Introduction à Matlab

Université de Fada N'Gourma-S3 Génie Civil

Yacouba SIMPORE

Associate professor at Université de Fada N'Gourma;

Introduction

MATLAB est une abréviation de Matrix LABoratory, écrit à l'origine, en Fortran, par C. Moler. MATLAB était destiné à faciliter l'accès au logiciel matriciel développé dans les projets LINPACK et EISPACK. La version actuelle, écrite en C par the MathWorks Inc., existe en version professionnelle et en version étudiant. Sa disponibilité est assurée sur plusieurs plateformes : Sun, Bull, HP, IBM, compatibles PC (DOS, Unix ou Windows), Macintoch, iMac et plusieurs machines parallèles.

MATLAB est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant calcul numérique et visualisation. C'est un environnement performant, ouvert et programmable qui permet de remarquables gains de productivité et de créativité. MATLAB est un environnement complet, ouvert et extensible pour le calcul et la visualisation. Il dispose de plusieurs centaines (voire milliers, selon les versions et les modules optionnels autour du noyau Matlab) de fonctions mathématiques, scientifiques et techniques.

L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numérique et symbolique de facon fiable et rapide. Grace aux fonctions graphiques de MATLAB, il devient très facile de modifier interactivement les différents paramètres des graphiques pour les adapter selon nos souhaits. L'approche ouverte de MATLAB permet de construire un outil sur mesure. On peut inspecter le code source et les algorithmes des bibliothèques de fonctions (Toolboxes), modifier des fonctions existantes et ajouter d'autres.

MATLAB possède son propre langage, intuitif et naturel qui permet des gains de temps de CPU spectaculaires par rapport à des langages comme le c, le TurboPascal et le Fortran. Avec MATLAB, on peut faire des liaisons de facon dynamique, à des programmes C ou Fortran, échanger des données avec d'autres applications (via la DDE Dynamic Data Exchange : MATLAB serveur ou client) ou utiliser MATLAB comme moteur d'analyse et de visualisation. MATLAB comprend aussi un ensemble d'outils spécifiques à des domaines, appelés Toolboxes (ou Boites à Outils). Indispensables à la plupart des utilisateurs, les Boites à Outils sont des collections de fonctions qui étendent l'environnement MATLAB pour résoudre des catégories spécifiques de problèmes.

Chapitre Introduction à l'environnement MATLAB MATLAB permet le travail interactif soit en mode commande, soit en mode programmation; tout en ayant toujours la possibilité de faire des visualisations graphiques. Considéré comme l'un des meilleurs langages de programmation, MATLAB possède les particularités suivantes par rapport à ces langages: la programmation facile, la continuité parmi les valeurs entières, réelles et complexes, la gamme étendue des nombres et leur précision, la bibliothèque mathématique très compréhensive, l'outil graphique qui inclut les fonctions d'interface graphique et les utilitaires, la possibilité de liaison avec les autres langages classiques de programmation (c ou Fortran).

La bibliothèque des fonctions mathématiques dans MATLAB donne des analyses mathématiques très simples. En effet, l'utilisateur peut exécuter dans le mode commande n'importe quelle fonction mathématique se trouvant dans la bibliothèque sans avoir à recourir à la programmation. Pour l'interface graphique, des représentations scientifiques et meme artistiques des objets peuvent être créées sur l'écran en utilisant les expressions mathématiques. Les graphiques sur MATLAB sont simples et attirent l'attention des utilisateurs, vu les possibilités importantes offertes par ce logiciel.

MATLAB n'est pas le seul environnement de calcul scientifique, il existe d'autres concurrents dont les plus importants sont Maple et Mathematica. Il existe même des logiciels libres qui sont des clones de MATLAB comme Scilab et Octave. 2. L'environnement MATLAB MATLAB affiche au démarrage plusieurs fenêtres. Selon la version on peut trouver les fenêtres suivantes :

Current Folder: indique le répertoire courant ainsi que les fichiers existants.
Workspace: indique toutes les variables existantes avec leurs types et valeurs.
Command History: garde la trace de toutes les commandes entrées par l'utilisateur.
Command Window: utilisée pour formuler nos expressions et interagir avec MATLAB. C'est la fenêtre que nous utilisons tout au long de ce chapitre.
MATLAB est beaucoup plus qu'un langage de programmation. Il s'agit d'une console d'exécution (shell) permettant d'exécuter des fonctions, d'attribuer des valeurs à des variables, etc. La console MATLAB permet d'effectuer des opérations mathématiques, de manipuler des matrices et de tracer facilement des graphiques.

Le langage MATLAB n'est pas un langage compilé, à chaque appel d'un SCRIPT (ou d'une FUNCTION), le logiciel lit et exécute les programmes ligne par ligne. L'utilisateur peut grâce à l'invite MATLAB affecter des valeurs à des variables et effectuer des opérations sur ces variables. Par exemple:

Ici, il faut noter que lorsque l'utilisateur ne fixe pas de variable de sortie, MATLAB place le résultat d'une opération dans ans. Il est toujours possible de connaitre les variables utilisées et leur type à l'aide de la fonction whos. Par exemple, pour les manipulations précédentes:

Calcul élémentaire

MATLAB permet de créer et d'initialiser des variables. La déclaration des variables en matlab suit les règles suivantes: toutes les variables sont des matrices; pas de déclaration de type. Le type est établi automatiquement à partir des valeurs affectées à la variable.
Les types de variables utilisés par MATLAB sont les suivants : réel; complexe; chaine de caractères logique

whos Permet d'afficher toutes les variables avec indication sur leur taille, leur codage et leur type.
On peut aussi retrouver le type d'une commandes: ischar, islogical et isreal:

ischar(x) retourne 1 si x est de type chaine de caractères et 0 sinon;
islogical(x) retourne 1 si x est de type logique et 0 sinon.
isreal(x) est à utiliser avec discernement: elle retourne 1 si x est réel ou de type chaine de caractères et 0 sinon (x est complexe à partie imaginaire non nulle ou n'est pas un tableau de valeurs réelles ou de caractères).

» who pour afficher toutes les variables.
» whos pour afficher toutes les variables avec indication sur leur taille.
» size(a) pour afficher les dimensions de la matrice a.
» clear a pour effacer la variable a.
» clear all pour effacer toutes les variables.
» clc pour effacer l'ecran.

Les vecteurs

» $V = [1,2,3]$ donne $V = (1 \quad 2 \quad 3)$;

» $V = [1;2;3;4]$ donne $V^T = (1 \quad 2 \quad 3 \quad 4)$.

»$x = -1 : 1/10 : 1$ donne :

»$y = linsapce(-1, 1/10, 1)$ donne:

» $y = linspace(-1, 1)$ donne les 100 éléments par defaut.

Pour afficher des éléments d'un vecteur:

» $V = [1, 4, 5, 6, 7]$

» $V(3)$ donne $ans = 5$;

» $V(2:1:5) = (4 \quad 5 \quad 6 \quad 7)$;

» $b = V - 5$ donne $b = (-4 \quad -1 \quad 0 \quad 1 \quad 2)$;

» $c = V.^2$ donne $c = (1 \quad 16 \quad 25 \quad 36 \quad 49)$;

» $d = V' * V$ donne produit scalaire de V par V;

» $W = [c, \quad b]$ donne (1 16 25 36 49 -4 -1 0 1 2);

[] énumération d'éléments

: description d'éléments

() ensemble d'arguments

, séparateur d'arguments

; séparateur des lignes; non affichage du résultat de l'exécution d'une instruction

. force l'opérateur à s'appliquer sur chaque éléments.

Les Matrices

Une matrice est un ensemble de lignes comportant toutes le même nombre de colonnes.

»$M = [1 \quad 2 \quad 4; 4 \quad 3 \quad 4; 11 \quad 0 \quad 0]$;

»$A = M.^2$,

»$B = M * A$

»ones(2,5),

» zeros(3,4),

» eye(5);

»$diag([3 \quad 4 \quad 5])$ ou diag(V);

»$[P, D] = eig(A)$; diagonalisation

»$M(1:2, 2:3)$ extraction de matrice dans la matrice M.

Fonctions

» $\exp(x)$: exponentielle de x
» $\log(x)$: logarithme n´eperien de x
»$\log 10(x)$: logarithme en base 10 de x
» x^n : x á la puissance n
»$sqrt(x)$: racine carrée de x
»$abs(x)$: valeur absolue de x
$sign(x)$: 1 si x > 0 et 0 si x < 0
$\sin(x)$: sinus de x
$\cos(x)$: cosinus de x
$\tan(x)$: tangente de x
$asin(x)$: sinus inverse de x (arcsin de x)
$\sinh(x)$: sinus hyperbolique de x
$asinh(x)$: sinus hyperbolique inverse de x

Evidemment les fonction hyperboliques et inverses sont aussi valables pour le cosinus et la tangente On pourra ´egalement utiliser les fonctions d'arrondis :
$round(x)$: entier le plus proche de x
$floor(x)$: arrondi par défaut de x
$ceil(x)$: arrondi par excès de x
ainsi que les fonctions d'arithm´etiques :
$rem(m,n)$: reste de la division enti'ere de m par n
$lcm(m,n)$: plus petit commun multiple de m et n
$\gcd(m,n)$:plus grand commun diviseur de m et n
$factor(n)$: décomposition en facteurs premiers de n
Enfin lorsque l'on travaille avec des complexe on pourra utiliser :
$conj(z)$: conjugué de z
$abs(z)$: module de z
$angle(z)$: argument de z
$real(z)$: partie réelle de z
$imag(z)$: partie imaginaire de z.

Quelques fonctions utiles

Nous présentons dans ce paragraphe un ensemble de fonctions usuelles liées à l'utilisation des matrices.
« size(M) » renvoie les dimensions de la matrice.
« max(M) » renvoie un vecteur-ligne contenant les valeurs maximales associées à chaque colonne.
« min(M) » renvoie un vecteur-ligne contenant les valeurs minimales associées à chaque colonne.
« rank(M) » renvoie le rang de la matrice.
« det(M) » renvoie le déterminant de la matrice.
« diag(M) » extrait la diagonale de la matrice.
« triu(M) » extrait la matrice-triangle supérieure de M. tril donne la matrice-triangle inférieure
« eig(M), det(A), inv(A) et poly(A) » renvoie respectivement un vecteur contenant les valeurs propres de la matrice; le determinant de la matrice A; l'inverse de la matrice A; un vecteur contenant les coefficients du polynome caracteristique de A.

Autres fonctions

»N;
» diag(ones(N-k,1),k)|| »diag(ones(N-k,1),-k)
»$A = [2 \quad 1 \quad -1; -1 \quad 2 \quad 3; -2 \quad 1 \quad 0]$;
» triu(A),
» tril(A),
» tril(A,-1),
» tril(A,1);
» kron(eye(N),A),
» kron(ones(N,N),A);

Graphique

Pour faire une répresentation graphique on utilise la fonction plot de Matlab:

Exemple:

»x=linspace(0,pi/2);

$y = cos(x) + x^2$;

»plot(x,y)

»plot(x,tan(x))

» hold on

» plot(x,sin(x))

» hold off

»clf:

»plot(x,cos(x),x,sin(x))

» legend('courbe de sinus','cours de cosinus')

» title('courbes de fonctions circulaire')

» plot(x,sin(x),' $r+$',x,cos(x),' $b<$')

» help plot

La fontion compass

La représentation polaire d'un ou plusieurs nombres complexes peut être obtenue avec la commande *compass*(z) où z est soit un unique nombre complexe, soit un vecteur dont les composantes sont des nombres complexes.

Exemple

$$>> z = 5 + 2 * i; compass(z);$$

La fonction $hist(x,n)$ répartit les valeurs de la liste (ou du vecteur) x en n classes et trace l'histogramme correspondant (par défaut $n = 10$). $[N,X] = hist(x,n)$ retourne dans N l'effectif de chacune des classes et dans X l'abscisse du centre de chaque classe.
La fonction $pie(x)$ dessine un diagramme des valeurs de x normalisées par

$$s = \sum_{i=1}^{n} x_i.$$

Exemple

$$a = randn(1,500);$$

$$figure(1)$$

$$hist(a)$$

$$[N,X] = hist(a);$$

$$figure(2)$$

$$pie(N)$$

Graphique en dimension 3

On utilse la fonction plot3 pour les courbes paramétrés et la fonction mesh pour les courbes de fonctions de deux variables.
La fonction meshgrid permet de faire un maillage de points entre x et y par exemple observons:
Exemple:
Répresenter la courbe paramétrée définie par:
$x(t) = cos(t) + sin(t) \quad y(t) = t + cos(t) \quad t \in [0, 4\pi]$

```
»t=linspace(0,4*pi);
»plot3(cos(t)+sin(t),t+cos(t),t)
```

Répresenter la fonction:

$f(x,y) = x^2 + y^2 \quad (x,y) \in [-\pi,\pi] \times [-1,1].$

»$x = -pi : 0.1 : pi;\ y = -1 : 0.1 : 1;$

»$[X, Y] = meshgrid(x, y);\ Z = X^2 + Y^2;$

»mesh(X,Y,Z).

Par la suite, on va voir on va voir comment MATLAB permet de représenter des surfaces définies par une relation $z = f(x, y)$ où f est une fonction continue, définie sur un domaine

$$[x_0, x_1] \times [y_0, y_1].$$

Modélisation du domaine

$$[x_0, x_1] \times [y_0, y_1].$$

- Fonction meshgrid:

Cette modélisation se fait en deux étapes:
Définition de deux subdivisions régulières:

$$x \text{ pour } [x_0, x_1] \text{ et } y \text{ pour } [y_0, y_1]$$

Construction d'une grille modélisant le domaine

$$[x_0, x_1] \times [y_0, y_1].$$

La grille est définie par deux matrices xx et yy résultant de

$$[xx, yy] = meshgrid(x, y).$$

Précisément:

$$xx(l, k) = x(k) \text{ et } yy(l, k) = y(l) \text{ pour tout } k, (1 <= k <= \text{length}(x)$$

et pour tout $l, (1 <= k <= \text{length}(y)$. De telle sorte que:

$$(xx(l, k), yy(l, k)) = (x(k), y(l)).$$

Il est alors possible d'évaluer les valeurs de f suivant cette grille en appliquant f au couple de tableaux xx et yy.

Tracé de la surface

-fonctions mesh et surf:
Une fois le domaine d'étude modélisé par deux tableaux xx et yy on évalue les valeurs de la fonction pour obtenir un tableau $z - f(xx, yy)$.
On dessine la surface $z = f(x, y)$ (tracée en perspective dans une plotting-box comme pour plot3) avec l'une des fonctions suivantes:
Fonction mesh:
$mesh(xx, yy, z)$ donne une représentation de la surface par un maillage "fil de fer".
Fonction surf: $surf(xx, yy, z)$ donne une représentation où les mailles sont colorées.
Comme pour les courbes de l'espace, la commande rotate3d on permet de déplacer la plottingbox à l'aide de la souris.

Exemple: courbe $\sin(xy)$ sur $[-\pi,\pi]\times[-\pi,\pi]$

$$x = -7.5 : .5 : 7.5; y = x$$

$$[X, Y] = meshgrid(x, y);$$

$$Z = \sin(X. * Y);$$

$$figure\ ;\ mesh(X, Y, Z)\ ;\ title('mesh')$$

$$figure\ ;\ surf(X, Y, Z)\ ;\ title('surf')$$

Surfaces et courbes de niveau

Comme pour le tracé d'une surface, on commence par modéliser le domaine d'étude par deux tableaux xx et yy, puis on évalue les valeurs de la fonction et on obtient une matrice $z = f(xx, yy)$.
Plusieurs fonctions permettent alors de dessiner les surfaces de niveau de f:
contour, **contour3** et **pcolor**.
Surfaces et courbes de niveau
La fonction contour:
$contour(xx, yy, z, n)$ détermine n surfaces de niveau (10 par défaut) et les projette sur le plan xoy.
u lieu de spécifier le nombre de niveaux, il est possible d'indiquer leur valeurs sous forme d'une liste $[z0 : pas : z1]$, en particulier pour obtenir la surface correspondant à un niveau donné $z0$ on utilisera $contour(xx, yy, z, [z0])$.
En niveaux de gris (cf. colormap), la couleur des courbes de niveau est d'autant plus claire que la valeur du niveau l'est.
Il est possible de fixer la couleur des courbes de niveau en utilisant un caractère (cf. codes des couleurs vus précédemment) comme dernier argument.

La fonction contour3:

Semblable à contours, contour3 détermine n surfaces de niveau et en donne une représentation en trois dimensions.

Comme pour contour, la couleur des courbes de niveau est d'autant plus claire que la valeur du niveau l'est.

La fonction pcolor $pcolor(xx, yy, z)$ génère une image plane à la même échelle que contour et dont les pixels ont une couleur qui si on utilise une échelle de gris, est d'autant plus claire que la valeur de $f(x, y)$ est grande.

Cette fonction est utilisée en conjonction avec contour.

Exemple 2:

$$[xx, yy, z] = peaks;$$

$$figure(1)\ ;\ mesh(xx, yy, z)\ ;\ title('peaks')$$

$$figure(2)\ ;\ contour3(xx, yy, z)\ ;\ title('contour3')$$

Exemple 3:

$$[xx, yy, z] = peaks;$$

$$figure(3)\ ;\ contour(xx, yy, z)\ ;\ title('contour')$$

$$figure(4)\ ;\ pcolor(xx, yy, z)$$

shading interp, % supprime la grille
hold on contour(xx,yy,z,'k'), % superpose les courbes de niveau en noir
title('Contour avec pcolor')
hold off

Si l'on souhaite afficher uniquement les valeurs de quelques lignes de niveau, on peut utiliser la commande clabel de la manière suivante :

$$>> [C, h] = contour(X, Y, Z, n)$$

$$>> clabel(C, h, 'manual')$$

On peut alors grace à la souris sélectionner les lignes de niveau pour lesquelles on souhaite afficher la valeur.

Exemple 4

$$[X, Y] = meshgrid(-2 : 0.2 : 2, -2 : 0.2 : 2);$$

$$Z = (X - 1).^2 + 10 * (X.^2 - Y).^2;$$

$$[C, h] = contour(X, Y, Z, 30);$$

$$clabel(C, h, 'manual')$$

Introduction à la programmation

Les fichiers SCRIPT et FUNCTION.
Pour des taches répétitives, il est pratique et judicieux d'écrire de courts programmes, qu'on sauvegarde, pour effectuer les calculs désirés. Il existe deux types de fichiers qui peuvent être programmés avec MATLAB: les fichiers SCRIPT et FUNCTION. Dans les deux cas, il faut lancer léditeur de fichier et sauvegarder le fichier avec l'extension .m. Le fichier SCRIPT permet de lancer les mêmes opérations que celles écrites directement à l'invite MATLAB. Toutes les variables utilisées dans un SCRIPT sont disponibles à l'invite MATLAB.

Cette approche est définie en Matlab par les M-Files, qui sont des fichiers pouvant contenir les données, les programmes (scripts) ou les fonctions que nous développons. Pour créer un M-Files il suffit de taper la commande edit, ou tout simplement aller dans le menu: $File \Rightarrow New \Rightarrow M-Files$ (ou cliquer sur l'icône). Une fenêtre d'édition comme celle-ci va apparaitre:

Nous avons vu jusqu'à présent comment utiliser MATLAB pour effectuer des commandes ou pour évaluer des expressions en les écrivant dans la ligne de commande, par conséquent les commandes utilisées s'écrivent généralement sous forme d'une seule instruction (éventuellement sur une seule ligne)

Cependant, il existe des problèmes dont la description de leurs solutions nécessite plusieurs instructions, ce qui réclame l'utilisation de plusieurs lignes. Comme par exemple la recherche des racines d'une équation de second degr(́avec prise en compte de tous les cas possibles). Une collection d'instructions bien structurées visant à résoudre un problème donné s'appelle un programme. Dans cette partie, on va présenter les mécanismes d'écriture et d'exécution des programmes en MATLAB.

Opérateurs de comparaison et opérateurs logiques

1. Les opérateurs de comparaison sont:
   - $==$: égale á (X==Y);
   - $>$: Strictement supérieur à $(X > Y)$;
   - $<$: Strictement inférieur à $X < Y$;
   - $>=$: supérieur ou égale à $X \geq Y$;
   - $<=$: inférieur ou égale à $X \leq Y$;
   - $\sim=$: différent de $X \sim= Y$.

Les opérateurs logiques sont:

- $\&$: et (and) $(X \& Y)$,
- $|$: ou (or) $(X|Y)$,
- $-$: Non(not) X $(-X)$

Les entrées et sorties

Entrée au clavier:

L'utilisateur peut saisir des informations au clavier grâce à la commande $x = input(....)$,

»$x = input$('saisir une valeur de x :'), Saisir une valeur de x: pi

Soritie à l'écran :

Pour afficher quelque chose à lécran, l'utilisateur peut utiliser la commande disp, qui affiche le contenu d'une variable (chaine de caractère, vecteur, matrice...).

»A=[1 3 4];

» disp(A)

Instructions de contrôle

Les instructions de contrôle sous Matlab sont très proches de celles existant dans d'autres langages de programmation.
L'instruction while:
L'instruction while répète l'exécution d'un groupe d'instructions un nombre indéterminé de fois selon la valeur d'une condition logique. Elle a la forme générale suivante:
»while (condition)
Ensenble d'instrucrions end.
Tant que l'expression de while est évaluée à vrai (true), l'ensemble d'instructions s'exécutera en boucle.

Faire un programme sous Matlab qui calcule la somme suivante:
$S = 1 + 2/2! + 3/3! + 4/4! + \ldots$ on arrête le calcul quand $S > 2,71$.

programme

```
»S=1; i=1;f=1;
»while S < 2,71
»i=i+1
»f=f*i
»S=S+i/f
end
```

L'instruction if:
L'instruction if est la plus simple et la plus utilisée des structures de contrôle de flux. Elle permet d'orienter l'exécution du programme en fonction de la valeur logique d'une condition. Sa syntaxe générale est la suivante:

programme

if (condition)
instruction 1
instruction 2
.....
instruction N
end
ou bien
if (condition)
ensemble d'instruction 1
else
ensemble d'instruction 2
end.

Si la condition est évaluée à vrai (true), les instructions entre le if et le end seront exécutées, sinon elles ne seront pas (ou si un else existe les instructions entre le else et le end seront exécutées). S'il est nécessaire de vérifier plusieurs conditions au lieu d'une seule, on peut utiliser des clauses elseif pour chaque nouvelle condition, et à la fin on peut mettre un else dans le cas ou aucune condition n'a été évaluée à vrai. Voici donc la syntaxe générale:

programme

if (expression 1)
Ensemble d'instructions 1
elseif (expression 2)
Ensemble d'instructions 2
.... elseif (expression n)
Ensemble d'instructions n
else
Ensemble d'instructions si toutes les expressions étaient fausses
end

Exemple1

Faire un programme sous MATLAB qui résout le problème suivant:

1. $y = x \quad si \quad x < 0;$
2. $y = x^2 \quad si \quad x > 0;$
3. $y = 10 \quad si \quad x == 0;$

Exemple 2

Créer un programme qui trouve les racines d'une équation de second degré désigné par: $ax^2 + bx + c = 0$. Voici le M-File qui contient le programme (il est enregistré avec le nom 'Equation2deg.m')

L'instruction switch: L'instruction switch exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé à une clause case qui définit si ce groupe doit être exécuté ou pas selon l'égalité de la valeur de ce case avec le résultat d'évaluation de l'expression de switch. Si tous les cases n'ont pas été acceptés, il est possible d'ajouter une clause otherwise qui sera exécutée seulement si aucun case n'est exécuté. Donc, la forme générale de cette instruction est:

switch (expression)
case valeur 1
Groupe d'instructions 1
case valeur 2
Groupe d'instructions 2
.....
case valeur n
Groupe d'instructions n
otherwise
end
Groupe d'instructions si tous les cases ont échoué

```
»x=input (’Entrez un nombre : ’);
switch(x)
case 0
disp(’x=0’)
case 10
disp(’x=10’)
case 100
disp(’x=100’)
otherwise
disp(’x n”est pas 0 ou 10 ou 100 ’)
end
```

L'instruction for

L'instruction for répète l'exécution d'un groupe d'instructions un nombre déterminé de fois. Elle a la forme générale suivante:
for variable $=$ *expression* $-$ *vecteur*
Groupe d'instructions
end
L'expression $-$ *vecteur* correspond à la définition d'un vecteur: début: pas: fin ou début: fin La variable va parcourir tous les éléments du vecteur défini par l'expression, et pour chacune il va exécuter le groupe d'instructions.
Une version plus explicite:
for i=initialvalue: **pas** :finalvalue
Bloc d'instructions
Le reste des instructions
Exemple:
Soit A=[1 0 1;-1 1 0; 1 0 0]. Calculer $\sum_{n=0}^{10} \frac{A^n}{n!}$

creation de fonction

On peut utiliser le caractère @ Pour créer les fonctions numériques:

Exemple:

»$f = @(x)2x^2 + 7x + 5;$

»f(0)=5

»f(1)=14.

En dimension n $n \geq 2)$ la variable x est considéré comme un vecteur.

Exemple:

»$f = @(x)x(1) * x(2) + (x(2))^2 + 3$

»$f([1,1]) = 5$

ou encore

» syms $f(x,y)$;

» $f(x,y) = x^2 + y^2;$

ou encore

» $f = @(x,y)x^2 + 3 * y^3.$

Dérivation intégration

Matlab est capable de dériver ou intégrer une des expressions symboliques.

»syms x;

»$diff(sin(x^2))$; *il affiche* $2*x*cos(x^2)$

»$diff(x^6,6)$; *il affiche* 720

»$int(-2*x/(1+x^2)^2)$; il affiche $1/(x^2+1)$

»$int(x*log(1+x),0,1)$; il affiche la valeur de $\int_0^1 x*log(1+x)dx$

»syms x t

»$diff(sin(x*t^2),t)$; il affiche $2*t*xcos(x*t^2)$

»$int(x/(1+t^2),t)$;il affiche $x*arctan(t)$

»$int(2*x,sin(t),1)$; il affiche $cos^2(t)$

Equations algébrique et systèmes d'équations

Matlab est capable de dériver ou intégrer une des expressions symboliques.

»syms a b c x t m;

»solve$(a*x+b*x+c==0,x)$

»solve$(sin(t+m)==0,t)$ il affiche $-m$

»solve$(a*x^2+b*x+c==0,x)$; il affiche les solutions de l'équation $ax^2+bx+c=0$

»syms x y;

» S=solve$(x+2y==1,x-y==5)$;

» $S=[S.x,S.y]$ il affiche $[11/3,-4/3]$

Pour les systèmes de Cramer $AX=B$ la solution est obtenue en utilisant le code suivant:

»$X=A\backslash B$.

Equations différentielles

Matlab est capable de dériver ou intégrer une des expressions symboliques.
»$ode(t) = diff(y,t) == exp(t) * y$ il affiche ode(t) $= diff(y(t),t) == exp(t) * y(t)$
($y' = e^t y$)
»sol=dsolve(ode): il affiche la solution de l'equation différentielle $C_1 \exp(\exp(t))$.
» syms y(t)
»$ode(t) = \cos(t) * diff(y,t) - \sin(t) * y == \sin(t);$
»$cond = y(0) == 1;$
»sol=dsolve(ode,cond) affiche la solution du problème de cauchy
$\cos(t)y' - \sin(t)y = \sin(t);\ y(0) = 1.$
» syms y(x) »ode(x)=diff(y,x,2)-2*diff(y,x)+y==(x+1)*exp(x);
»sol=dsolve(ode);

Systèmes différentiels

```
» syms u(t) v(t)
» ode1=diff(u,t)==3*u+2*v;
» ode2=diff(v,t)==u-4*v;
» ode=[ode1,ode2];
» S=dsolve(ode);
» S.u;
» S.v;
```

Intégrales Multiples

La syntaxe pour calculer une integrale double (valeur appochée) sur matlab est:
integral2(fun,xmin,xmax,ymin,ymax)
La syntaxe pour calculer une integrale triple (valeur appochée) sur matlab est:
integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)
Exemple 1: Calculer la surface de la region
$D = \{(x,y) \in \mathbb{R}^2 \quad 0 \leq x \leq 2 \text{ et } 0 \leq y \leq x+1\}$.
Exemple 2: calculer le volume de domaine
$D = \{(x,y,z) \in \mathbb{R}^3 \quad x \geq 0 \; y \geq 0 \text{ et } x^2+y^2+z^2 \leq 9\}$
Voir help de matlab.

Applications des méthodes numériques

L'analyse numérique est utilisée pour trouver des approximations à des problèmes difficiles tels que la résolution des équations non linéaires, l'intégration impliquant des expressions complexes. Elle est appliquée à une grande variété de disciplines telles que tous les domaines de l'ingénierie, de l'informatique, l'éducation, la géologie, la météorologie, et bien d'autres. Il y a des années, les ordinateurs à haute vitesse n'existaient pas, par conséquent, le calcul manuel exigeait beaucoup de temps et de travail laborieux. Mais maintenant que les ordinateurs sont devenus indispensables pour les travaux de recherche dans la science, l'ingénierie et d'autres domaines, l'analyse numérique est devenue une tache beaucoup plus facile et plus agréable.

Dans cette section, nous considérons une structure mecanique constitué de N éléments (structure dont les éléments sont des barres ou des poutres).
En elasticité linéaire, le principe des travaux virtuels nous permet d'établir dans le cas statique une matrice de rigidité pour l'élement barre et pour lélément poutre.
Ainsi, la matrice de l'élement barre en dimension 1 est donnée par:

$$\left[K_{(i)}\right] = \frac{EA}{L}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

et la relation qui lie les déplacements aux deux noeuds aux forces appliqués aux différents noeuds est:

$$\frac{EA}{L}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix}$$

La matrice de l'élement poutre en dimension 1 est donnée par:

$$\left[K_{(i)}\right] = \frac{EI}{L^3}\begin{pmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{pmatrix}$$

et la relation qui lie les déplacements et rotations aux deux noeuds aux forces et moments appliqués aux différents noeuds est donnée par:

$$\frac{EI}{L^3}\begin{pmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{pmatrix}\begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ M_1 \\ F_2 \\ M_2 \end{Bmatrix}$$

Passage en dimension deux

Soit u_1, v_1, u_2 et v_2 les composantes de déplacement aux noeuds 1 et 2 exprimés dans le repére global. Soit $F_{x_1}, F_{y_1}, F_{x_2}$ et F_{y_2} les composantes des forces aux noeuds 1 et 2 exprimés dans le repère global. Posons

$$\left[K'_{(i)}\right] = \frac{EA}{L}\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{et} \quad [P] = \begin{pmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & \cos\phi & \sin\phi \\ 0 & 0 & -\sin\phi & \cos\phi \end{pmatrix}$$

et soit $\left[K'_{iXY}\right]$ la matrice de rigidité locale de l'élément de barre dans le repère global (X, Y).

On montre que

$$\left[K'_{(i)XY}\right] = [P]^{-1}\left[K'_{(i)}\right][P].$$

Finalement

$$\left[K'_{(i)XY}\right] = \frac{EA}{L}\begin{pmatrix}[A] & -[A]\\ -[A] & [A]\end{pmatrix} \text{ ; avec } [A] = \begin{pmatrix}C^2\phi & C\phi S\phi\\ C\phi S\phi & S^2\phi\end{pmatrix},$$

où $C\phi = \cos\phi$ et $S\phi = \sin\phi$

Passage en trois dimensions

Si on note (X_1, Y_1, Z_1) et (X_2, Y_2, Z_2) les coordonnées des noeuds 1 et 2 dans le repère global (X, Y, Z), on a:

$$L^2 = (X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2$$

et la matrice de rigidité local de l'élément de barre dans le repère global est donnée par

$$\left[K'_{(i)XYZ}\right] = \frac{EA}{L}\begin{pmatrix} \alpha^2 & \alpha\beta & \alpha\gamma & -\alpha^2 & -\alpha\beta & -\alpha\gamma \\ \alpha\beta & \beta^2 & \beta\gamma & -\alpha\beta & -\beta^2 & -\beta\gamma \\ \alpha\gamma & \beta\gamma & \gamma^2 & -\alpha\gamma & -\beta\gamma & -\gamma^2 \\ -\alpha^2 & -\alpha\beta & -\alpha\gamma & \alpha^2 & \alpha\beta & \alpha\gamma \\ -\alpha\beta & -\beta^2 & -\beta\gamma & \alpha\beta & \beta^2 & \beta\gamma \\ -\alpha\gamma & -\beta\gamma & -\gamma^2 & \alpha\gamma & \beta\gamma & \gamma^2 \end{pmatrix}$$

où

$$\alpha = \frac{X_2 - X_1}{L}, \quad \beta = \frac{Y_2 - Y_1}{L} \text{ et } \gamma = \frac{Z_2 - Z_1}{L}.$$

Représentation de la structure

Definition

- Le nombre d'élément de la est noté N.
- Le nombre de noeud de la structure est noté N_d.
- Le nombre de noeud par élément est noté n.

Table de noeud

Indice du noeud	Coordonnées x	Coordonnées y	Coordonnées z
1	x_1	y_1	z_1
2	x_2	y_2	z_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
N_d	x_{N_d}	y_{N_d}	z_{N_d}

La table des noeuds contient tout simplement les coordonnées cartésiennes des noeuds sous la forme ci-dessus. Le numéro de noeud de cette table (allant de 1 à N_d) est un numéro global du noeud au sein du maillage, qu'il ne faut pas confondre avec l'indice d'un noeud au sein d'un élément.

Table de connectivité

Cette table représente essentiellement la manière dont les neuuds de la liste précédente sont reliés les uns aux autres pour former des éléments. On considère ici, pour simplifier, que tous les éléments sont du même type et donc qu'ils ont tous le même nombre de noeuds. Cette table se présente sous la forme d'un tableau à deux entrées $\mathrm{CONNEC}(i,j)$; où i (indice de ligne du tableau) correspond à l'indice de l'élément considéré (et donc i varie entre 1 et N), et j (indice de colonne) correspond à l'indice du noeud au sein de l'élément (et donc varie entre 1 et n).
Ensuite, le contenu du tableau est la liste des numéros des noeuds qui forment l'indice de l'élément j.

Principe

De manière générale, la taille des matrices de rigidité locale est

$$(n \times NDDL) \times (n \times NDDL)$$

où

- n est le nombre de noeud par élément;
- $NDDL$ est le nombre de degré de liberté par noeud.

Definition

Pour les éléments volumiques classiques (tétraèdres et hexaèdres, par exemple), on a $NDDL = 3$. Ce qui correspond aux composantes de déplacement dans les trois directions de l'espace.
Certains éléments peuvent également avoir (c'est le cas des éléments finis poutres et des coques, par exemple) des degrés de liberté connu en rotation.

Regle

1. La taille des matrices de rigidité locales est: $(n \times NDDL) \times (n \times NDDL)$.
2. La taille des vecteurs forces locaux est: $n \times NDDL$.
3. La taille des matrices de rigidité globales et des matrices de rigidité locales expansées est: $(N_d \times NDDL) \times (N_d \times NDDL)$.
4. La taille des vecteurs forces globales et des vecteurs forces locaux expansés est: $N_d \times NDDL$.

Vecteur de localisation

On définit pour chaque élément de la structure, un vecteur de localisation noté: $\langle \mathrm{LOC}_{(i)} \rangle$ pour l'élément (i).
Cet outil permet facilement d'envoyer les termes des matrices locales et vecteurs locaux au bon endroit dans les matrices locales expansées et vecteurs locaux expansés. La taille de ces vecteurs de localisation est: $n \times NDDL$.
La K-iéme composante du vecteur de localisation de l'élément (i) est définit grace à la formule

$$\mathrm{LOC}_{(i)}(k) = (\text{numéro noeud global} - 1)\, NDDL + \text{indice ddl},$$

où indice ddl est l'indice du degré du noeud considéré, il varie donc entre 1 et $n \times NDDl$.

On forme les valeurs des composantes du vecteur de localisation de l'élément (i) en bouclant sur les noeuds de l'élément (i), et sur les degrés de liberté de chaque noeud selon l'algorithme. Dans cet algorithme, k correspond à l'indice du composant du vecteur de localisation et sera incrémenté de 1 à $n \times NDDL$.

algorithme

[Formation du vecteur de localisation pour l'élément (i)]

$k = 1$

Pour j allant de 1 a n

Pour l allant de 1 a $NDDL$

$$\mathrm{LOC}_{(i)}(k) = (\mathrm{CONNEC}(i,j) - 1) \times NDDL + l$$

$$k = k + 1$$

Fin Pour

Fin Pour

Utilisation des vecteurs de localisation

Comme mentionné ci-dessous, on utilise les vecteurs de localisation pour former les matrices et les vecteurs locaux expansés.

Definition

On utilise, dans la suite du texte, les notations suivantes:

- $[K_i]$ est la matrice de rigidité locale de l'élément (i);
- $\{F_i\}$ est le vecteur force local pour l'élément (i);
- $[K_i^e]$ est la matrice de rigidité locale expansée pour l'élément (i);
- $\{F_i^e\}$ est le vecteur force local expansé pour l'élément (i).

On commence par initialiser toutes les composantes des matrices locales expansées et des vecteurs locaux expansés à 0.

Algorithme

[Formation du vecteur local expansé pour l'élément (i)]

Pour k allant de 1 a $n \times NDDL$

$$F_i^e\left(\mathrm{LOC}_{(i)}(k)\right) = F_{(i)}(k)$$

Fin Pour

Algorithme

[Formation de la matrice local expansé pour l'élément (i)]

Pour k allant de 1 a $n \times NDDL$

Pour l allant de 1 a $n \times NDDL$

$$K_i^e \left(\mathrm{LOC}_{(i)}(k), \mathrm{LOC}_{(i)}(l) \right) = K_{(i)}(k,l)$$

Fin Pour

Fin Pour

Le terme à l'intersection de la ligne k et la colonne l de $\left[K_{(i)}\right]$ est envoyé à l'intersection de la ligne $\mathrm{LOC}_{(i)}(k)$ et de la colonne $\mathrm{LOC}_{(i)}(l)$ de $\left[K^{e}_{(i)}\right]$.

Utilisation des matrices et vecteurs expansés

De cette manière, la contribution de chaque (i) à la forme intégrale faible peut maintenant s'écrire

$$W_{(i)} = \langle \delta U_{N_d} \rangle \left[- \left[K^e_{(i)} \right] \{ U_{N_d} \} + \left\{ F^e_{(i)} \right\} \right],$$

où le vecteur $\{U_{N_d}\}$ est tel que

$$\langle U_{N_d} \rangle = \langle u_1 \ v_1 \ w_1 \quad \cdots \quad u_n \ v_n \ w_n \rangle .$$

Ce vecteur regroupe tout simplement toutes les composantes des déplacements au noeud du maillage. On a globalement

$$\langle \delta U_{N_d} \rangle = \langle \delta u_1 \ \delta v_1 \ \delta w_1 \quad \cdots \quad \delta u_n \ \delta v_n \ \delta w_n \rangle .$$

On a:

$$W = \sum_{i=1}^{N} W_{(i)} = 0 \Longrightarrow \sum_{i=1}^{N} \left[\langle U_{N_d} \rangle \left(- \left[K^e_{(i)} \right] \{U_{N_d}\} + \{F^e_i\} \right) \right] = 0$$

$$\Leftrightarrow \langle \delta U_{N_d} \rangle \left[\sum_{i=1}^{N} - \left[K^e_{(i)} \right] \{U_{N_d}\} + \sum_{i=1}^{N} \{F^e_i\} \right] = 0.$$

Et finalement, on introduit $[K]$ et $\{F\}$, ce qui donne

$$\sum_{i=1}^{N} W_{(i)} = \langle \delta U_{N_d} \rangle \left[-[K]\{U_{N_d}\} + \{F\} \right].$$

Regle

On calcule la matrice de rigidité globale $[K]$ et le vecteur force global $\{F\}$ à partir des matrices de rigidité locales expanées $\left[K^e_{(i)}\right]$ et les vecteurs forces locaux expansés $\{F^e_i\}$ de la manière suivante:

$$[K] = \sum_{i=1}^{N} \left[K^e_{(i)}\right] \qquad \text{et} \qquad \{F\} = \sum_{i=1}^{N} \left\{F^e_{(i)}\right\}.$$

Ici, on a un système linéaire particulier puisque:

$$-[K]\{U_{N_d}\}+\{F\}=0 \Leftrightarrow [K]\{U_{N_d}\}=\{F\}.$$

- La matrice $[K]$ est complètement connue.
- Le vecteur $\{U_{N_d}\}$ est majoritairement inconnu, car on impose des déplacements et donc certaines composantes de $\{U_{N_d}\}$ sont connues.
- Le vecteur $\{F\}$ est majoritairement connu, car les seuls composantes de $\{F\}$ qui sont inconnues sont les composantes correspondant au déplacement imposé.

En général, les déplacements imposés correspondent aux appuis.