```
myData = dlmread('donnees.txt');
```

### Conditions

```
if (condition)
        %instructions
elseif (condition)
        %instructions
else
        %instructions
end
```

### Boucles

On a la boucle **for** qui prend en argument une série :

```
for cpt = debut:fin %Le pas est possible
        %instructions
end

for val = sort(rand(1,8));
        %instructions
end
```

La boucle **while** et **do-while** :

```
while (x < 5)
        %instructions
end

do
        %instructions
until (x < 5)
```

Bien sûr, les instructions **continue** et **break** fonctionnent.

### Fonctions

Elles sont à définir au DÉBUT.
Ci-dessous, *valueX* sont des valeurs de retours et *argX* les arguments.

```
function [value1[, value2, ...]]
= name( [arg1[, arg2, ...]] )
    value1 = ... % pour le retour
    return; % pour sortir de la fonction
endfunction
```

Il y a aussi les fonctions anonymes, qui sont des fonctions sans nom stockées dans une variable.

```
f = @(x) x*x % soit x au carré
f(2)         % retourne 4
arrayfun(f, [2 3 4]) % appel du C
% ou Fortran dans le cas vecteur !
```

## Algèbres

Calcul d'une intégrale, il faut mettre la fonction en pointeur et renseigner les limites :

```
myFonc = @uneFonction; % e.g. @sin
quad(myFonc,0,pi); % fonc, début, fin
```

Calcule d'une valeur :

```
myFonc = @uneFonction; % e.g. @sin
feval(myFonc, arg); % fonc, valeur
%On peut aussi faire
myFonc(arg);
```

On peut analyser une fonction :

```
s = input('Entrez une fonction :','s');
myFonc = inline(s);
```

## Équations différentielles

On a dans le cas de **Lorentz** :

- $\frac{dx}{dt} = a(y(t) - x(t))$
- $\frac{dy}{dt} = cx(t) - x(t)z(t) - y(t)$
- $\frac{dz}{dt} = x(t)y(t) - bz(t)$

```
function [f] = lorentz(u,t)
        a = 10; b = 8/3; c = 28;
        f(1) = a * (u(2) - u(1));
        f(2) = c * u(1) - u(1) * u(3) - u(2);
        f(3) = u(1) * u(2) - b * u(3);
endfunction
t = linspace(0,50,1001);
initVal = [1; 1; 1];
[myRes] = lsode('lorentz', initVal, t);
plot3(myRes(:,1), myRes(:,2), myRes(:,3));
figure;
```



# #Jaizappé ... ... l'Octave

## Histoire

**Octave** est un projet **GNU**. Apparu au début des années 90. Ce projet est maintenu et développé par **John W. Eaton**.
**Octave** peut exécuter des scripts MatLab, ainsi que du code C++ compilé (*.oct*). Un script Octave est suffixé en *.m*.

## Bases

### Commentaire

Un commentaire est un message caché à l'exécution.

```
%Ceci est un commentaire
```

### Afficher du texte

Pour afficher du texte ou des données, on dispose de **disp** (Le résultat est mis en commentaire et *ans* signifie *réponse*) :

```
disp('Mon texte'); % Mon texte
maVariable = 4     % ans = 4
MAVARIABLE = 4     % ans = 4
c = 3 + 1          % ans = 4
disp(maVariable);  % 4
```

### Demander une valeur

```
a = input('Entrer la valeur de a :');
s = input('Entrer une chaine :', 's');
```

### Calculs basiques

```
a = 5 + 1; %addition
b = a * 3; %multiplication
c = b - 6; %soustraction
d = c / 1; %division
e = d ^ 3; %exposant
```

De même, on dispose de valeurs de référence :

- **pi** : 3.14
- **i** : Imaginaire $\sqrt{-1} = 1$
- **e** : Nombre d'Euler
- **inf** : Infini
- **NaN** : Not a Number

Ainsi que des fonctions indispensables :

| Description | Commande |
|---|---|
| Modulo de $x$ par $y$ | mod(x,y); |
| Cos, Sin, Tan | cos(x); sin(x); tan(x); |
| Les arcs | acos(x); asin(x); atan(x); |
| Exp., Log et $\text{Log}_{10}$ | exp(x); log(x); et log10(x); |
| Val. abs. et Racine | abs(x); sqrt(x); |
| Arrondi, ↗ et ↘ | round(x); ceil(x); et floor(x); |
| Aléatoire | rand; |

# Vecteurs et matrices

## Déclaration

Déclarations manuelles des valeurs :

```
%Un vecteur
v1 = [0 1 2 3];
v2 = 1:10    % debut:fin
v3 = 1:2:10 % debut:pas:fin
%Une matrice
m = [0 1;2 3];
```

Déclarations programmées :

| Description | Commande |
|---|---|
| Transposé de A et trace | A'; trace(A); |
| Mat. aléa. de $n \times n$ et $n \times m$ | rand(n); rand(n,m); |
| Matrice de 1 | ones(n); ou ones(n,m); |
| Matrice de 0 | zeros(n); ou zeros(n,m); |
| Mat. de 0 et diag. à 1 | eye(n); |
| Vect. de $a$ à $b$ à $n$ élém. | linspace(a, b, n); |
| Vect. de $10^a$ à $10^b$ à $n$ élém. | logspace(a, b, n); |
| Vect. et Mat. diag. de A | diag(A); et diag(diag(A)); |

## Calculs basiques

On peut calculer élément par élément : $C_{ij} = A_{ij} \bigstar B_{ij}$, il suffit de préfixer +, ∗, −, / et ^ d'un point :

```
C = A .+ B
```

Opération entre deux matrices :

```
C = A + B
```

## Manipulation de matrices

Modifier la taille d'une matrice, *e.g.* $2 \times 3 \rightarrow 1 \times 6$ :

```
A = rand(2,3);      % 2 lignes, 3 colonnes
B = reshape(A,1,6);% 1 ligne, 6 colonnes
```

Pour ordonner un vecteur :

```
A = sort(A); % Tri dans l'ordre croissant
```

Pour faire des rotations matricielles :

```
A = rot90(A);     % Rotation de 90 deg.
A = rot90(A, -2);% Rotation de -180 deg.
```

Récupérer des valeurs :

```
B = A(2,3)     % Récupère la val. en 2 3
C = A(2,[1,3])% Récupère en 2 1 et 2 3
D = A(:,3)     % Récupère la colonne 3
E = A(2,:)     % Récupère la ligne 2
```

Ajouter des vecteurs et matrices ensemble,
Par exemple on veut une matrice $5 \times 5$ :

```
A11 = rand(2,2);A12 = rand(2,3);
A21 = rand(3,2);A22 = rand(3,3);
A = [A11 A12; A21 A22];
```

Pour obtenir la matrice triangulaire ...

```
B = tril(A);    % ... inférieur
C = triu(A);    % ... supérieur
[L,U] = lu(A); % Décomposition LU
[Q,R] = qr(A); % Décomposition QR
```

Inverse, déterminant d'une matrice carrée :

```
AA = inv(A);
b = det(A);
```

Norme et conditionnement d'un vecteur ou d'une matrice :

```
c = norm(A);
d = cond(A); % norm(A) * norm(inv(A))
```

Valeurs propres et vecteur diagonale :

```
[VectPro, Diag] = eig(A);
```

# Graphiques

Pour afficher un graphique 2D, on utilise :

```
title('Mon titre');          %facultatif
label('label Y', 'label X'); %facultatif
plot(myData(:,1), myData(:,2));
figure; %sert à afficher la fenetre
```

Si l'on veut 2 séries de données :

```
hold on %prends 2 séries à la fois
plot(myData(:,1), myData(:,2));
plot(myData(:,1), myData(:,3));
figure;
```

On peut dessiner des grilles :

```
[X,Y] = meshgrid(debut:pas:fin);
```

On peut personnaliser le graphique avec des options :

| Descri. | Arg. | Descri. | Arg. | Descri. | Arg. |
|---|---|---|---|---|---|
| Noir | k | Rouge | r | Vert | g |
| Bleu | b | Magenta | m | Cyan | c |
| Croix | + x | Rond | o | Étoile | * |
| Point | . | Carré | s | Diamant | d |
| △ ▽ ◁ ▷ | ^ v <> | Pent. | p | Hexa. | h |

```
plot(myData(:,2), myData(:,3), 'ro');
figure;
```

On a aussi :

- bar(...); et barh(...); : Histogramme et Histo. Horizon.;
- semilogx(...); semilogy(...); : Log sur $X$ ou $Y$;
- loglog(...); : Log sur $X$ et $Y$;

Dessin d'une fonction sur pointeur :

```
myFonc = @fonction; % e.g. @sin
fplot(myFonc, [limit1,limit2]); %3eme arg :
figure;                          %couleur et forme
```

## Sous-graphique

```
subplot(1,2,1); %ligne, col, index
plot([0:10], sin([0:10]));
subplot(1,2,2); %ligne, col, index
plot([0:10], cos([0:10]));
figure;
```

# Polynômes

## Déclaration

Un polynôme se déclare par ses valeurs tel un vecteur, *e.g.* : $-4x^4 - x^3 + 3x + 2$ :

```
p = [-1, 0, 3, 2];
polyout(p, 'x')
% ans = - 1*x^3 + 0*x^2 + 3*x^1 + 2
```

Racine, dérivée et intégration :

```
q1 = roots(p);
q2 = polyder(p);
q3 = polyint(p);
```

Multiplication et division polynômiale :

```
result = conv(p, q)
[b, r] = deconv(y, a) % y = ab + r
```

# Scripts

## Lire un fichier

Le fichier doit se présenter sous forme *CSV* avec l'espace comme séparateur. On obtient une matrice :