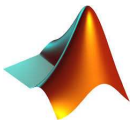
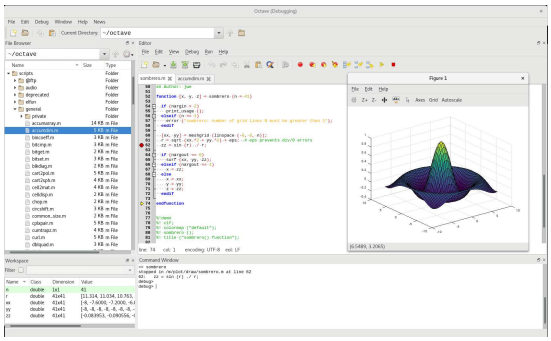


GNU Octave

Travaux Pratiques MATLAB : Les matrices et instructions de base

CONTENU :

- **Création et manipulation des matrices**
- **Recherche de valeurs et d'indices**
- **Les boucles et tests**
- **Matrices et limites**
- **Optimisation du code**



Opérations sur les matrices

`A'`	transposée de `A`
`rank(A)`	rang de `A`
`inv(A)`	inverse de `A`
`expm(A)`	exponentielle de `A`
`det(A)`	déterminant de `A`
`trace(A)`	trace de `A`
`poly(A)`	polynôme caractéristique de `A`
`eig(A)`	valeurs propres de `A`
`[U,D]=eig(A)`	vecteurs propres et valeurs propres de `A`
`+ -`	addition, soustraction
`* ^`	multiplication, puissance (matricielles)
`.* .^`	multiplication, puissance terme à terme
`A\b`	solution de $Ax = b$
`b/A`	solution de $xA = b$
`./`	division terme à terme

EXERCICE 1 - *Création de matrices et accès aux éléments de la matrice*

1. Créez la matrice A de dimension 4x4 telle que : $A = \begin{pmatrix} 1 & 4 & 5 & 7 \\ 7 & 8 & 9 & 10 \\ 2 & 26 & 14 & 56 \\ 78 & 11 & 43 & 12 \end{pmatrix}$
2. Que font les commandes suivantes ?
   (a) A(3,4)
   (b) A(1, :)
   (c) A(:,3)
   (d) A.'
   (e) $[L, C]$ = find (A > 75) et A(L(1),C(1))
   (f) $[L, C]$ = find (A > 50) et A(L(1),C(1)) et A(L(2),C(2))

EXERCICE 2 - *Opérations de base sur les matrices*

Soient les matrices A et B suivantes de dimension 3x3 :

$$A = \begin{pmatrix} 3 & 5 & 8 \\ 2 & 7 & 14 \\ 6 & 1 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 10 & 4 \\ 3 & 7 & 15 \\ 11 & 13 & 1 \end{pmatrix}$$

Réalisez les opérations suivantes :

1. C = A+B
2. D = A-D
3. E = A*B

EXERCICE 3 - *Inversion de matrices 2x2*

On se propose de déterminer la matrice inverse d'une matrice 2x2 en utilisant la formule permettant d'obtenir l'inverse d'une matrice, puis en utilisant la commande matlab ***inv(A)*** *donnant directement l'inverse d'une matrice A.*

Inversion de matrices

Si le déterminant d'une matrice A est non nul, alors A est inversible, son inverse étant donnée par :

$$A^{-1} = \frac{1}{\det A} (\operatorname{com} A)^{\mathsf{T}}$$

où $(\operatorname{com} A)^{\mathsf{T}}$ est la transposée de la comatrice de A.

Inversion des matrices 2×2

L'équation des cofacteurs ci-dessus permet de calculer l'inverse des matrices de dimensions 2×2 : si $ad - bc \neq 0$,

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \ \operatorname{com} A = \begin{pmatrix} d & -c \\ -b & a \end{pmatrix}, \ (\operatorname{com} A)^{\mathsf{T}} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix},$$

$$A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Exemple

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^{-1} = \frac{1}{-2} \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix}.$$

1. Créez la matrice A telle que : $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
2. Créez la comatrice comA correspondant à la matrice A (voir Rappels de l'encadré ci-dessus)
3. Calculez le déterminant de A par la commande matlab correspondante ; vérifiez que vous obtenez le même résultat par $\det(A) = ad - bc$
4. Enfin, obtenez la matrice inverse A^{-1} par la formule :

$$A^{-1} = \frac{1}{det(A)} (\text{comA})^T$$

5. Vérifiez que vous obtenez le même résultat par inv(A)
6. Vérifiez que vous obtenez la matrice identité $I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ en calculant $A.A^{-1}$

EXERCICE 4 - *Résolution d'un système d'équations par inversion de matrices*

On se propose de trouver les valeurs de x,y,z d'un système d'équations en utilisant l'outil matriciel sous Matlab.

Soit le système à 3 équations suivant :

$$\begin{cases} x + y + 2z &= 3 \\ x + 2y + z &= 1 \\ 2x + y + z &= 0 \end{cases}$$

1. Traduire le système d'équations sous une forme matricielle $AX = Y$ avec :

   A : matrice 3x3 à déterminer

   $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, $Y = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$

2. Trouver les valeurs de x,y et z par inversion de la formule précédente : $X = A^{-1}Y$

EXERCICE 5 - *Diagonalisation d'une matrice*

La diagonalisation d'une matrice consiste à générer à partir d'une matrice A, une matrice D dont seuls les éléments diagonaux sont non nuls :

$D = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$ (exemple ici avec matrice de dimension 3x3)

Pour trouver la matrice diagonale, il faut connaître évidemment la matrice A de départ et la matrice des vecteurs propres V (obtenue à partir de la matrice A). On constitue la matrice D par la formule :

$$D = V^{-1}AV$$

À quoi peut servir de diagonaliser une matrice ?

1. Si on est amené à calculer A^4 (par exemple) : ceci peut être fastidieux si on applique des multiplications matricielles successives $A.A$ puis $A.A^2$ et enfin $A.A^3$

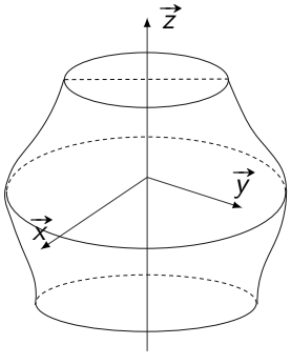
   $\Rightarrow$ En remarquant que $A = VDV^{-1}$, on peut facilement vérifier que :

   $A^4 = (VDV^{-1})(VDV^{-1})...(VDV^{-1}) = VD^4V^{-1}$ où D^4 est simplement $D^4 = \begin{pmatrix} \lambda_1^4 & 0 & 0 \\ 0 & \lambda_2^4 & 0 \\ 0 & 0 & \lambda_3^4 \end{pmatrix}$

   $\Rightarrow$ Ce qui rend le calcul de A^4 plus facile
2. En mécanique, chercher à diagonaliser la matrice d'inertie d'un solide revient à chercher des propriétés de symétrie et d'axes de révolution

Opérateur d'inertie

Solide avec une symétrie de révolution



Si l'axe $(O, \vec{z})$ est un axe de révolution matérielle, le solide possède alors une infinité de plan de symétrie orthogonaux. Les produits d'inertie sont donc tous nuls et la matrice est diagonale dans toute base contenant l'axe de révolution et en tout point de cet axe.

$$\overline{\overline{\mathcal{I}_O(S)}} = \begin{pmatrix} A & 0 & 0 \\ 0 & A & 0 \\ 0 & 0 & C \end{pmatrix}_{O,\mathcal{B}}$$

La commande $[V, D]$ = eig(A) permet d'obtenir respectivement :

- V : matrice des vecteurs propres
- D : matrice diagonale associée à A

Vérifier que vous retrouvez la matrice diagonale D par :

$$D = V^{-1}AV$$

EXERCICE 6 - *Programmes simples utilisant la boucle for et la condition if*

1. En utilisant la boucle for, écrire un programme qui affiche tous les nombres pairs de 0 à 20
2. En utilisant le reste d'une division par 2 (commande *mod()*), écrire un programme qui indique à l'écran si un nombre entré par l'utilisateur est pair ou impair.

EXERCICE 7 - *Boucle for et convergence de suites*

Soit la suite définie par :

$$\begin{cases} u_0 &= 1 \\ u_n &= 0.5(u_{n-1} + a/u_{n-1}) \end{cases}$$

avec $a \in \mathbb{R}^+$

1. Proposez un programme qui calcule tous les éléments de la suite dans un vecteur u_n, voici le corps du programme

```
for n = ...........
     u(n) = ..............
end
```

2. On prendra $a = 16$ et on prendra $N = 10^5$ avec N l'indice du dernier terme de la suite.

3. Prévoir la préallocation en mémoire du vecteur u par $u = \text{zeros}(1, N)$

4. Prévoir l'initialisation du premier terme : $u_0 \equiv u(1) = 1$ (les indices négatifs ou nuls ne sont pas autorisés dans Matlab).

5. Lancer le programme pour différentes valeurs de a

6. Conjecturer la limite de la suite u_n

7. En positionnant les mots clés *tic* et *toc* respectivement au début et à la fin d'un jeu d'instructions, on peut mesurer le temps d'exécution de ce jeu d'instructions : mesurez le temps d'exécution de votre programme

On se propose, maintenant de modifier le programme ci-dessus, en n'utilisant que deux variables un_1 qui stockera l'élément précédent de la suite (u_{n-1}) et un qui calculera l'élément courant de la suite (u_n)

1. Voici le corps du programme

```
a = 16;
N = 10^5;
un = 1;
for n = ...........
     un_1 = ............;
     un = ............;
end
```

2. Lancer le programme pour $a = 16$

3. Comparer le temps d'exécution avec le code précédent : vous devez conclure que le programme impliquant les termes d'un vecteur est plus long que celui impliquant simplement les deux variables un_1 et un.

En effet, la lecture des termes dans un vecteur (lecture du terme $u(n-1)$) et l'écriture du résultat dans un élément d'un vecteur (écriture du résultat dans $u(n)$) est coûteux en termes de temps de calcul.

EXERCICE 8 - *Boucle for versus matrices*

Soit le code suivant permettant de générer une sinusoide sous forme d'un vecteur :
» t = 0 :0.01 :10 ;
» y = sin(t) ;
» N = length(y) ;

On cherche à réaliser la dérivée de la fonction y(t) = sin(t) :

$$y'(t) = \frac{dy}{dt} = \frac{y(t+dt) - y(t)}{dt}$$

On peut réaliser cette opération de dérivée $y_d(t) = y'(t)$ sous MATLAB,
soit en vectoriel :

dt = t(2)-t(1) ;
y_d = (y(2 :N)-y(1 :N-1))/dt ;

La ligne du dessus s'interprète comme :

	y(2)	y(3)	y(4)	...	y(N)
-					
	y(1)	y(2)	y(3)	...	y(N-1)
=	dy(1)	dy(2)	dy(3)	...	dy(N)
/dt	dy(1)/dt	dy(2)/dt	dy(3)/dt	...	dy(N)/dt
=	y_d(1)	y_d(2)	y_d(3)	...	y_d(N)

soit en scalaire :

```
dt = t(2)-t(1) ;
for i = 1 :N-1
    yd(i) = (y(i+1)-y(i))/dt ;
end
```

1. Programmez ces deux programmes et comparer les temps d'exécution et conclure : vous devriez conclure que le programme avec la boucle *for* et instructions "lecture/écriture" des termes d'un vecteur (lecture $y(i)$ et $y(i+1)$ et écriture dans $y_d(i)$ est plus coûteux en termes de charge de calcul que le programme faisant la simple différence y_d = (y(2 :N)-y(1 :N-1))/dt

EXERCICE 9 - *Somme avec la boucle while*

Lorsque l'on veut calculer une somme de termes d'une suite du type $\sum_{n=1}^{N} U_n$*, on utilise souvent la boucle for :*

- On initialise la somme en dehors de la boucle : $S = 0$ (en général)
- On calcule dans la boucle le terme à ajouter à la somme précédente : $U(i)$
- On additionne le terme précédent à la somme
- Le résultat est réinjecté dans la somme : $S = S + U(i)$
- Et ainsi de suite ...

Le programme ci-dessous permet de calculer la somme $\sum_{n=1}^{N} 1/n^2$. Cette série converge vers $\pi^2/6$. On utilise une variante de la boucle *for* avec la boucle *while* :

```
N = 1e4 ;
S = 0 ;
i = 1 ;

while ( abs(S-pi^2/6) >= 0.1 )
    U = 1/i^2 ;
    S = S + U ;
    i = i+1 ;
end

disp(['La valeur de S est egale a ',num2str(S)]) ;
```

1. Que réalise le programme ci-dessus ?

2. Modifier le programme pour s'approcher de $\pi^2/6$ à 10^{-3} près.