TP 3 : Les fonctions dans Matlab

Dans ce TP, nous verrons l'utilisation des fonctions dans Matlab. Nous verrons comment écrire une fonction dans Matlab, ainsi que comment appeler une fonction codée en C dans Matlab.

Vous devez avoir terminé le TP2 avant de commencer ce TP. De plus, vous devez réaliser ce TP sous Matlab Linux.

1 Ecrire des fonctions en Matlab

1.1 Ecrire des fonctions simples

Pour écrire une fonction dans Matlab, la première règle à respecter est de donner au fichier .m le même nom que la fonction que l'on est en train d'écrire. Par exemple, une fonction qui s'appellerait **mafact** devra être écrite dans le fichier **mafact.m**.

Pour écrire, une fonction dans Matlab, on doit d'abord donner le noms des valeurs en sortie générées par la fonction, puis le nom de la fonction, et enfin le noms des paramètres en entrée de la fonction :

```
function [sortie1,sortie2, ...] = nom_fonction(entree1,entree2, ...)
...
end
```

Par exemple, si on souhaite faire une fonction **produit**, qui calcule et renvoie en sortie le produit de deux scalaires passés en paramètre, on écrira dans le fichier **produit.m** :

```
function [res] = produit(a,b)
    res = a*b;
end
```

Pour tester cette fonction, placez-vous (à l'aide de l'explorateur de fichiers Matlab) dans le répertoire qui contient le fichier **produit.m** et on écrira dans la fenêtre de commande :

```
e = produit(2,4);
```

Une fois cette commande exécutée, on récupère bien dans **e** la valeur de sortie de la fonction, c'est à dire le produit des deux paramètres en entrée.

On remarquera que l'on ne voit pas, dans la zone des variables de Matlab, une variable nommée **res**. En effet, la variable **res** est interne à la fonction **produit**, et ne vit que pendant l'appel de cette fonction. Une fois la fonction terminée, la variable est effacée. De plus, la variable **res** n'existe que pour la fonction **produit** : elle ne peut pas être lue par une autre fonction ou par une commande directement dans Matlab. On récupère la valeur de **res** en la déclarant comme une valeur de sortie de la fonction, et en récupérant cette sortie dans la variable **e**.

Considérez cette autre fonction :

```
function [res] = somme
    res = a+b;
end
```

Cette fois-ci, on appellera cette fonction ainsi :

```
a=2;
b=4;
e = somme();
```

Cette fonction ne marche pas : même s'il existe des variables **a** et **b** dans les variables de Matlab, ce ne sont pas des variables internes à la fonction **somme**. Or, une fonction ne peut pas lire les variables de Matlab, elle ne connaît que ses variables internes et les variables passées en paramètre. Voilà pourquoi,

dans la fonction **produit**, on donnait les valeurs à multiplier comme des paramètres d'entrée de la fonction : c'est le seul moyen de transmettre à la fonction une ou plusieurs valeurs.

Si on récapitule, on doit, pour écrire une fonction, décider de plusieurs éléments :

— Un nom de fonction, qui devra être aussi le nom du fichier dans lequel sera écrit la fonction.

— Des paramètres d'entrée de la fonction, qui seront les valeurs des éléments dont aura besoin la fonction pour réaliser son calcul, et qu'elle ne pourra pas décider d'elle-même. Il est important de comprendre que la fonction ne peut pas lire les variables de la zone des variable de Matlab : une fonction ne connaît que ses variables internes, c'est à dire celles déclarées en paramètres d'entrée ou déclarées dans le code même de la fonction.

— Des paramètres en sortie de la fonction, qui seront les valeurs calculées par la fonction, et que l'on souhaite récupérer une fois la fonction terminée. Si vous regardez bien l'appel que vous avez fait de la fonction **produit** juste avant, vous verrez que la variable **res** interne à la fonction n'est pas dans la zone des variables de Matlab. En effet, une fois la fonction terminée, toutes ses variables internes sont détruites. Ici, on récupère le résultat du calcul grâce à la variable **e** qui n'est pas interne à la fonction (elle n'est pas déclarée dans la fonction) et qui est utilisée, lors de l'appel à la fonction, comme une variable de sortie.

Enfin, considérez cette fonction qui ne prend aucun paramètre de sortie :

```
function incrementer(a)
    a=a+1;
end
```

Cette fonction ajoute simplement 1 à son entrée. Testez-la ainsi dans Matlab :

```
e = 1;
incrementer(e);
```

La fonction s'exécute, mais la variable e n'est pas incrémentée de 1. Ceci est du au fait que Matlab transmet les paramètres d'entrée par valeur : lorsque la fonction **incrementer** est appelée, la variable **e** n'est pas substituée à la variable **a** ; c'est la valeur de **e** qui est recopiée dans **a**. Toutes les modifications réalisées ensuite dans **a** sont perdues lorsque la fonction se termine, et **e** n'a jamais changé de valeur.

En C, c'est la même chose : par défaut, les variables sont passées par valeur, ce qui signifie que si une fonction modifie la valeur d'un de ses paramètres d'entrée, ceci n'a aucune conséquence sur les variables de la partie principale du programme. Il existe un mécanisme, utilisant les pointeurs, afin de modifier cette limitation et faire ce que l'on appelle le passage de paramètres par référence (et non plus par valeur).

Questions

— Comment faire, selon vous, pour écrire une fonction **incrémenter** qui incrémente la valeur donnée en entrée ?

— Ecrivez une fonction qui renvoie en sortie le maximum de trois valeurs passées en paramètre d'entrée.

— Ecrivez une fonction qui renvoie la valeur et le numéro d'ordre de l'élément d'une matrice **A** qui est le plus proche d'une valeur **b**. A vous de bien réfléchir aux paramètres d'entrée et de sortie de la fonction.

1.2 Ecrire des fonctions avec un nombre variable de paramètres en entrée

On peut écrire des fonctions dont le nombre de paramètres n'est pas fixe, mais peut grandir. Pour ce faire, on déclare la fonction ainsi :

```
function [sortie1,sortie2, ...] = nom_fonction(entree1,entree2,...,varargin)
...
end
```

Ici, on déclare une fonction avec un certain nombre de paramètres de sortie obligatoires, ainsi qu'un certain nombre de paramètres d'entrée obligatoires. Le mot clef **varargin** permet de spécifier qu'en plus des entrées obligatoires, il pourra y avoir des entrées supplémentaires à la fonction.

Considérez ce code qui permet de faire le produit de tous les entiers passés en paramètre de la fonction :

```
function [res] = produit(a,b, varargin)
    res = a*b;
    for i=1:length(varargin)
```

```
        res = res*varargin{i};
    end
end
```

Si l'on appelle ensuite, dans l'éditeur de code Matlab, la fonction ainsi :

```
r = produit(2,4)
```

La fonction place dans **r** la valeur 8, comme on pouvait s'y attendre. Si l'on appelle la fonction ainsi :

```
r = produit(2,4, 3, 7)
```

La fonction place dans **r** la valeur 168. En réalité, la fonction a placé dans son paramètre d'entrée **a** la valeur 2, dans son paramètre d'entrée **b** la valeur 4, et dans le paramètre d'entrée **varargin** une liste contenant les nombres 3 et 7. Dans le code de la fonction, on parcourt cette liste et on multiplie la variable **res** avec les valeurs de la liste.

Enfin, si on appelle la fonction ainsi

```
r = produit(2)
```

Matlab affiche un erreur, expliquant que nous n'avons pas saisi suffisamment de paramètres. En effet, on a bien spécifié dans le corps de la fonction qu'il fallait au moins, en paramètres d'entrée, deux valeurs **a** et **b**, ce qui n'est pas le cas ici...

> **Questions**
>
> Ecrivez une fonction qui renvoie le maximum de toutes les valeurs passées en paramètre d'entrée. Il faut passer au moins une valeur à la fonction.

1.3 Ecrire des fonctions avec un nombre variable de paramètres en sortie

Afin d'écrire des fonctions avec un nombre variable de paramètres de sortie, on utilise le même principe que précédemment, mais avec le mot clef **varargout**. Par exemple, voici le code d'une fonction permettant de demander des entiers à l'utilisateur pour remplir les variables de sortie :

```
function [varargout] = demander()
    for i=1:nargout
        varargout{i} = input('Entrez une valeur : ');
    end
end
```

La grande différence avec la partie précédente vient de l'utilisation du mot clef **nargout** qui permet de connaître le nombre de paramètres de sortie choisi par l'utilisateur. EN effet, on peut pas utiliser la fonction **length** sur la variable **varargout** car elle n'est pas encore construite et initialisée lorsque la fonction début. C'est au fur et à mesure que l'on remplit les valeurs de **varargout{k}** dans la boucle for que cette variable est construite.

Si l'on appelle cette fonction ainsi :

```
[a b c] = demander();
```

Alors la fonction demandera trois éléments qu'elle rangera dans les variables **a**, **b** et **c**.

> **Questions**
>
> 1. Ecrivez une fonction qui affiche le maximum de toutes les valeurs passées en paramètre, ainsi que, si demandé, la position de ce maximum. La fonction doit prendre au un paramètre en entrée, et produire au moins un élément en sortie.
> 2. Ecrivez une fonction qui prend en paramètre une matrice, et renvoie en sortie les n plus grands éléments de cette matrice, où n est le nombre de paramètres en sortie de la fonction. Le taille de la matrice doit nécessairement être supérieure àn, ou la fonction produira une erreur (avec la fonction **error**).

2 Utiliser MEX dans Matlab

2.1 Un premier programme C sous Matlab

MEX (Matlab EXecutable) est une commande Matlab permettant de compiler du code C afin de pouvoir être appelé dans Matlab. Pour l'utiliser, il faut évidemment savoir coder en C, ce que vous devriez savoir faire grâce aux cours d'Informatique de Base. Pour commencer, nous allons écrire un très court programme C qui affichera simplement bonjour.

A l'aide de votre éditeur de code préféré (**gedit** ou autre), écrivez ce code dans un fichier **hellomat.c** :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
       printf("Bonjour !\n"); /* On affiche Bonjour */
}
```

Ensuite, dans Matlab, assurez vous d'être, dans l'explorateur de fichiers, dans le même dossier que celui qui contient le fichier **hellomat.c**. Tapez alors cette commande dans l'éditeur de commandes :

```
mex hellomat.c
```

Si tout s'est bien passé, vous devriez n'avoir eu aucun message d'erreur (peut-être un message d'avertissement concernant la version de **gcc**), et un fichier **hellomat.mexa64**. Il vous suffit, pour exécuter votre programme C sous Matlab, d'écrire, dans l'éditeur de commandes Matlab :

```
hellomat
```

Vous devriez voir s'afficher à l'écran le message Bonjour !

Si on regarde attentivement notre programme C, on remarque deux éléments :

— On inclue le fichier **mex.h**, qui contient l'équivalent des nombreuses commandes utiles pour faire communiquer Matlab et C.

— On écrit la fonction principale comme **mexFunction**, et non pas comme **main**.

2.2 Gérer les entrées dans le programme

Connaître le nombre d'entrées d'un programme

Simplement afficher bonjour à l'écran est un premier pas, mais pour pouvoir faire des programmes intéressants, il faut pouvoir échanger des variables entre Matlab et le programme C. Pour ce faire, on utilise les paramètres de la fonction **mexFunction**. Le premier paramètre qui nous intéresse est l'entier **nrhs** qui nous donne le nombre de paramètres en entrée de la fonction. Ecrivez ce programme dans le fichier **param.c**, et compilez-le avec **mex** :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
       printf("%d\n", nrhs);
}
```

Si vous appelez la fonction directement, elle devrait afficher 0 :

```
>> param
0
```

Cependant, si vous appelez cette fonction en lui passant des paramètres, la sortie est différente :

```
>> param(1, 3)
2

>> param(7, 9, 2, 9)
4
```

```
>> param(7, 8, [1 2 3; 2 3 4])
3
```

L'entier **nrhs** nous permet de connaître, dans le code C, combien de paramètres ont été donné en entrée de la fonction sous Matlab. Ces paramètres peuvent être des nombres ou des tableaux.

> **Questions**
>
> Ecrivez (et testez) un programme C qui permet de vérifier que l'utilisateur a bien entré, dans la fonction, trois ou quatre paramètres. Si ce n'est pas le cas, on affiche le message *Erreur* et on interromp le programme avec le mot clef **return**. Sinon, on affiche *Ok*.

Récupérer en entrée des valeurs de type double

On peut connaitre le nombre d'entrées d'un programme, il nous manque maintenant à pouvoir récupérer ces entrées. Nous allons commencer par tenter de récupérer des scalaires (des double). Tout se passe avec la variable **prhs**, qui est un tableau. Chaque case de ce tableau contient une des variables passées en entrée de la fonction sous Matlab. Ecrivez ce code dans le fichier **param2.c**, et compilez-le avec **mex** :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
	int i;
	double e;

	for (i=0; i<nrhs; i++)
	{
		e=mxGetScalar(prhs[i]);
		printf("%f\n", e);
	}
}
```

Ici, on écrit une boucle **for** qui parcourt les éléments du tableau **prhs** et affiche chacun d'eux. On utilise la fonction **mxGetScalar** sur chaque élément de ce tableau afin de le transformer en **double** et le stocker dans la variable **e**, que l'on affiche ensuite.

Sous Matlab, l'appel de la fonction avec des paramètres scalaires nous permet d'afficher chacun de ces paramètres :

```
>> param2(3, 4, 8, 9)
3.000000
4.000000
8.000000
9.000000
```

Ce code nous permet d'identifier deux éléments importants du code C avec Matlab :

- Le tableau **prhs** contient tous les paramètres passés à la fonction dans Matlab. On ne peut pas utiliser directement ces paramètres, il faut utiliser une fonction pour les convertir.
- Lorsque ces paramètres sont des scalaires de type **double**, on peut les stocker dans un **double** avec la fonction **mxGetScalar**.

> **Questions**
>
> 1. Ecrivez une fonction Mex **prod** qui affiche le produit de tous les scalaires passés en paramètre de la fonction :
>
>    ```
>    >> prod(2, 6, 8)
>    96
>    ```

2. Ecrivez une fonction Mex **monmax** qui affiche le maximum de tous les scalaires passés en paramètre.
3. Que se passe-t-il si vous renommez la fonction **mymax** en **max** et recompilez le fichier avec **mex** ?

Récupérer en entrée des valeurs de type matrice

On peut évidemment, dans Matlab, récupérer des matrices passées en paramètre d'une fonction. Matlab propose des fonctions afin de récupérer, pour chaque matrice passée en paramètre, sa taille, son nombre de dimensions, et ses valeurs. Il est important de savoir que, dans un code C Mex, les matrices sont vues comme des vecteurs, où chaque élément est rangé à la place correspondant à son numéro d'ordre (cependant, en C, le numéro d'ordre commence à 0). Considérez le code suivant à copier (et compiler) dans le fichier **matrice.c** :

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
	int i, nb_dim, nb_elem;
	const int *size;

	if(nrhs < 1)
	{
		printf("Il faut donner au moins un parametre en entree\n");
		return;
	}

	nb_dim = mxGetNumberOfDimensions(prhs[0]);
	printf("Le tableau en premier parametre possede %d dimensions\n", nb_dim);

	size = mxGetDimensions(prhs[0]);
	for (i=0; i<nb_dim; i++)
	{
		printf("La dimension %d possede %d elements\n", i, size[i]);
	}

	nb_elem = mxGetNumberOfElements(prhs[0]);
	printf("Le tableau possede %d elements en tout\n", nb_elem);
}
```

Appelez la fonction en lui passant en paramètre un tableau, en faisant par exemple :

```
matrice([1 2 3; 4 3 2; 3 3 3; 9 8 7])
```

Le code affiche tous les paramètres du tableau. On peut en déduire le rôle important de certaines fonctions :

— La fonction **mxGetNumberOfDimensions** permet de récupérer le nombre de dimensions du tableau (ici, 2).
— La fonction **mxGetDimensions** permet de récupérer un tableau qui contient, dans chaque case, la taille de la dimension associée. La première case contient le nombre de lignes, la seconde case contient le nombre de colonnes, etc...
— La fonction **mxGetNumberOfElements** permet de récupérer le nombre d'éléments au total dans le tableau.

Questions

Que se passe-t-il si vous passez un simple double en paramètre de la fonction **matrice** ? Que produit l'affichage ? Comment séparer le cas où un paramètre est un double du cas où un paramètre est un tableau ?

Pour afficher les éléments du tableau, on peut faire une boucle simple et afficher chaque élément les uns après les autres ainsi :

```
#include <stdio.h>
#include "mex.h"
```

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
	int i, nb_elem;
	double max;
	double* e;

	e = mxGetPr(prhs[0]);
	nb_elem = mxGetNumberOfElements(prhs[0]);

	max = e[0];
	for(i=1; i<nb_elem; i++)
	{
		if(e[i]>max)
			max = e[i];
	}

	printf("%f\n", max);
}
```

On peut aussi, si c'est une matrice, faire une différenciation en lignes et colonnes, et afficher les éléments à l'aide d'une double boucle :

```
#include <stdio.h>
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
	int i, j, nb_lig, nb_col;
	const int *size;
	double* e;

	e = mxGetPr(prhs[0]);
	size = mxGetDimensions(prhs[0]);
	nb_lig = size[0];
	nb_col = size[1];

	for(j=0; j<nb_col; j++)
	{
		for(i=0; i<nb_lig; i++)
		{
			printf("%f ", e[j*nb_lig + i]);
		}
		printf("\n");
	}
}
```

Ce dernier code permet d'afficher les éléments de la matrice sous la forme ligne/colonne. Quelle que soit la méthode, on peut noter que la fonction **mxGetPr** permet de récupérer, dans un tableau de double (de type **double ***), l'ensemble des valeurs de la matrice, afin de les utiliser dans le programme C.

Questions

1. Le dernier code proposé affiche la matrice transposée. Comment le modifier afin d'afficher la matrice comme le ferait la fonction **disp** ?
2. Ecrivez un script, entièrement en Matlab et sans aucune vectorisation, qui calcule l'élément maximum d'un tableau à l'aide d'une boucle for. Testez-le sur une matrice de 100000 nombres choisis au hasard entre 0 et un million. Mesurez son temps d'exécution.
3. Ecrivez une fonction C Mex qui réalise la même opération : affiche l'élément maximum d'un tableau à l'aide d'une boucle for. Testez-la sur la même matrice que précédemment, et mesurez

son temps d'exécution. Quelle méthode est la plus rapide ?

4. Trouvez l'élément maximum du tableau avec un code vectorisé, qui n'utilise plus de boucle for. Testez votre code sur la même matrice que précédemment, et mesurer son temps d'exécution. Quelle méthode est la plus rapide ?

2.3 Gérer les sorties dans le programme

Connaître le nombre de sorties d'un programme

Pour connaître le nombre de sorties d'un programme, il suffit d'examiner la variable **nlhs**, qui spécifie le nombre de paramètres en sortie attendus lors de l'appel de la fonction. Essayez ce code C Mex, à copier dans le fichier **outputtest.c** :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
       printf("%d\n", nlhs);
}
```

Après l'avoir compilé, appelez-le avec différents paramètres de sortie :

```
>> [a b c] = outputtest()
3

>> [a b c d e] = outputtest()
5
```

Ce code permet de récupérer le nombre de paramètres en sortie attendus par l'utilisateur. Vous verrez s'afficher un message d'erreur en rouge : ce dernier vous informe que votre fonction n'a pas attribué de valeur aux paramètres de sortie (ce qui est normal).

Envoyer en sortie des valeurs de type double

Le tableau **plhs** permettra de renvoyer les éléments de sortie du programme vers Matlab. Il devra contenir **plhs** cases, chacune contenant une variable compatible avec Matlab. Nous allons voir comment construire une variable double compatible avec Matlab à l'aide de la fonction **mxCreateDoubleScalar** :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
       double a1, a2, r;

       if(nrhs != 2)
       {
              mexErrMsgTxt("Seulement deux parametres d'entree svp.");
       }

       if(nlhs != 1)
       {
              mexErrMsgTxt("Seulement un parametre en sortie svp.");
       }

       a1=mxGetScalar(prhs[0]);
       a2=mxGetScalar(prhs[1]);

       r = a1*a2;

       plhs[0] = mxCreateDoubleScalar(r);
}
```

La fonction **mxCreateDoubleScalar** permet de convertir une valeur double d'un code C, en une valeur double de Matlab. Elle est à utiliser à la toute fin, lorsqu'on remplit le tableau **plhs** des paramètres de sortie de la fonction. Chaque case du tableau **plhs** doit être remplie avec un des paramètres de sortie de la fonction ; ce tableau contient autant de cases que de paramètres de sortie demandés lors de l'appel de la fonction (**nlhs** cases).

Si l'on écrit ce code dans le fichier **testout.c**, et qu'on le compile dans Matlab avec Mex, on pourra appeler ce code avec :

```
a = testout(3, 4);
```

Remarquez l'utilisation de la fonction **mexErrMsgTxt**, propre à Matlab Mex, qui permet d'interrompre la fonction et d'afficher une erreur qui sera transmise à Matlab.

Astuce : si vous avez du mal à vous rappeler lequel de **nrhs** ou **nlhs** contrôle les variables d'entrée, il suffit de se souvenir de la chose suivante. Le "r" dans **nrhs** signifie *right* (droite) : lorsqu'on appelle une fonction en matlab, ce sont les variables d'entrée que l'on écrit à droite du nom de la fonction. Le "l" dans **nlhs** signifie *left* (gauche) : lorsqu'on appelle une fonction en matlab, ce sont les variables de sortie que l'on écrit à gauche du nom de la fonction.

Questions

1. Ecrivez une fonction C Mex qui prend autant de paramètres de sortie que d'entrée, et calcule, dans chaque variable de sortie, la somme partielle des paramètres d'entrée. Le premier paramètre de sortie sera égal au premier paramètre d'entrée, le second paramètre de sortie sera égal à la somme des deux premiers paramètres d'entrée, le troisième paramètre de sortie sera égal à la somme des trois premiers paramètres d'entrée, etc. La fonction doit prendre autant de paramètres qu'on le souhaite.
2. Ecrivez une fonction C Mex qui prend en paramètre un tableau de double, et produit un ou trois paramètres en sortie. Si l'utilisateur appelle la fonction avec un seul paramètre de sortie, alors on y calcule la moyenne des éléments du tableau. Si l'utilisateur appelle la fonction avec trois paramètres, on y calcule la valeur moyenne, la variance ainsi que la valeur maximale du tableau.

   Votre fonction donne-t-elle la même valeur de variance que la fonction **var** ?

   Attention : vous n'avez le droit d'utiliser qu'une seule boucle dans votre code.

   La fonction **ne donne pas** la même valeur de variance que la fonction **var**. Dans la documentation de la fonction, on peut lire *var normalizes V by N – 1*, ce qui signifie que la somme des écarts à la moyenne au carré est divisé par le nombre d'éléments moins 1. Ceci permet, en statistique, d'obtenir un estimateur sans biais de la variance.

   Donc, en bref, Matlab ne calcule pas ce à quoi les élèves s'attendent. Pour faire le calcule "classique" de variance d'un vecteur v, on fera **var(v,1)**.

Envoyer en sortie des valeurs de type matrice

La dernière partie de ce TP consistera à récupérer en sortie des matrices allouées par notre programme. En C Mex, pour construire un tableau, on utilisera la fonction **mxCreateDoubleMatrix**, qui permet de construire une matrice matlabd'une certaine taille. Par exemple, ce code permet de construire une matrice dont la taille est donnée par les paramètres d'entrée, et initialiser chaque case avec son numéro d'ordre :

```
#include <stdio.h>
#include "mex.h" /* On doit toujours inclure ce fichier */

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double r=0.0;
    int i, hauteur, largeur;
    mxArray *S;
    double *e;

    if(nrhs!=2 || nlhs!=1)
    {
        mexErrMsgTxt("Un parametre en sortie, et deux prametres en entree");
```

```
    }

    hauteur = (int)mxGetScalar(prhs[0]);
    largeur = (int)mxGetScalar(prhs[1]);

    //On construit une matrice :
    S = mxCreateDoubleMatrix(hauteur, largeur, mxREAL);
    //On recupere le tableau de double de la matrice
    e = mxGetPr(S);

    //Et on initialise chaque case de cette matrice avec son numero d'ordre
    for (i=0; i<largeur*hauteur; i++)
    {
        e[i] = i;
    }

    //Et on ecrit cette matrice dans les parametres de sortie
    plhs[0] = S;
}
```

On remarquera ici plusieurs éléments intéressants :

- On utilise la fonction **mxCreateDoubleMatrix** afin de construire une matrice Matlab. Cette fonction prend en premier paramètre la hauteur de la matrice, puis sa largeur. Avec cette fonction, les cases de la matrice de sortie sont toutes initialisées à 0.
- Une fois la matrice construite, on utilise la fonction **mxGetPr**, que l'on a vue auparavant, afin de récupérer le tableau de double de la matrice. Ceci nous permet ensuite de pouvoir manipuler le tableau de double dans le code C Mex.
- On récupère largeur et hauteur, passés en paramètres, à l'aide de la fonction **mxGetScalar** : cette fonction retournant une valeur réelle, il est nécessaire de la convertir en entier avec le mot clef **int**.
- Enfin, on écrit la matrice construite auparavant dans le tableau **plhs** contenant les paramètres de sortie. Ceci nous permettra de récupérer, une fois la fonction terminée, la matrice dans le paramètre de sortie sous Matlab.

Si l'on écrit ce code dans le fichier **testoutmatrice.c**, que l'on compile ensuite avec **mex**, et que l'on exécute le code, on obtient ceci :

```
>> M=outputtest(3, 4)

M =

     0 3 6 9
     1 4 7 10
     2 5 8 11
```

On obtient bien, dans le paramètre de sortie, une matrice avec les dimensions souhaitées.

Questions

1. Ecrivez un code C Mex qui prend en paramètre une matrice, et produit en sortie la matrice transposée. Ce code est-il plus rapide que l'opération de transposée sous Matlab (l'apostrophe) ?
2. Ecrivez un code qui prend en paramètre une matrice, ainsi qu'un second paramètre entier facultatif. S'il n'y a pas de second paramètre, le programme calcule le produit de tous les éléments du tableau. S'il y a un second paramètre, ce dernier doit nécessairement valoir 1 ou 2. Dans ce cas, le programme calcule le produit de tous les éléments de la matrice sur une même ligne (si le paramètre vaut 1) ou sur une même colonne (si le paramètre vaut 2). On produit alors en sortie un vecteur colonne (qui contient le produit de chaque ligne) ou un vecteur colonne (qui contient le produit de chaque colonne).