regardez l'aide de cette fonction. Essayez de dessiner la variable x dans une autre figure qui n'occuperait que le quart de la fenêtre et positionnée en haut à gauche.

3.2 • Ecriture d'un titre sur une figure.

```
>> t = 0:0.1:10 ;
>> x = sin(pi*t) ; y=sinc(pi*t) ; z=cos(pi*t) ;
```



Affichez x, y et z sur une même figure (avec 3 couleurs différentes) avec t en abscisse. En utilisant la fonction `title` donnez un titre à votre figure. Nommez les axes verticaux et horizontaux grâce aux fonctions `xlabel` et `ylabel`. Utilisez la fonction `legend` pour donner le nom à chaque courbe comme illustré ci dessus.

3.3 • Sous-figures.

Utilisez la commande `subplot` pour créer trois sous-figures sur la même figure sur lesquelles vous dessinerez x y et z.

3.4 • Le *handle*.

Le *handle* est un concept Matlab qui peut être utilisé pour tous les objets Matlab. Le *handle* contient la liste de toutes les propriétés d'un objet Matlab. Chaque fois que vous ne saurez pas manipuler un objet, il y a toujours un moyen de le faire avec le *handle*. Pour récupérer le handle d'un objet, il vous faut utiliser la fonction `get`. Par exemple pour récupérer le handle de la figure 1 il vous faut taper :

```
>> h = get(figure(1)) ;
```

Si vous affichez `h` vous verrez l'ensemble des variables que vous pouvez consulter ou

modifier sur la figure. Par exemple essayez :

```
>> set(1,'Visible','off') ;
```

suivit de :

```
>> set(1,'Visible','on') ;
```

Essayez de modifier la position de la figure sur l'écran.
Faites une petite routine avec une boucle for qui baladerait la fenêtre en cercle sur votre écran de visualisation. Pour se faire, repérez les coordonnées qu'a votre fenêtre aux différents coins de votre écran.
Expérimentés quelques unes des possibilités de manipulation des figures dont vous disposez.

3.5 • Visualisation de l'animation d'un graphique.

Dans de nombreux cas, on a besoin d'afficher graphiquement les résultats d'un calcul au fur et à mesure du calcul. Par exemple essayez le code suivant :

```
>> x=-3:0.1:3 ;
>> for alpha=0:30
>> z=sinc(sin(alpha*pi/60)*x)'*sinc(sin(alpha*pi/60)*x) ;
>> figure(1) ; surf(z) ; drawnow ;
>> end
```

Il se peut que dans les nouvelles versions, l'usage de la fonction `drawnow` qui oblige Matlab à tout afficher à chaque itération soit inutile.

4 • Sauvegarde de variables, lecture de fichiers.

4.1 • Sauvegarde de variables.

Certains calculs peuvent avoir été très long et on peut souhaiter sauvegarder certains résultats en vue d'une exploitation ultérieure. Pour sauvegarder des variables, il vous faut utiliser la fonction `save`. Regardez l'aide de cette fonction et sauvegardez vos variables sous différents formats. Essayez d'ouvrir les fichiers sauvegardés (ils sont sauvegardés avec l'extension `.mat`).

4.2 • Lecture de fichiers.

La procédure inverse de la sauvegarde est la lecture de fichier. Pour cela on utilise la fonction `load`. Pour tester ces deux fonctions duales, sauvegardez quelques variables de votre environnement (pour connaître le nom des variables utilisées employez la commande `whos`). Effacez la totalité de vos variables :

```
>> clear all
```

Vérifiez que votre environnement est vide (`whos`) puis rechargez vos variable (`load`).

5 • Un peu d'algèbre linéaire.

5.1 • Un peu de calculs de moments en 2D.

Dans un premier temps, on va créer un nuage de points aléatoires orientés :

```
>> a=2 ; b=-3 ; x=randn(1,200)*10 ;
>> y= a*x + b + randn(size(x))*5 ;
```

Visualisez ces points.

```
>> figure(1) ; plot(x,y,'.') ; axis equal ;
```

Créez un vecteur $X = \begin{bmatrix} x_1 & y_1 \\ \dots & \dots \\ x_n & y_n \end{bmatrix}$. Calculez le barycentre de ces n points.

On rappelle que le barycentre n'est autre que le point dont les coordonnées x_0 y_0 sont les moyennes des coordonnées du nuage de point.

La matrice d'inertie du nuage de point est donnée par : $M = \frac{1}{n} X^T X$

Calculez les valeurs propres et les vecteurs propres de cette matrice .

Multipliez la matrice X^T par la matrice de vecteur propre pour obtenir le vecteur Y^T.

Visualisez le nuage de point du vecteur Y.

```
>> figure(2) ; plot(Y(:,1),Y(:,2),'.') ; axis equal ;
```

On appelle λ_1 et λ_2 les deux valeurs propres de la matrice M.

Tracez, sur cette même figure l'ellipse dont les allongements sont égaux à trois fois les racines carrées des valeurs propres de la matrice et dont le centre est le barycentre des nouveaux points. Pour vous aider, on rappelle qu'une ellipse de centre (x_0, y_0) et d'allongements α_x et α_y a pour équation : $\left(\frac{x - x_0}{\alpha_x}\right)^2 + \left(\frac{y - y_0}{\alpha_y}\right)^2 = 1$ ce qui peut s'écrire en utilisant un paramètre angulaire $\theta \in [-\pi,\pi]$:

$x = \alpha_x \cos\theta + x_0$, $y = \alpha_y \sin\theta + y_0$

Pour tracer l'ellipse, créez une trentaine de points. Expliquez ce que vous constatez si vous le pouvez.

Déduire de cet exercice une méthode pour tracer une ellipse ayant la même position dans le nuage des points de départ (avant transformation).

5.2 • Fonction.

Regroupez toute cette procédure dans une seule fonction dont le prototype serait :

```
[barycentre, allongement] = AnalyseNuageDePoint(Nuage) ;
```

Cette fonction prendrait en entrée un nuage de point 2D, afficherait sur une figure le nuage de points en bleu et l'ellipse englobante en rouge et renverrait le barycentre des points (`barycentre`) et les deux allongements de l'ellipse (`allongement`).

5.3 • Rotation et translation.

Une matrice de rotation d'un angle θ dans le plan s'écrit : $\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$. Faites une fonction qui a comme entrée l'angle de rotation et comme sortie une matrice de rotation. Utilisez cette fonction pour créer une animation faisant tourner l'ensemble de vos points

autour du point (0,0). Essayez de les faire tourner autour d'un autre point.

Créez un objet 3D.

```
>> t = 0:pi/10:2*pi;
>> [X,Y,Z] = cylinder(4*cos(t));
>> figure(9) ; mesh(X,Y,Z) ;
```

Faites le tourner selon le même principe en remarquant qu'en 3D un objet peut tourner autour de trois axes. La rotation résultant de cette rotation autour de 3 axes est obtenue en multipliant toutes les rotation entre elles :

$$\mathrm{Rot}(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Utilisez les différentes options d'affichages des objets 3D. En utilisant le handle, faites tourner la figure elle même.

Conseil : pour faire cette opération, vous aurez besoin de regrouper les points dans une seule matrice. Une telle procédure peut être obtenue en faisant :

```
>> k=1 ;
>> for i=1:21
>> for j=1:21
>> P(k,1)=X(i,j) ; P(k,2)=Y(i,j) ; P(k,3)=Z(i,j) ; k=k+1 ;
>> end ;
>> end ;
```

Une procédure identique dans l'autre sens doit vous permettre de visualisez cette forme.