EPREUVE SPECIFIQUE - FILIERE PSI

INFORMATIQUE POUR TOUS

SUJET 0

VERSION CORRIGEE POUR LA CLASSE DE PC DU LYCEE CASSIN

Les calculatrices sont autorisées

Le sujet comporte 11 pages dont :
– 11 pages de texte de présentation et énoncé du sujet.

Toute documentation autre que celle fournie est interdite.

REMARQUES PRELIMINAIRES

Il est conseillé d'utiliser des feuilles de papier brouillon afin de mettre au point les développements mathématiques, schémas, graphes et courbes, avant de les recopier au propre sur la copie à rendre.

Il est demandé au candidat de bien vouloir inscrire les résultats et les développements nécessaires aux différentes questions sur sa copie, **en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions.** Les résultats attendus seront obligatoirement entourés.

Simulation des vibrations en automobile

I Introduction

Le niveau de bruit qui règne dans l'habitacle d'une voiture influe considérablement sur son confort. Maîtriser les nuisances sonores dans l'habitacle est un problème extrêmement complexe. En effet, les sources de vibrations, et donc de bruits, sont multiples sur une voiture en mouvement. Si le moteur constitue à lui seul une source importante de bruits de par les phénomènes de combustion et les mouvements des nombreuses pièces mobiles qu'il comporte, il est loin d'être le seul. Des bruits aérodynamiques, liés au frottement de l'air sur la carrosserie, apparaissent dès que la vitesse s'élève. Les pneus sont aussi à l'origine de vibrations et de bruits de roulement. Enfin, la carrosserie elle-même peut adopter des états vibratoires issus de phénomènes tant aérodynamiques que mécaniques provenant du train roulant et transmis par les suspensions.

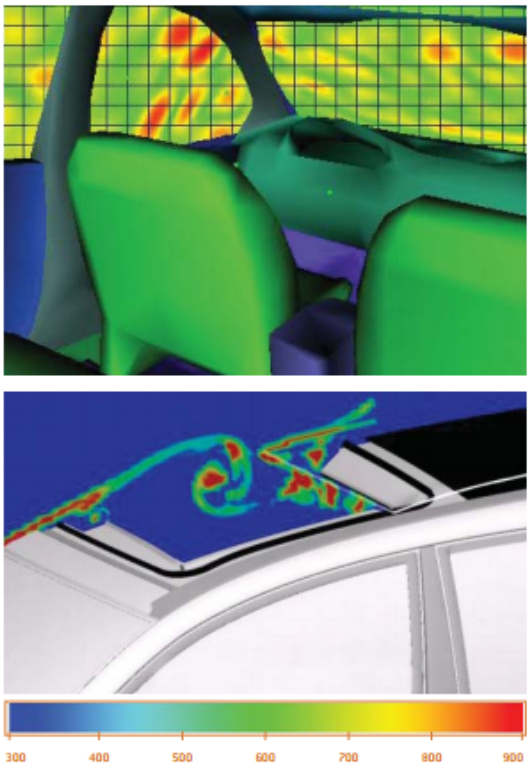


Figure 1 – En bas, simulation de l'écoulement de l'air au-dessus d'un toit ouvrant (Mégane). En haut, champ de vitesse sur les vitrages d'une Laguna soumise à un écoulement

La maîtrise des états vibratoires de la carrosserie passe par la modélisation numérique associée à des mesures sur prototype. Certains problèmes peuvent alors être résolus par ajout de renforts sur la caisse, de façon à accroître sa raideur, ou de dispositifs permettant d'introduire un amortissement structurel. Pour cela, il est nécessaire de savoir identifier très précisément leur origine. La modélisation numérique de la carrosserie permet de simuler ses états vibratoires en les amplifiant jusqu'à déterminer très précisément les points où il est possible d'agir et d'évaluer l'efficacité de la solution envisagée.

Objectif

L'objectif du sujet est de mettre en oeuvre une démarche de résolution d'un problème vibratoire d'une carrosserie simplifiée. On suppose que cette carrosserie est constituée d'un ensemble d'éléments massiques en liaison les uns aux autres. Sous une sollicitation cyclique donnée de durée T, on cherchera à déterminer les zones où les déplacements sont les plus importants et on mettra en place les fonctions permettant d'analyser l'influence des liaisons et matériaux utilisés pour limiter ces vibrations.

II Modélisation et problème à résoudre

On considère donc que la structure étudiée est constituée de n éléments de masse m_i (i variant de 1 à n) reliés par des liaisons visco-élastiques de raideur k_i et d'amortissement c_i (voir figure 2). La structure est supposée unidimensionnelle de longueur L.

Le nombre d'éléments peut être de l'ordre de quelques milliers dans une approche unidimensionnelle. Des approches plus complexes (par éléments finis) sont basées sur les mêmes principes (mais en trois dimensions notamment ce qui nécessite beaucoup plus de performances informatiques).

Le déplacement au cours du temps de l'élément i autour de sa position d'équilibre est noté $u_i(t)$. Une force $f_n(t)$ est appliquée sur l'élément n uniquement. L'extrémité gauche de la structure est bloquée. Les effets de la pesanteur sont négligés.

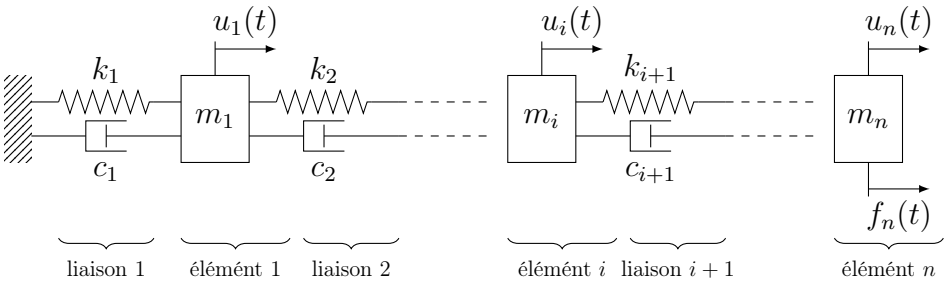


Figure 2 – Modélisation de la poutre par un ensemble de masses-ressorts-amortisseurs.

Le principe fondamental de la dynamique appliqué à un élément i permet d'aboutir à l'équation suivante pour $i = 2$ jusqu'à $n - 1$:

$$m_i \frac{\mathrm{d}^2 u_i(t)}{\mathrm{d}t} = -k_i \left(u_i(t) - u_{i-1}(t)\right) - k_{i+1} \left(u_i(t) - u_{i+1}(t)\right) - c_i \left(\frac{\mathrm{d}u_i(t)}{\mathrm{d}t} - \frac{\mathrm{d}u_{i-1}(t)}{\mathrm{d}t}\right) - c_{i+1}\left(\frac{\mathrm{d}u_i(t)}{\mathrm{d}t} - \frac{\mathrm{d}u_{i+1}(t)}{\mathrm{d}t}\right) \quad (1)$$

Les équations des éléments situés aux extrémités sont les suivantes

$$m_1 \frac{\mathrm{d}^2 u_1(t)}{\mathrm{d}t} = -(k_1 + k_2) u_1(t) + k_2 u_2(t) - (c_1 + c_2) \frac{\mathrm{d}u_1(t)}{\mathrm{d}t} + c_2 \frac{\mathrm{d}u_2(t)}{\mathrm{d}t} \quad (2)$$

$$m_n \frac{\mathrm{d}^2 u_n(t)}{\mathrm{d}t} = -k_n \left(u_n(t) - u_{n-1}(t)\right) - c_n \left(\frac{\mathrm{d}u_n(t)}{\mathrm{d}t} - \frac{\mathrm{d}u_{n-1}(t)}{\mathrm{d}t}\right) + f_n(t) \quad (3)$$

Dans toute la suite de l'étude, on imposera $f_n(t) = f_{\max} \sin(\omega t)$.

Un calcul est défini par :

- une géométrie : longueur L de la structure, nombre d'éléments et de liaisons n, masse des éléments stockée dans un tableau m, raideur des liaisons stockée dans un tableau k, amortissement des liaisons stocké dans un tableau c.
- des paramètres de simulation : durée de la simulation T, pulsation de l'excitation ω, amplitude de l'excitation f_{max}, pas de temps dt (distance entre deux instants de calcul), et le nombre de pas de temps $npts$.

III Structure générale du programme

Le programme est composé d'une fonction principale `main()` dont l'ébauche est donnée ci-dessous.

En python :

```
def main():
 ## FONCTIONS A APPELER ICI
 ##Définition des paramètres du calcul
    [T,omega,fmax,dt,nbpts]=parametres_calcul()
    ##Sauvegarde dans la base de données
    sauve(L,n,m,k,c,T,omega,fmax,dt,nbpts)
    ##Création des opérateurs
    [M,K,C]=creation_operateur(n,m,k,c)
    ##resolution du problème
    X=calcul(n,M,K,C)
    #analyse des résultats
    posttraitement(X,L,n,c)
```

En Scilab :

```
function main()
    // FONCTION A APPELER ICI
    //Définition des paramètres du calcul
    [T,omega,fmax,dt,nbpts]=parametres_calcul()
    //Sauvegarde dans la base de données
    sauve(L,n,m,k,c,T,omega,fmax,dt,nbpts)
    //Création des opérateurs
    [M,K,C]=creation_operateur(n,m,k,c)
    //resolution du problème
    X=calcul(n,M,K,C,npts,dt,fmax,omega)
    //analyse des résultats
    posttraitement(X,L,n,c)
endfunction
```

Les variables utilisées dans cette fonction seront définies dans la suite.

La fonction principale `main()` fait également appel au tout début (zone commentée) à la fonction `geometrie` définie ci-dessous, qui permet de définir les différents paramètres de la géométrie.

En python :

```
def geometrie(mode_md):
    #Définition manuelle
    if mode_md == 0:
        L_local = float(input('Longueur de la poutre : '))
        n_local = int(input('Nombre de masses représentant de la poutre : '))
        m_local = []
        for i in range(n_local):
            m_local.append(float(input('Masse de l''élement '+str(i)+' :')))
        k_local = []
        c_local = []
        for i in range(n_local):
            k_local.append(float(input('Raideur de la liaison '+str(i)+' :')))
            c_local.append(float(input('Amortissement de la liaison'+str(i)+' :')))
    #Définition dans un fichier
    elif mode_md == 1:
        nom_fic = input('Donner le nom du fichier : ')
        [L_local,n_local,m_local,k_local,c_local] = lire_fichier(nom_fic)
    #Définition dans la base de données
    elif mode_md == 2:
        requete1 = "" #A DEFINIR
        liste_calcul = interroge_bdd(requete1)
        print(liste_calcul)
        id_calcul = int(input('Choix du numéro du calcul : '))
        #ZONE A COMPLETER
        #
        #
    return [L_local,n_local,m_local,k_local,c_local]
```

En Scilab :

```
function [L_local,n_local,m_local,k_local,c_local]=geometrie(mode_md)
    //Définition manuelle
    if mode_md == 0 then
        L_local = input('Longueur de la poutre : ')
        n_local = input('Nombre de masses représentant de la poutre : ')
        m_local = []
        for i = 1:n_local
            m_local($+1) = input('Masse de l''élement '+string(i)+ ' : ')
        end
        k_local = []
        c_local = []
        for i = 1:n_local
            k_local($+1) = input('Raideur de la liaison '+string(i)+ ' : ')
            c_local($+1) = input('Amortissement de la liaison '+string(i)+ ' : ')
        end
    //Définition dans un fichier
    elseif mode_md == 1 then
        nom_fic = input('Donner le nom du fichier : ')
        [L_local,n_local,m_local,k_local,c_local] = lire_fichier(nom_fic)
    //Définition dans la base de données
    elseif mode_md == 2 then
```

```
        requete1 = ""//A DEFINIR
        liste_calcul = interroge_bdd(requete1)
        disp(liste_calcul)
        id_calcul = input('Choix du numéro du calcul : ')
        //ZONE A COMPLETER
        //
        //
    end
endfunction
```

Q1. Compléter la fonction principale `main` permettant de renseigner une géométrie en mode manuel.

Lors de l'éxécution de de la fonction `geometrie`, l'utilisateur tape la séquence suivante : 0.5, 3, 0.1, 0.2, 0.1, 20000, 1000, 20000, 10, 1000, 10. L'utilisateur aura appuyé sur Entrée après chaque valeur.

Q2. Donner la valeur de `L`, `n` et des listes `m`, `k` et `c` à la sortie de la fonction `geometrie`.

IV Étude de la fonction `geometrie()`

IV.1 Initialisation par lecture d'un fichier

Dans le cas d'une définition complexe avec plusieurs centaines d'éléments, rentrer les données à la main est trop compliqué. Il est possible de lire la géométrie dans un fichier texte.

Le stockage dans le fichier texte contient les différentes valeurs numériques sous forme d'une unique colonne. Celle-ci contient :

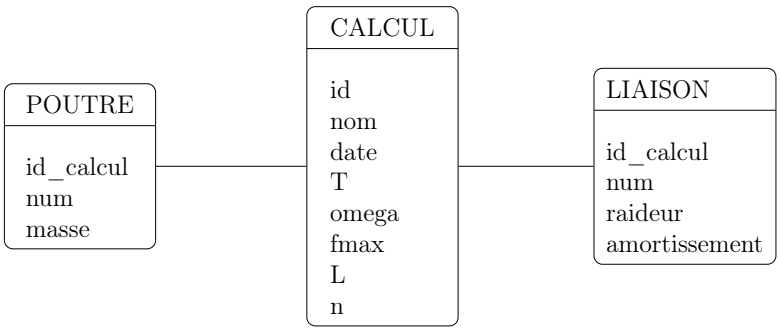
- la longueur L sur la première ligne,
- le nombre d'éléments n sur la deuxième,
- la masse m_1,
- la masse m_2,
- ...
- la masse m_n,
- la raideur k_1,
- la raideur k_2,
- ...
- la raideur k_n,
- l'amortissement c_1,
- l'amortissement c_2,
- ...
- l'amortissement c_n,

Q3. En utilisant la documentation fournie en annexe, proposer une écriture de la fonction `lire_fichier(nom_fic)` utilisée dans le mode 1 de la fonction `geometrie`.

IV.2 Initialisation à partir de la base de données

L'ensemble des calculs exécutés sont stockés dans une base de données, un calcul peut être initialisé avec la géométrie d'un calcul précédent.

La base de données est composée de trois tables dont la structure est la suivante :



Tous les attributs sont de type numérique sauf le nom du calcul qui est une chaîne de caractères.

La table CALCUL contient un attribut id qui est une clé primaire, un nom de calcul donné par l'utilisateur, la date à laquelle le calcul a été effectué puis les paramètres de la simulation T, ω, f_{max}, L et le nombre d'éléments du modèle n.

La table POUTRE contient pour une géométrie l'identifiant correspondant au calcul. Les éléments de la géométrie sont caractérisés par leur numéro num et leur masse associée $masse$.

La table LIAISON contient pour une géométrie l'identifiant correspondant au calcul id_calcul. Les liaisons de la géométrie sont caractérisées par leur numéro num, leur raideur et amortissement associé.

Le tableau ci-dessous montre un seul calcul correspondant à la géométrie entrée manuellement à la question Q2.

CALCUL							
id	nom	date	T	omega	fmax	L	n
102	calcul1	18-02-2015	0.1	2000	10000	0.5	3

POUTRE		
id_calcul	num_elem	masse
102	1	0.2
102	2	0.1
102	3	0.2

LIAISON			
id_calcul	num_liaison	raideur	amortissement
102	1	20000	10
102	2	1000	1000
102	3	20000	10

On suppose que la fonction `resultat=interroge_bdd(requete)` permet de se connecter à la base de données, d'envoyer la requête et de retourner la réponse sous la forme d'un tableau `resultat` dont les colonnes correspondront aux colonnes sélectionnées par la commande SELECT et les lignes correspondent aux différents résultats de la requête.

Q4. Définir la chaîne de caractère `requete1` (voir fonction `geometrie`), permettant de renvoyer l'ensemble des noms de calcul avec leur date et leur identifiant.

L'utilisateur choisit ensuite l'identifiant du calcul dont il souhaite reprendre la géométrie, il est stocké dans la variable `id_calcul`.

Q5. Compléter la « Zone à compléter » de la fonction `geometrie` sur votre feuille, permettant de définir les différentes variables de la géometrie `L_local`, `n_local`, `m_local`, `k_local` et `c_local`.

V Simulation numérique

Objectif

Les paramètres étant définis, la deuxième partie du programme consiste à résoudre le problème spécifié par le système d'équations différentielles.

V.1 Création des matrices

Le système d'équations différentielles défini initialement peut s'écrire sous une forme matricielle :

$$M\ddot{X}(t) + C\dot{X}(t) + KX(t) = F(t) \tag{4}$$

avec $X(t) = {}^T(u_1(t), \cdots, u_n(t))$. La taille des matrices carrées est égale à $n \times n$.

Q6. En reprenant les équations 1, 2 et 3, déterminer les expressions des matrices M, C et K. Bien préciser les indices des termes non nuls.

Q7. Écrire une fonction `[M,K,C]=creation_operateur(n,m,k,c)` qui réalise la construction de ces différentes matrices.

V.2 Résolution

La résolution de l'équation différentielle matricielle (4) est obtenue à l'aide d'un schéma d'Euler explicite.

L'intervalle de temps [0,T] est discrétisé en pas de temps de même durée, notés dt. Ainsi, le piquet de temps t_{q+1} est donné par $t_{q+1} = dt + t_q$ soit par récurrence, $t_{q+1} = (q+1)dt$ avec $t_0 = 0$.

A l'instant t_q, le vecteur déplacement est noté X_q, la vitesse V_q et l'accélération A_q.

Q8. En appliquant le schéma d'Euler explicite deux fois, exprimer V_q en fonction de dt, X_q et X_{q-1}, puis A_q en fonction de dt, X_q, X_{q-1} et X_{q-2}. Écrire le système matriciel à l'instant t_q et le mettre sous la forme $HX_q = G_q$ où vous exprimerez H en fonction de M, K, C et le pas de temps dt et G_q en fonction de M, K, C, X_{q-1}, X_{q-2}, dt et F_q le vecteur force calculé à l'instant t_q.

La matrice H est calculée une fois pour toute avant les itérations sur les piquets de temps. Elle a la forme suivante :

$$H = \begin{pmatrix} H_{11} & H_{12} & 0 & 0 & 0 \\ H_{12} & H_{22} & \ddots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & H_{n-1n-1} & H_{n-1n} \\ 0 & 0 & 0 & H_{n-1n} & H_{nn} \end{pmatrix}$$

Le second membre G_q est réévalué à chaque piquet de temps puis le système est résolu par la fonction `Xq=resoud(H,Gq)`. Le vecteur de force extérieure `Fq` est calculé dans la boucle itérative du temps afin de ne pas stocker sa valeur sur tout l'intervalle de temps.

Le résultat de la simulation sera stocké dans une matrice de taille $npts \times n$, `X` telle que $X = (X_0, X_1, \cdots, X_q, \cdots)$ avec $npts$ le nombre de pas de temps et n le nombre d'éléments de structure (chaque ligne contient un vecteur déplacement pour un pas de temps donné).

Q9. Écrire une fonction `X=calcul(n,M,K,C,npts,dt,fmax,omega)` qui permette de résoudre le problème en utilisant la fonction `resoud`, qui ne sera pas détaillée ici. On choisira un pas de temps dt égal à $T/npts$.

La résolution `Xq=resoud(H,Gq)` peut être faite à l'aide d'un pivot de Gauss classique. Cependant, compte-tenu de la forme tridiagonale de la matrice H, la procédure peut être simplifiée.

Q10. Écrire une fonction `Xq=resoud(H,Gq)` basée sur la méthode du pivot de Gauss adaptée à la forme de la matrice H.

Q11. Calculer la complexité en nombre d'opérations de la fonction `resoud` en fonction de n. Comparer ce résultat à la complexité basée sur une résolution du système matriciel à l'aide d'un pivot de Gauss classique.

V.3 Évaluation du coût de stockage

En pratique, les matrices M, K, C, H et le vecteur G contiennent très peu de termes et sont stockés de manière particulière (stockage appelé "sparse"). Ceci ne constitue pas un coût de stockage important ; par contre le stockage de la matrice X peut s'avérer très lourd si le nombre de masses n est grand et le pas de temps dt est petit.

Une étude typique consiste à résoudre ce système matriciel avec les caractéristiques suivantes :

- $n = 200$,
- $T = 0{,}3\,\text{s}$,
- pas de temps de $2 \times 10^{-6}\,\text{s}$,
- stockage de la solution au format réel double précision sur toute la durée de simulation T.

Le calculateur utilisé possède 32 Go de mémoire RAM.

Q12. Déterminer la quantité de mémoire RAM en Go nécessaire pour stocker le résultat d'un calcul sur la durée T. On ne tiendra compte que de la matrice X.

VI Traitement des résultats

Objectif

Le calcul étant réalisé, la dernière étape consiste à post-traiter les résultats pour en extraire des informations telles que l'énergie dissipée dans la structure.

Les résultats du calcul sont stockés dans la matrice `X` de taille $npts \times n$ (où $npts$ est le nombre de pas de temps et n le nombre d'éléments).

VI.1 Affichage du résultat

Q13. Écrire une fonction `affiche_deplacement_pdt(q,X,L,n,ampl)` qui affiche le déplacement X_q au pas de temps t_q. L'affichage devra représenter la poutre comme sur la figure 3, la position des masses étant matérialisée par des diamants reliés entre eux (option "D" en Python

et "diamond" en Scilab) et donnée par la position initiale plus le déplacement amplifié d'un facteur *ampl*.

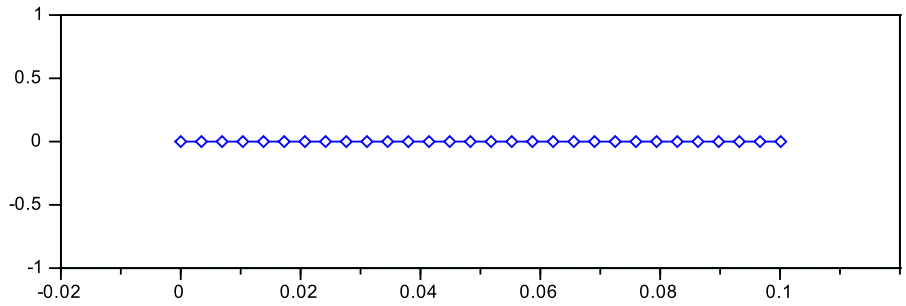


Figure 3 – Représentation graphique de la poutre.

VI.2 Détermination du pas de temps et du lieu du déplacement maximal

Q14. Écrire une fonction `lieu_temps_depl_max(X)` qui renvoie le numéro du nœud, le pas de temps et la valeur du déplacement maximal en valeur absolue.

VI.3 Détermination de l'énergie dissipée

La puissance dissipée dans la poutre est donnée par la relation :

$$P_{\text{diss}}(t) = \sum_{i=2}^{n} c_i(\dot{u}_i(t) - \dot{u}_{i-1}(t))^2$$

L'énergie imposée est donnée par la relation :

$$E_{\text{diss}}(t) = \int_0^t P_{\text{diss}}(\tau)\mathrm{d}\tau$$

Q15. Écrire une fonction `Ediss=calcul_energie(X,c,dt)`qui calcule l'énergie dissipée, puis trace son évolution en fonction du temps et qui renvoie `Ediss`. Vous préciserez quelle méthode d'intégration vous utiliserez.

Les différentes fonctions mises en place permettraient de tester différentes structures et analyser l'influence de celles-ci sur les vibrations.

VII Annexes

VII.1 Extraits de documentations pour lire dans un fichier

En Python

```
f=open(name[, mode[, buffering]])
```

Open a file, returning an File type object *f*. If the file cannot be opened, IOError is raised. *name* is the file name to be opened, and mode is a string indicating how the file is to be opened. The most commonly-used values of mode are 'r' for reading, 'w' for writing (truncating the file if it already exists), and 'a' for appending.

```
f.readline()
```

Reads a single line from the file; a newline character (
n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn't end in a newline. If f.readline() returns an empty string, the end of the file has been reached, while a blank line is represented by
n, a string containing only a single newline.

```
f.close()
```

When you're done with a file, call this function to close it and free up any system resources taken up by the open file. After calling f.close(), attempts to use the file object will automatically fail.

En Scilab

```
fd = mopen(file [, mode ])
```

opens a file in Scilab fd, a scalar : a file descriptor (it's a positive integer). file : a character string containing the path of the file to open. mode : a character string specifying the access mode requested for the file. The parameter can have one of the following values : r : opens for reading (default). The file must exist, otherwise it fails. w : opens for writing. If the file exists, its contents are destroyed. a : opens for appending. It creates the file if it does not exist.

```
mclose(fd)
```

closes an opened file. fd, a scalar : the fd parameter returned by the function mopen is used as a file descriptor.

```
txt = mgetl(file\_desc [,m])
```

mgetl function allows to read a lines from an text file. file_desc, a character string giving the file name or an integer giving a logical unit returned by mopen. m, an integer scalar : a number of lines to read. Default value is -1. txt, a column vector of strings. If m is omitted or is -1 all lines till end of file occurs are read. If m is given mgetl tries to read exactly m lines.

VII.2 Rappels des syntaxes en Python et Scilab

Remarque : Sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import * Les indices de ces tableaux commencent à 0.

Remarque : Sous Scilba, les indices des tableaux commencent à 1.

	Python	Scilab
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)	`v=[1, 2, 3]` ou `[1 2 3]`
accéder à un élément	`v[0]` renvoie 1	`v(1)` renvoie 1
ajouter un élément	`L.append(5)` uniquement sur les listes	`v($+1) = 5`
tableau à deux dimensions (matrice) :	`M=array(([1,2,3],[3,4,5]))`	`M=[1,2,3;3,4,5]`
accéder à un élément	`M[1,2]` ou `M[1][2]` donne 5	`M(2,3)` donne 5
Extraire une portion de tableau (2 premières colonnes)	`M[:,0:2]`	`M(:,1:2)`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3),float)`	`zeros(2,3)`
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	`arange(0,10.1,0.1)`	`[0:0.1:10]`
Définir une chaîne de caractères	`mot="Python et Scilab"`	`mot="Python et Scilab"`
Taille d'une chaine	`len(mot)`	`length(mot)`
Extraire des caractères	`mot[2:7]`	`part(mot,[11:16])`

Boucle For	`for i in range(10):`
	`  print(i)`
		`for i=1:10`
		`  disp(i);`
		`end`

Condition If	`if (i>3):`
	`  print(i)`
	`else`
	`  print("hello")`
		`if (i>3) then`
		`  disp(i)`
		`else`
		`  disp("hello")`
		`end`

Définir une fonction qui possède un argument et renvoie 2 résultats	`def fonction(param):`
	`  res=param`
	`  res2=param*param`
	`  return res1,res2`
		`function [res,res2]=fonction(`
		`  param)`
		`  res=param`
		`  res2=param*param`
		`endfunction`

Effectuer le produit de deux matrices	`dot(M,P)` (dans numpy)	

Fin de l'énoncé