

```
% phd.m
%
% author: Cecilia
% date: 09/08/05

load THESIS_TOPIC

while (funding==true)
  data = run_experiment(THESIS_TOPIC);
  GOOD_ENOUGH = query(advisor);
  if (data > GOOD_ENOUGH)
    graduate();
    break
  else
    THESIS_TOPIC = new();
    years_in_gradschool += 1;
  end
end
```



www.phdcomics.com



# INTRODUCTION A GNU OCTAVE

Ph. Truillet

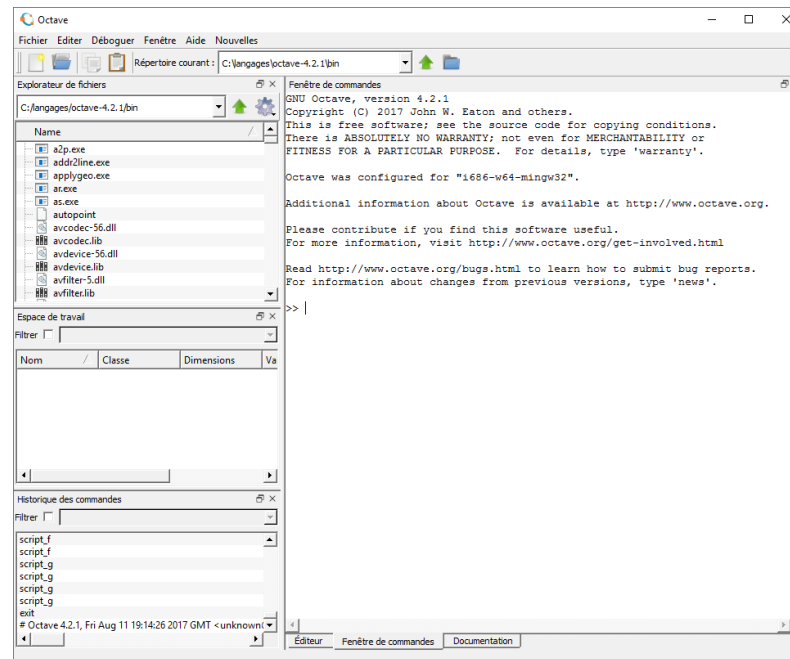
Août 2018, v. 1.1

# OÙ OBTENIR GNU OCTAVE

Vous pouvez le télécharger (gratuitement) depuis :

- <https://www.gnu.org/software/octave> (Download ...)

- Version au 09 août 2018 : **4.4.1**

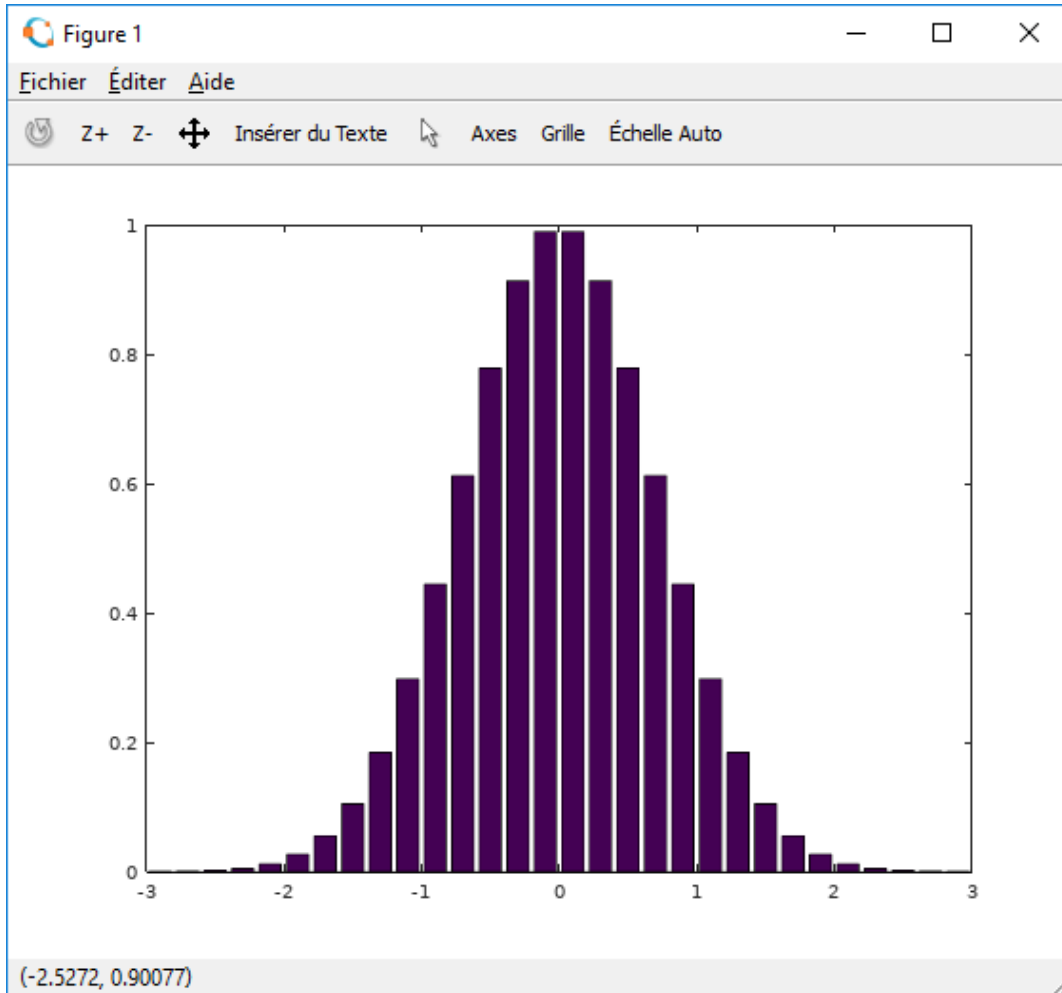


# GNU OCTAVE, C'EST QUOI ?

C'est version libre de **Matlab** (acronyme pour **MAT**rix **LAB**oratory)

- un langage interprété (comme Python) compatible Matlab
- de multiples possibilités de visualisation
- de nombreuses fonctions mathématiques embarquées (notamment pour la manipulation de matrices)
- facile à apprendre et simple à utiliser !

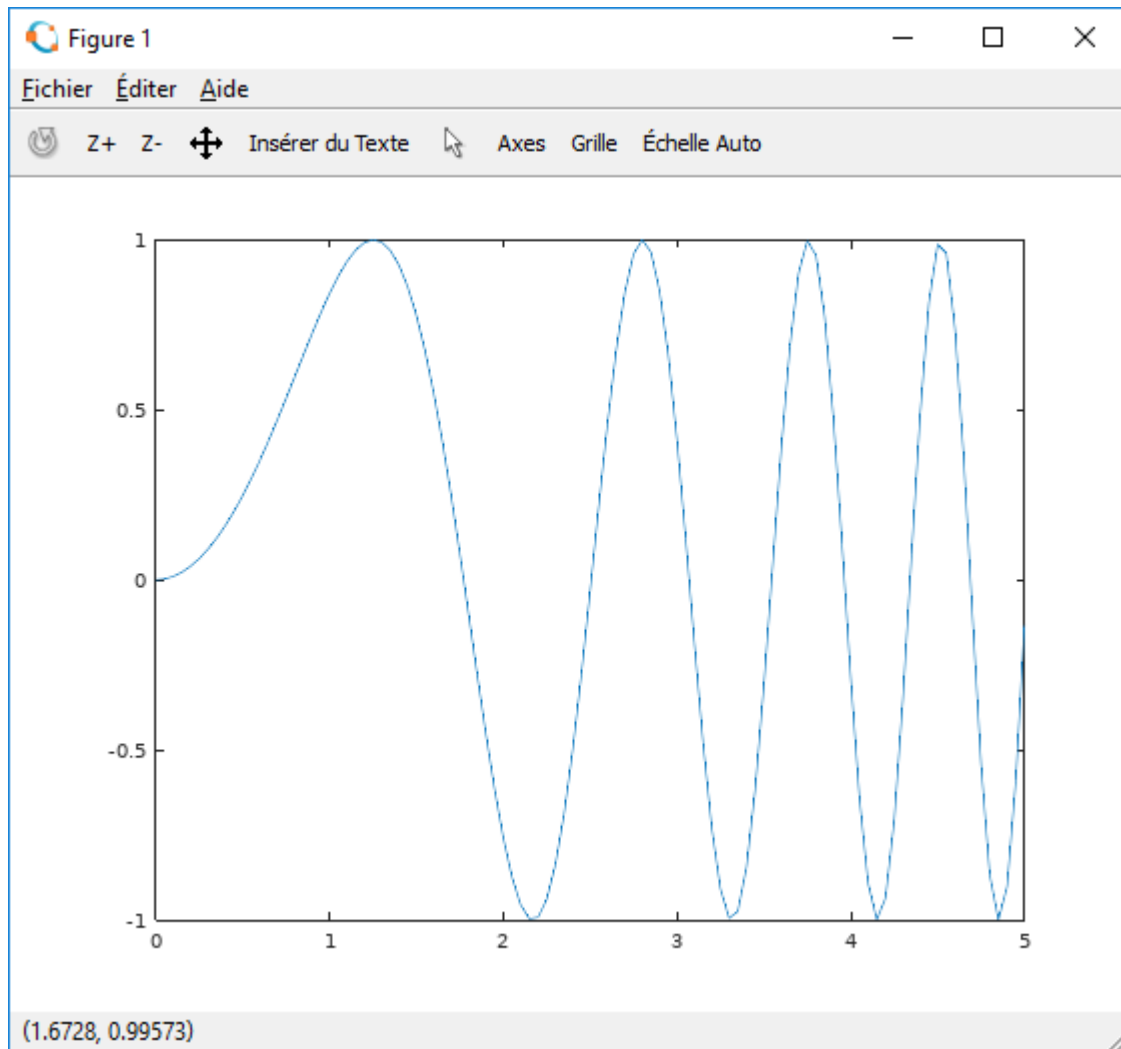
# QUELLES VISUALISATIONS ?



Des barres

```
x = -2.9:0.2:2.9;  
bar(x,exp(-x.*x));
```

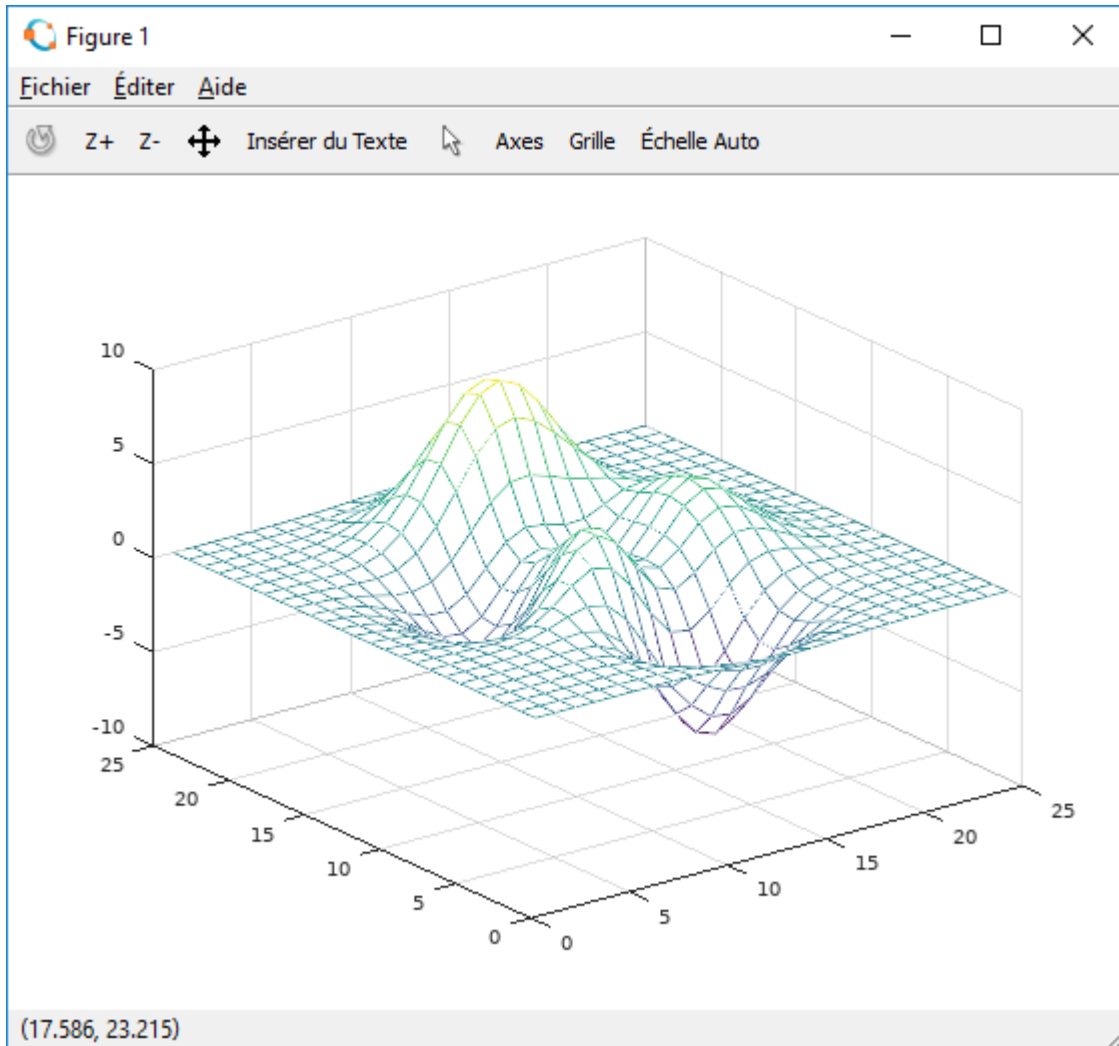
# QUELLES VISUALISATIONS ?



Des fonctions mathématiques

```
x=0:0.05:5;  
y=sin(x.^2);  
plot(x,y);
```

# QUELLES VISUALISATIONS ?



Des fonctions 3D

```
z=peaks(25);  
mesh(z);
```

# QUELLES VISUALISATIONS ?

Et plein d'autres choses ... comme :

- l'affichage de séries
- l'affichage de plusieurs fonctions dans une fenêtre,
- des multi-vues
- l'affichage d'équations différentielles,
- etc...

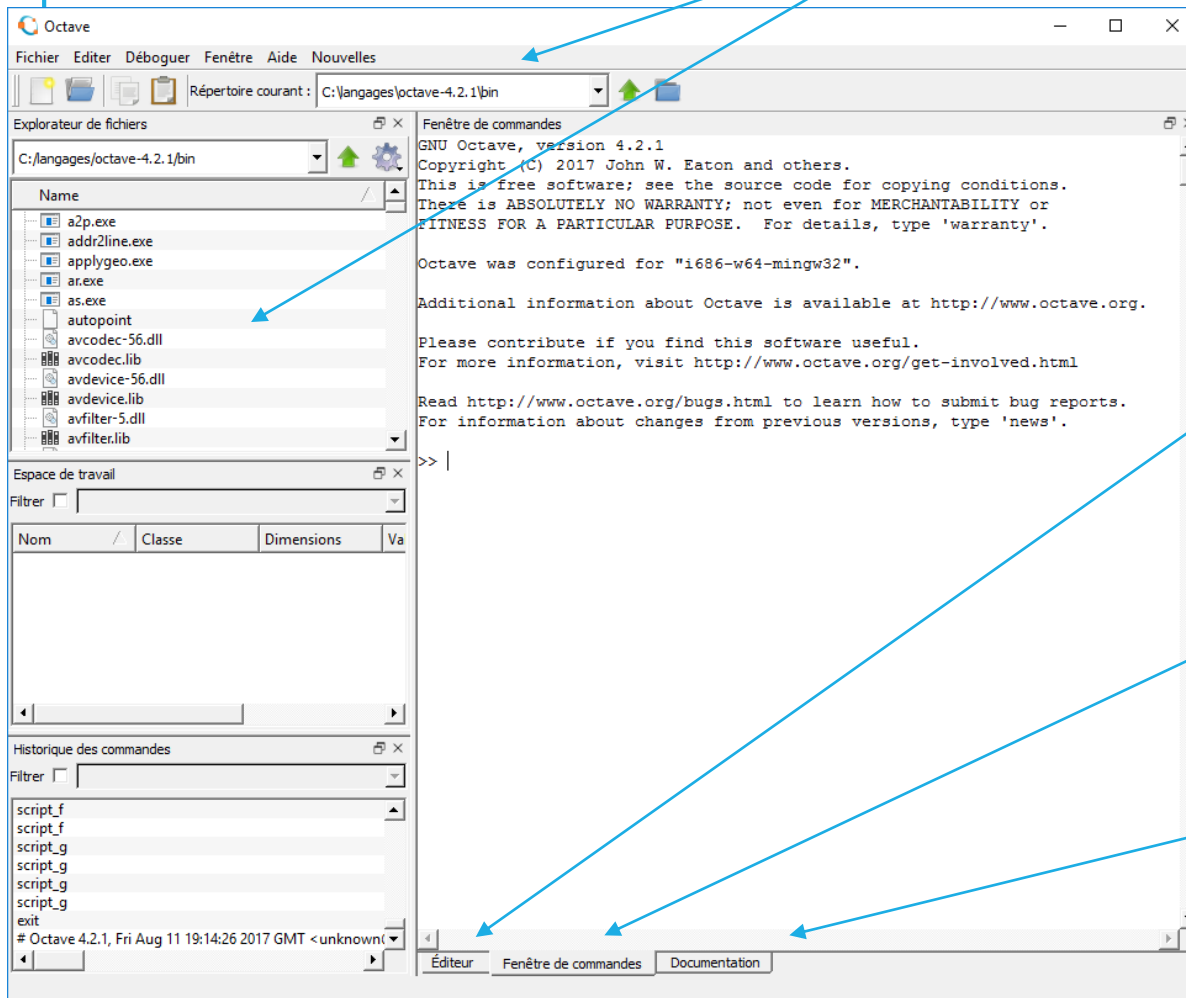
# GNU OCTAVE

Le répertoire courant utilisé (**TRES UTILE !**)

L'éditeur de texte  
(pour écrire des scripts  
ou des fonctions)

La fenêtre de commandes

L'aide du logiciel





# LES VARIABLES

Il n'est pas nécessaire de déclarer ses variables 😊 ni de les initialiser avant usage !

Il suffit de s'en servir quand on en a besoin !

>>

>> a=12; % la variable a contient la valeur 12

“Prompt”  
GNU  
Octave

Opérateur  
d'assignation

Supprime  
l'affichage du  
résultat de  
l'opération

Commentaire



Essayer de taper la même chose sans le ; de la fin

# LES VARIABLES

Les types manipulés sont nombreux dont les plus courants :

- Les booléens `true (1) et false (0)`
- Les entiers `12`
- Les « flottants » `12.1`
- Les nombres complexes `2+12*i (ou 2+12*j)`
- Les caractères `'a'`
- Les chaînes de caractères `"a" (attention 'a' ≠ "a")`
- (et d'autres que l'on va voir par ailleurs)

Par ailleurs, GNU Octave définit l' $\infty$  (`Inf`) et le résultat de la division zéro par zéro (`NaN` : **Not a Number**)

# LES VARIABLES

Pour voir le contenu d'une variable, il suffit de taper son nom

```
>> a
```

```
a = 12
```

Vous pouvez aussi manipuler des variables stockées dans l'espace de travail

```
>> b = 10;
```

```
>> c = a + b
```

```
c = 22
```

# COMMANDES UTILES

- **clc** : clear console : efface la console
- **clear a** : efface le contenu de la variable a
- **clear all** : efface toutes les variables
- **help** <cmd> : cherche de l'aide pour la commande <cmd>
- **lookfor** <chaine> : cherche de l'aide qui contient la chaîne de caractères <chaine>
- **whos** : donne le contenu de l'espace de travail

```
Fenêtre de commandes
>> a = 12
a = 12
>> b=14
b = 14
>> whos
Variables in the current scope:

Attr Name      Size      Bytes  Class
====
a            1x1         8  double
b            1x1         8  double

Total is 2 elements using 16 bytes
>> |
```

# VECTEURS ET MATRICES

Ce sont des tableaux à une (vecteur) ou x dimensions (matrice)  $m \times n$

- On distinguera les vecteurs-ligne (1 colonne, m lignes) des vecteurs-colonne (n colonnes, 1 ligne)
- Les matrices à deux dimensions seront codées sous la forme ligne/colonne (m/n)

Par convention, les noms des **matrices** débiteront par une **lettre en majuscule** tandis que les **vecteurs et variables** le seront par une **lettre en minuscule**

En réalité, toutes les entités en GNU Octave sont des matrices !

# VECTEURS ET MATRICES

Définir une matrice est simple 😊

```
A = [1, 2 ; 3, 4]
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Accéder à un élément est aussi simple !

```
>> A(1, 2)
```

```
ans=2
```

1<sup>ère</sup> ligne

2<sup>ème</sup> colonne

Utiliser un « , » ou un « » pour  
séparer les colonnes

Utiliser un « ; » ou un « <CR> » pour  
séparer les lignes

# VECTEURS ET MATRICES

## Il existe plusieurs façons de créer des vecteurs

Créer un vecteur avec n intervalles fixes

```
>> x = 0:0.5:pi % créer un ensemble de données entre 0 et pi  
avec un intervalle de n=0,5
```

Créer un vecteur avec m intervalles égaux

```
>> x = linspace(0,pi,7) % m=7 intervalles entre 0 et pi
```

Créer un vecteur dans un espace logarithmique

```
>> x = logspace(1,2,7) % m=7 intervalles entre  $\log_{10}(1)$  et  
 $\log_{10}(2)$ 
```

# VECTEURS ET MATRICES

## Il existe plusieurs façons de créer des matrices

`zeros(m, n)` : créer une matrice  $m \times n$  remplie de 0

`ones(m, n)` : créer une matrice  $m \times n$  remplie de 1

`eye(m, n)` : créer une matrice  $m \times n$  identité

`randn(m, n)` : créer une matrice  $m \times n$  distribuée normalement (moyenne à 0 et variance à 1)

`magic(m)` : créer un carré magique (matrice carrée où la somme des éléments est la même pour chaque colonne, ligne et diagonale)

`pascal(m)` : créer une matrice de Pascal d'ordre  $m$



# VECTEURS ET MATRICES

**Un opérateur très important : « : »** (que l'on peut traduire par « à »)

`1:10` : créé un vecteur (ligne) de la valeur 1 à 10 (avec un pas de 1)

```
ans= 1 2 3 4 5 6 7 8 9 10
```

Si l'on veut définir le *pas*, il faut insérer la valeur entre les valeurs de départ et d'arrivée

`1:2:10` : créé un vecteur (ligne) de la valeur 1 à 10 (avec un pas de 1)

```
ans= 1 3 5 7 9
```



Essayer avec les valeurs

`x=0:0.01:2*pi`

# VECTEURS ET MATRICES

Cet opérateur est aussi utilisé pour sélectionner des plages de données

```
>> A (3,2:3)
```

```
ans= 1 7
```

A =

3	2	1
5	1	0
2	1	7

```
>> A(:,2)
```

```
ans= 2
```

```
1
```

```
1
```



Que se passe t-il si vous tapez A(:, :) ?

# VECTEURS ET MATRICES

A =

3	2	1
5	1	0
2	1	7

## Quelques opérateurs utiles

B =

1	3	1
4	9	5
2	7	2

- >> A'                    % calcule la transposée de A
- >> B\*A                % multiplie les matrices A et B
- >> B.\*A               % multiplie élément par élément les matrices A et B
- >> B/A                % divise les matrices A et B
- >> B./A               % divise élément par élément les matrices
- >> [B A]               % fusionne les matrices (horizontalement)
- >> [B; A]              % fusionne les matrices (verticalement)



Créer les matrices A and B et essayez les différents opérateurs  
(**Attention**, il peut y avoir des erreurs !)

# VECTEURS ET MATRICES

**De manière globale, on peut appliquer les opérateurs mathématiques suivants sur les matrices et vecteurs**

**+** : addition

**–** : soustraction

**\*** : multiplication

**/** : division

**^** : exposant

**\** : division à gauche. L'opération  $A \setminus B$  est la même que  $\text{inv}(A) * B$  mais plus rapide

# VECTEURS ET MATRICES

A ceci, s'ajoutent des opérateurs « *spéciaux* » (sans équivalence mathématique mais **TRES** utiles).

Ce sont les opérations « *élément par élément* » caractérisées par un « . » avant l'opérateur mathématique :  $.^*$ ,  $. /$  et  $.^{\wedge}$

# VECTEURS ET MATRICES

**Il existe aussi de nombreuses fonctions de manipulation de vecteurs et de matrices**

`mean(a)` : calcule la moyenne d'un vecteur `a`

`max(a)`, `min(a)` : calcule le maximum et le minimum d'un vecteur `a`

`sum(a)` : somme les éléments d'un vecteur `a`

`sort(a)` : trie le vecteur `a`

`median(a)` : calculi la valeur médiane d'un vecteur `a`

# VECTEURS ET MATRICES

`std(a)` : calcule la deviation standard de a

`det(A)` : calculi le determinant d'une matrice carrée A

`dot(a,b)` : calcule le produit scalaire de 2 vecteurs

`cross(a,b)` : calcule le produit vectoriel de 2 vecteurs

`inv(A)` : inverse la matrice carrée A

# GRAPHIQUES

Afficher des données sous forme de graphiques est un point très important de GNU Octave.

Vous allez pouvoir afficher des données sous différentes formes : séries, fonctions 2D, 3D, surfaces, etc, etc ...

Généralement, il faut commencer par générer des données (vecteurs) sur les différents axes

Par exemple :

```
>> x = 0:0.01:2*pi;
```

```
>> y = sin(x);
```



# GRAPHIQUES

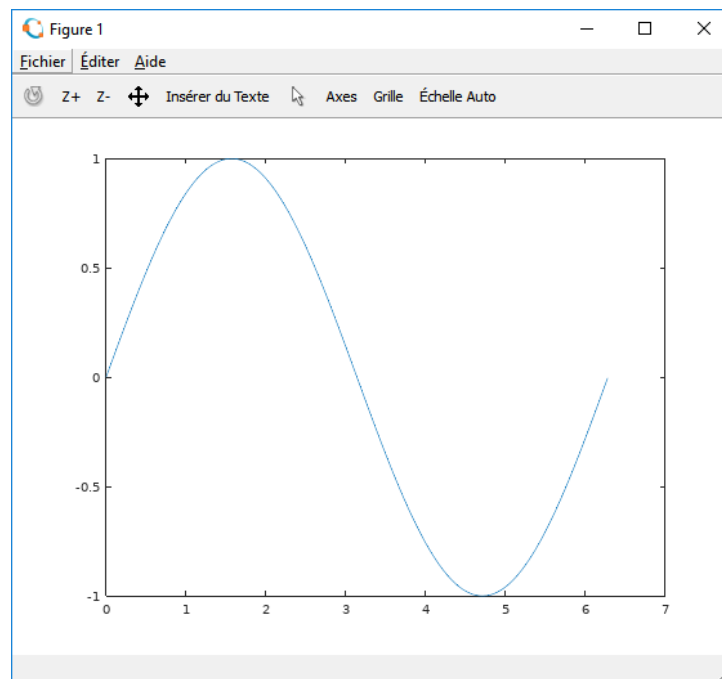
Enfin, on va pouvoir générer la figure, le plus souvent avec la fonction `plot`

```
>> plot(x,y)
```

La forme générale est :

```
plot(x_axis,y_axis,'style')
```

Où le style va permettre d'afficher les données avec différentes couleurs ou sous forme de +, o, etc...



# GRAPHIQUES

Il est bien évidemment possible d'ajouter des informations au graphique : nom des axes, titre ou légende grâce à différentes fonctions !

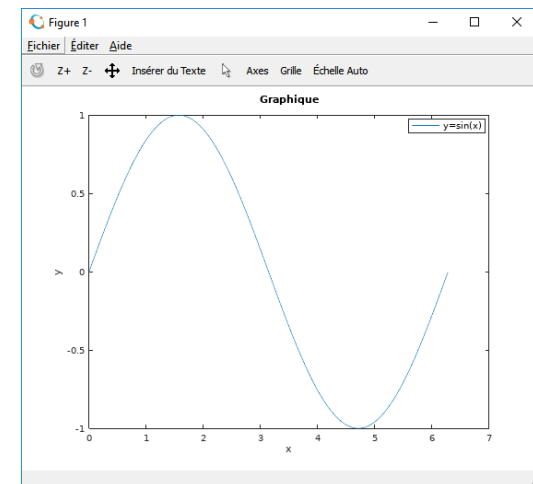
```
xlabel('nom de l'axe x')
```

```
ylabel('nom de l'axe y')
```

```
title('titre de la figure')
```

```
legend('légende du graphique')
```

```
fenêtre de commandes  
>> xlabel('x')  
>> ylabel('y')  
>> title('Graphique')  
>> legend('y=sin(x)')  
>> |
```



# GRAPHIQUES

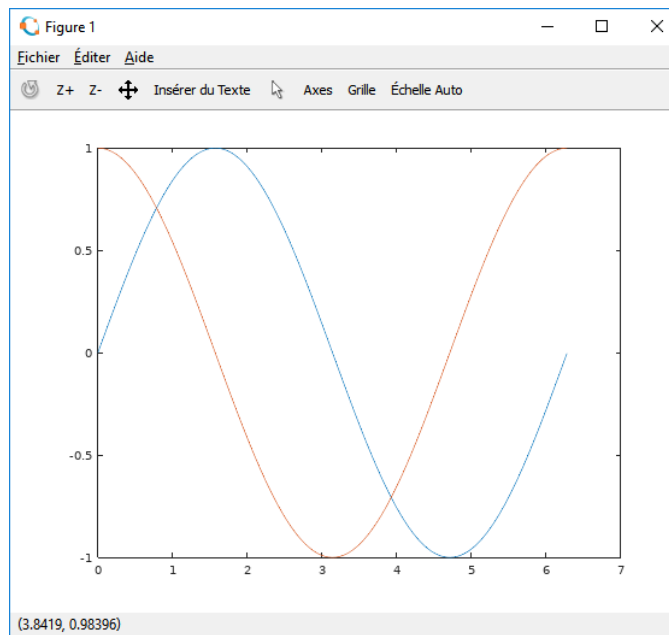
Il est possible de **superposer** plusieurs graphiques dans la même figure.

Par défaut, chaque appel à `plot` réinitialise l'affichage.

Néanmoins l'appel à la commande `hold on` permet de superposer les affichages (`hold off` annulera cette action)

Ex :

```
>> x = 0:0.01:2*pi;  
>> y1 = sin(x);  
>> y2 = cos(x);  
>> plot(x,y1);  
>> hold on;  
>> plot(x,y2);
```



# GRAPHIQUES

De la même manière, il est possible d'afficher plusieurs graphiques séparés dans une même figure en utilisant la commande `subplot`

Tout d'abord, il faut sélectionner où vous voulez afficher vos données

`subplot(2,2,1);`

nombre de lignes

nombre de colonnes

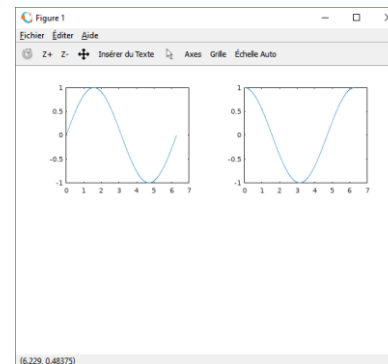
sous-graphique sélectionné

1	2
3	4

Puis utiliser la commande `plot` comme vu précédemment

Ex :

```
>> subplot(2,2,1);  
>> plot(x,y1);  
>> subplot(2,2,2);  
>> plot(x,y2);  
>> |
```



# GRAPHIQUES

De nombreux affichages sont possibles avec différentes fonctions remplaçant `plot` comme :

`stem` : affiche des données de manière discrète

`plot3` : affiche une courbe dans un espace 3d (attention à l'axe z !)

`bar`, `barh` : affiche les données sous forme de barres verticales ou horizontales,

`pie` : affiche les données sous forme de camembert

...

# GRAPHIQUES

On peut enfin afficher des surfaces 3D suivant un processus un peu spécifique détaillé ici :

1. créer des tableaux suivant les axes x et y

```
x= 0:0.1:10
```

```
y= 0:0.1:10
```

2. créer un réseau de mailles en x,y

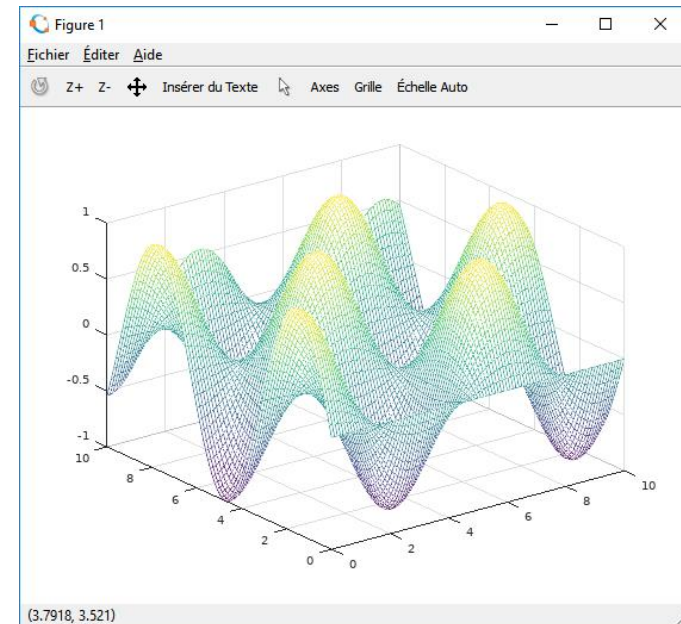
```
[X,Y] = meshgrid(x,y)
```

3. appliquer la fonction  $z=f(x,y)$  sur ce maillage

```
Z = cos(X).*sin(Y);
```

4. afficher les données

```
mesh(X,Y,Z); % ou surf(X,Y,Z)
```



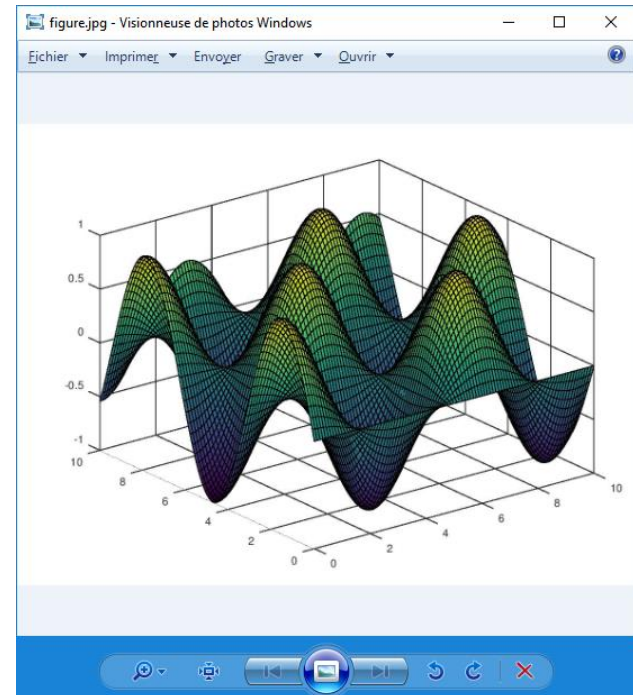
# GRAPHIQUES

Dernière étape intéressante, il est possible de sauver les figures dans différents formats (jpeg, png, eps, ...) avec la fonction `saveas`

Ex :

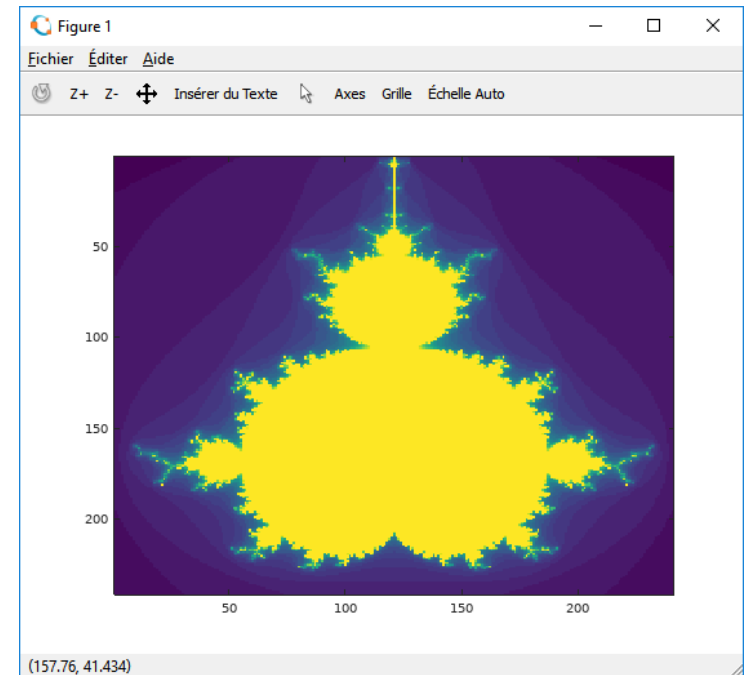
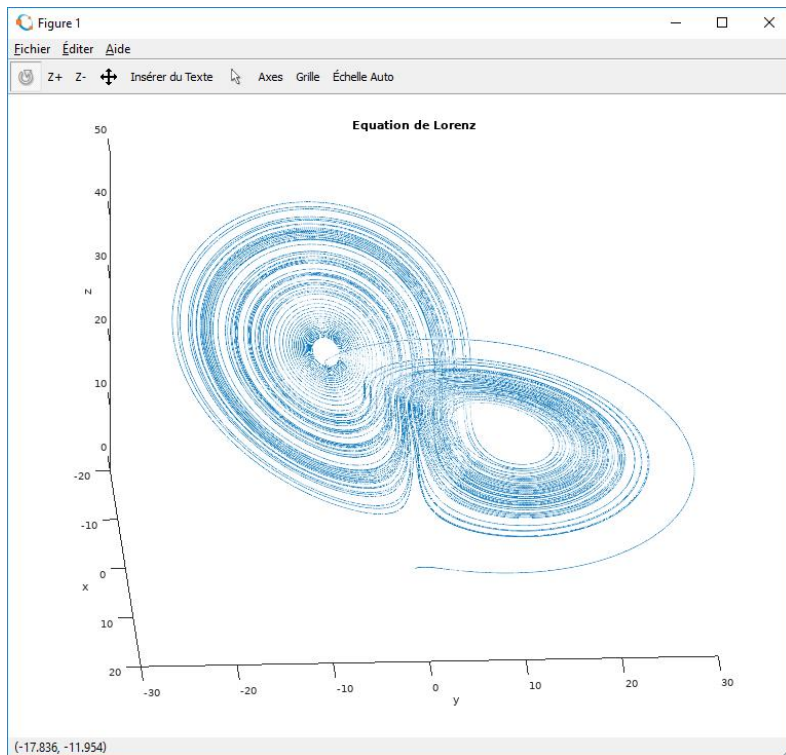
```
f1= mesh(X,Y,Z);  
saveas(f1,'figure.jpg');
```

**Attention :** le processus peut être assez long !



# GRAPHIQUES

Nous verrons plus tard comment afficher des équations différentielles ou des fractales (mais ce n'est pas compliqué)



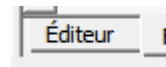


# SCRIPTS ET FONCTIONS

Tout ce qui a été vu précédemment a été saisi au clavier commande par commande ... Mais le plus intéressant reste d'automatiser ces processus de saisie grâce à des **scripts** (que l'on apparentera à des **programmes**) et des **fonctions** (sous-programmes génériques)

scripts et fonctions sont sauvés dans un **fichier texte** avec l'extension **.m** (matlab)

Ces fichiers peuvent être créés, ouverts, modifiés dans l'onglet « *Editeur* » de GNU Octave



# SCRIPTS ET FONCTIONS

Les scripts sont équivalents aux commandes tapées directement dans la fenêtre de commandes mais rassemblées dans un fichier et ré-appelables autant de fois que nécessaire.

L'avantage est de pouvoir prototyper (essayer) différentes méthodes sans avoir à tout retaper !

Si le nom du script est « **monscript.m** », il suffit de taper « **monscript** » dans la fenêtre de commande pour que les différentes actions soient exécutées.

De plus, les scripts peuvent utiliser les structures fondamentales de l'algorithmique à savoir les répétitions (`while`, `for`, ...) et les sélections (`if`, `switch`, ...) en sus des actions

# SCRIPTS ET FONCTIONS

Les fonctions, même si elles ressemblent à des scripts sont différentes de ceux-ci.


Le mot clé « `function` » est utilisé. **Le nom de fichier est de celui de la fonction.**

Les fonctions ne manipulent pas directement des **variables effectives** (qui contiennent des valeurs) mais des **variables formelles** (qui servent à expliciter les calculs dans la fonction). Ces variables sont **nécessaires** et **suffisantes** pour effectuer le travail

Enfin, les fonctions acceptent des paramètres en entrées et fournit des valeurs en sortie.

# SCRIPTS ET FONCTIONS

Un exemple de définition d'une fonction

```
de6faces.m   
1 function val=de6faces()  
2     % renvoie une valeur entre 1 et 6 - tir de dé  
3     val =ceil(rand()*6); % renvoie la valeur entière  
4                     % par excès entre ]0,6[ soit entre [1,6]  
5 endfunction
```

Et son appel dans la fenêtre de commandes

```
>> de6faces()  
ans = 5  
>> de6faces()  
ans = 1  
>> |
```

# SCRIPTS ET FONCTIONS

L'objectif sera la plupart du temps :

1. d'analyser le problème posé
2. de décomposer son problème en sous-problèmes
  - Le cas échéant, écrire des fonctions génériques utiles à la résolution
3. d'écrire un script GNU Octave qui résout ce problème (qui appellera un ensemble de fonctions)
4. afficher, sauver et analyser les résultats (souvent sous forme graphique)

# SCRIPTS ET FONCTIONS

De très nombreuses fonctions et bibliothèques sont déjà implémentées dans GNU Octave

Au-delà de l'aspect informatique, il rend d'énormes services :

- comme vérifier ses calculs rapidement
- comparer différentes méthodes numériques
- effectuer des simulations dans différents domaines (mécanique, thermodynamique, électronique, ...)
- s'amuser un peu 😊

# POUR FINIR ...

```
x=[-2:.001:2];  
y=(sqrt(cos(x)).*cos(200*x)+sqrt(abs(x))-0.7).*(4-x.*x).^0.01;  
plot(x,y);
```

