TP3 – LES FONCTIONS

1 – LES BASES DES FONCTIONS

Dans la programmation, une **fonction** est une séquence d'instructions qui accomplit une tâche spécifique. Une fonction peut avoir des arguments d'entrée (**input arguments**), qui lui sont rendus par l'**utilisateur**, l'entité appelant la fonction. Les fonctions ont aussi des arguments de sorties (**output arguments**), qui sont les résultats de la fonction que l'utilisateur s'attend à recevoir une fois que la fonction a terminé sa tâche.

Nous définirons une fonction en termes de son **entête**. Dans MATLAB, l'entête d'une fonction est une liste des arguments de sortie de la fonction entourée de **crochets**, suivie par un signe égal " = ", le nom de la fonction, et les arguments d'entrée de la fonction, entourés de **parenthèses**. Par exemple, l'entête de la fonction **sin** ressemble à ceci : **[y] = sin (x)**.

Pour programmer vos propres fonctions, vous aurez besoin d'utiliser une nouvelle partie de l'environnement MATLAB appelée l'**éditeur**. L'éditeur vous permet de construire, d'éditer et de sauvegarder vos fonctions. Vous pouvez ouvrir l'éditeur en cliquant sur le bouton "**new→m-file**", situé dans le coin supérieur gauche de l'environnement MATLAB.

Nous commencerons par la construction d'une fonction très simple définie par :

[S] = monAdditionneur (a,b,c), où **S** est la somme de **a**, **b** et **c**.

La première ligne d'une nouvelle fonction doit toujours être le mot **function**, suivi par l'entête de la fonction. La première ligne de la fonction **monAdditionneur** est alors :

```
function  [S] = monAdditionneur (a,b,c)
```

Vous pouvez remarquer que le mot **function** est en bleu, parce que **function** est un **mot-réservé** qui dénote le début d'une fonction. Les mots-réservés ne peuvent pas être affectés comme une variable ou des noms de fonctions. Ensuite, vous devez écrire le corps de la fonction. C'est la séquence d'instructions qui produiront les sorties désirées basées sur les entrées données.
S'il y a de multiples lignes à l'intérieur de la fonction, MATLAB les exécute en ordre. Puisque notre fonction est très simple, le corps exigera seulement une seule instruction.

```
function [S] = monAdditionneur (a,b,c)
S= a + b + c ;
end
```

Remarque : Mettez toujours un point-virgule à la fin de toutes expressions d'affectation. Utilisez plutôt la fonction display si vous voulez que votre fonction affiche sur l'écran le résultat de l'exécution des instructions.

Constatez que le mot **end** est aussi un mot-réservé qui dénote la fin d'une fonction. Cette fonction continuera à travailler si le mot **end** n'est pas placé à la fin, mais il causera des problèmes plus tard. Donc, pour notre but, **toujours terminer la fonction avec le mot end**.

Nous introduisons maintenant une bonne pratique de programmation. Un **commentaire** est une ligne à l'intérieur d'une fonction qui n'est pas lue par MATLAB comme code. Vous pouvez dénoter un commentaire en plaçant le symbole % au début d'une ligne. Vous constaterez que tous les caractères de cette ligne sont en vert. MATLAB n'exécutera aucun code qui est en vert. Quand votre programme devient plus long et plus compliqué, les commentaires vous aident à comprendre ce que vous êtes en train d'essayer de faire, à vous éviter des erreurs de programmation, et à trouver des erreurs quand vous faites des fautes. Nous recommandons fortement que vous commentez énormément votre propre programme.

Astuce : Vous pouvez mettre en commentaires de grands blocs de code en sélectionnant un bloc de code avec la souris et en appuyant ensuite sur **ctrl+r**. Vous pouvez enlever les commentaires du code en appuyant sur **ctrl+t**. Mettre de grands blocs de code en commentaire est utile quand vous voulez essayer une solution différente à un problème sans risquer de perdre de l'information de votre essai précédent.

Astuce : Construisez de bonnes pratiques de programmation en donnant aux variables et aux fonctions des noms descriptifs, en commentant souvent, et en évitant des lignes superflues de code.

Ex1: Ecrire la fonction **monAdditionneur** en y ajoutant des commentaires.

Pour sauvegarder votre fonction, cliquez sur le bouton **save**, situé dans le coin supérieur gauche ($3^{ème}$ bouton) de l'éditeur. Vous pouvez aussi sauvegarder votre fonction en appuyant sur les touches **Alt→d→s** ou **ctrl+s**. Quand la fenêtre qui s'affiche vous demande de nommer le fichier, le nom du fichier doit être le même que celui de la fonction. Le type du fichier doit être .m, un m-file, qui est le type de fichier standard pour les fonctions MATLAB. Sauvegardez cette fonction sous le nom **monAdditionneur.m.**

Astuce : C'est une bonne pratique de programmation de sauvegarder assez souvent avec Ctrl+s pendant que vous êtes en train d'écrire votre fonction.

Les fonctions doivent se conformer à un schéma de nomination similaire aux variables. Elles peuvent contenir seulement des caractères alphanumériques et des soulignés, et le premier caractère doit être une lettre. Le nom de la fonction doit être moins de 255 caractères de long. Une fois que votre fonction est sauvegardée dans le répertoire courant de travail, elle se comporte exactement comme une des fonctions construites de MATLAB et peut être appelée de l'invite des commandes ou par d'autres fonctions.

Ex2: Utilisez votre fonction **monAdditionneur** à l'invite des commandes pour calculer la somme de quelques nombres. Vérifier que le résultat est correct. Essayer la commande **help** pour votre fonction **monAdditionneur**.

Vous pouvez **composer** des fonctions en affectant des appels de fonctions comme entrées à d'autres fonctions. Dans l'ordre des opérations, MATLAB exécutera d'abord l'appel de fonction le plus interne. Vous pouvez aussi affecter des expressions mathématiques comme entrées des fonctions. Dans ce cas, MATLAB exécutera d'abord les expressions mathématiques.

Ex3: Utiliser la fonction monAdditionneur pour calculer la somme de sin π, cos π et tan π. Utiliser les expressions 5+2, 3*4, et 12/6 comme entrées de la fonction monAdditionneur et vérifier qu'elle effectue les opérations correctement.

Les fonctions MATLAB peuvent avoir de multiples arguments de sorties. Quand vous appelez une fonction avec de multiples arguments de sorties, vous pouvez placer une liste de variables auxquelles vous voulez affecter, à l'intérieur des crochets, séparées par des virgules.

Considérez la fonction suivante :

```
function  [A,B] = maSomTrig(a,b)
A = sin (a) + cos(b) ;
B = sin (b) + cos(a) ;
End
```

Ex4: Calculer la fonction **maSomTrig**(a,b) pour **a = 2** et **b = 3**. Affecter le **premier argument de sortie** à la variable **C** et le **second argument de sortie** à la variable **D**.

Si vous faites moins d'affectations qu'il n'y a de variables de sorties, MATLAB sera seulement les premières affectations et le reste est ignoré.

En écrivant des fonctions, vous pouvez oublier d'assigner une des sorties si votre fonction est compliquée. Si c'est le cas, MATLAB s'arrêtera, et vous obtiendrez une erreur.

Un autre mot-clé utile est **return**. Quand MATLAB voit une instruction **return**, il termine **immédiatement** la fonction comme s'il a exécuté l'instruction **end** de la fonction. Si un des arguments de sorties n'a pas été assigné quand une instruction **return** est exécutée, vous obtiendrez une erreur.

2 – L'ESPACE DE TRAVAIL DE LA FONCTION

Le TP2 a introduit l'idée de l'espace de travail où les variables créées à l'invite des commandes sont stockées. Une fonction a aussi un espace de travail. **L'espace de travail de la fonction** est un espace dans la mémoire qui est réservé pour les variables créées à l'intérieur de cette fonction. Cet espace de travail n'est pas partagé avec celui de la fenêtre des commandes. Donc, une variable avec un nom donné peut être affectée à l'intérieur d'une fonction sans changer une variable avec le même nom à l'extérieur de la fonction. L'espace de travail d'une fonction est ouvert à chaque fois qu'une fonction est utilisée.

Ex5: Quelle sera la valeur de **S** après que les lignes du code suivant sont exécutées ?

```
>> S = 1 ;
>> d = monAdditionneur (1, 2, 3) ;
>> S
```

Dans la fonction **monAdditionneur**, la variable **S** est une **variable locale**. C'est-à-dire qu'elle est définie seulement dans l'espace de travail de la fonction **monAdditionneur**. Donc, elle ne peut pas affecter des variables dans des espaces de travail externes à la fonction, et les actions entreprises dans des espaces de travail extérieurs à la fonction ne peuvent pas l'affecter, même si elles ont le même nom. Ainsi, dans l'exercice précédent, il y a une variable **S**, définie dans l'espace de travail de la fenêtre des commandes. Quand la fonction **monAdditionneur** est appelée à la ligne suivante, MATLAB ouvre un nouvel espace de travail pour les variables de cette fonction. Une des variables créées dans cet espace de travail est une autre variable S. Cependant, puisqu'elles ont des espaces de travail différents, l'affectation à **S** à l'intérieur de la fonction monAdditionneur ne change pas la valeur affectée à **S** dans l'espace de travail de la fenêtre des commandes.

Pourquoi avoir des espaces de travail de fonctions séparés plutôt qu'un seul espace de travail ? Parce que c'est très efficace pour de grands projets consistant en de nombreuses fonctions travaillant ensemble.

Par exemple, si un programmeur est chargé d'écrire la fonction **sin** de MATLAB et un autre d'écrire la fonction **cos** de MATLAB, nous ne voudrions pas que chaque programmeur ait à se soucier des noms des variables que l'autre va utiliser. Nous voulons qu'ils puissent travailler d'une manière indépendante et avoir confiance que leur propre travail n'interfère pas avec celui de l'autre et vice versa. Donc, les espaces de travail séparés protègent une fonction de l'influence externe.

Les seules choses externes à l'espace de travail de la fonction qui peuvent affecter ce qui se passe à l'intérieur d'une fonction sont les arguments d'entrées, et les seules choses qui peuvent échapper au monde extérieur de l'espace de travail d'une fonction quand la fonction se termine, sont les arguments de sortie.

Exemple : Considérer la fonction suivante :

```
function [x , y] = TestEspaceTravail(a , b)
x = a+b ;
y= x*b ;
z= a+b ;
end
```

Ex6: Quelles seront les valeurs de a, b, x, y, et z après que le code suivant soit exécuté ?

```
>> clear all
>> a = 2 ;
>> b = 3 ;
>> z = 1 ;
>> [x , y] = TestEspaceTravail(b , a)
```

Ex7: Quelles seront les valeurs de a, b, x, y, et z après que le code suivant soit exécuté ?

```
>> clear all
>> x = 2 ;
>> y = 3 ;
>> [a , b] = TestEspaceTravail(y , x)
```

3 – LE CHEMIN DE MATLAB

Quand une fonction est appelée à l'invite des commandes ou de l'intérieur d'une fonction, MATLAB doit rechercher le fichier (m-file) qui lui dit comment exécuter cette fonction. MATLAB recherche cette fonction d'abord dans le répertoire courant ou comme une sous fonction (décrite dans la section suivante). S'il ne la trouve pas, MATLAB regardera dans les dossiers le long du **chemin** de MATLAB jusqu'à ce qu'il trouve le fichier approprié à exécuter. Si MATLAB atteint la fin du fichier sans trouver la fonction demandée, il retourne une erreur.

Vous pouvez visualiser le chemin de MATLAB en cliquant sur **File → Set Path**. Vous pouvez modifier le chemin sous **Set Path**.

4 – LES SOUS FONCTIONS

Une **sous fonction** est une fonction qui est définie dans le même fichier que sa **fonction parent**. Seulement, la fonction parent est capable d'appeler la sous fonction. Cependant, la sous fonction retient un espace de travail séparé de sa fonction parent. Une sous fonction est déclarée après l'instruction **end** de sa fonction parent, et elle doit avoir une instruction **end** à la fin de sa propre définition.

Exemple : Considérer la fonction suivante **maDistanceXYZ** et sa sous fonction **maDistance** :

```
function [D] = maDistanceXYZ(x,y,z)
D(1) = maDistance(x,y) ;
D(2) = maDistance(x,z) ;
D(3) = maDistance(y,z) ;
end
function [D] = maDistance(x,y)
D=sqrt((x(1)-(y(1))^2+(x(2)-y(2))^2) ;
end
```

Remarquez que les variables **D**, **x** et **y** apparaissent à la fois dans la fonction **maDistanceXYZ** et sa sous fonction **maDistance**. Ceci est permis parce qu'une sous fonction a un espace de travail séparé de sa fonction parent.

Les sous fonctions sont utiles quand une tâche doit être accomplie plusieurs fois à l'intérieur de la fonction mais non à l'extérieur de la fonction. De cette manière, les sous fonctions aident leur fonction parent à accomplir sa tâche sans encombrer le répertoire de travail avec des fichiers supplémentaires.

Ex8: Appeler la fonction **maDistanceXYZ** pour x=[0 0], y=[0 1], et z=[1 1]. (**Rép.:** 1, 1.4142, 1).

Notez que MATLAB regardera à travers la liste des sous fonctions avant d'aller voir le chemin de MATLAB.

Ex9: Réécrire la fonction **maDistanceXYZ** directement sans utiliser de sous fonctions.

5 – LES HANDLES DE FONCTIONS

Jusqu'à maintenant, nous avons affecté diverses structures de données aux noms des variables. Etre capable d'affecter une structure de données à une variable nous permet de passer de l'information aux fonctions et d'obtenir en retour de l'information d'elles d'une manière propre et ordonnée.

Des fois, il est utile de pouvoir passer une fonction comme une variable à une autre fonction. En d'autres termes, l'entrée de certaines fonctions peut être d'autres fonctions. Pour accomplir ceci, nous avons besoin des handles de fonctions.

Les **handles de fonctions** sont des variables auxquelles ont été affectées des fonctions comme leur valeur. Le handle d'une fonction est créé en plaçant le symbole **@** devant une fonction dans le chemin courant et en utilisant ensuite l'opérateur d'affectation.

Ex10: Affecter la fonction **exp** à la variable F. Vérifier que F a le type de handle de fonction en utilisant la fonction **class**.

```
>>  F = @ exp ;
```

Dans l'exercice précédent, F est maintenant équivalent à la fonction **exp**. Juste comme x = 1 signifie que x et 1 sont interchangeables, F et la fonction **exp** sont maintenant interchangeables.

Ex11: Calculer e^2 en utilisant le handle de fonction F. Vérifier que c'est correct en utilisant la fonction **exp**.

Ex12: Programmer une fonction appelée **maFoncPlusUn** qui prend un handle de fonction, H, et une variable x comme arguments d'entrées. Cette fonction doit retourner H évalué à x, et le résultat augmenté de 1. Vérifier que cela marche pour différentes fonctions et valeurs de x.

L'utilisation des handles de fonctions exige que la fonction affectée à une variable soit sauvegardée dans le répertoire courant. Une **fonction anonyme** est un handle de fonction qui est affecté à une fonction non stockée. Elle est créée selon la construction suivante :

F = @ (entrées) définition de la fonction

Exemple : Créer la fonction **monAdditionneur** vue précédemment en utilisant les handles d'une fonction anonyme. Utiliser la variable F. Vérifier que la variable F et la fonction **monAdditionneur** se comportent de la même façon pour a=1, b=2 et c=3.

```
>> F = @(a, b, c) a + b + c
>> y = F(1, 2, 3)
>> y = monAdditionneur (1, 2, 3)
```

6 – LES FICHIERS SCRIPT

Un **fichier script** est un fichier de type **m (m-file)** qui contient une séquence d'instructions mais ce n'est pas une fonction. Contrairement à une fonction, un fichier script partage son espace de travail avec le répertoire courant. Un fichier script est créé en écrivant des lignes de code juste comme vous auriez fait à l'invite des commandes. Il est **exécuté** quand vous pressez le bouton **run** dans la barre d'outils supérieure. Quand le fichier script est exécuté, MATLAB exécute les instructions en ordre comme si vous les avez tapées dans l'invite des commandes une à la fois.

Ex13: Ecrire un fichier script qui calcule le volume et la surface du cylindre pour un rayon et une hauteur donnés.

Astuce: Au début de chaque fichier script, commencez toujours par écrire **clc, clear all** et **close all**. Ceci parce que les fichiers script partagent leur espace de travail avec le répertoire courant, et c'est une bonne pratique de s'assurer que votre fichier script n'utilise pas des variables d'anciennes sessions de MATLAB.

Remarque: Les fonctions sont les plus utilisées quand on doit faire un tâche plusieurs fois.

Vous pouvez organiser le code de vos fichiers script en cellules (**cells**), qui peuvent être exécutées sans exécuter le reste du script. Une cellule est un morceau de code du script qui peut être exécuté individuellement. Les cellules de code peuvent être créées en insérant une ligne qui commence avec deux symboles **%%**. Vous pouvez exécuter une cellule en cliquant sur le bouton **evaluate-cell** ou le bouton **evaluate-cell-and-advance**.

Ex14: Organiser le fichier script précédent en cellules. Exécuter les cellules une à la fois en utilisant le mode cellule. Vous pouvez activer le mode cellule sous **cell→Enable Cell Mode** si ce n'est déjà fait.