

Travaux Pratiques n°1 :

1 Opérations sur les matrices

Opérations arithmétiques Les opérations arithmétiques sont $+$, $-$, $*$, $/$ et pour la puissance $^$ ainsi que l'usage des parenthèses et les priorités respectives sont standard. En tapant une opération après `>>`, **Matlab** retourne la réponse dans la variable `ans`. Par exemple

```
>> 2 + 3/(4*5)
ans =
2.1500
>> 2 + 3 ^ 2
ans =
11
>>
```

Variables. Il n'est pas nécessaire de déclarer le type des variables. **Matlab** devine si la variable x est un réel ou un vecteur en fonction de ce qu'il y a à droite de l'égalité.

```
>> x = 3+ 9^7
x =
4782973
>> a29 = 29
a29 =
29
>>
```

Il est possible d'utiliser les variables dans les opérations.

Vecteurs et matrices. Nous fabriquons un vecteur ligne par

```
v1 = [1 2 3]
v1 =
1 2 3
>>
```

Pour un vecteur colonne nous ajoutons des `;` pour passer à la ligne

```
vc = [1; 2; 3]
vc =
1
2
3
>>
```

Et donc pour les matrices

```
>> a = [1 2 3; 4 5 6 ]
a =
```

```
1 2 3
4 5 6
>>
```

On accède aux éléments de ces variables de façon naturelle en tapant $a(i,j)$. Par exemple

```
>> vl(1)
ans =
1
>> vc(2)
ans =
2
>> a(2,3)
ans =
6
>>
```

Exercices: Pour vous entraîner effectuer les opérations suivantes et commenter les résultats.

- Créer la matrice A donnée par

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad u = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 2 \end{pmatrix}.$$

- calculer A^{-1} à l'aide de `inv(A)`, le conditionnement `cond(A,1)`, `cond(A)`, `cond(A, 2)`, la transposée A^T à l'aide de `B=A'`, la taille à l'aide de `[n,m] = size(A)`, la norme 1, 2 et infini à l'aide des commandes `norm(A,1)`, `norm(A,2)`, `norm(A,inf)`.
- calculer le produit $A*u$, puis $A \setminus b$. Que constatez-vous?
- créer une matrice ne contenant que des 0, 1 et l'identité pour $n = 4$

```
>> n =4;
>> 0 = zeros(n,n)
>> I = ones(n,n)
>> Id = eye(n)
>>
```
- calculer le scalaire $b^T u$, la matrice $B = u b^T$, tester la commande `B(1,1 : 4)`, puis `B(2 : 4,2)` et commenter;
- tester les commandes `sqrt(B)`, `exp(A)`, `expm(A)`, `2*B`, `A*B` et `A.*B`; commenter.
- Créer la matrice M donnée par

$$M = \begin{pmatrix} +2 & -1 & 0 & 0 \\ -1 & +2 & -1 & 0 \\ 0 & -1 & +2 & -1 \\ 0 & 0 & -1 & +2 \end{pmatrix},$$

en utilisant les commandes

```
>> A = diag(2*ones(n,1),0) - diag(ones(n-1,1),1) -diag(ones(n-1,1),-1)
>>
```

- utiliser deux méthodes différentes pour résoudre le problème linéaire suivant

$$\begin{cases} 2x + y + 5z = 5 \\ 2x + 2y + 3z = 7 \\ x + 3y + 3z = 6. \end{cases}$$

2 Intervalles et fonctions

Pour les nombre complexes, on utilise aussi i ou j .

```
>> i
ans =
0. + 1.000000e+00 i
>> j
ans =
0. + 1.000000e+00 i
```

On peut aussi générer un vecteur comme

```
>> 1:5
ans =
1 2 3 4 5
>> 1:0.33:2
1. 1.33 1.66 1.99
```

Pour calculer π , nous faisons

```
>> pi
ans =
3.141592653589793
```

Pour calculer la valeur d'une fonction classique en un point, nous faisons

```
>> cos(pi/2)
ans =
6.1232e-17
```

Exercice : tracer une fonction Pour tracer une fonction, il faut diviser un intervalle en sous-intervalles, puis on utilise `plot`

- créer un vecteur x contenant l'ensemble des points de $-\pi$ à π avec un écart de 0.1
- tracer la fonction $\cos(x)$
- à l'aide de `Help`, vous pouvez ajouter des commentaires sur le graphique.

3 Fichiers Matlab

Nous voulons créer un fichier `test.m` (utiliser **File** → **New** → **create a M-file**, puis le sauves **File** → **Save as**) et l'on écrit dedans

```
% script test.m
% cree un vecteur ligne delements entre 0 et pi
v = [0:pi/4:pi];
s =size(v);
% genere une matrice carree avec un vecteur ligne de i
a = v' * ones(s);
% calcule le cosinus de tout les elements
cos(a)
```

On peut executer cette série de commandes en tapant

```
>> test
```

Pour sauver un résultat de calcul en faisant

```
>> a = rand(5,10)
>> save matrice.dat a -ascii -double
```

4 Opérateurs logiques, boucles et tests

Matlab permet de faire des tests et des boucles comme dans les langages standards. Les `true` et `false` sont représentés par 1 et 0. Les tests logiques sont donnés par

```
x == y, x est-il egal a y?  
x ~= y, x est-il different de y?  
x > y, x est-il superieur a y?  
x >= y, x est-il superieur a y?
```

Par exemple

```
>> x = pi  
x =  
3.1416  
>> x ~= 3  
ans =  
1  
>> x == pi  
ans =  
1
```

Les opérateurs logiques permettant de combiner les tests sont `&` pour AND | pour OR et `~` pour NOT.
>> x ~= pi & x ~= 3

On utilise ces constructions logiques en partie pour faire des tests à l'aide de la syntaxe classique

```
if <test>  
<commandes>  
elseif <test>  
<commandes>  
else  
<commandes>  
end
```

Pour construire les boucles, nous utilisons “while” qui signifie “tant que”

```
while <test>  
<commandes>  
end
```

L'autre possibilité pour faire des boucles est l'utilisations d'un compteur. La syntaxe est alors

```
for i=imin:imax  
<commandes>  
end
```

5 Création de fonctions

Une fonction est un fichier ascii de nom `nomdefonction.m` de syntaxe

```
function [argument en sortie] = nomdefonction(arguments en entree)  
%  
% commentaires eventuels  
%  
commandes et fonctions Matlab a exécuter sur les arguments
```

Donnons un exemple pour calculer l'aire d'un triangle dont les longueurs de cotés sont `a`, `b`, et `c`.

```
function [A] = area(a,b,c)  
%  
% surface du triangle avec cotes de longueur a, b, et c
```

```
%
s= (a+b+c)/2 ;
A = sqrt(s*(s-a)*(s-b)*(s-c)) ;
end
```

On peut lors évaluer la surface d'un triangle en tapant

```
>> area(10,15,20)
ans =
72.6184
```

Exercice : la suite de Fibonacci La suite de Fibonacci est définie par

$$f_1 = 0, \quad f_2 = 1, \quad f_n = f_{n-1} + f_{n-2}, \quad n \geq 3.$$

- On souhaite disposer d'une fonction Matlab qui prend n et retourne en sortie f_n .

```
function f = Fib1(n)
%
% Calcule le neme terme de la suite de Fibonacci
%
if (n==1)
f = 0;
elseif (n==2)
f = 1;
else a = 0; b = 1
for i=3:n
c = a + b;
a = b; b = c;
end
f = c
end
```

- On souhaite renvoyer aussi tous les calculs intermédiaires.

```
function f = Fib2(n)
%
% Calcule tous les termes de la suite de Fibonacci
%
F = zeros(1,n);
for i=3:n
F(i) = F(i-1) + F(i-2);
end
f = F
```

- On souhaite garder seulement les deux derniers.

```
function f = Fib3(n)
%
% Calcule tous les termes de la suite de Fibonacci et garde 2 derniers
%
F = zeros(1,n);
for i=3:n
F(i) = F(i-1) + F(i-2);
end
f = [F(n-1) F(n)]
```

- On souhaite écrire une version récursive.

```
function f = Fib4(n)
%
% Calcule de maniere recursive
%
if (n==1)
f = 0
elseif (n==2)
f = 1
else
f = Fib4(n-1) + Fib4(n-2);
end
```

- On écrit une version par un calcul matriciel et on ne renvoie que le dernier terme.

```
function f = Fib5(n)
%
% Calcule de maniere matriciel
%
A = [ 0 1 ; 1 1];
y = A ^ n * [0; 1];
f = y(1)
```